EDA on 911 Emergencies – Python

In this project, we will be using Python to practice EDA skills using various libraries such as numpy, matplotlib, and seaborn. We will be using a csv flat file with 911 call data to create several visualizations and gain insights into our data.

Contents

Skills Demonstrated:

- Data Preparation
- Data Visualization
- Data Manipulation

Required Materials:

- The dataset *911.csv* available *here*.
- A Python notebook. (I will be using Jupyter.)

Walkthrough:

Data Understanding

1. First, let's open up the CSV and get an idea of the data we will be working with.

   Our file is composed of the following columns:

   - **lat**: The latitude the emergency occurred.
   - **lng:** The longitude the emergency occurred.
   - **desc:** The description of the emergency which is composed of the address, the town, the station, and a date-time stamp.
   - **zip:** The zipcode of the emergency.
   - **title:** The service used for the emergency and the emergency reason.

- **timeStamp:** A time stamp of when the emergency occurred.
- **twp:** The town the emergency occurred in.
- **addr:** The address of the emergency
- **e:** A column filled with 1s.

2. With an understanding of the data present, let's construct a set of questions we would like answered:

Questions:

- Which stations see the most calls?
- How many stations are there?
- How many different zip codes are being tracked?
- Which zip code sees the most emergencies?
- What is the most common type of emergency?
- Which type of service is utilized the most?
- What is the frequency for when these emergencies occur?
- Around what date did the most emergencies in a single day occur?
- Around what time of the week do the most emergencies occur?
- Where do the most emergencies occur?

3. With these questions in mind, let's begin by importing our libraries and loading our data.

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

%matplotlib inline

# Will need to be adjusted as needed.
df = pd.read_csv(r'C:\Users\Patrick Robison\Documents\Business Documents\Portfolio Projects\911.csv')
```

In [2]:
```
# Viewing data
df.head()
```

Out[2]:

	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

4. With our data loading properly, let's get some more insights into it using info(), describe(), and some code to determine the amount of missing entries.

In [3]:
```
# Rows vs Columns
df.shape
```

Out[3]:
```
(663522, 9)
```

In [4]:
```
# File info
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 663522 entries, 0 to 663521
Data columns (total 9 columns):
 #   Column     Non-Null Count   Dtype  
---  ------     --------------   -----  
 0   lat        663522 non-null  float64
 1   lng        663522 non-null  float64
 2   desc       663522 non-null  object 
 3   zip        583323 non-null  float64
 4   title      663522 non-null  object 
 5   timeStamp  663522 non-null  object 
 6   twp        663229 non-null  object 
 7   addr       663522 non-null  object 
 8   e          663522 non-null  int64  
dtypes: float64(3), int64(1), object(5)
memory usage: 45.6+ MB
```

In [7]:
```
# We can get more insights into our data using describe.
# Here I set the format to two decimal places so as to not take up as much space.
# The maximum and minimum values can let us see potential outliers.
pd.set_option("display.float_format", "{:.2f}".format)
df.describe()
```

Out[7]:

	lat	lng	zip	e
count	663522.00	663522.00	583323.00	663522.00
mean	40.16	-75.30	19236.06	1.00
std	0.22	1.67	298.22	0.00
min	0.00	-119.70	1104.00	1.00
25%	40.10	-75.39	19038.00	1.00
50%	40.14	-75.31	19401.00	1.00
75%	40.23	-75.21	19446.00	1.00
max	51.34	87.85	77316.00	1.00

In [5]:
```
# Using this cell we can see the percentage of missing or null data.
df.isnull().sum()/len(df)
```

Out[5]:
```
lat          0.000000
lng          0.000000
desc         0.000000
zip          0.120869
title        0.000000
timeStamp    0.000000
twp          0.000442
addr         0.000000
e            0.000000
dtype: float64
```

In [6]:
```
# If we want the numbers we can use this.
df.isnull().sum()
```

Out[6]:
```
lat              0
lng              0
desc             0
zip          80199
title            0
timeStamp        0
twp            293
addr             0
e                0
dtype: int64
```

We didn't learn too much from shape() or info() that we didn't quite know from the file, but using describe() gave us insights into some of the outliers of our columns and the general area that most emergencies occur with the longitude and latitude columns. Additionally, we also found some missing information in the zip code and township columns. Now that we have the gist of what we are dealing with, let's start preparing our data for our questions.

Data Preparation

1. First, let's change the name of our columns so that they are easier to reference.
2. Additionally, we'll be dropping the 'e' column as there is no useful information in there for our analysis.

In [9]:
```
# We'll use an array in order to change the names of the columns for readability.
df = pd.read_csv(r'C:\Users\Patrick Robison\Documents\Business Documents\Portfolio Projects\911.csv')
column = ['Latitude', 'Longitude', 'Description', 'Zipcode', 'Title', 'Timestamp'
         , 'Township', 'Address', 'e']
df.columns = column
df.head()
```

Out[9]:

	Latitude	Longitude	Description	Zipcode	Title	Timestamp	Township	Address	e
0	40.30	-75.58	REINDEER CT & DEAD END; NEW HANOVER; Station ...	19525.00	EMS: BACK PAINS/INJURY	2015-12-10 17:10:52	NEW HANOVER	REINDEER CT & DEAD END	1
1	40.26	-75.26	BRIAR PATH & WHITEMARSH LN; HATFIELD TOWNSHIP...	19446.00	EMS: DIABETIC EMERGENCY	2015-12-10 17:29:21	HATFIELD TOWNSHIP	BRIAR PATH & WHITEMARSH LN	1
2	40.12	-75.35	HAWS AVE; NORRISTOWN; 2015-12-10 @ 14:39:21-St...	19401.00	Fire: GAS-ODOR/LEAK	2015-12-10 14:39:21	NORRISTOWN	HAWS AVE	1
3	40.12	-75.34	AIRY ST & SWEDE ST; NORRISTOWN; Station 308A;...	19401.00	EMS: CARDIAC EMERGENCY	2015-12-10 16:47:36	NORRISTOWN	AIRY ST & SWEDE ST	1
4	40.25	-75.60	CHERRYWOOD CT & DEAD END; LOWER POTTSGROVE; S...	NaN	EMS: DIZZINESS	2015-12-10 16:56:52	LOWER POTTSGROVE	CHERRYWOOD CT & DEAD END	1

In [10]:
```
# Removing the irrelevant row: e.
df.drop(['e'], axis=1, inplace=True)
df.head()
```

Out[10]:

	Latitude	Longitude	Description	Zipcode	Title	Timestamp	Township	Address
0	40.30	-75.58	REINDEER CT & DEAD END; NEW HANOVER; Station ...	19525.00	EMS: BACK PAINS/INJURY	2015-12-10 17:10:52	NEW HANOVER	REINDEER CT & DEAD END
1	40.26	-75.26	BRIAR PATH & WHITEMARSH LN; HATFIELD TOWNSHIP...	19446.00	EMS: DIABETIC EMERGENCY	2015-12-10 17:29:21	HATFIELD TOWNSHIP	BRIAR PATH & WHITEMARSH LN
2	40.12	-75.35	HAWS AVE; NORRISTOWN; 2015-12-10 @ 14:39:21-St...	19401.00	Fire: GAS-ODOR/LEAK	2015-12-10 14:39:21	NORRISTOWN	HAWS AVE
3	40.12	-75.34	AIRY ST & SWEDE ST; NORRISTOWN; Station 308A;...	19401.00	EMS: CARDIAC EMERGENCY	2015-12-10 16:47:36	NORRISTOWN	AIRY ST & SWEDE ST
4	40.25	-75.60	CHERRYWOOD CT & DEAD END; LOWER POTTSGROVE; S...	NaN	EMS: DIZZINESS	2015-12-10 16:56:52	LOWER POTTSGROVE	CHERRYWOOD CT & DEAD END

3. If you noticed earlier, we have most of the information present in the description aside from the station ID. We can extract this information using str.split which effectively will act like a substring. We will start after station and end after the ; mark.
4. Doing this will mean we have all the information we would need from the description in other columns, so we will drop the description column as well.

In [11]:
```python
# We can break up columns into substrings using split.
df['StationNumber'] = df.Description.str.split('Station', expand=True)[1].str.split(';', expand=True)[0]
df.head()
```

Out[11]:

	Latitude	Longitude	Description	Zipcode	Title	Timestamp	Township	Address	StationNumber
0	40.30	-75.58	REINDEER CT & DEAD END; NEW HANOVER; Station ...	19525.00	EMS: BACK PAINS/INJURY	2015-12-10 17:10:52	NEW HANOVER	REINDEER CT & DEAD END	332
1	40.26	-75.26	BRIAR PATH & WHITEMARSH LN; HATFIELD TOWNSHIP...	19446.00	EMS: DIABETIC EMERGENCY	2015-12-10 17:29:21	HATFIELD TOWNSHIP	BRIAR PATH & WHITEMARSH LN	345
2	40.12	-75.35	HAWS AVE; NORRISTOWN; 2015-12-10 @ 14:39:21-St...	19401.00	Fire: GAS-ODOR/LEAK	2015-12-10 14:39:21	NORRISTOWN	HAWS AVE	:STA27
3	40.12	-75.34	AIRY ST & SWEDE ST; NORRISTOWN; Station 308A;...	19401.00	EMS: CARDIAC EMERGENCY	2015-12-10 16:47:36	NORRISTOWN	AIRY ST & SWEDE ST	308A
4	40.25	-75.60	CHERRYWOOD CT & DEAD END; LOWER POTTSGROVE; S...	NaN	EMS: DIZZINESS	2015-12-10 16:56:52	LOWER POTTSGROVE	CHERRYWOOD CT & DEAD END	329

In [12]:
```python
# As we have access to most of the information in the description, we will remove this column too.
df.drop(['Description'], axis=1, inplace=True)
```

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

Out[13]:

	Latitude	Longitude	Zipcode	Title	Timestamp	Township	Address	StationNumber
0	40.30	-75.58	19525.00	EMS: BACK PAINS/INJURY	2015-12-10 17:10:52	NEW HANOVER	REINDEER CT & DEAD END	332
1	40.26	-75.26	19446.00	EMS: DIABETIC EMERGENCY	2015-12-10 17:29:21	HATFIELD TOWNSHIP	BRIAR PATH & WHITEMARSH LN	345
2	40.12	-75.35	19401.00	Fire: GAS-ODOR/LEAK	2015-12-10 14:39:21	NORRISTOWN	HAWS AVE	:STA27
3	40.12	-75.34	19401.00	EMS: CARDIAC EMERGENCY	2015-12-10 16:47:36	NORRISTOWN	AIRY ST & SWEDE ST	308A
4	40.25	-75.60	NaN	EMS: DIZZINESS	2015-12-10 16:56:52	LOWER POTTSGROVE	CHERRYWOOD CT & DEAD END	329

Now our data is ready to visualize for the purpose of our analysis.

Data Visualization

1. Let’s start by answering our first question using value counts.

In [15]:
```python
# What are the top ten stations by call volume?
df['StationNumber'].value_counts()
```

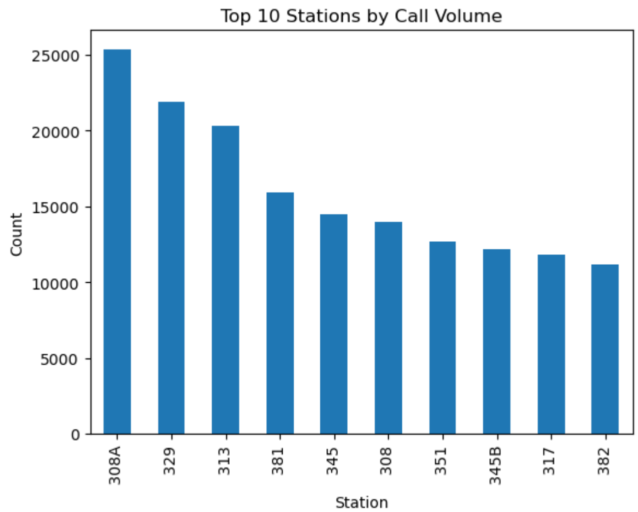
Out[15]:
```
StationNumber
 308A      25346
 329       21895
 313       20325
 381       15949
 345       14504
           ...
 03RAD         1
               1
:FIRE          1
:56FD          1
:56            1
Name: count, Length: 139, dtype: int64
```

1. **308A: 25346**

This gives us good information regarding our first question, but we can gain some more with a bar chart visualization. While we are at it, we can also quickly determine the answer to our second question using nunique which will return the number of unique stations.

```
ax = df['StationNumber'].value_counts() \
    .head(10) \
    .plot(kind='bar', title='Top 10 Stations by Call Volume')
ax.set_xlabel('Station')
ax.set_ylabel('Count')
```

Out[16]: Text(0, 0.5, 'Count')



In [17]:
```
# How many stations are there?
print(df.StationNumber.nunique())
```

139

2. 139 Stations

2. We can perform the same analysis on the zip code column to answer our questions regarding it.

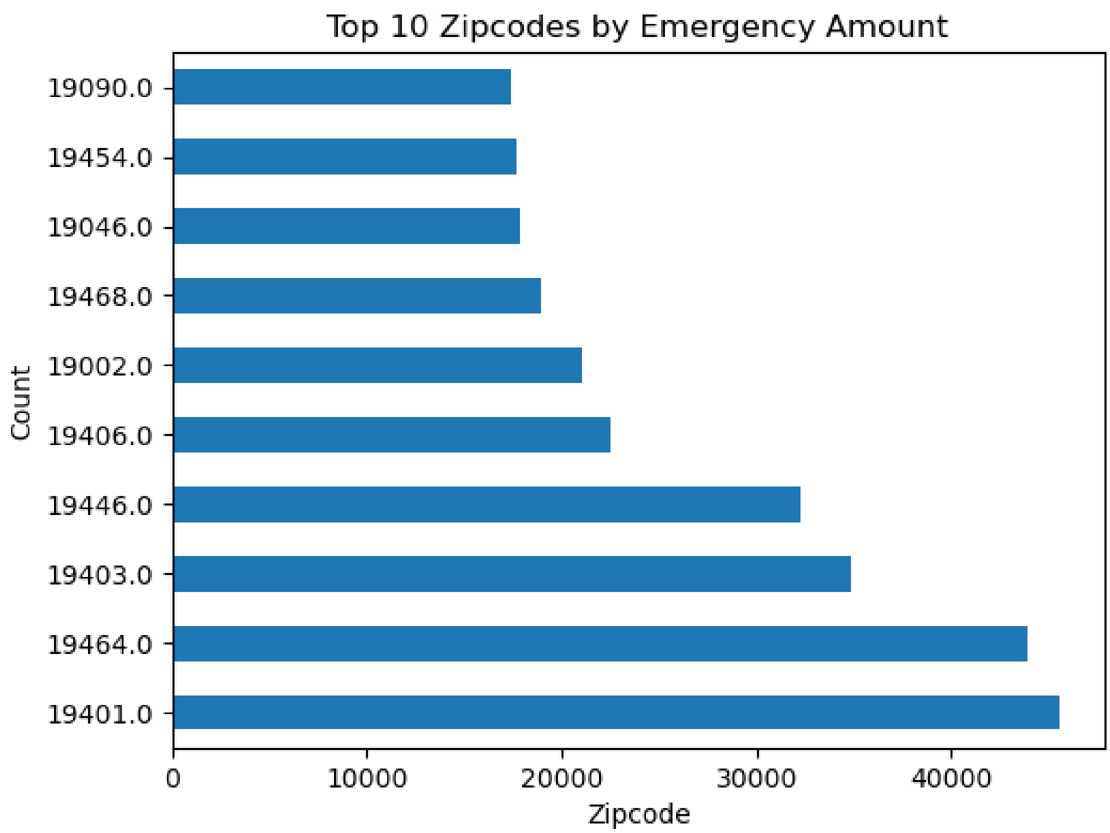
3. 204 zip codes

4. 19401: 45606 emergencies

In [18]:
```
# The same analysis for Zipcode.
print(df.Zipcode.nunique())  # Will return the number of unique zipcodes.
df['Zipcode'].value_counts()
```

204

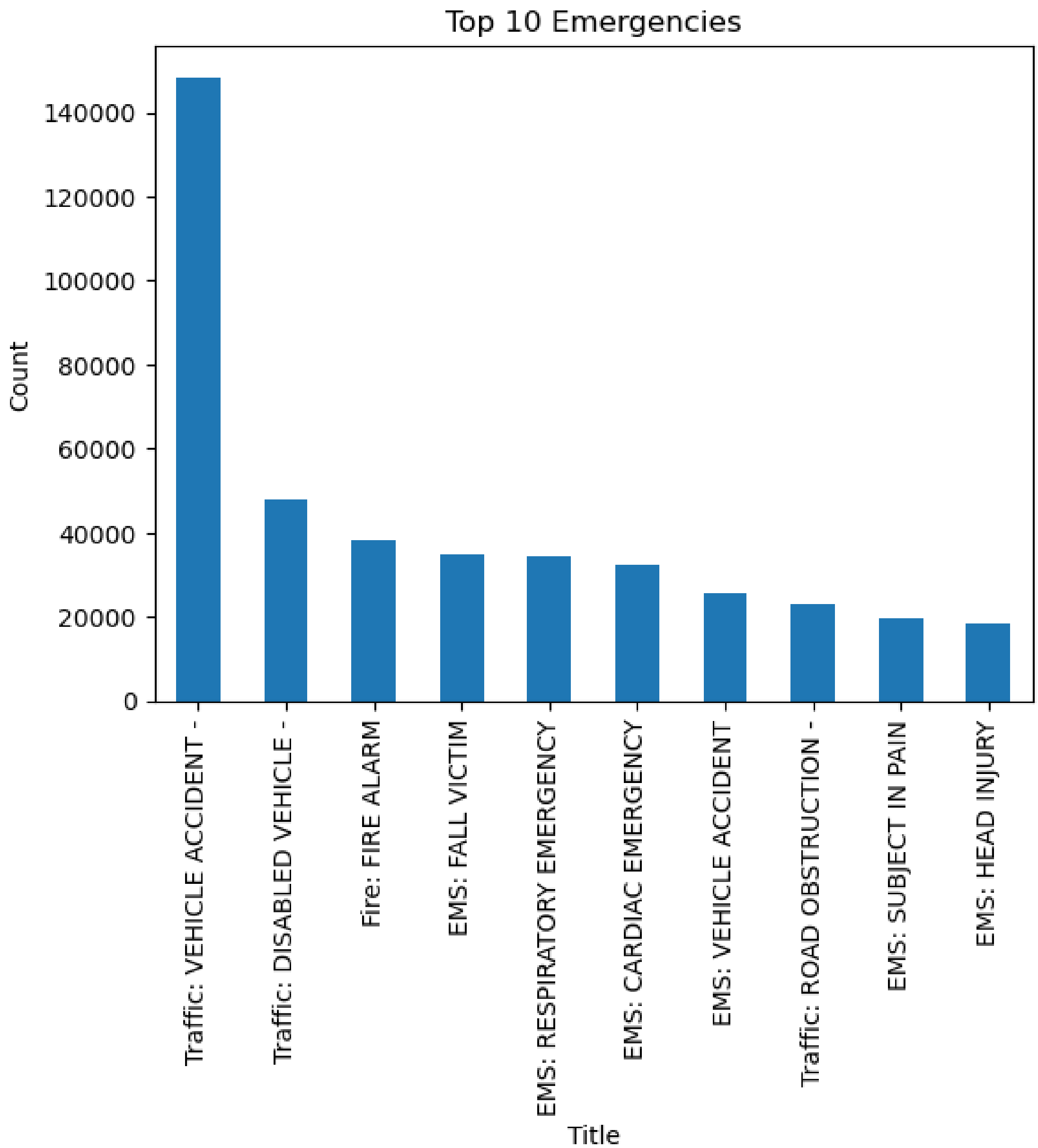
Out[18]:
```
Zipcode
19401.00    45606
19464.00    43910
19403.00    34888
19446.00    32270
19406.00    22464
            ...
77316.00        1
19134.00        1
19135.00        1
8502.00         1
18938.00        1
Name: count, Length: 204, dtype: int64
```



3. Again, we can perform this same analysis on our emergency title column.

```
148
```

```
Out[20]: Title
Traffic: VEHICLE ACCIDENT -    148372
Traffic: DISABLED VEHICLE -     47909
Fire: FIRE ALARM                38336
EMS: FALL VICTIM                34676
EMS: RESPIRATORY EMERGENCY      34248
                                 ...
EMS: DISABLED VEHICLE               1
Fire: PRISONER IN CUSTODY           1
Fire: GENERAL WEAKNESS              1
Fire: SUSPICIOUS                    1
Fire: BARRICADED SUBJECT            1
Name: count, Length: 148, dtype: int64
```



5. **Vehicle Accident: 148372**

4. Again, we see that the service used in the emergency is a part of the title column. Using the str.split method we utilized earlier, we can split the column into the emergency reason and emergency service utilized in order to gain more insights into the service specifically.

In [22]:
```
# Further categorizing the data by breaking up the title column.
df['Service'] = df.Title.str.split(':', expand=True)[0]
df['Reason'] = df.Title.str.split(':', expand=True)[1].str.replace(' -', '')
```

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

Out[23]:

	Latitude	Longitude	Zipcode	Title	Timestamp	Township	Address	StationNumber	Service	Reason
0	40.30	-75.58	19525.00	EMS: BACK PAINS/INJURY	2015-12-10 17:10:52	NEW HANOVER	REINDEER CT & DEAD END	332	EMS	BACK PAINS/INJURY
1	40.26	-75.26	19446.00	EMS: DIABETIC EMERGENCY	2015-12-10 17:29:21	HATFIELD TOWNSHIP	BRIAR PATH & WHITEMARSH LN	345	EMS	DIABETIC EMERGENCY
2	40.12	-75.35	19401.00	Fire: GAS-ODOR/LEAK	2015-12-10 14:39:21	NORRISTOWN	HAWS AVE	:STA27	Fire	GAS-ODOR/LEAK
3	40.12	-75.34	19401.00	EMS: CARDIAC EMERGENCY	2015-12-10 16:47:36	NORRISTOWN	AIRY ST & SWEDE ST	308A	EMS	CARDIAC EMERGENCY
4	40.25	-75.60	NaN	EMS: DIZZINESS	2015-12-10 16:56:52	LOWER POTTSGROVE	CHERRYWOOD CT & DEAD END	329	EMS	DIZZINESS

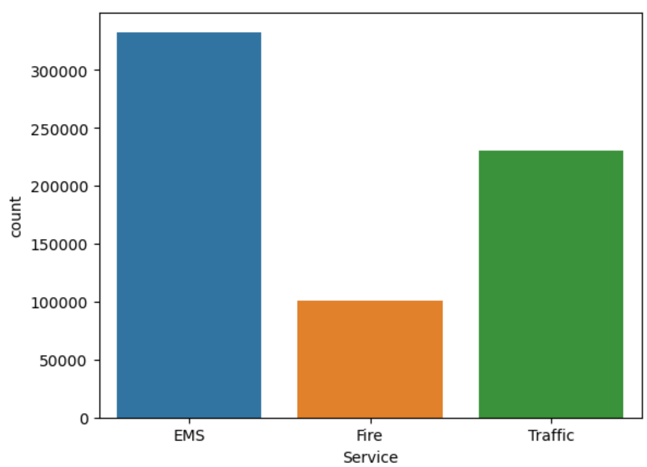
In [24]:
```
# What are the amount of emergencies based on the service?
df['Service'].value_counts()
```

Out[24]:
```
Service
EMS        332692
Traffic    230208
Fire       100622
Name: count, dtype: int64
```

We can also use a seaborn count plot to visualize this data.

In [25]:
```
# We can also use Seaborn plots to get an idea.
sns.countplot(x=df.Service, data=df)
```

Out[25]: <Axes: xlabel='Service', ylabel='count'>



6. EMS is the most utilized service.

5. Our next set of questions related to times and dates which will utilize our timestamp column. From experience, we can't immediately extract any date or time related information as it is as earlier we saw that the column was being identified as an object and not a datetime type. We can convert it and then break it up into several columns using the following cells:

Service

In [26]:
```
# Let's get better insights as to when these incidents occur.
```

In [27]:
```
# Currently this will throw an error, so we need to convert this column to the date format.
# df['DayOfWeek'] = df.Timestamp.dt.weekday
# df['Hour'] = df.Timestamp.dt.hour
# df['Month'] = df.Timestamp.dt.month
```

In [28]:
```
# Conversion
df['Timestamp'] = pd.to_datetime(df.Timestamp)
```

In [29]:
```
df['DayOfWeek'] = df.Timestamp.dt.weekday
df['Hour'] = df.Timestamp.dt.hour
df['Month'] = df.Timestamp.dt.month
```

In [30]:
```
df.head()
```

Out[30]:

	Latitude	Longitude	Zipcode	Title	Timestamp	Township	Address	StationNumber	Service	Reason	DayOfWeek	Hour	Month
0	40.30	-75.58	19525.00	EMS: BACK PAINS/INJURY	2015-12-10 17:10:52	NEW HANOVER	REINDEER CT & DEAD END	332	EMS	BACK PAINS/INJURY	3	17	12
1	40.26	-75.26	19446.00	EMS: DIABETIC EMERGENCY	2015-12-10 17:29:21	HATFIELD TOWNSHIP	BRIAR PATH & WHITEMARSH LN	345	EMS	DIABETIC EMERGENCY	3	17	12
2	40.12	-75.35	19401.00	Fire: GAS-ODOR/LEAK	2015-12-10 14:39:21	NORRISTOWN	HAWS AVE	:STA27	Fire	GAS-ODOR/LEAK	3	14	12
3	40.12	-75.34	19401.00	EMS: CARDIAC EMERGENCY	2015-12-10 16:47:36	NORRISTOWN	AIRY ST & SWEDE ST	308A	EMS	CARDIAC EMERGENCY	3	16	12
4	40.25	-75.60	NaN	EMS: DIZZINESS	2015-12-10 16:56:52	LOWER POTTSGROVE	CHERRYWOOD CT & DEAD END	329	EMS	DIZZINESS	3	16	12

Our new Day of week column works, but we can change it from an int to the day of the week it represents using an array and a map.

In [31]:
```
# Currently our days of the week are formatted as integers, but we can format them as strings using a map which will perform
# a function based on the iterables within.
weekmap = {0:'Mon', 1:'Tue', 2:'Wed', 3:'Thu', 4:'Fri', 5:'Sat', 6:'Sun'}
df['DayOfWeek'] = df.DayOfWeek.map(weekmap)
df.head()
```

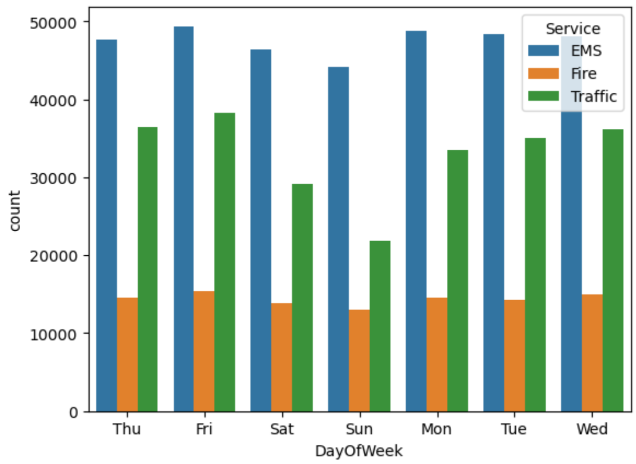
Out[31]:

	Latitude	Longitude	Zipcode	Title	Timestamp	Township	Address	StationNumber	Service	Reason	DayOfWeek	Hour	Month
0	40.30	-75.58	19525.00	EMS: BACK PAINS/INJURY	2015-12-10 17:10:52	NEW HANOVER	REINDEER CT & DEAD END	332	EMS	BACK PAINS/INJURY	Thu	17	12
1	40.26	-75.26	19446.00	EMS: DIABETIC EMERGENCY	2015-12-10 17:29:21	HATFIELD TOWNSHIP	BRIAR PATH & WHITEMARSH LN	345	EMS	DIABETIC EMERGENCY	Thu	17	12
2	40.12	-75.35	19401.00	Fire: GAS-ODOR/LEAK	2015-12-10 14:39:21	NORRISTOWN	HAWS AVE	:STA27	Fire	GAS-ODOR/LEAK	Thu	14	12
3	40.12	-75.34	19401.00	EMS: CARDIAC EMERGENCY	2015-12-10 16:47:36	NORRISTOWN	AIRY ST & SWEDE ST	308A	EMS	CARDIAC EMERGENCY	Thu	16	12
4	40.25	-75.60	NaN	EMS: DIZZINESS	2015-12-10 16:56:52	LOWER POTTSGROVE	CHERRYWOOD CT & DEAD END	329	EMS	DIZZINESS	Thu	16	12

6. Using a plot similar to our earlier seaborn plot, we can visualize the different service emergency counts side by side depending on the day of the week. The same can also be done for the hour and month of the year.

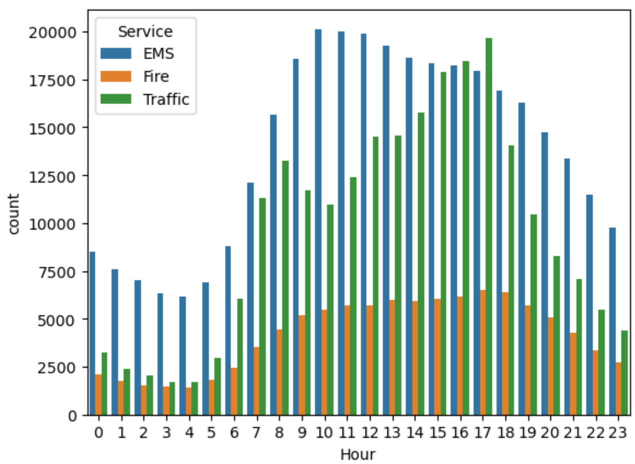
In [32]:
```
# Using Seaborn we can get an idea of the frequency of service based on time/day/month factors.
sns.countplot(x=df.DayOfWeek, data=df, hue='Service')
```

Out[32]: <Axes: xlabel='DayOfWeek', ylabel='count'>

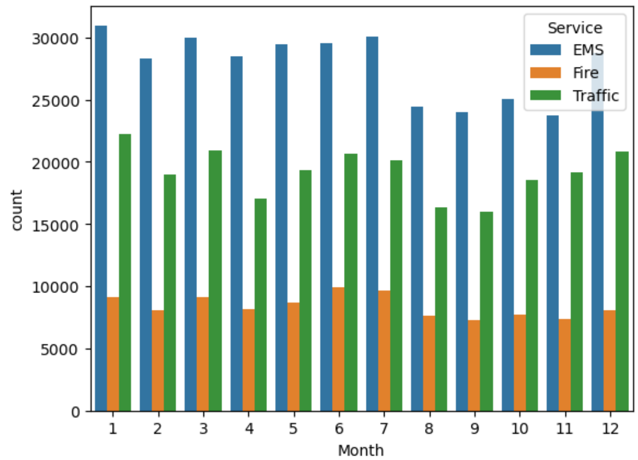


In [33]:
```
sns.countplot(x=df.Hour, data=df, hue='Service')
```

Out[33]: <Axes: xlabel='Hour', ylabel='count'>



In [34]:
```
sns.countplot(x=df.Month, data=df, hue='Service')
```
Out[34]:
```
<Axes: xlabel='Month', ylabel='count'>
```



Most of the data looks similar between each timepoint with the exception of our second plot. The reason behind this certainly being tied to when more people are awake for more emergencies to both occur and be reported. We can roughly say that the peak time period is between 7am to 6pm, which answers **question 7.**

7. Next, let's make a date column out of our timestamp so that we can get an idea of how many emergencies occurred on each day.

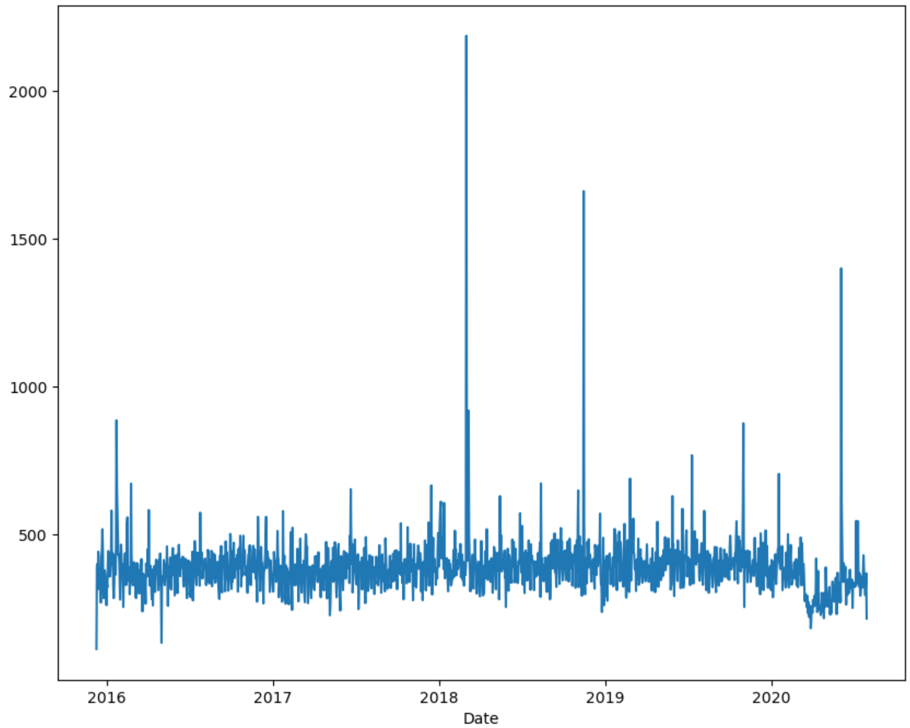
In [35]:
```
# Next, let's try to get a timeline of all of our data. We can do this by creating a new date column from our timestamp
# column.
df['Date'] = df.Timestamp.dt.date
df.Date
```
Out[35]:
```
0         2015-12-10
1         2015-12-10
2         2015-12-10
3         2015-12-10
4         2015-12-10
             ...
663517    2020-07-29
663518    2020-07-29
663519    2020-07-29
663520    2020-07-29
663521    2020-07-29
Name: Date, Length: 663522, dtype: object
```

In [36]:
```
# We will use groupby in order to get an aggregate on each day.
byDate = df.groupby("Date").count()
byDate.head()
```
Out[36]:

Date	Latitude	Longitude	Zipcode	Title	Timestamp	Township	Address	StationNumber	Service	Reason	DayOfWeek	Hour	Month
2015-12-10	114	114	100	114	114	114	114	73	114	114	114	114	114
2015-12-11	391	391	332	391	391	391	391	254	391	391	391	391	391
2015-12-12	402	402	334	402	402	402	402	257	402	402	402	402	402
2015-12-13	316	316	279	316	316	316	316	241	316	316	316	316	316
2015-12-14	444	444	386	444	444	443	444	260	444	444	444	444	444

We can then plot it and also get value counts for this column.

```
In [37]: byDate.Township.plot(figsize=(10, 8))
Out[37]: <Axes: xlabel='Date'>
```



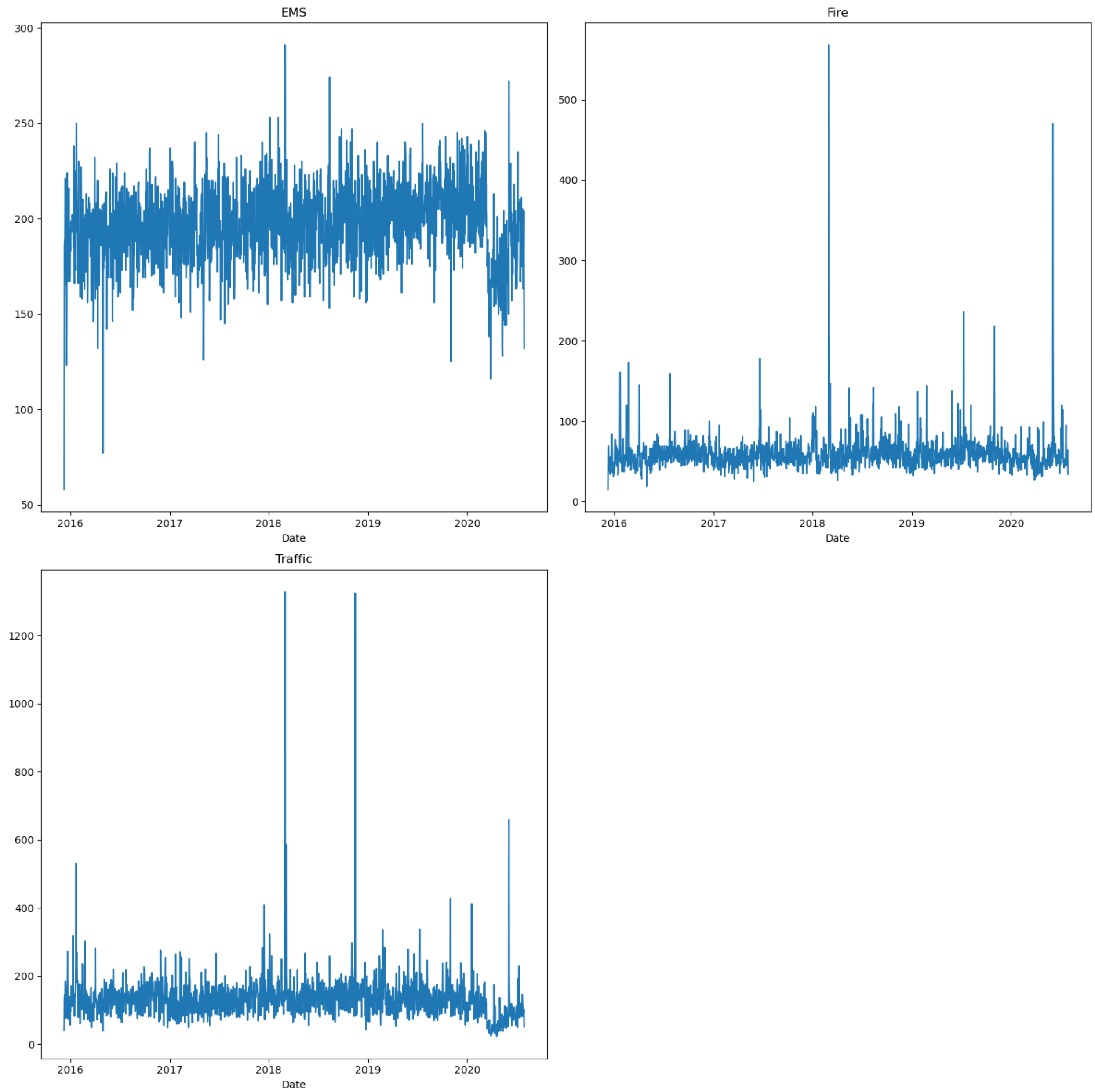

```
In [46]: df['Date'].value_counts()
Out[46]: Date
         2018-03-02    2187
         2018-11-15    1662
         2020-06-03    1401
         2018-03-07     920
         2018-03-03     917
                       ...
         2020-07-29     217
         2020-03-29     204
         2020-03-28     184
         2016-05-01     136
         2015-12-10     114
         Name: count, Length: 1694, dtype: int64
```

8. The most emergencies in a single day occurred on March 2nd, 2018.

8. We can also create subplots based on the service type by using a for loop.

```
In [38]: plt.figure(figsize=(15, 15))

         for i, Reason in enumerate(df.Service.unique(), 1):
             plt.subplot(2, 2, i)
             df[df['Service']==Reason].groupby('Date').count()['Township'].plot()
             plt.title(Reason)
             plt.tight_layout()
```



9. For our next question, let's try using a heatmap. We can count the amount of times a service was used on a day of the week at a specific hour using split(), which will allow us to view the data in a matrix with hours as the columns and the day of the week as the row.

In [39]:

```python
# Next, let's do some heatmaps to get an idea of the magnitude of some time periods.
# First we'll do the day of the week and the hour.
# We can use unstack to view the rows and columns together.
dayHour = df.groupby(by=['DayOfWeek','Hour']).count()['Service'].unstack()
dayHour.head()
```

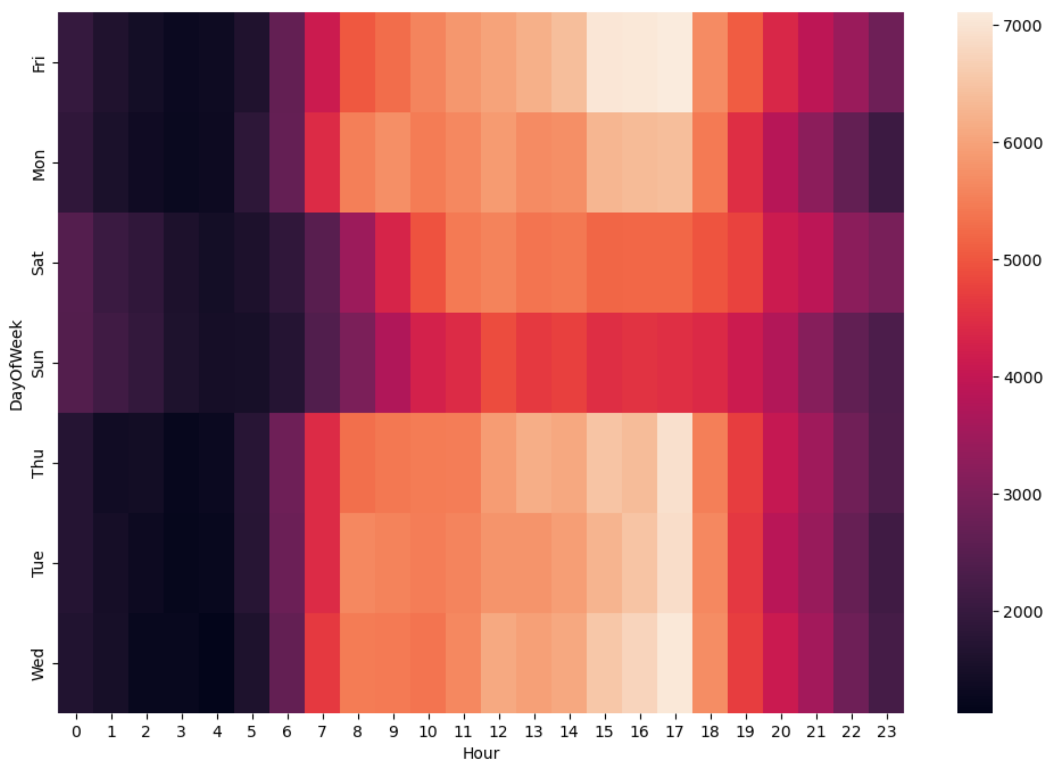
Out[39]:

Hour DayOfWeek	0	1	2	3	4	5	6	7	8	9	...	14	15	16	17	18	19	20	21	22	23
Fri	1983	1635	1449	1296	1339	1639	2670	4143	5018	5288	...	6394	7040	7065	7113	5668	5056	4375	3913	3422	2834
Mon	1894	1571	1368	1272	1336	1844	2675	4430	5504	5724	...	5713	6289	6346	6408	5441	4488	3823	3254	2658	2072
Sat	2447	2059	1883	1592	1451	1580	1880	2489	3457	4315	...	5421	5181	5211	5213	4980	4753	4127	3895	3226	2965
Sun	2424	2135	1946	1614	1471	1488	1726	2408	3001	3728	...	4744	4475	4560	4505	4402	4135	3748	3161	2629	2323
Thu	1731	1408	1426	1236	1293	1775	2816	4432	5297	5412	...	6079	6493	6375	6935	5512	4703	4045	3490	2844	2354

In [40]:

```python
plt.figure(figsize=(12, 8))
sns.heatmap(dayHour)
```

Out[40]: <Axes: xlabel='Hour', ylabel='DayOfWeek'>



9. Around 5pm most days emergencies seem to spike, with Friday having a more noticeable increase.

We can also perform this analysis on month if we'd want to.

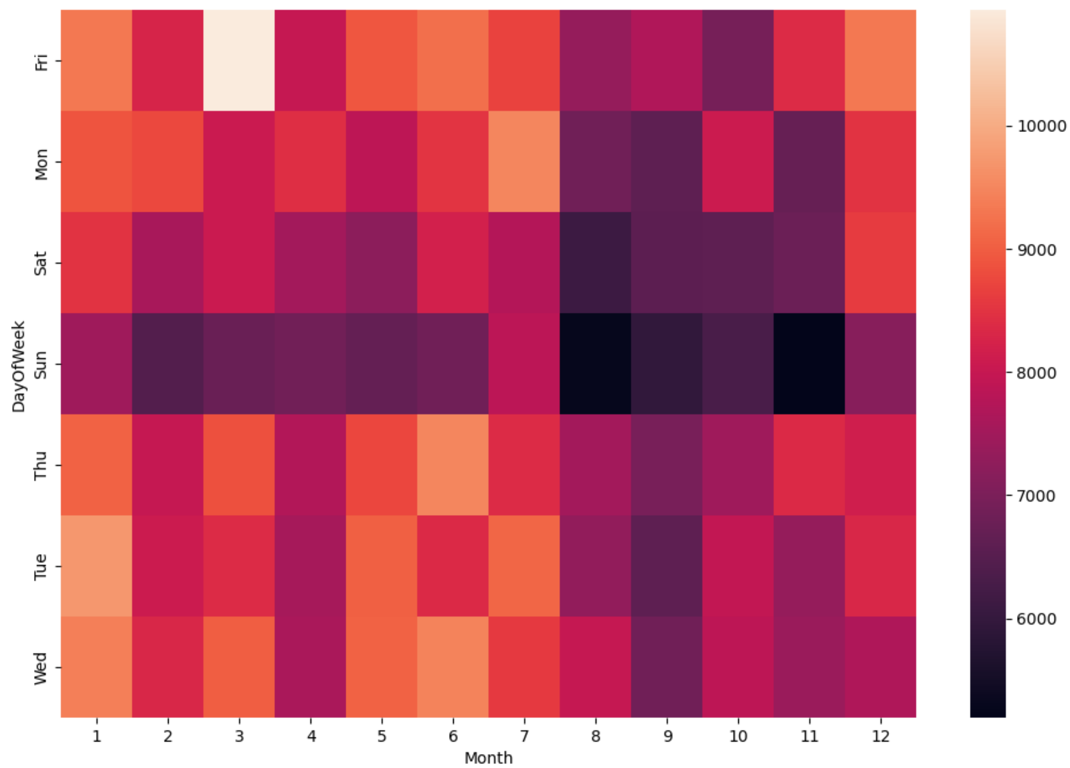
```
In [41]: # Let's do the same for month instead of hour.
         dayMonth = df.groupby(by=['DayOfWeek','Month']).count()['Service'].unstack()
         dayMonth.head()
```

Out[41]:

Month DayOfWeek	1	2	3	4	5	6	7	8	9	10	11	12
Fri	9309	8255	10941	7997	8904	9207	8681	7336	7694	6934	8379	9305
Mon	8896	8747	8060	8410	7881	8511	9499	6854	6598	8075	6722	8492
Sat	8475	7593	8050	7514	7223	8198	7748	6111	6566	6609	6773	8592
Sun	7478	6452	6766	6865	6694	6837	7859	5275	5956	6316	5196	7165
Thu	9055	7997	8849	7722	8740	9489	8378	7508	6954	7482	8358	8151

```
In [42]: plt.figure(figsize=(12, 8))
         sns.heatmap(dayMonth)
```

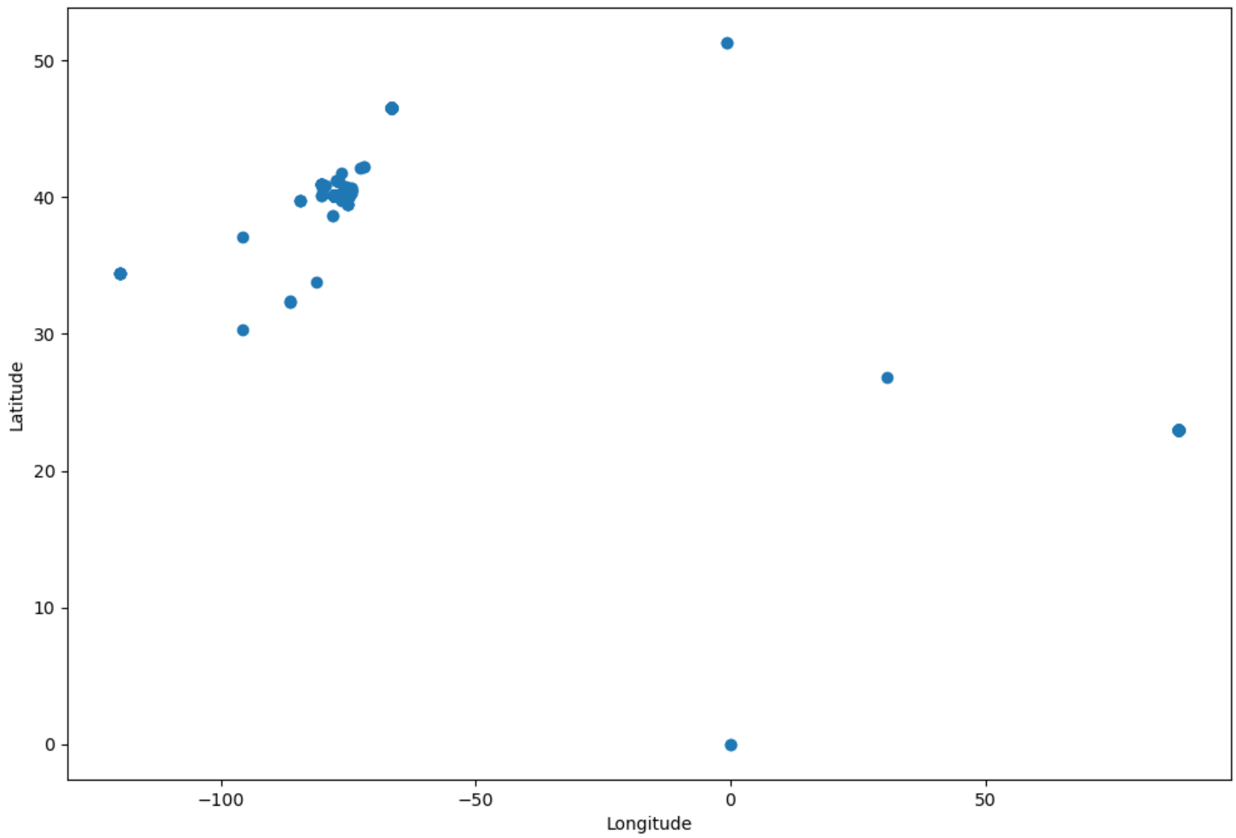
Out[42]: <Axes: xlabel='Month', ylabel='DayOfWeek'>



10. Finally, let's use the longitude and latitude columns to get an idea of the most emergency-dense area. We can do this using a scatterplot.

```
In [43]: # Finally let's get an idea of where the emergencies took place by using a scatter plot with longitude and Latitude chart.
         plt.figure(figsize=(12, 8))
         plt.scatter(df.Longitude, df.Latitude)
         plt.ylabel("Latitude")
         plt.xlabel("Longitude")
```

Out[43]: Text(0.5, 0, 'Longitude')

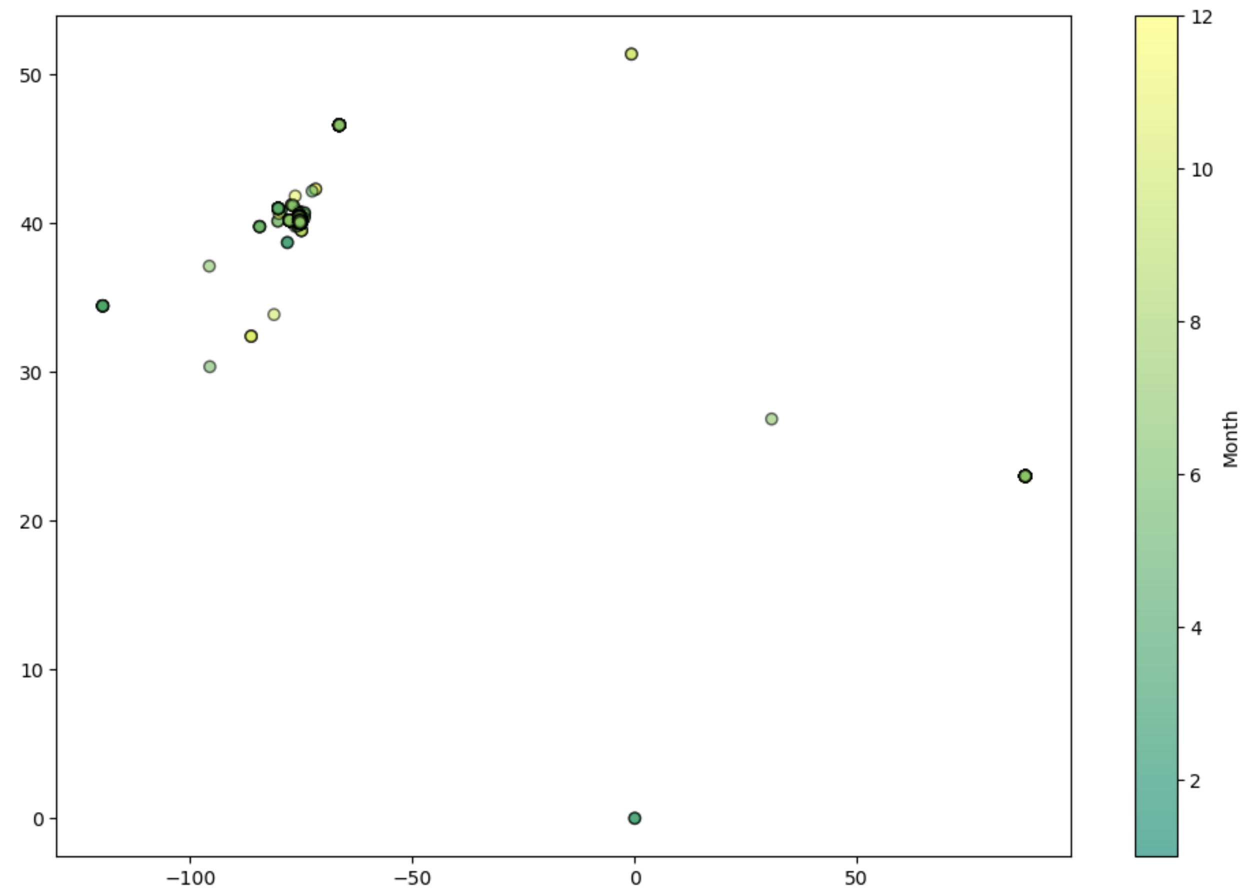


10. This graph seems to correlate with what we saw earlier using df.describe(), showing that on average an emergency will occur around the coordinates of 40, -75.

If we wanted to we could additionally create a scatter plot that includes a time field by using a colorbar, which will change the color of each data point according to the time period.

In [44]:
```
# If we want to, we can also gain insight as to when (DOW, Month, Hour) and where these occured by using a color map.
plt.figure(figsize=(12, 8))
plt.scatter(df.Longitude, df.Latitude, c=df.Month, cmap='summer',
            edgecolor='black', linewidth=1, alpha=0.60)

cbar = plt.colorbar()
cbar.set_label('Month')
```



Using the data and these libraries, we've gained effective insights from our created visualizations and answered our initial questions.