In [1]:
```
# section 1 importing Libs:

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import os

# my style:
sns.set(style= "whitegrid")
```

In [2]:
```
from plotly import __version__
import cufflinks as cf

from plotly.offline import download_plotlyjs, init_notebook_mode, plot, iplot
init_notebook_mode(connected=True)

cf.go_offline()
```

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

2. Exploring Data set:

In [4]:
```
# head of DEFAULT data frame:
df.head(2)
```

Out[4]:

	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

In [5]:
```
# all unique enteries in respective columns:
df.nunique()
```

Out[5]:
```
lat            20843
lng            20864
desc          326362
zip              138
title            133
timeStamp     315575
twp               68
addr           33219
e                  1
dtype: int64
```

In [6]:
```
#droping non usefull columns or dummy cols:
del df["e"]
```

In [7]:
```
# information about the DEFAULT data frame:
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 326425 entries, 0 to 326424
Data columns (total 8 columns):
lat          326425 non-null float64
lng          326425 non-null float64
desc         326425 non-null object
zip          286835 non-null float64
title        326425 non-null object
timeStamp    326425 non-null object
twp          326310 non-null object
addr         326425 non-null object
dtypes: float64(3), object(5)
memory usage: 19.9+ MB
```

In [8]:
```
# as we can see timestamp column is an object not a date time type:
# coverting:

df["timeStamp"] = pd.to_datetime(df["timeStamp"])

df["timeStamp"].loc[0]
```

Out[8]:
```
Timestamp('2015-12-10 17:10:52')
```

3 Creating New Features and Columns for data analyis:

Creating columns like year, month, date etc on the basis of timeStamp column:

In [9]:
```
df["reason category"] = df["title"].apply(lambda x: x.split(":")[0])

# The 3 category of reasons for calls.
df["reason category"].value_counts()
```

Out[9]:
```
EMS        161441
Traffic    116065
Fire        48919
Name: reason category, dtype: int64
```

In [10]:
```
p = df["timeStamp"].loc[0]
print(p)
print (p.year)
print (p.month)
print (p.date())
print (p.dayofweek)
print (p.time())
```

```
2015-12-10 17:10:52
2015
12
2015-12-10
3
17:10:52
```

In [11]:
```
# adding new date time columns:

df["year"] = df["timeStamp"].apply(lambda x: x.year)

df["month"] = df["timeStamp"].apply(lambda x: x.month)

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

df["day of week"] = df["timeStamp"].apply(lambda x : x.dayofweek)

df["time"] = df["timeStamp"].apply(lambda x : x.time())
```

In [12]:
```
# Mapping day of week (numberic) column into categorical:
dow = {0:"Mon", 1:"Tues", 2:"Wed", 3:"Thurs" , 4:"Fri", 5:"Sat", 6:"Sun" }

df["Day of Week"] = df["day of week"].map(dow)
```

In [13]:
```
# Creating a column of the basis of sunlight
df["day/night"] = df["timeStamp"].apply(lambda x : "night" if int(x.strftime("%H")) > 19 else "day")
```

4 Basic Q/A section:

In [14]:
```
# Top 5 zipcode for 911 calls?
df["zip"].value_counts().head(5)
```

Out[14]:
```
19401.0    22136
19464.0    21588
19403.0    16488
19446.0    15793
19406.0    10800
Name: zip, dtype: int64
```

In [15]:
```
# Top 5 reason for 911 calls:
df["title"].value_counts().head(5)
```

Out[15]:
```
Traffic: VEHICLE ACCIDENT -    76179
Traffic: DISABLED VEHICLE -    23957
Fire: FIRE ALARM               18436
EMS: RESPIRATORY EMERGENCY     16602
EMS: FALL VICTIM               16438
Name: title, dtype: int64
```

In [16]:
```
# Top 5 township from where calls were recieved:
df["twp"].value_counts().head(5)
```

Out[16]:
```
LOWER MERION    28073
ABINGTON        20206
NORRISTOWN      18433
UPPER MERION    17276
CHELTENHAM      15026
Name: twp, dtype: int64
```

In [17]:
```
# Top 10 dates of receiving the maximum in calls in all the years:
df["date"].value_counts().head(10)
```

Out[17]:
```
2018-03-02    2187
2018-03-07     920
2018-03-03     917
2016-01-23     887
2016-02-24     673
2017-12-15     667
2016-01-24     657
2017-06-21     654
2018-01-05     612
2018-01-12     608
Name: date, dtype: int64
```

In [18]:
```
# Most busy year with total number of calls recieved:
df["year"].value_counts().head(1)
```

Out[18]:
```
2016    142360
Name: year, dtype: int64
```

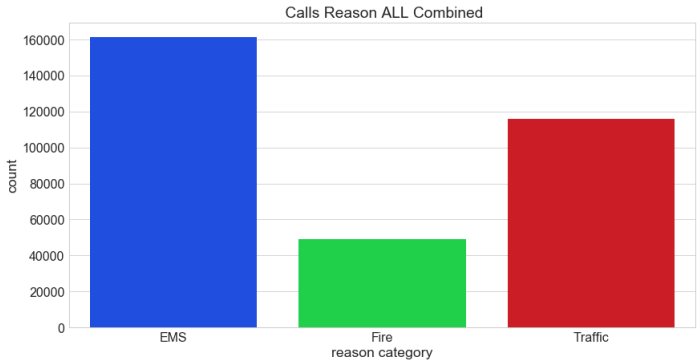
5 Data Visualizing and Analysing

In [19]:
```
# New customized Data Frame with additional features and columns:
df.head(1)
```

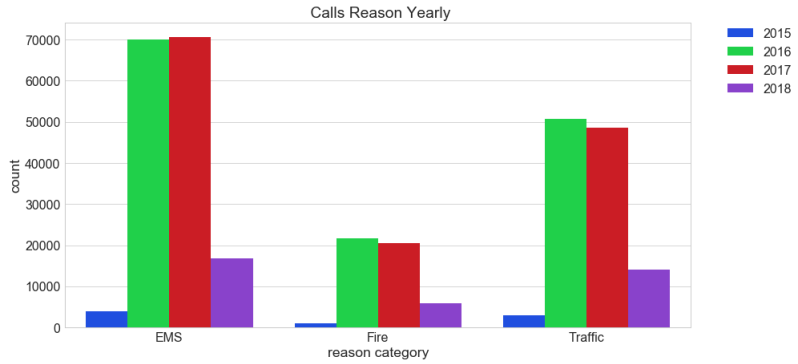
Out[19]:

	lat	lng	desc	zip	title	timeStamp	twp	addr	reason category	year	month	date	day of week	time	Day of Week	day/night
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	EMS	2015	12	2015-12-10	3	17:10:52	Thurs	day

In [20]:
```
# Plot for Category of reasons:

plt.figure(figsize=(14,7))
sns.set_context("paper", font_scale = 2)
sns.countplot(x= "reason category", data= df, palette="bright")
plt.title(" Calls Reason ALL Combined")
plt.show()
```



In [21]:
```
plt.figure(figsize=(14,7))
sns.set_context("paper", font_scale = 2)
sns.countplot(x= "reason category", data= df, palette="bright" ,hue= "year")
plt.title(" Calls Reason Yearly")
plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)
plt.show()
```



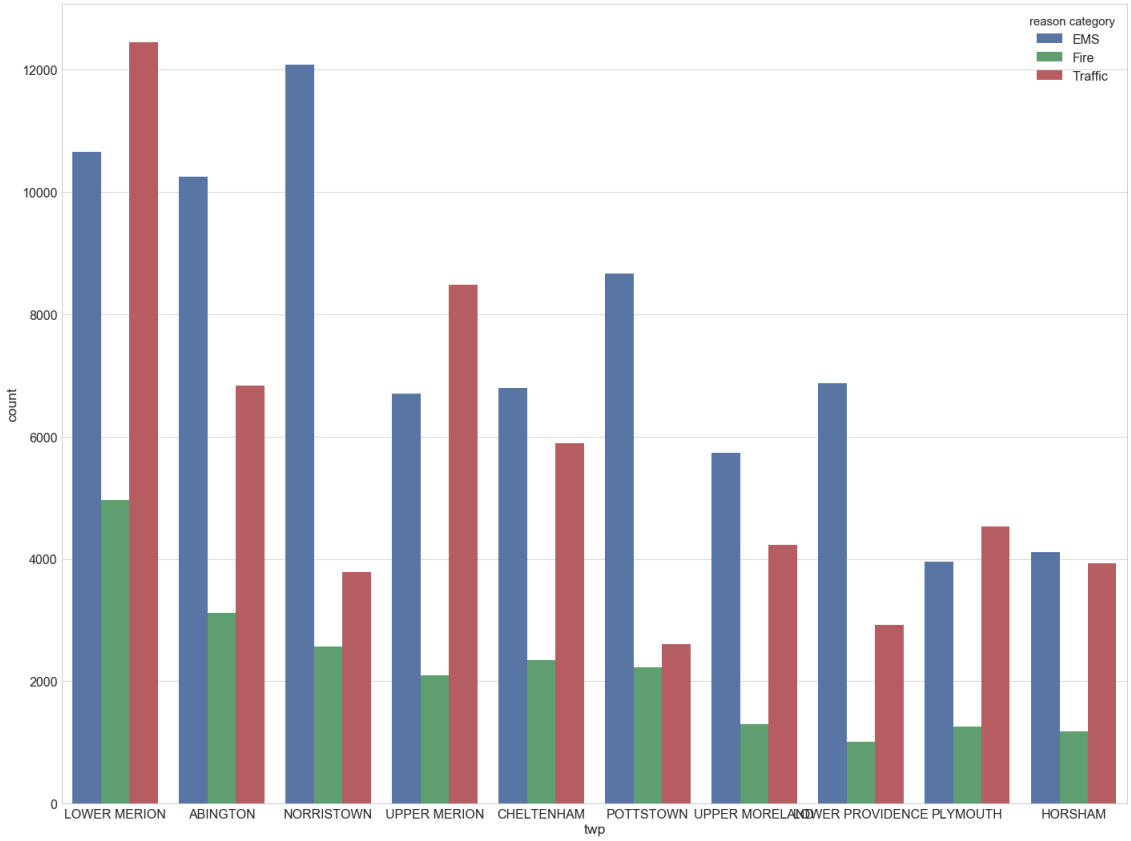
In [22]:
```
df[df["twp"]=="LOWER MERION"]["title"].value_counts()
```

Out[22]:
```
Traffic: VEHICLE ACCIDENT -             7745
Traffic: DISABLED VEHICLE -             2589
Fire: FIRE ALARM                        2475
Traffic: ROAD OBSTRUCTION -             1455
EMS: VEHICLE ACCIDENT                   1267
EMS: FALL VICTIM                        1053
EMS: RESPIRATORY EMERGENCY               942
EMS: CARDIAC EMERGENCY                   894
EMS: HEAD INJURY                         679
EMS: SUBJECT IN PAIN                     576
Traffic: HAZARDOUS ROAD CONDITIONS -     492
EMS: MEDICAL ALERT ALARM                 483
Fire: VEHICLE ACCIDENT                   455
EMS: UNKNOWN MEDICAL EMERGENCY           448
Fire: FIRE INVESTIGATION                 446
EMS: SYNCOPAL EPISODE                    438
Fire: GAS-ODOR/LEAK                      339
EMS: GENERAL WEAKNESS                    323
Fire: CARBON MONOXIDE DETECTOR           302
EMS: UNCONSCIOUS SUBJECT                 290
EMS: CVA/STROKE                          275
EMS: SEIZURES                            275
EMS: ABDOMINAL PAINS                     272
EMS: HEMORRHAGING                        272
EMS: ALTERED MENTAL STATUS               250
EMS: OVERDOSE                            212
EMS: NAUSEA/VOMITING                     203
Fire: BUILDING FIRE                      181
EMS: DIZZINESS                           167
Fire: VEHICLE FIRE                       156
                                         ...
Fire: DEBRIS/FLUIDS ON HIGHWAY             6
Fire: CARDIAC ARREST                       6
Fire: BURN VICTIM                          5
Fire: TRANSFERRED CALL                     4
Fire: RESCUE - WATER                       4
EMS: RESCUE - ELEVATOR                     3
EMS: STABBING                              3
EMS: RESCUE - WATER                        3
EMS: HAZARDOUS MATERIALS INCIDENT          2
Fire: RESCUE - TECHNICAL                   2
EMS: APPLIANCE FIRE                        2
EMS: FIRE SPECIAL SERVICE                  2
EMS: RESCUE - TECHNICAL                    2
Fire: HAZARDOUS MATERIALS INCIDENT         2
EMS: FIRE INVESTIGATION                    2
EMS: AMPUTATION                            2
Fire: EMS SPECIAL SERVICE                  1
Fire: POLICE INFORMATION                   1
EMS: TRAIN CRASH                           1
Fire: TRAIN CRASH                          1
EMS: SHOOTING                              1
EMS: FIRE ALARM                            1
EMS: BOMB DEVICE FOUND                     1
EMS: ELECTROCUTION                         1
EMS: ELECTRICAL FIRE OUTSIDE               1
EMS: UNKNOWN TYPE FIRE                     1
EMS: DROWNING                              1
Fire: CARDIAC EMERGENCY                    1
Fire: S/B AT HELICOPTER LANDING            1
Fire: PUMP DETAIL                          1
Name: title, Length: 95, dtype: int64
```

In [23]:
```
'''plt.figure(figsize=(20,15))
sns.countplot(x= df["twp"]=="LOWER MERION", data=df ,order = df["title"].value_counts().index[:10],
              hue ="reason category")
plt.tight_layout()'''
```

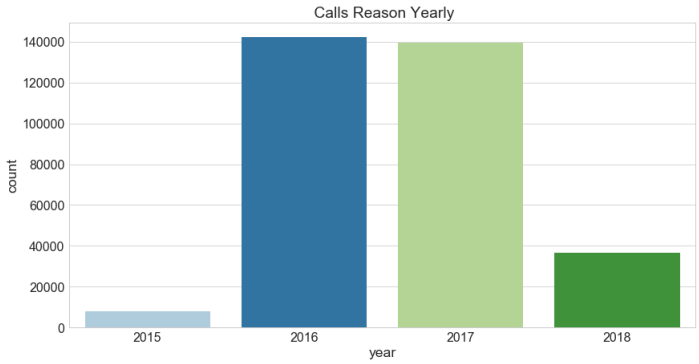
Out[23]:
```
'plt.figure(figsize=(20,15))\nsns.countplot(x= df["twp"]=="LOWER MERION", data=df ,order = df["title"].value_counts().index[:10],\n              hue ="reason category")\nplt.tight_layout()'
```

In [24]:
```
plt.figure(figsize=(20,15))
sns.countplot(x='twp', data=df , order= df['twp'].value_counts().index[:10],
              hue ="reason category")
plt.tight_layout()
```

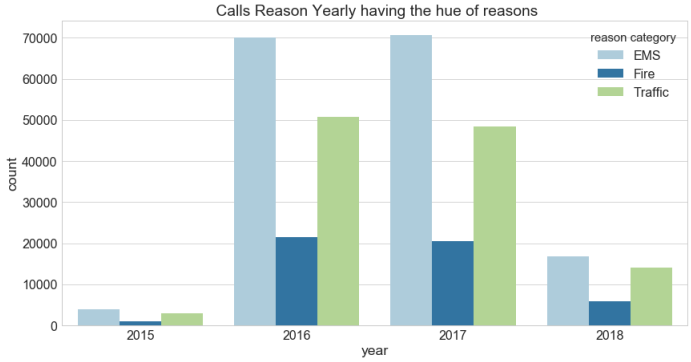


In [ ]:
```
```

In [25]:
```
# Plot for calls recieved yearly:
plt.figure(figsize=(14,7))
sns.set_context("paper", font_scale = 2)
sns.countplot(x= "year", data= df, palette="Paired")
plt.title(" Calls Reason Yearly")
plt.show()
```



In [26]:
```
plt.figure(figsize=(14,7))
sns.set_context("paper", font_scale = 2)
sns.countplot(x= "year", data= df, palette="Paired", hue = "reason category")
plt.title(" Calls Reason Yearly having the hue of reasons")
plt.show()
```

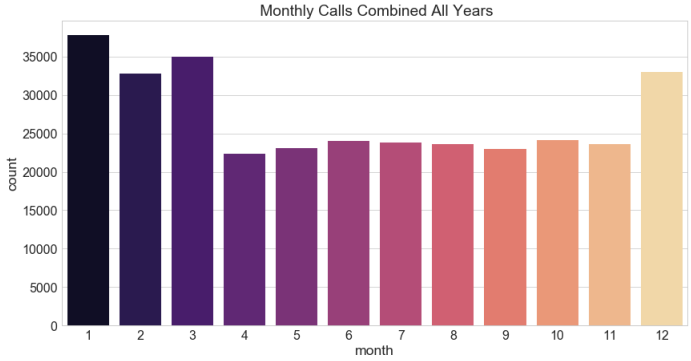


In [ ]:

```
```

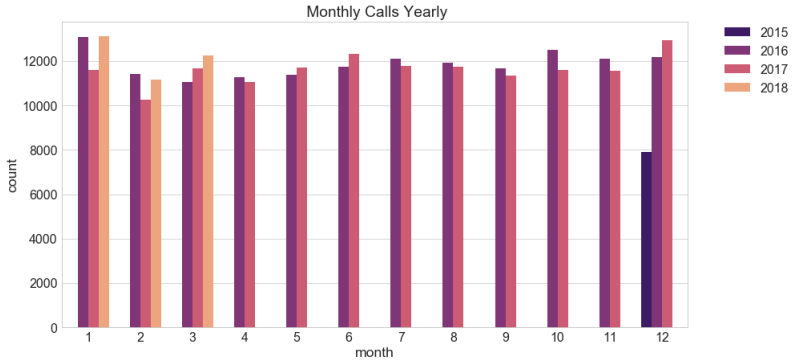
In [27]:

```
# Plot for calls recieved monthly combined of all years:
plt.figure(figsize=(14,7))

sns.set_context("paper", font_scale = 2)
sns.countplot(x= "month", data= df, palette="magma")

plt.title(" Monthly Calls Combined All Years")
plt.show()
```



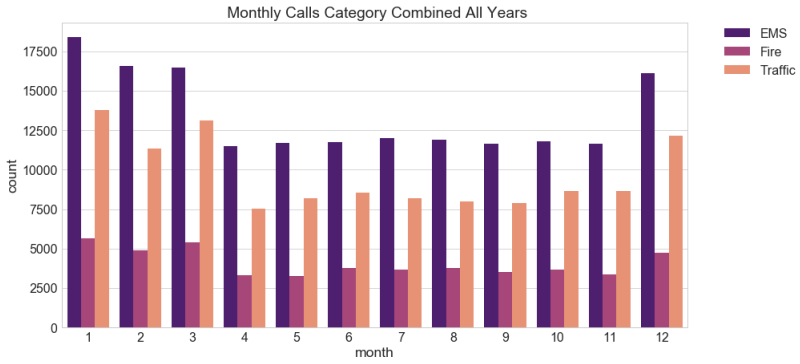
In [28]:

```
plt.figure(figsize = (14,7))

sns.set_context("paper", font_scale=2)
sns.countplot(data= df, x= "month", hue= "year", palette="magma")

plt.title(" Monthly Calls Yearly")
plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)
plt.show()
```



In [29]:

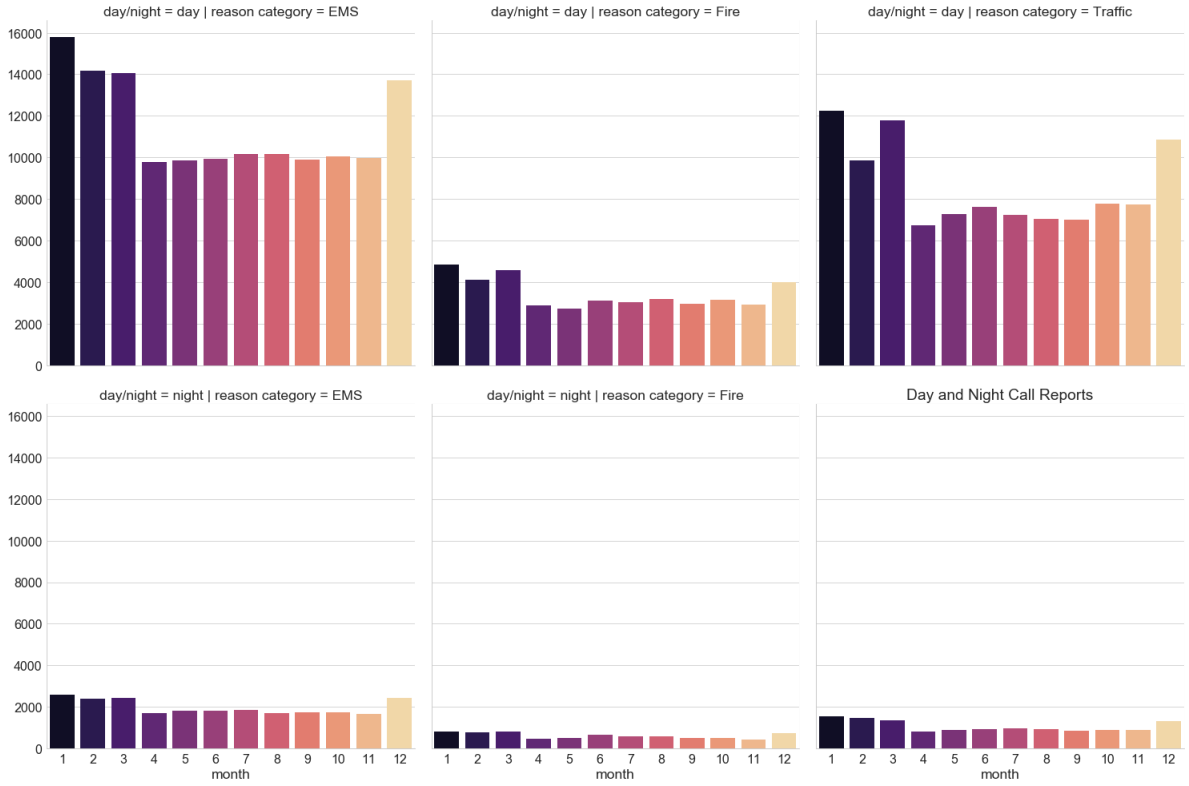
```
plt.figure(figsize=(14,7))
sns.set_context("paper", font_scale = 2)
sns.countplot(x= "month", data= df, palette="magma", hue= "reason category")
plt.title(" Monthly Calls Category Combined All Years")
plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)
plt.show()
```



In [30]:
```
# Day and Night Call Reports
g = sns.FacetGrid(df, row ="day/night", col="reason category", size = 7)
g.map(sns.countplot, "month", palette= "magma")
plt.title("Day and Night Call Reports")
```

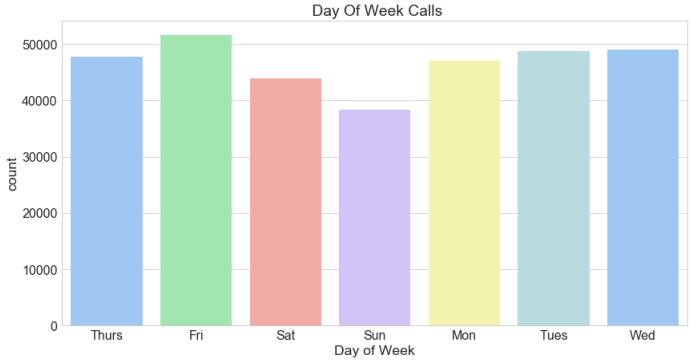
```
/Users/keshavrastogi/anaconda3/lib/python3.6/site-packages/seaborn/axisgrid.py:703: UserWarning:

Using the countplot function without specifying `order` is likely to produce an incorrect plot.
```

Out[30]:
```
Text(0.5,1,'Day and Night Call Reports')
```

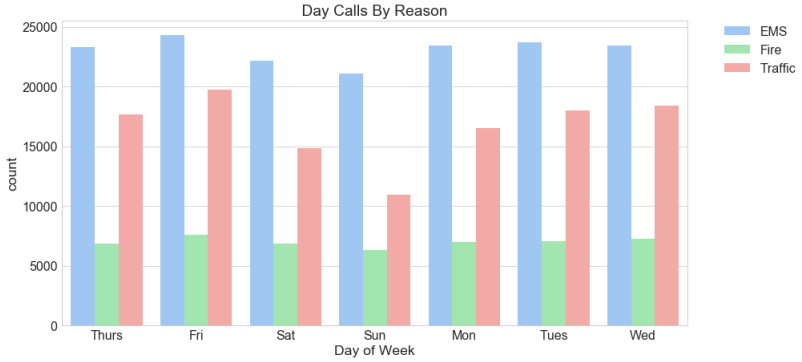


In [ ]:
```
```

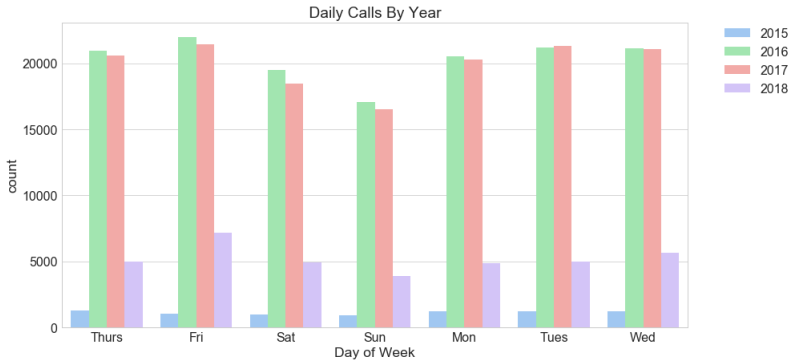
In [31]:
```
# Calls report Daily:

plt.figure(figsize=(14,7))
sns.set_context("paper", font_scale = 2)
sns.countplot(x= "Day of Week", data= df, palette="pastel")
plt.title(" Day Of Week Calls")
plt.show()
```



In [32]:
```
plt.figure(figsize=(14,7))
sns.set_context("paper", font_scale = 2)
sns.countplot(x= "Day of Week", data= df, palette="pastel", hue= ("reason category") )
plt.title(" Day Calls By Reason ")
plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)
plt.show()
```



In [33]:
```
plt.figure(figsize=(14,7))
sns.set_context("paper", font_scale = 2)
sns.countplot(x= "Day of Week", data= df, palette="pastel", hue= "year" )
plt.title(" Daily Calls By Year ")
plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)
plt.show()
```



In [ ]:
```
```

5 Line Plots on basis of aggregation:

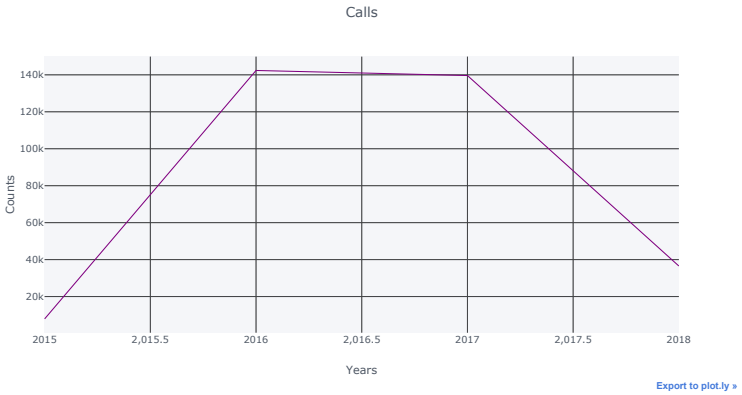
Using interactive plots: Plotly

In [34]:
```
df.groupby("year").count()
# this will return a df having Years as its index
```

Out[34]:

year	lat	lng	desc	zip	title	timeStamp	twp	addr	reason category	month	date	day of week	time	Day of Week	day/night
2015	7916	7916	7916	6902	7916	7916	7911	7916	7916	7916	7916	7916	7916	7916	7916
2016	142360	142360	142360	124495	142360	142360	142317	142360	142360	142360	142360	142360	142360	142360	142360
2017	139617	139617	139617	123123	139617	139617	139562	139617	139617	139617	139617	139617	139617	139617	139617
2018	36532	36532	36532	32315	36532	36532	36520	36532	36532	36532	36532	36532	36532	36532	36532

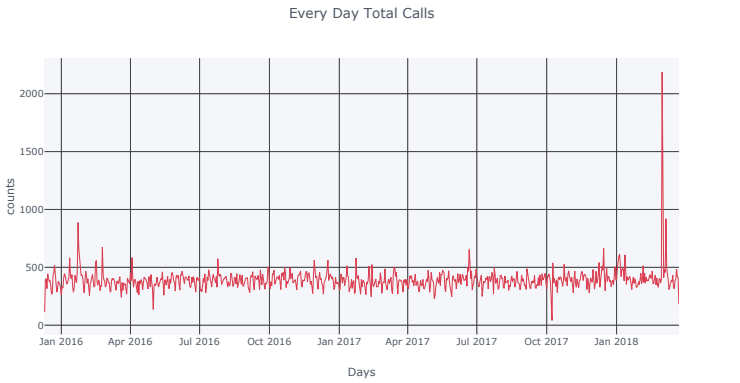
In [35]:
```
by_year = df.groupby("year").count()
df_year = by_year.reset_index()

df_year.iplot( x= "year", y= "lat", title =" Calls", colors="purple", xTitle="Years",
              yTitle = "Counts")
```



In [36]:
```
by_month= df.groupby("month").count()
# Changing index from months to default index(0:)
df_month= by_month.reset_index()
df_month.iplot(x="month", y = "lat", colors= "pink", title="Monthly calls in all years", xTitle="Month",
               yTitle = "Counts")
```

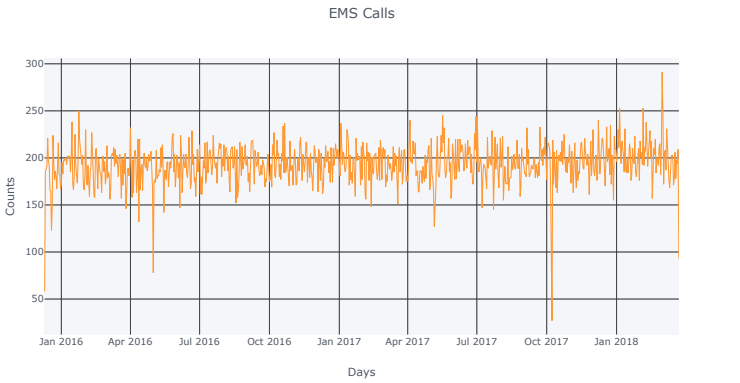
Export to plot.ly »

In [37]:
```
date_gr = df.groupby("date").count()
df_date = date_gr.reset_index()

df_date.iplot(x= "date", y="lat", size= 6, color= "red", title= "Every Day Total Calls",
              xTitle = "Days", yTitle = "counts")
```



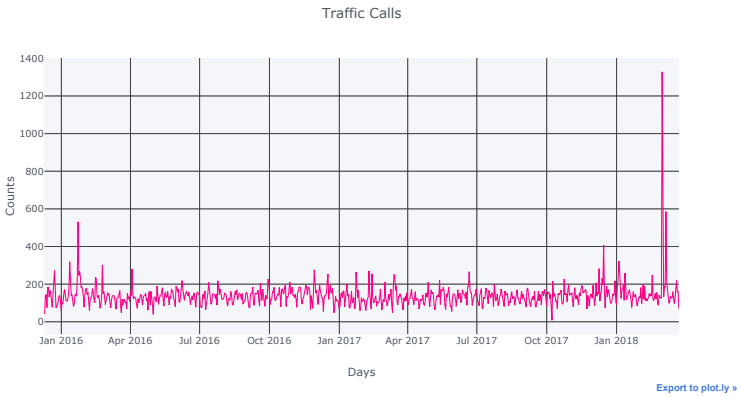
Export to plot.ly »

In [38]:
```
ems_gr = df[df["reason category"]== "EMS"].groupby("date").count()

df_ems = ems_gr.reset_index()

df_ems.iplot(x= "date", y= "lat", colors= "orange", title="EMS Calls", xTitle="Days",
             yTitle = "Counts")
```

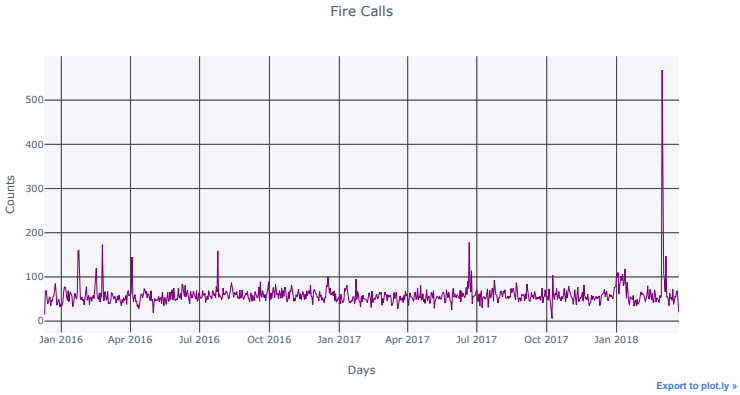


Export to plot.ly »

In [39]:
```
tra_gr = df[df["reason category"]== "Traffic"].groupby("date").count()

df_tra= tra_gr.reset_index()

df_tra.iplot(x= "date", y= "lat", colors= "pink", title="Traffic Calls", xTitle="Days",
             yTitle = "Counts")
```



In [40]:
```
fire_gr = df[df["reason category"]== "Fire"].groupby("date").count()

df_fire= fire_gr.reset_index()

df_fire.iplot(x= "date", y= "lat", colors= "purple", title="Fire Calls", xTitle="Days",
             yTitle = "Counts")
```



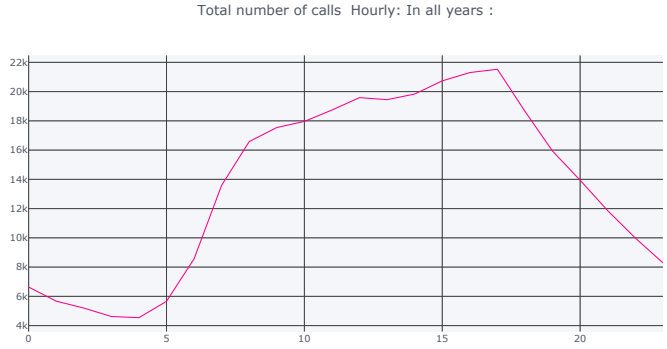
In [ ]:
```
```

In [41]:
```
df["year"].value_counts()
```

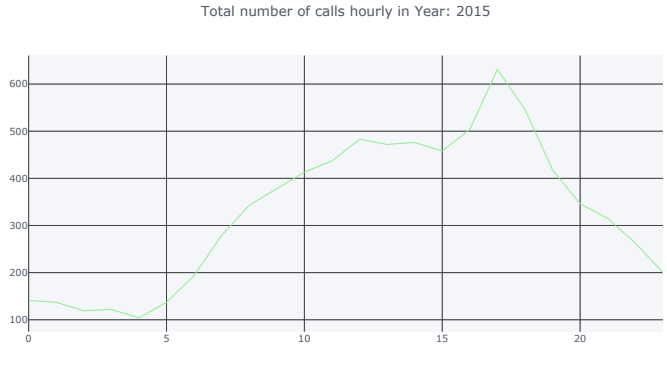
Out[41]:
```
2016    142360
2017    139617
2018     36532
2015      7916
Name: year, dtype: int64
```

In [42]:
```
df["hour"] = df["time"].apply(lambda x: x.hour)
tmap = {0:1, 1:2, 2:3, 3:4, 4:5, 5:6, 6:7, 7:8, 8:9, 9:10, 10:11, 11:12, 12:13, 13:14,
        14:15, 15:16, 16:17, 17:18, 18:19, 19:20, 20:21, 21:22, 22:23, 23:24}
df["hour"] = df["hour"].map(tmap)
print(df["hour"].value_counts())

df["hour"] = df["time"].apply(lambda x: x.hour)
hr_grp = df.groupby("hour").count()
df_time = hr_grp.reset_index()


df_time.iplot(x= "hour", y = "lat", title= "Total number of calls  Hourly: In all years :",
              colors = "pink")
```

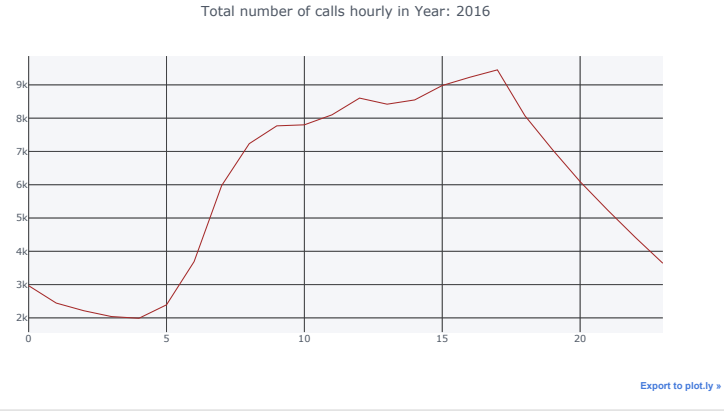
```
18    21527
17    21302
16    20733
15    19833
13    19588
14    19449
12    18739
19    18650
11    17961
10    17544
9     16584
20    15933
21    13942
8     13595
22    11860
23     9993
7      8568
24     8293
1      6634
2      5672
6      5661
3      5199
4      4623
5      4542
Name: hour, dtype: int64
```



In [43]:
```
yl5_grp = df[df["year"] == 2015].groupby("hour").count()

df_yl5 = yl5_grp.reset_index()

df_yl5.iplot( x="hour", y ="lat" , title = "Total number of calls hourly in Year: 2015",
            colors = "lightgreen")
```

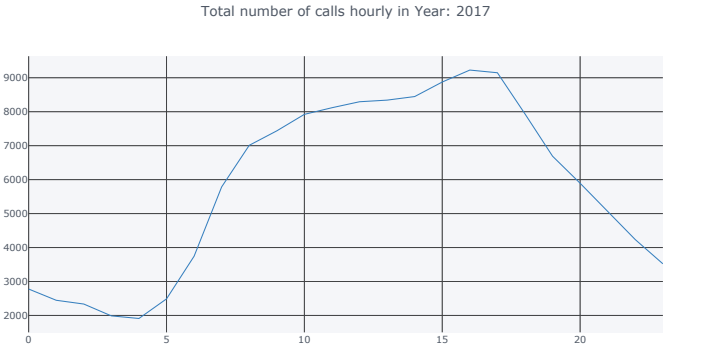


In [44]:
```
y16_grp = df[df["year"] == 2016].groupby("hour").count()

df_y16 = y16_grp.reset_index()

df_y16.iplot( x="hour", y ="lat",title = "Total number of calls hourly in Year: 2016",
             colors = "brown")
```



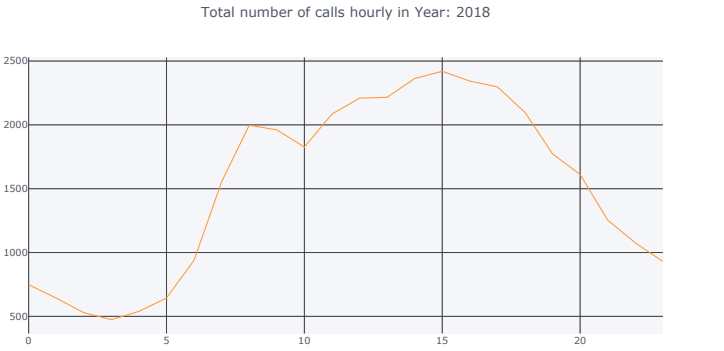
Export to plot.ly »

In [45]:
```
y17_grp = df[df["year"] == 2017].groupby("hour").count()

df_y17 = y17_grp.reset_index()

df_y17.iplot( x="hour", y ="lat", title = "Total number of calls hourly in Year: 2017",
             colors = "blue")
```



Export to plot.ly »

In [46]:
```
y18_grp = df[df["year"] == 2018].groupby("hour").count()

df_y18 = y18_grp.reset_index()

df_y18.iplot( x="hour", y ="lat", title = "Total number of calls hourly in Year: 2018" )
```



Export to plot.ly »

In [ ]:

6 Heat Maps:

In [47]:
```
dayvshour= df.groupby(["Day of Week", "hour"]).count()["lat"].unstack()
dayvshour.head()
```

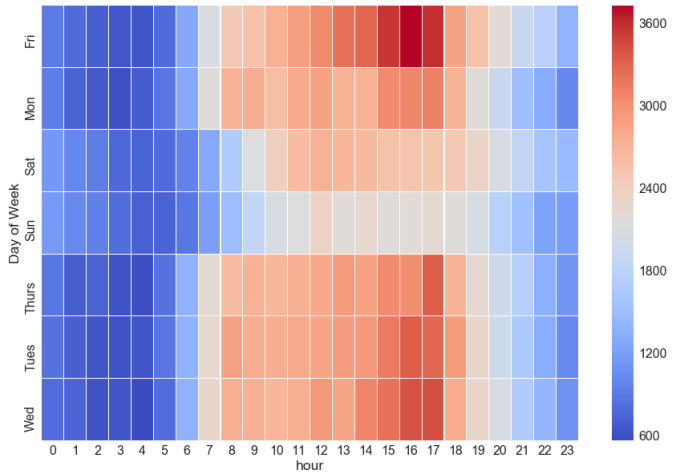
Out[47]:

Day of Week \ hour	0	1	2	3	4	5	6	7	8	9	...	14	15	16	17	18	19	20	21	22	23
Fri	896	789	701	644	633	786	1286	2087	2487	2570	...	3290	3562	3726	3596	2858	2562	2205	1916	1765	1396
Mon	931	732	663	585	683	862	1291	2175	2714	2766	...	2729	3034	3082	3116	2729	2179	1912	1503	1303	1003
Sat	1130	999	908	767	742	788	957	1295	1674	2132	...	2643	2563	2529	2525	2477	2320	2074	1839	1579	1469
Sun	1159	1026	955	800	717	736	880	1196	1492	1849	...	2276	2170	2200	2234	2179	2067	1753	1536	1235	1177
Thurs	871	690	727	603	573	833	1386	2254	2613	2719	...	2863	3044	3017	3335	2707	2226	1974	1724	1360	1120

5 rows × 24 columns

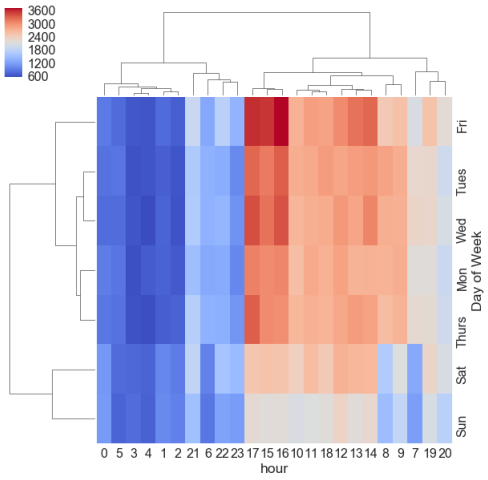
In [48]:
```
plt.figure(figsize= (15,10))
sns.heatmap(dayvshour, cmap = "coolwarm", linewidths=.1)
```

Out[48]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1a1672a940>
```



In [49]:
```
plt.figure(figsize= (15,10))
sns.clustermap(dayvshour, cmap = "coolwarm")
```

Out[49]:
```
<seaborn.matrix.ClusterGrid at 0x1112863c8>

<matplotlib.figure.Figure at 0x111286a58>
```



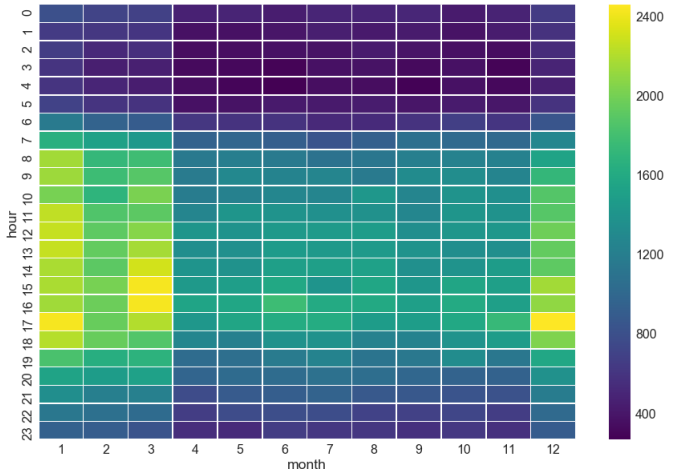
In [50]:
```
monthvshour = df.groupby(["hour", "month"]).count()["lat"].unstack()
monthvshour.head()
```

Out[50]:

hour \ month	1	2	3	4	5	6	7	8	9	10	11	12
0	799	710	691	466	485	442	491	495	501	428	475	651
1	651	617	604	364	366	390	453	415	422	377	430	583
2	654	515	564	343	350	363	377	422	382	357	338	534
3	592	449	460	320	312	291	367	372	320	364	290	486
4	597	497	439	342	298	266	348	341	280	368	309	457

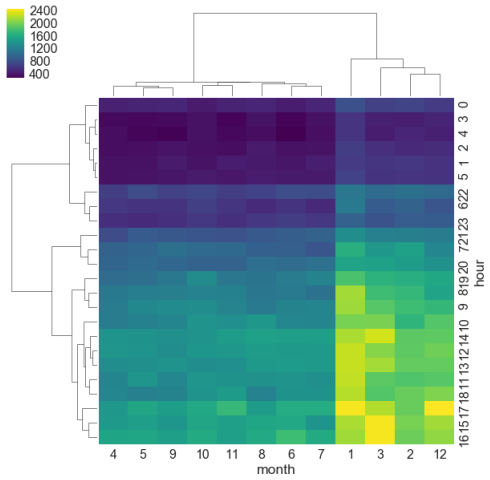
In [51]:
```
plt.figure(figsize=(15,10))
sns.heatmap(monthvshour, cmap="viridis", linewidths=.5)
```

Out[51]: <matplotlib.axes._subplots.AxesSubplot at 0x1a196456d8>



In [52]:
```
sns.clustermap(monthvshour, cmap = "viridis")
```

Out[52]: <seaborn.matrix.ClusterGrid at 0x1112864a8>



In [ ]:

Please, free to leave any comment or feedback.

Thanks!