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

In [2]:
```
df=pd.read_csv(r"C:\Users\priya\Downloads\Telegram Desktop\911.csv")
df
```

Out[2]:

	lat	lng	desc	zip	title	timeStamp	tw
0	40.297876	-75.581294	REINDEER CT & DEAD END; NEW HANOVER; Station ...	19525.0	EMS: BACK PAINS/INJURY	2015-12-10 17:40:00	NE HANOVE
1	40.258061	-75.264680	BRIAR PATH & WHITEMARSH LN; HATFIELD TOWNSHIP...	19446.0	EMS: DIABETIC EMERGENCY	2015-12-10 17:40:00	HATFIEL TOWNSH
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	NORRISTOW
3	40.116153	-75.343513	AIRY ST & SWEDE ST; NORRISTOWN; Station 308A;...	19401.0	EMS: CARDIAC EMERGENCY	2015-12-10 17:40:01	NORRISTOW
4	40.251492	-75.603350	CHERRYWOOD CT & DEAD END; LOWER POTTSGROVE; S...	NaN	EMS: DIZZINESS	2015-12-10 17:40:01	LOWE POTTSGRO
...	...	...	...	...	...	...	
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	NORRISTOW
99488	40.006974	-75.289080	LANCASTER AVE & RITTENHOUSE PL; LOWER MERION; ...	19003.0	Traffic: VEHICLE ACCIDENT -	2016-08-24 11:07:02	LOWE MERIO
99489	40.115429	-75.334679	CHESTNUT ST & WALNUT ST; NORRISTOWN; Station ...	19401.0	EMS: FALL VICTIM	2016-08-24 11:12:00	NORRISTOW
99490	40.186431	-75.192555	WELSH RD & WEBSTER LN; HORSHAM; Station 352; ...	19002.0	EMS: NAUSEA/VOMITING	2016-08-24 11:17:01	HORSHA
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	UPPE GWYNED

99492 rows × 9 columns

In [3]:
```
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
```

In [4]:
```
df.head()
```

Out[4]:

	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	REINI & DE
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 WHITE
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	HA
3	40.116153	-75.343513	AIRY ST & SWEDE ST; NORRISTOWN; Station 308A;...	19401.0	EMS: CARDIAC EMERGENCY	2015-12-10 17:40:01	NORRISTOWN	AI SV
4	40.251492	-75.603350	CHERRYWOOD CT & DEAD END; LOWER POTTSGROVE; S...	NaN	EMS: DIZZINESS	2015-12-10 17:40:01	LOWER POTTSGROVE	CHERR' CT

In [5]:
```
zip_column = 'zip'
```

In [6]:
```
top_zipcodes = df[zip_column].value_counts().head(5)
```

In [7]:
```
"Top 5 zip codes for 911 calls:"
top_zipcodes
```

Out[7]:
```
19401.0    6979
19464.0    6643
19403.0    4854
19446.0    4748
19406.0    3174
Name: zip, dtype: int64
```

In [8]:
```
twp_column = 'twp'
top_townships = df[twp_column].value_counts().head(5)
```

In [9]:
```
"Top 5 townships for 911 calls:"
top_townships
```

Out[9]:
```
LOWER MERION    8443
ABINGTON        5977
NORRISTOWN      5890
UPPER MERION    5227
CHELTENHAM      4575
Name: twp, dtype: int64
```

In [10]:
```
title_column = 'title'
unique_title_codes = df[title_column].nunique()
"Number of unique title codes:", unique_title_codes
```

Out[10]:
```
('Number of unique title codes:', 110)
```

In [11]:
```
df['Reasons'] = df[title_column].apply(lambda title: title.split(':')[0].strip())
df
```

Out[11]:

	lat	lng	desc	zip	title	timeStamp	tw
0	40.297876	-75.581294	REINDEER CT & DEAD END; NEW HANOVER; Station ...	19525.0	EMS: BACK PAINS/INJURY	2015-12-10 17:40:00	NE\ HANOVE
1	40.258061	-75.264680	BRIAR PATH & WHITEMARSH LN; HATFIELD TOWNSHIP...	19446.0	EMS: DIABETIC EMERGENCY	2015-12-10 17:40:00	HATFIEL TOWNSH
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	NORRISTOW
3	40.116153	-75.343513	AIRY ST & SWEDE ST; NORRISTOWN; Station 308A;...	19401.0	EMS: CARDIAC EMERGENCY	2015-12-10 17:40:01	NORRISTOW
4	40.251492	-75.603350	CHERRYWOOD CT & DEAD END; LOWER POTTSGROVE; S...	NaN	EMS: DIZZINESS	2015-12-10 17:40:01	LOWE POTTSGROV
...	...	...	...	...	...	...	
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	NORRISTOW
99488	40.006974	-75.289080	LANCASTER AVE & RITTENHOUSE PL; LOWER MERION; ...	19003.0	Traffic: VEHICLE ACCIDENT -	2016-08-24 11:07:02	LOWE MERIO
99489	40.115429	-75.334679	CHESTNUT ST & WALNUT ST; NORRISTOWN; Station ...	19401.0	EMS: FALL VICTIM	2016-08-24 11:12:00	NORRISTOW
99490	40.186431	-75.192555	WELSH RD & WEBSTER LN; HORSHAM; Station 352; ...	19002.0	EMS: NAUSEA/VOMITING	2016-08-24 11:17:01	HORSHA
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	UPPE GWYNED

99492 rows × 10 columns

In [12]:
```
df.columns
```

Out[12]:
```
Index(['lat', 'lng', 'desc', 'zip', 'title', 'timeStamp', 'twp', 'addr', 'e',
       'Reasons'],
      dtype='object')
```

In [13]:
```
reasons_column = 'Reasons'
```

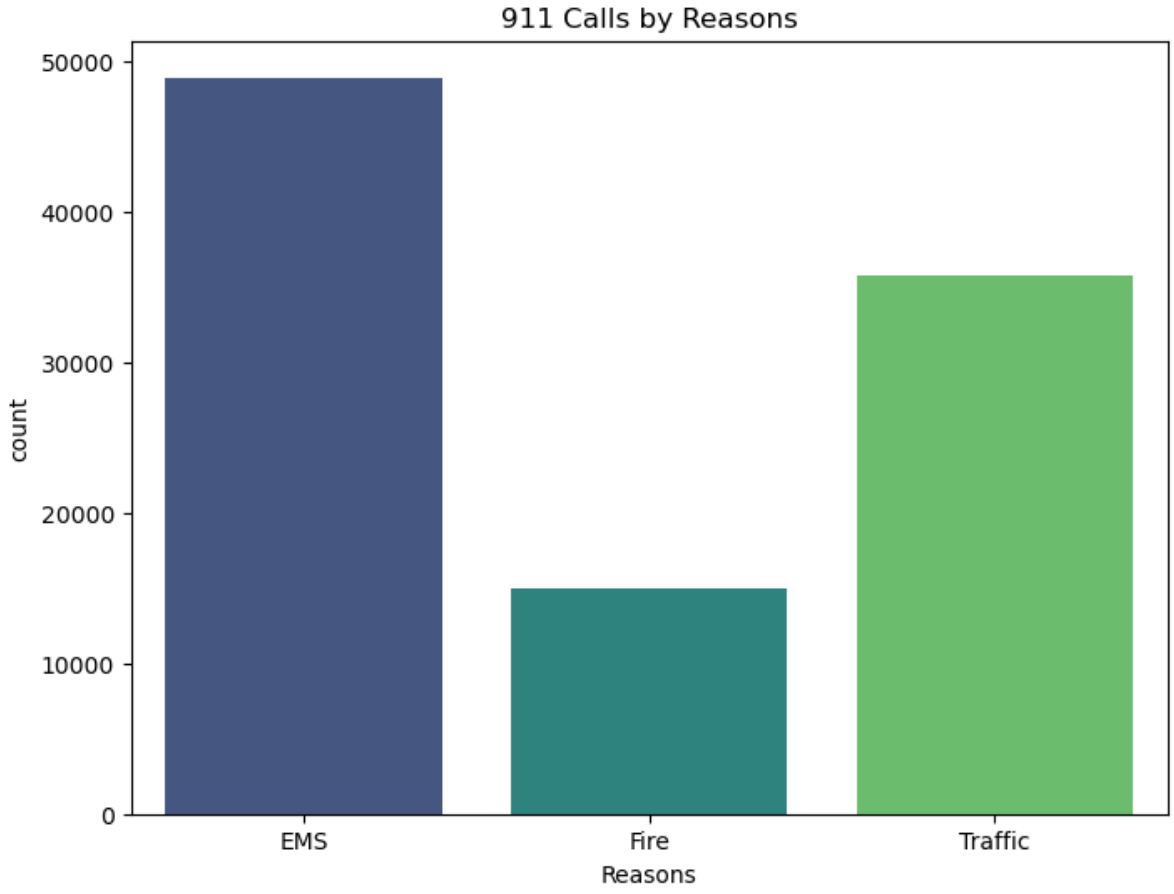
In [14]:
```
reasons_counts = df[reasons_column].value_counts()
```

In [15]:
```
reasons_counts
```

Out[15]:
```
EMS        48877
Traffic    35695
Fire       14920
Name: Reasons, dtype: int64
```

In [16]:
```
plt.figure(figsize=(8, 6))
plt.title('911 Calls by Reasons')
sns.countplot(x=reasons_column, data=df, palette='viridis')
```

Out[16]:
```
<Axes: title={'center': '911 Calls by Reasons'}, xlabel='Reasons', ylabel='count'>
```



In [17]:
```
type(df['timeStamp'].iloc[0])
```

Out[17]:
```
str
```

In [18]:
```
df['timeStamp'] = pd.to_datetime(df['timeStamp'], format='%Y-%m-%d %H:%M:%S')
```

In [19]:
```
time = df['timeStamp'].iloc[0]
time.hour
```

Out[19]:
```
17
```

In [20]:
```
df['Date'] = df['timeStamp'].dt.to_period('M')
```

In [21]:
```
month_names = {
    1: 'Jan', 2: 'Feb', 3: 'Mar', 4: 'Apr',
    5: 'May', 6: 'Jun', 7: 'Jul', 8: 'Aug',
    9: 'Sep', 10: 'Oct', 11: 'Nov', 12: 'Dec'}
```

In [22]:
```
"Data type of 'timeStamp' column after conversion:", df['timeStamp'].iloc[0]
```

Out[22]:
```
("Data type of 'timeStamp' column after conversion:",
 Timestamp('2015-12-10 17:40:00'))
```

In [23]:
```
df['Hour'] = df['timeStamp'].apply(lambda time: time.hour)
df['Month'] = df['timeStamp'].apply(lambda time: time.month)
df['Day of Week'] = df['timeStamp'].apply(lambda time: time.dayofweek)
```

In [24]:
```
df[['Hour', 'Month', 'Day of Week']].head()
```

Out[24]:

	Hour	Month	Day of Week
0	17	12	3
1	17	12	3
2	17	12	3
3	17	12	3
4	17	12	3

In [25]:
```
dmap = {0: 'Mon', 1: 'Tue', 2: 'Wed', 3: 'Thu', 4: 'Fri', 5: 'Sat', 6: 'Sun'}
```

In [26]:
```
df['Day of Week'] = df['Day of Week'].map(dmap)
```

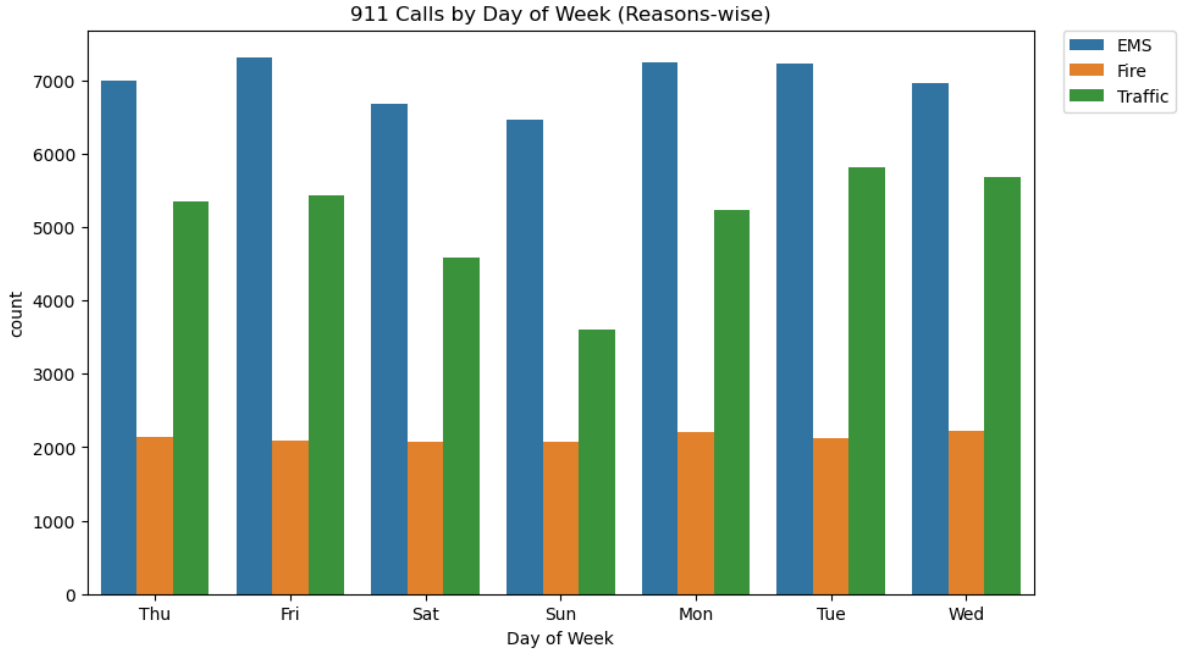
In [27]:
```
df[['Hour', 'Month', 'Day of Week']].head()
```

Out[27]:

	Hour	Month	Day of Week
0	17	12	Thu
1	17	12	Thu
2	17	12	Thu
3	17	12	Thu
4	17	12	Thu

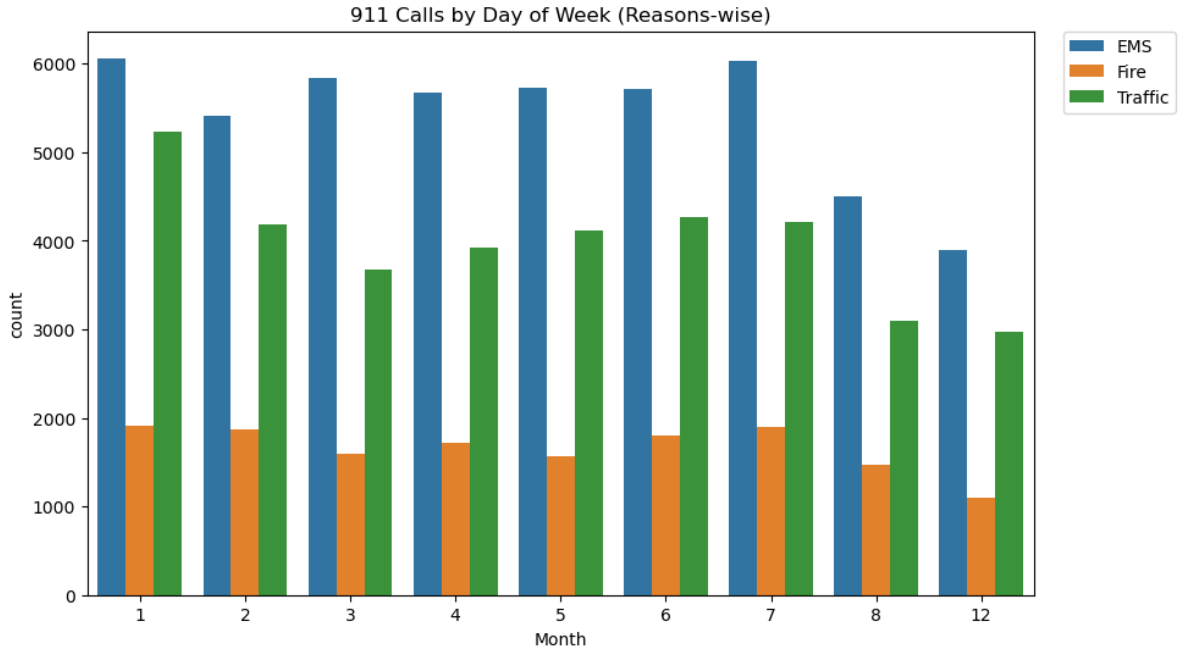
In [28]:
```
plt.figure(figsize=(10, 6))
sns.countplot(x='Day of Week', data=df, hue='Reasons')
plt.title('911 Calls by Day of Week (Reasons-wise)')
plt.legend(loc='upper right', bbox_to_anchor=(1.15, 1),borderaxespad=0)
```

Out[28]:
```
<matplotlib.legend.Legend at 0x230627ae8d0>
```



In [29]:
```
plt.figure(figsize=(10, 6))
sns.countplot(x='Month', data=df, hue='Reasons')
plt.title('911 Calls by Day of Week (Reasons-wise)')
plt.legend(loc='upper right', bbox_to_anchor=(1.15, 1),borderaxespad=0)
```

Out[29]:
```
<matplotlib.legend.Legend at 0x2305f945c50>
```



In [30]:
```
byMonth = df.groupby('Month').count()
```

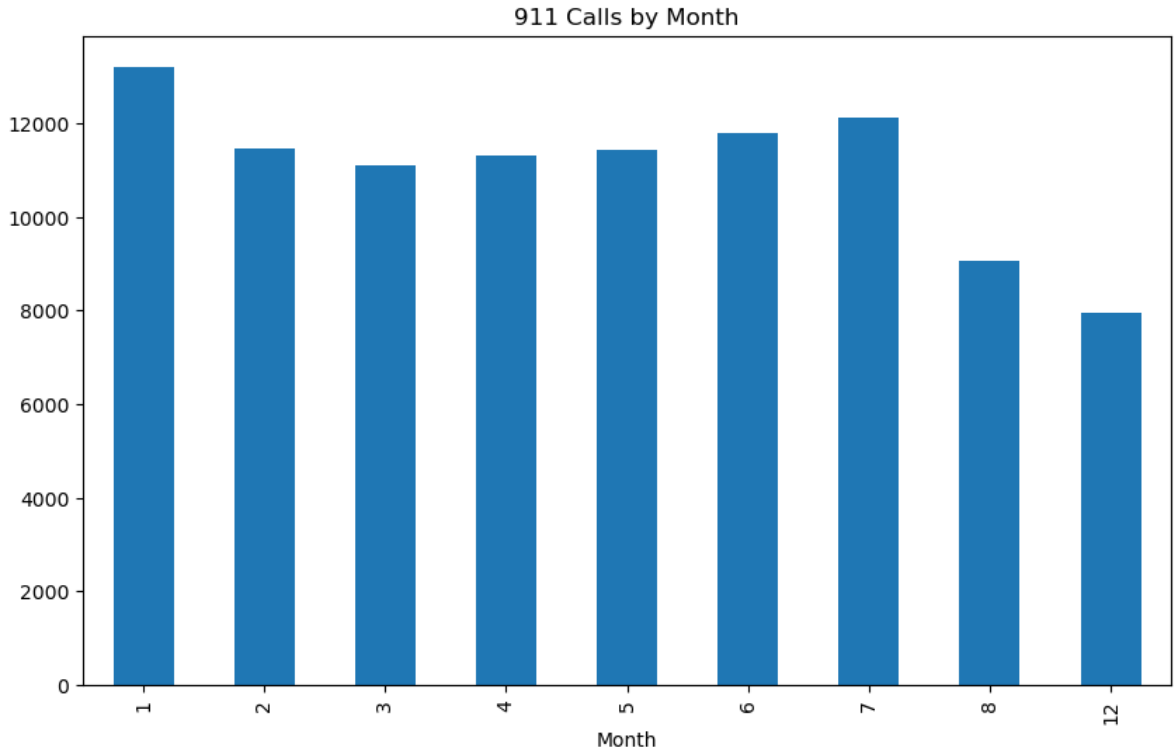
In [31]:
```
byMonth.head()
```

Out[31]:

Month	lat	lng	desc	zip	title	timeStamp	twp	addr	e	Reasons	Date	H
1	13205	13205	13205	11527	13205	13205	13203	13096	13205	13205	13205	13
2	11467	11467	11467	9930	11467	11467	11465	11396	11467	11467	11467	11
3	11101	11101	11101	9755	11101	11101	11092	11059	11101	11101	11101	11
4	11326	11326	11326	9895	11326	11326	11323	11283	11326	11326	11326	11
5	11423	11423	11423	9946	11423	11423	11420	11378	11423	11423	11423	11

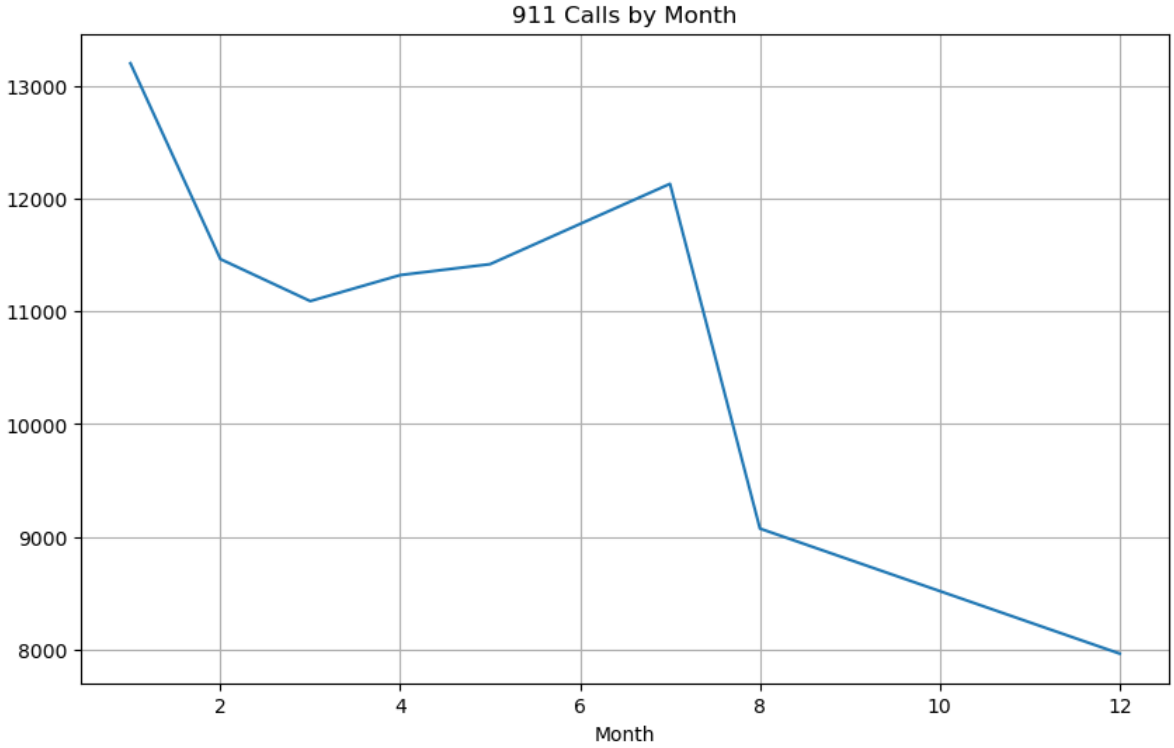
In [32]:
```
plt.figure(figsize=(10, 6))
byMonth['twp'].plot(kind='bar')
plt.title('911 Calls by Month')
```

Out[32]:
```
Text(0.5, 1.0, '911 Calls by Month')
```



In [33]:
```
plt.figure(figsize=(10, 6))
byMonth['twp'].plot(kind='line')
plt.grid(True)
plt.title('911 Calls by Month')
```

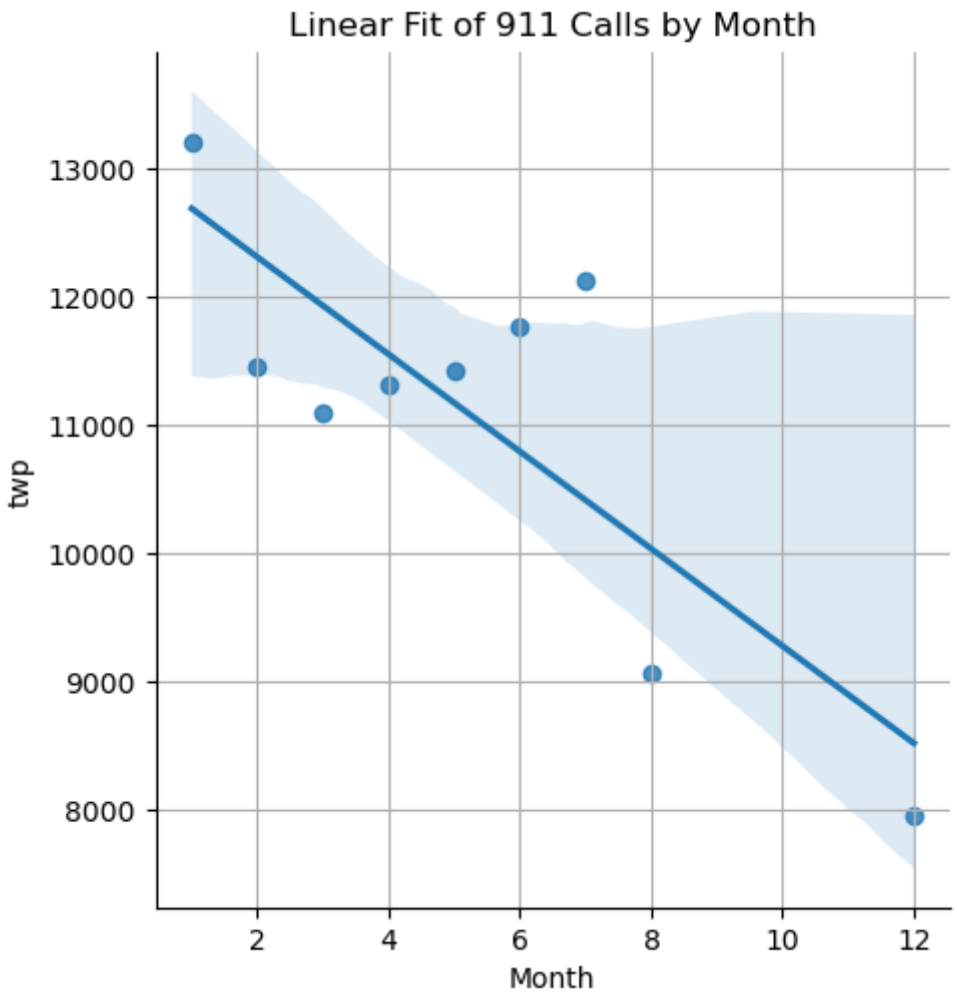
Out[33]:
```
Text(0.5, 1.0, '911 Calls by Month')
```



In [34]:
```
byMonth = byMonth.reset_index()
```

In [35]:
```
plt.figure(figsize=(10, 6))
sns.lmplot(x='Month', y='twp', data=byMonth)
plt.grid(True)
plt.title('Linear Fit of 911 Calls by Month')
```

Out[35]:
```
Text(0.5, 1.0, 'Linear Fit of 911 Calls by Month')

<Figure size 1000x600 with 0 Axes>
```



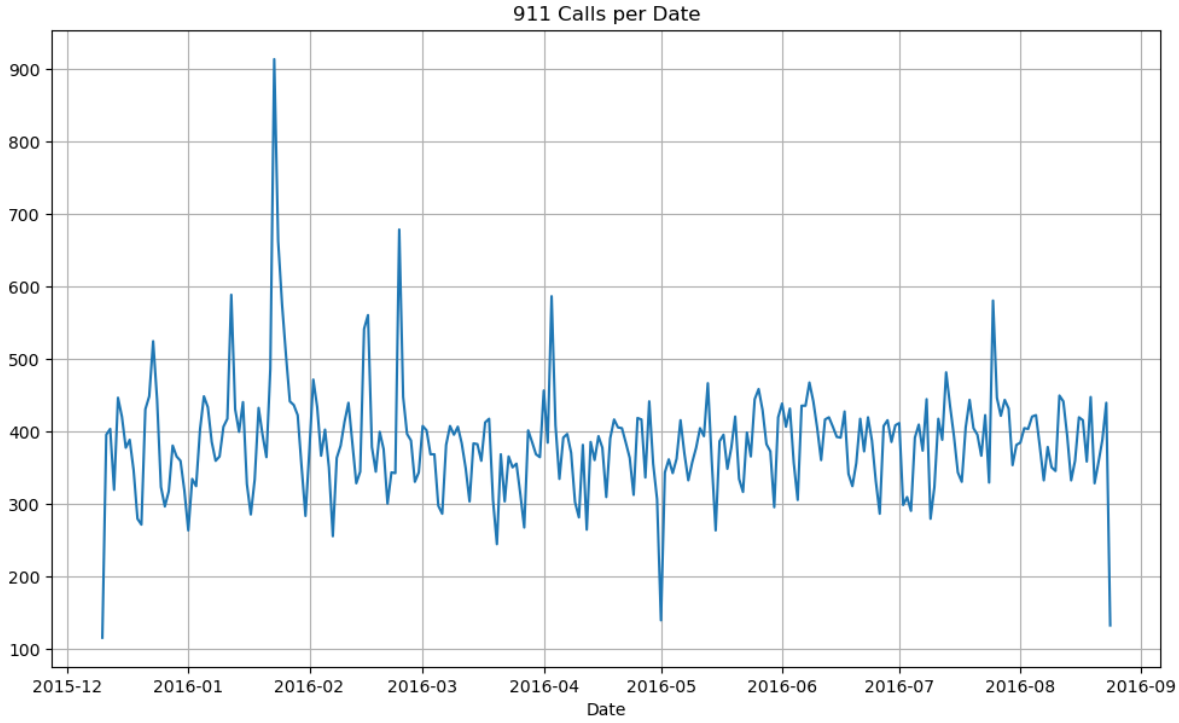
In [36]:
```
df['Date'] = df['timeStamp'].apply(lambda x:x.date())
```

In [37]:
```
df['Date'].head()
```

Out[37]:
```
0    2015-12-10
1    2015-12-10
2    2015-12-10
3    2015-12-10
4    2015-12-10
Name: Date, dtype: object
```

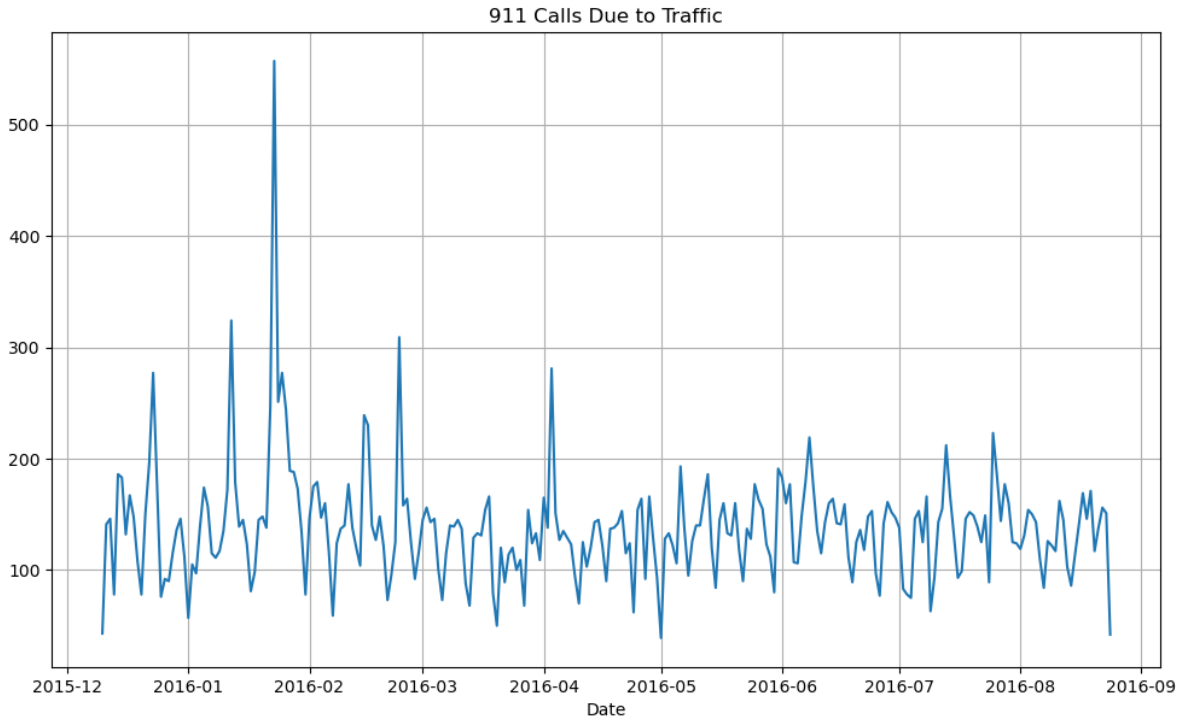
In [38]:
```
plt.figure(figsize=(10, 6))
Date = df.groupby('Date').count()['twp'].plot()
plt.tight_layout()
plt.grid(True)
plt.title('911 Calls per Date')
```

Out[38]:
```
Text(0.5, 1.0, '911 Calls per Date')
```



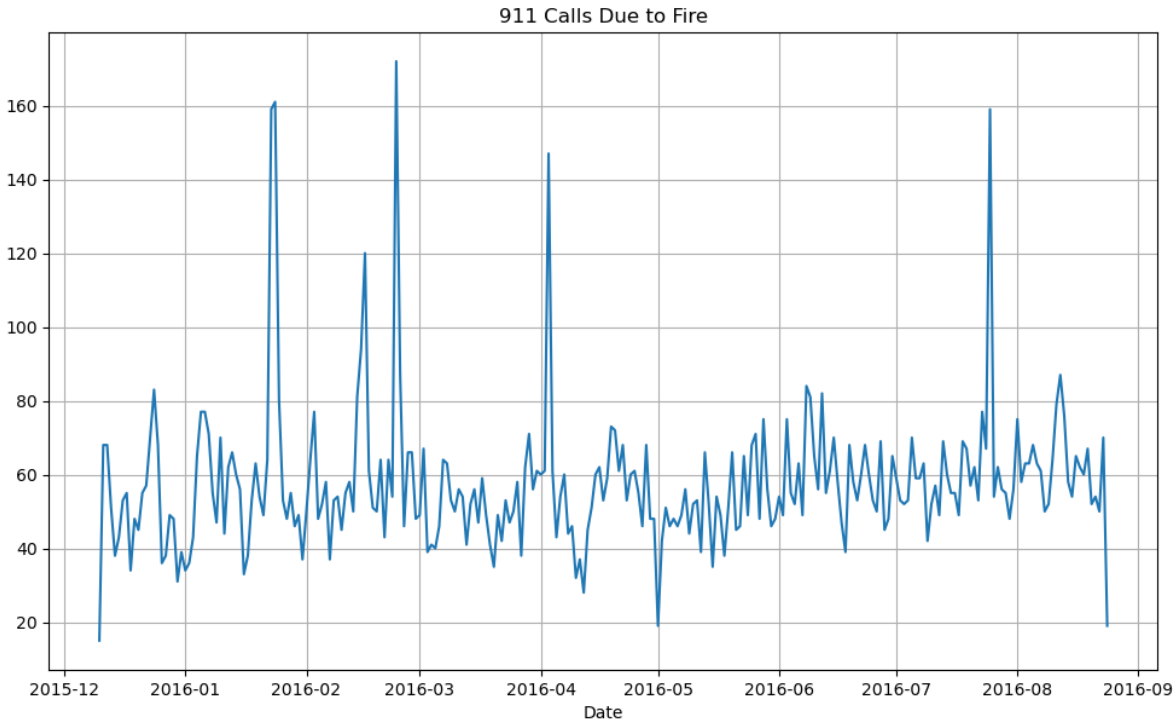
In [39]:
```
plt.figure(figsize=(10, 6))
df_traffic = df[df['Reasons'] == 'Traffic']
df_traffic.groupby('Date').count()['twp'].plot()
plt.tight_layout()
plt.grid(True)
plt.title('911 Calls Due to Traffic')
```

Out[39]:
```
Text(0.5, 1.0, '911 Calls Due to Traffic')
```



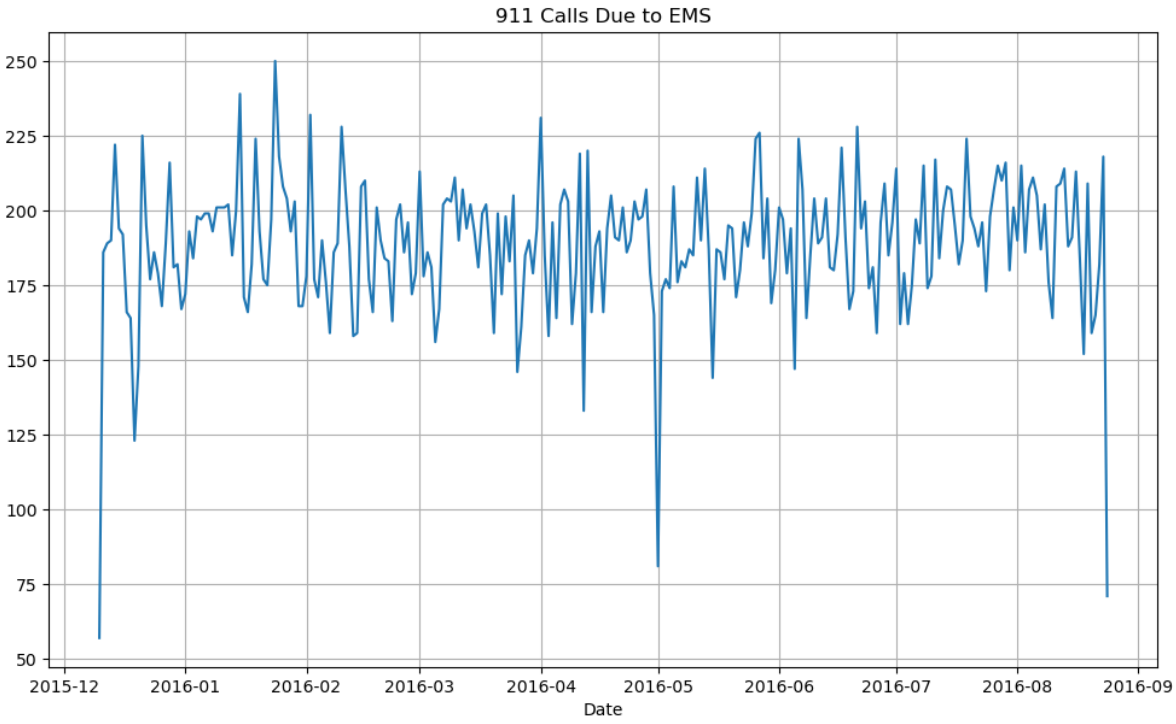
In [40]:
```
plt.figure(figsize=(10, 6))
df_traffic = df[df['Reasons'] == 'Fire']
df_traffic.groupby('Date').count()['twp'].plot()
plt.tight_layout()
plt.grid(True)
plt.title('911 Calls Due to Fire')
```

Out[40]:
```
Text(0.5, 1.0, '911 Calls Due to Fire')
```



In [41]:
```
plt.figure(figsize=(10, 6))
df_traffic = df[df['Reasons'] == 'EMS']
df_traffic.groupby('Date').count()['twp'].plot()
plt.tight_layout()
plt.grid(True)
plt.title('911 Calls Due to EMS')
```

Out[41]:
```
Text(0.5, 1.0, '911 Calls Due to EMS')
```



In [42]:
```
dayHour = df.groupby(['Day of Week','Hour']).count().unstack()['Reasons']
dayHour.head()
```

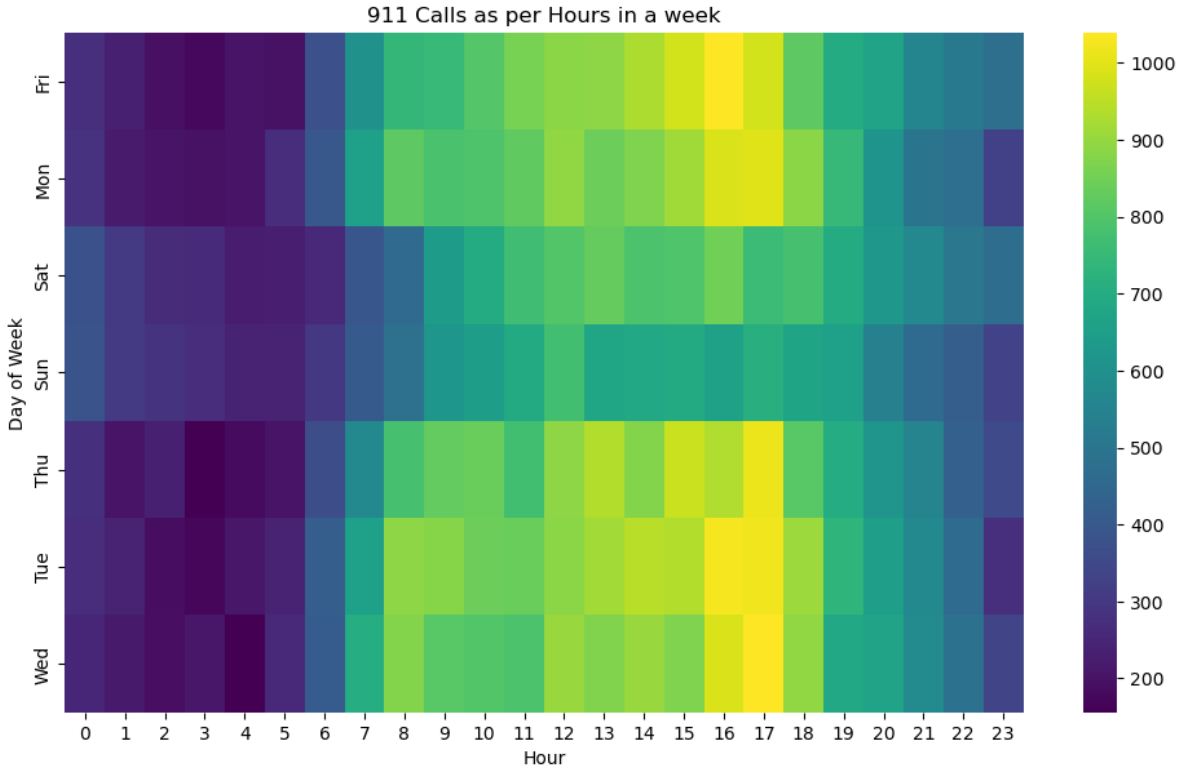
Out[42]:

Hour	0	1	2	3	4	5	6	7	8	9	...	14	15	16	17	18	19	2
Day of Week																		
Fri	275	235	191	175	201	194	372	598	742	752	...	932	980	1039	980	820	696	6(
Mon	282	221	201	194	204	267	397	653	819	786	...	869	913	989	997	885	746	6
Sat	375	301	263	260	224	231	257	391	459	640	...	789	796	848	757	778	696	6
Sun	383	306	286	268	242	240	300	402	483	620	...	684	691	663	714	670	655	5
Thu	278	202	233	159	182	203	362	570	777	828	...	876	969	935	1013	810	698	6

5 rows × 24 columns

In [43]:
```
plt.figure(figsize=(10, 6))
sns.heatmap(dayHour,cmap='viridis')
plt.tight_layout()
plt.title('911 Calls as per Hours in a week')
```

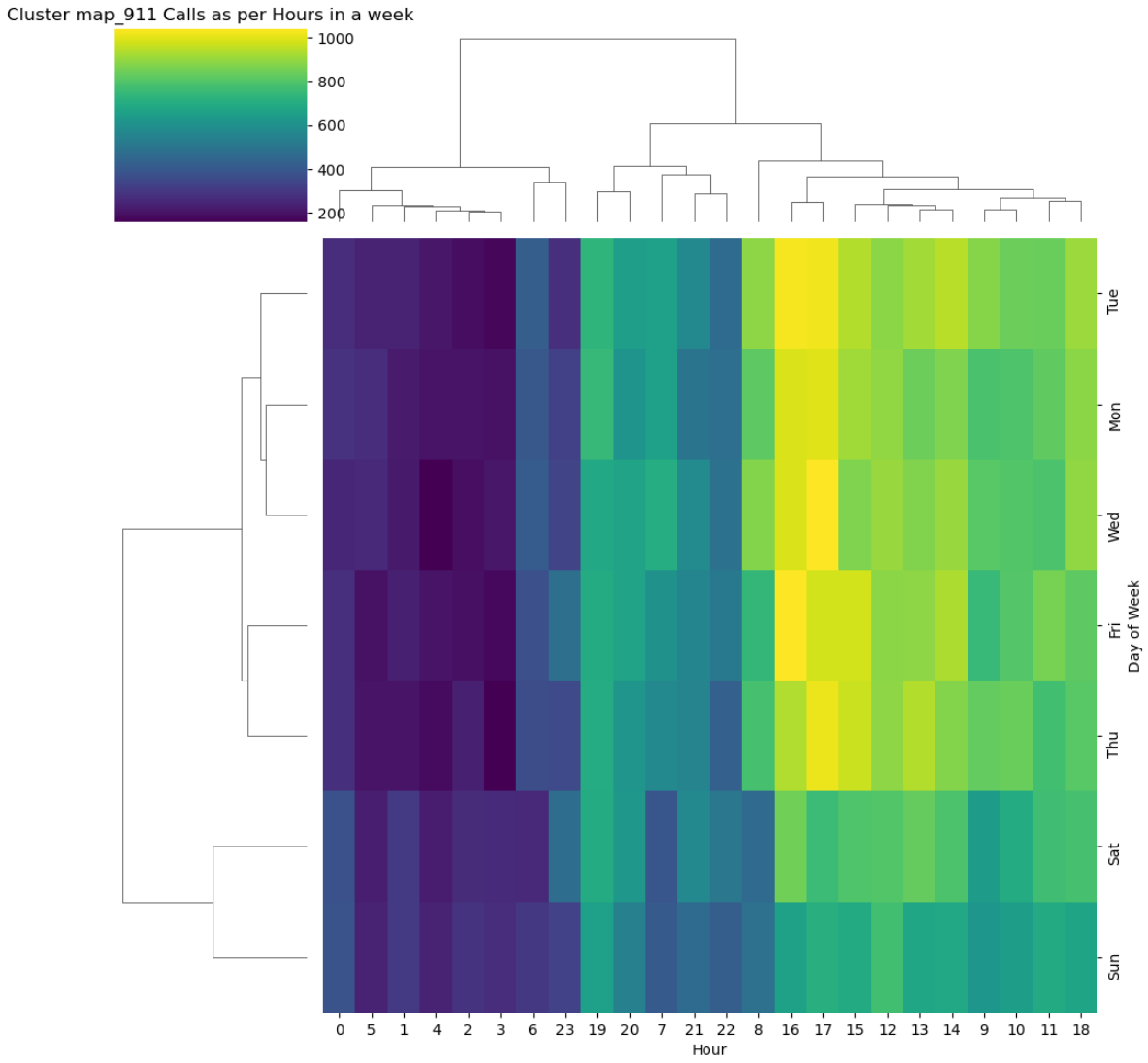
Out[43]:
```
Text(0.5, 1.0, '911 Calls as per Hours in a week')
```



In [44]:
```
plt.figure(figsize=(10,6))
sns.clustermap(dayHour, cmap='viridis')
plt.tight_layout()
plt.title('Cluster map_911 Calls as per Hours in a week')
```

Out[44]:
```
Text(0.5, 1.0, 'Cluster map_911 Calls as per Hours in a week')
```

```
<Figure size 1000x600 with 0 Axes>
```



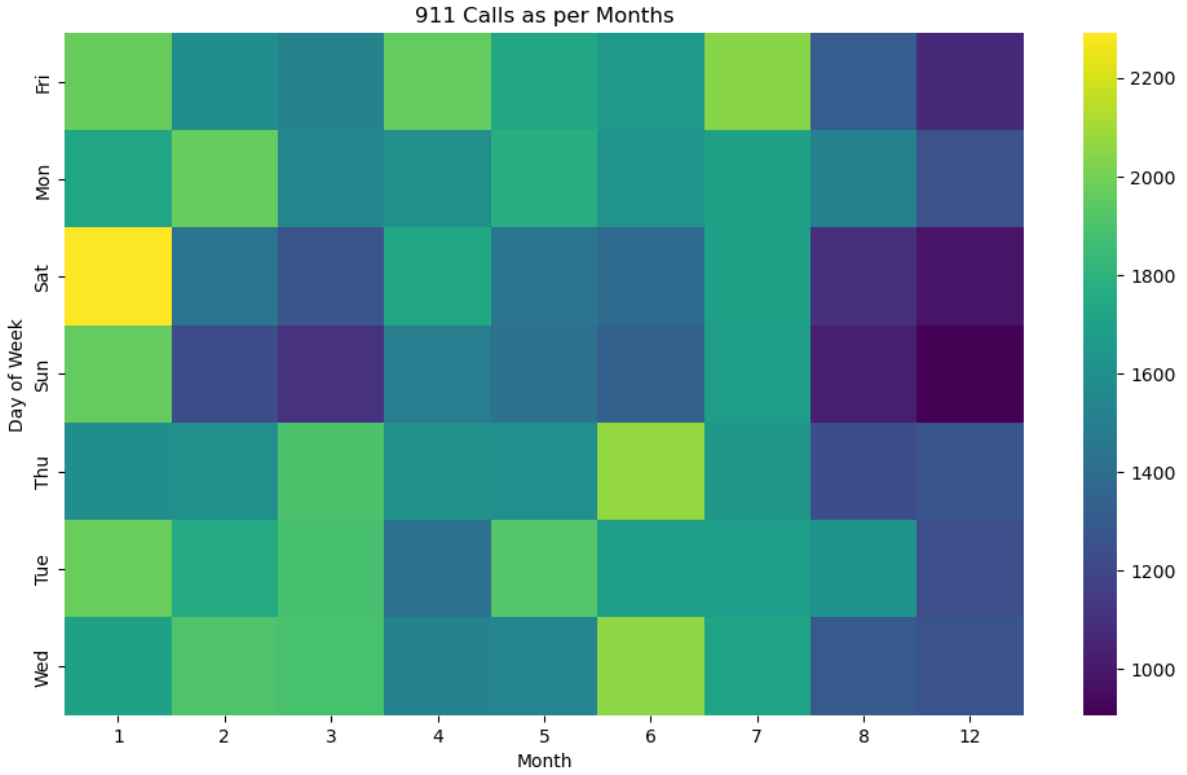
In [45]:
```
dayMonth = df.groupby(['Day of Week','Month']).count().unstack()['Reasons']
dayMonth.head()
```

Out[45]:

Month	1	2	3	4	5	6	7	8	12
Day of Week									
Fri	1970	1581	1525	1958	1730	1649	2045	1310	1065
Mon	1727	1964	1535	1598	1779	1617	1692	1511	1257
Sat	2291	1441	1266	1734	1444	1388	1695	1099	978
Sun	1960	1229	1102	1488	1424	1333	1672	1021	907
Thu	1584	1596	1900	1601	1590	2065	1646	1230	1266

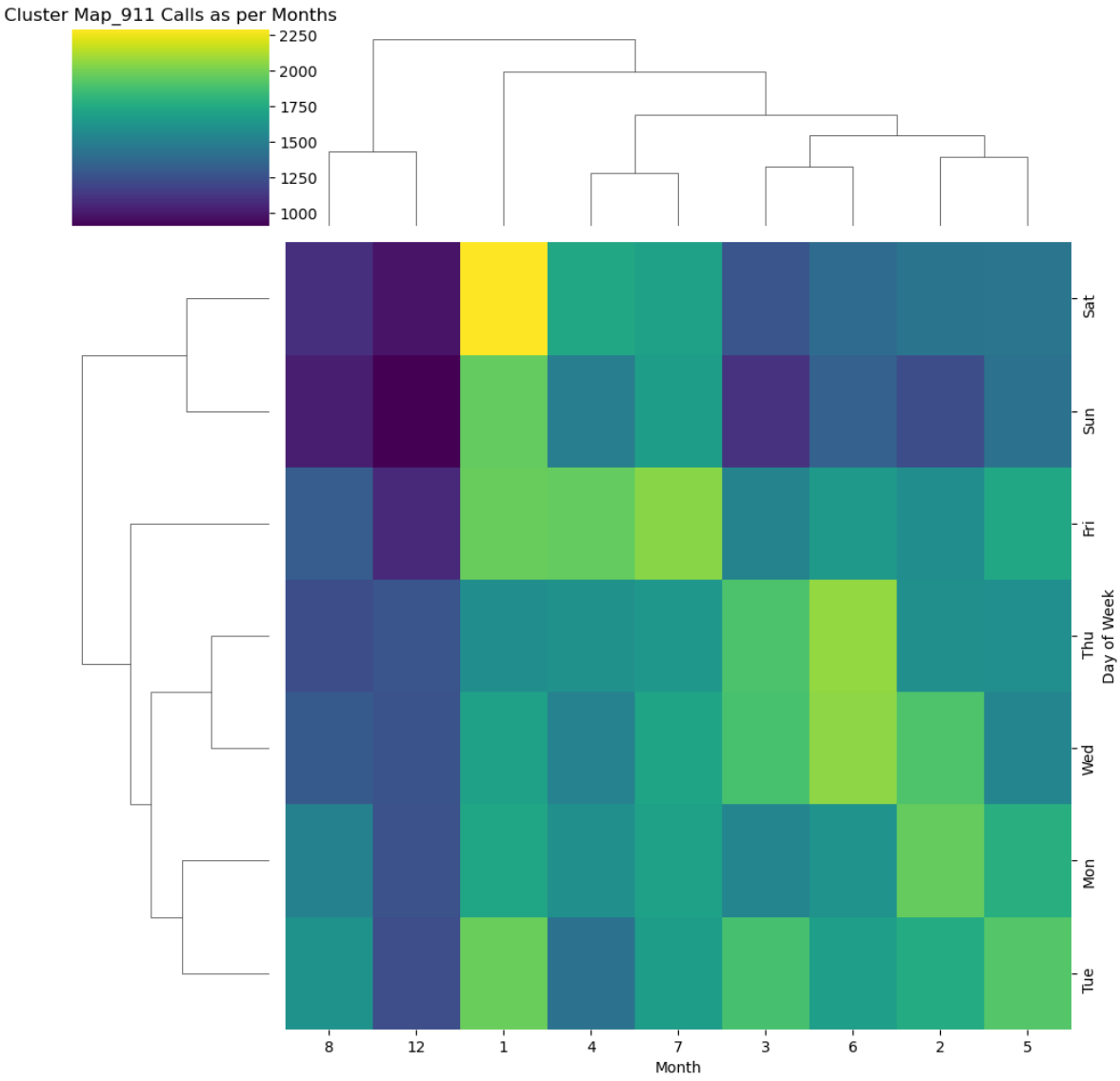
In [46]:
```
plt.figure(figsize=(10,6))
sns.heatmap(dayMonth,cmap='viridis')
plt.tight_layout()
plt.title('911 Calls as per Months')
```

Out[46]:
```
Text(0.5, 1.0, '911 Calls as per Months')
```



In [47]:
```
plt.figure(figsize=(10,6))
sns.clustermap(dayMonth, cmap='viridis')
plt.tight_layout()
plt.title('Cluster Map_911 Calls as per Months')
```

Out[47]:
```
Text(0.5, 1.0, 'Cluster Map_911 Calls as per Months')

<Figure size 1000x600 with 0 Axes>
```



Further Analysis of 911 Calls

In [48]:
```
### percentage of 911 calls for each reason
total_calls = len(df)
total_calls
```

Out[48]:
```
99492
```

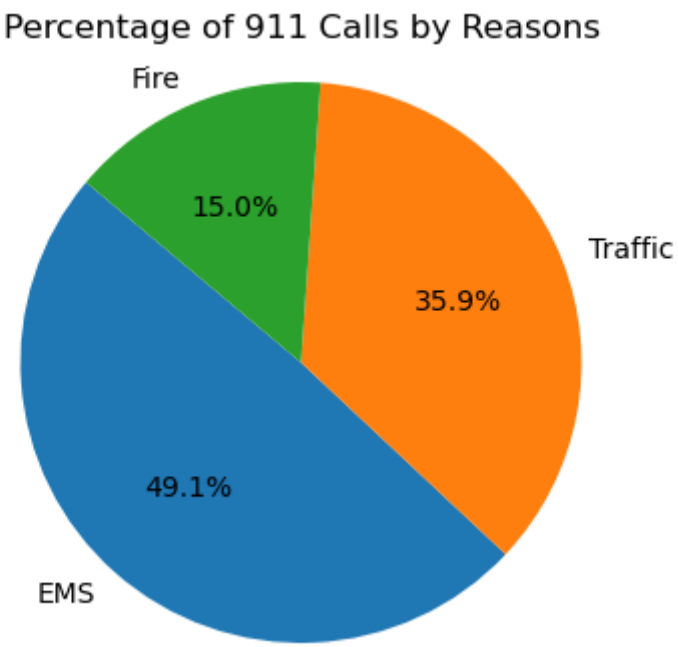
In [49]:
```
reasons_counts = df['Reasons'].value_counts()
```

In [50]:
```
percentage_by_reason = (reasons_counts / total_calls * 100).round(2)
```

In [51]:
```
percentage_by_reason
```

Out[51]:
```
EMS        49.13
Traffic    35.88
Fire       15.00
Name: Reasons, dtype: float64
```

In [52]:
```
plt.figure(figsize=(6, 4))
plt.pie(percentage_by_reason, labels=percentage_by_reason.index, autopct='%1.1f%%'
plt.title('Percentage of 911 Calls by Reasons')
plt.axis('equal')
plt.show()
```



In [53]:
```
# correaltion between numeric columns in df dataframe
numeric_columns = df.select_dtypes(include=['float64', 'int64']).columns
```

In [54]:
```
numeric_columns
```

Out[54]:
```
Index(['lat', 'lng', 'zip', 'e', 'Hour', 'Month'], dtype='object')
```

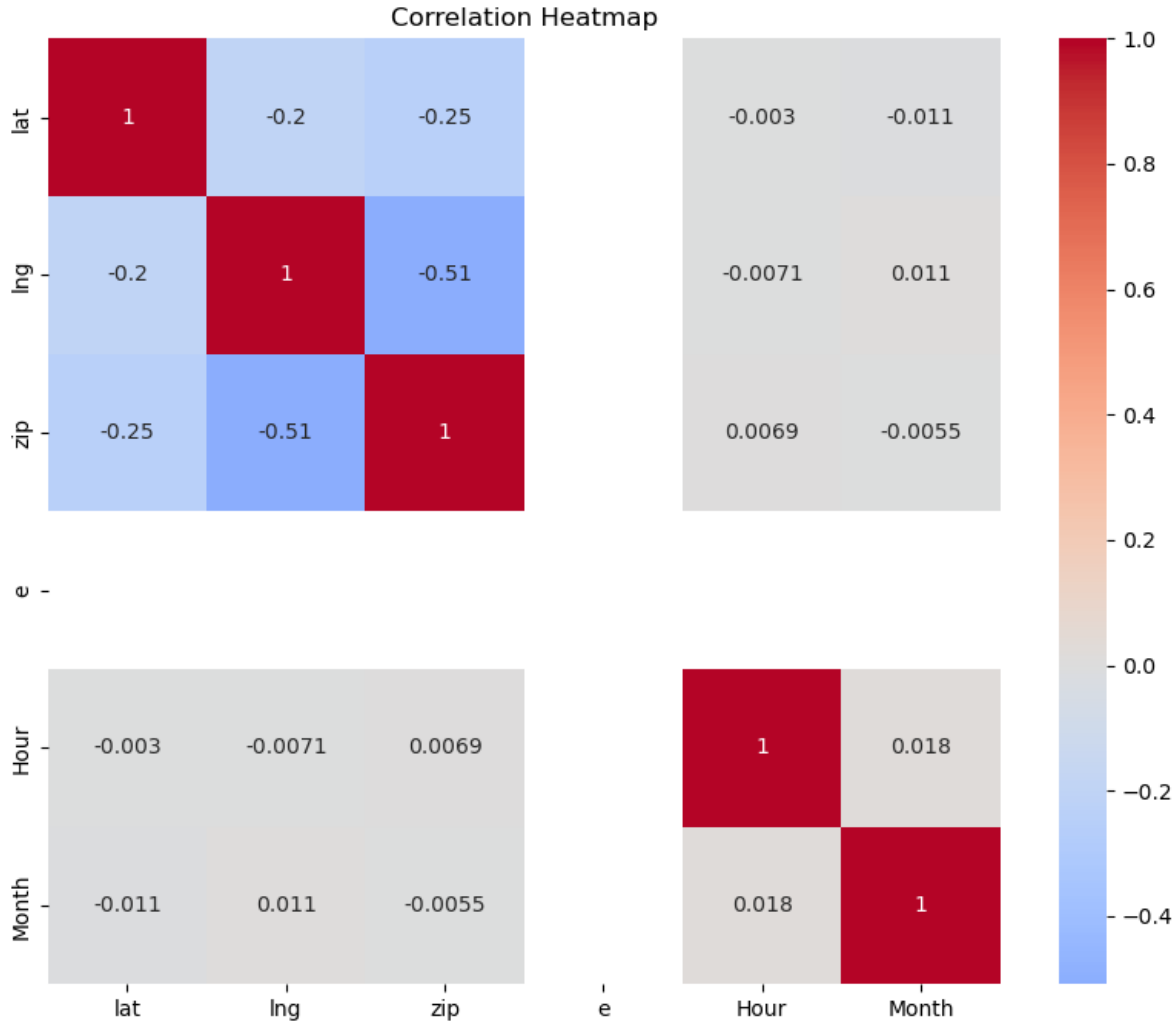
In [55]:
```
correlation_matrix = df[numeric_columns].corr()
df[numeric_columns]
```

Out[55]:

	lat	lng	zip	e	Hour	Month
0	40.297876	-75.581294	19525.0	1	17	12
1	40.258061	-75.264680	19446.0	1	17	12
2	40.121182	-75.351975	19401.0	1	17	12
3	40.116153	-75.343513	19401.0	1	17	12
4	40.251492	-75.603350	NaN	1	17	12
...	...	...	...	...	...	...
99487	40.132869	-75.333515	19401.0	1	11	8
99488	40.006974	-75.289080	19003.0	1	11	8
99489	40.115429	-75.334679	19401.0	1	11	8
99490	40.186431	-75.192555	19002.0	1	11	8
99491	40.207055	-75.317952	19446.0	1	11	8

99492 rows × 6 columns

In [56]:
```
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', center=0)
plt.title('Correlation Heatmap')
plt.show()
```



Response Time Analysis

Calculating and visualising the response times of 911 emergency services can analyse how response times vary based on time of day, day of the week, or type of reason. This also helps us understand the minimum response time of emergency services in critical situations.

In [57]:
```
df.columns
```

Out[57]:
```
Index(['lat', 'lng', 'desc', 'zip', 'title', 'timeStamp', 'twp', 'addr', 'e',
       'Reasons', 'Date', 'Hour', 'Month', 'Day of Week'],
      dtype='object')
```

In [58]:
```
df['timeStamp'] = pd.to_datetime(df['timeStamp'])
```

In [59]:
```
df['Hour'] = df['timeStamp'].dt.hour
df['Minute'] = df['timeStamp'].dt.minute
df['Second'] = df['timeStamp'].dt.second
```

In [60]:
```
df['ResponseTime'] = (df['timeStamp'] - df['timeStamp'].dt.normalize() +
    pd.to_timedelta(df['Hour'], unit='h') +
    pd.to_timedelta(df['Minute'], unit='m') +
    pd.to_timedelta(df['Second'], unit='s')).dt.total_seconds() / 60
```

In [61]:
```
avg_response_time = df['ResponseTime'].mean()
median_response_time = df['ResponseTime'].median()
```

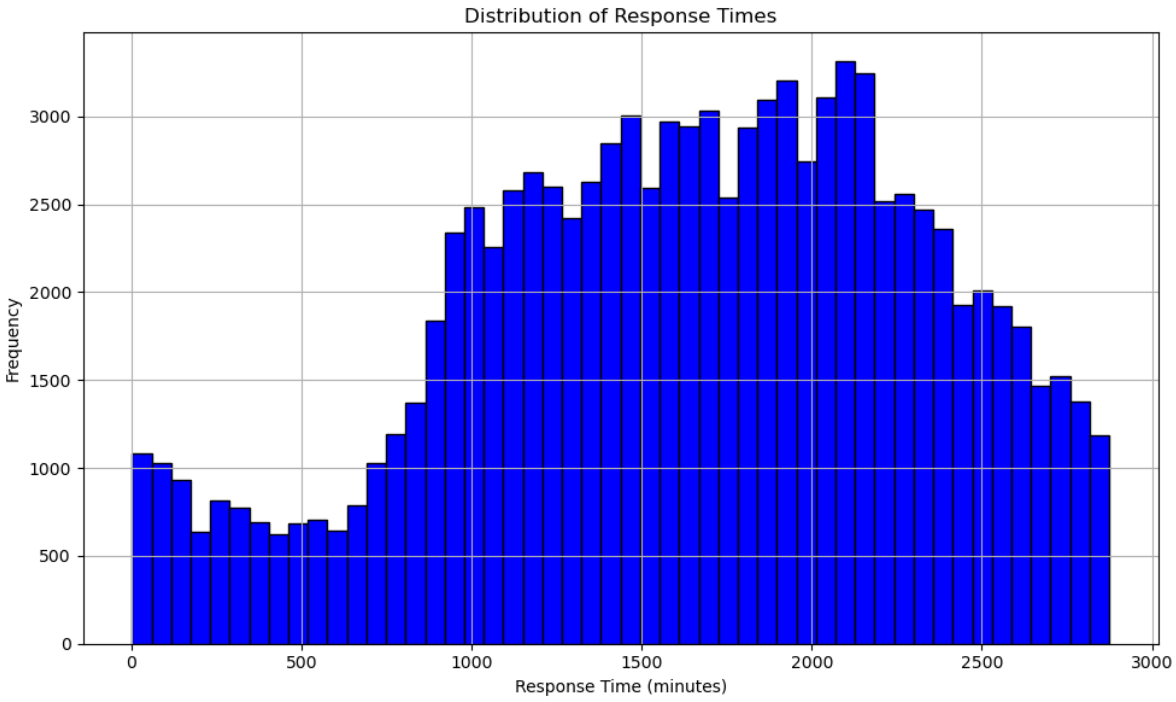
In [62]:
```
f"Average Response Time: {avg_response_time:.1f} min"
```

Out[62]:
```
'Average Response Time: 1632.7 min'
```

In [63]:
```
f"Median Response Time: {median_response_time:.1f} min"
```

Out[63]:
```
'Median Response Time: 1684.0 min'
```

In [64]:
```
###histogram to visualize the distribution of response times
plt.figure(figsize=(10, 6))
plt.hist(df['ResponseTime'], bins=50, color='b', edgecolor='black')
plt.title('Distribution of Response Times')
plt.xlabel('Response Time (minutes)')
plt.ylabel('Frequency')
plt.grid(True)
plt.tight_layout()
plt.show()
```



In [65]:
```
###average response time per day
avg_response_by_day = df.groupby('Day of Week')['ResponseTime'].mean().round(2)
```

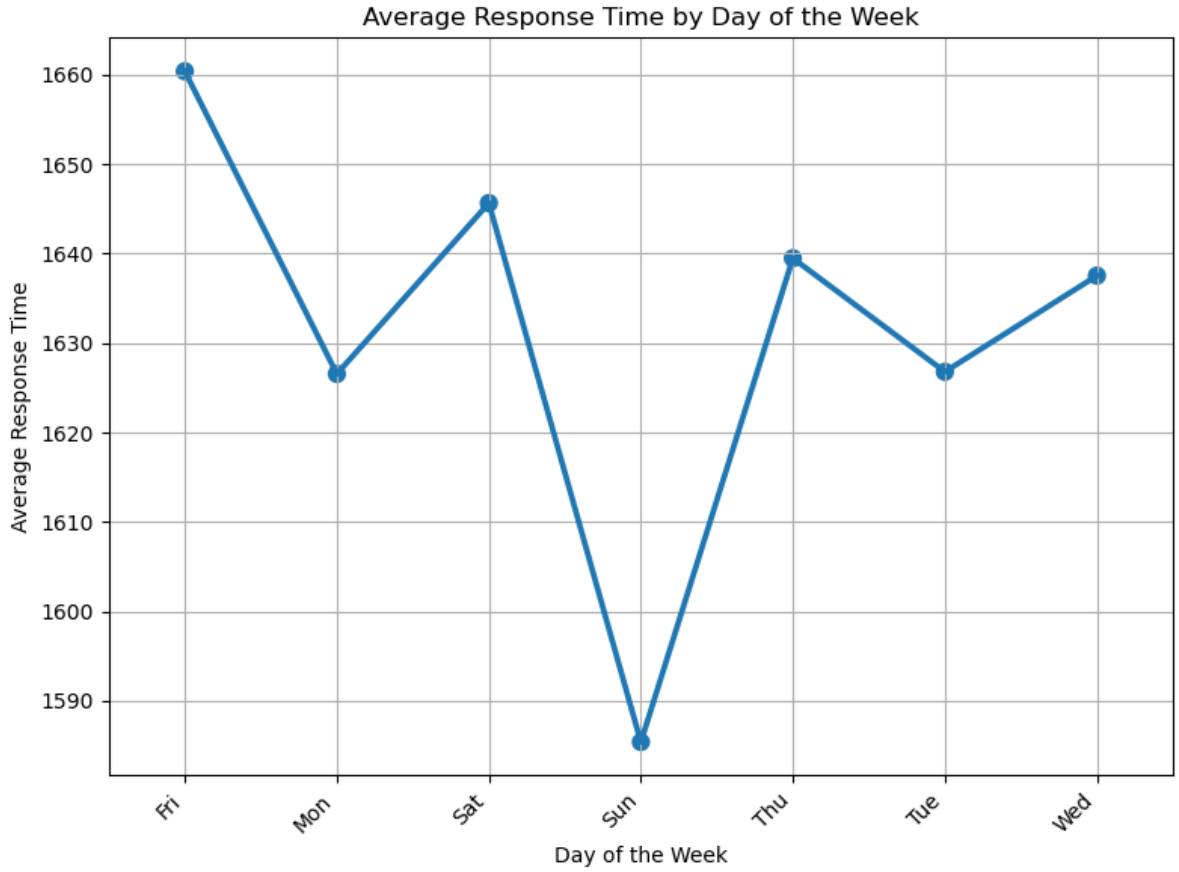
In [66]:
```
avg_response_by_day
```

Out[66]:
```
Day of Week
Fri    1660.40
Mon    1626.55
Sat    1645.66
Sun    1585.38
Thu    1639.49
Tue    1626.79
Wed    1637.54
Name: ResponseTime, dtype: float64
```

In [67]:
```
avg_response_by_day
```

Out[67]:
```
Day of Week
Fri    1660.40
Mon    1626.55
Sat    1645.66
Sun    1585.38
Thu    1639.49
Tue    1626.79
Wed    1637.54
Name: ResponseTime, dtype: float64
```

In [68]:
```
plt.figure(figsize=(8, 6))
sns.pointplot(x=avg_response_by_day.index, y=avg_response_by_day.values)
plt.xlabel('Day of the Week')
plt.ylabel('Average Response Time')
plt.title('Average Response Time by Day of the Week')
plt.xticks(rotation=45, ha='right')
plt.grid(True)
plt.tight_layout()
plt.show()
```

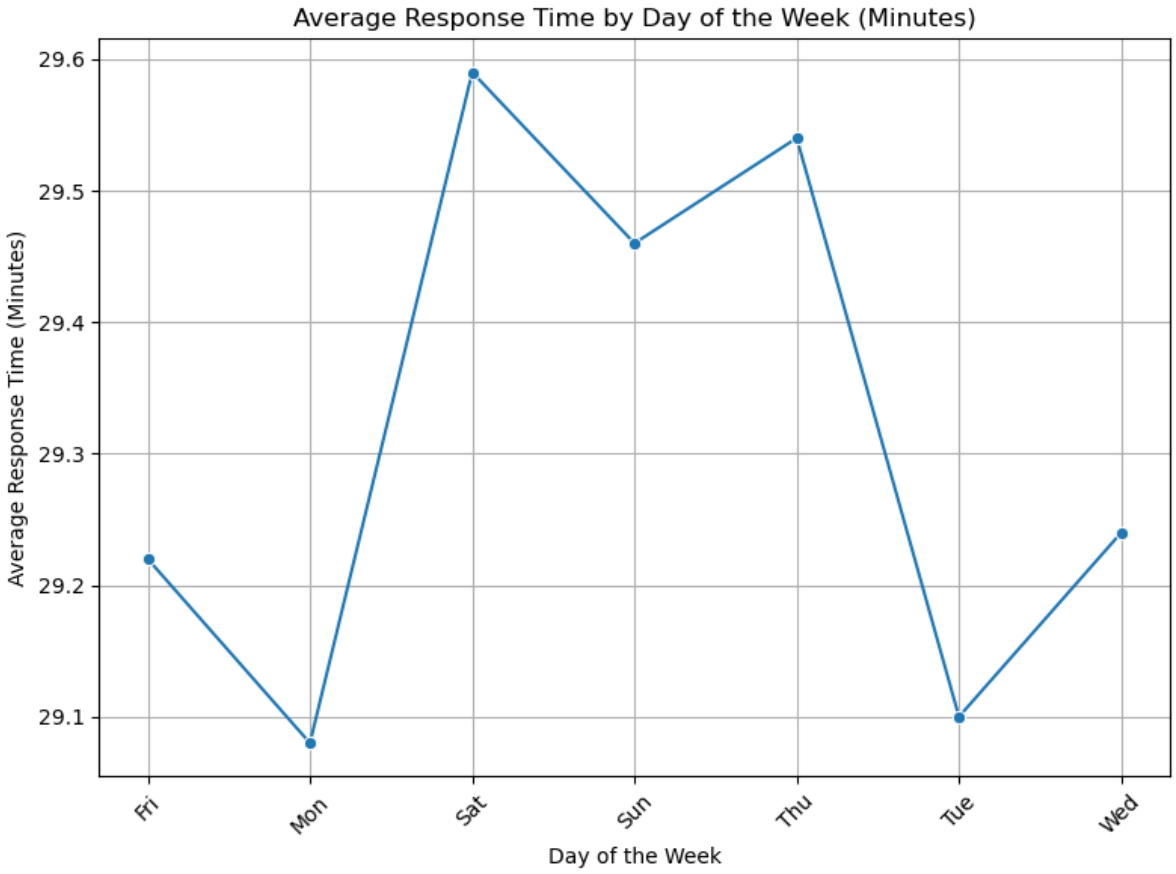


In [69]:
```
avg_response_by_day_minutes = df.groupby('Day of Week')['Minute'].mean().round(2)
```

In [70]:
```
avg_response_by_day_minutes
```

Out[70]:
```
Day of Week
Fri    29.22
Mon    29.08
Sat    29.59
Sun    29.46
Thu    29.54
Tue    29.10
Wed    29.24
Name: Minute, dtype: float64
```

In [71]:
```
plt.figure(figsize=(8, 6))
sns.lineplot(x=avg_response_by_day_minutes.index, y=avg_response_by_day_minutes.va
plt.xlabel('Day of the Week')
plt.ylabel('Average Response Time (Minutes)')
plt.title('Average Response Time by Day of the Week (Minutes)')
plt.xticks(rotation=45)
plt.grid(True)
plt.tight_layout()
plt.show()
```

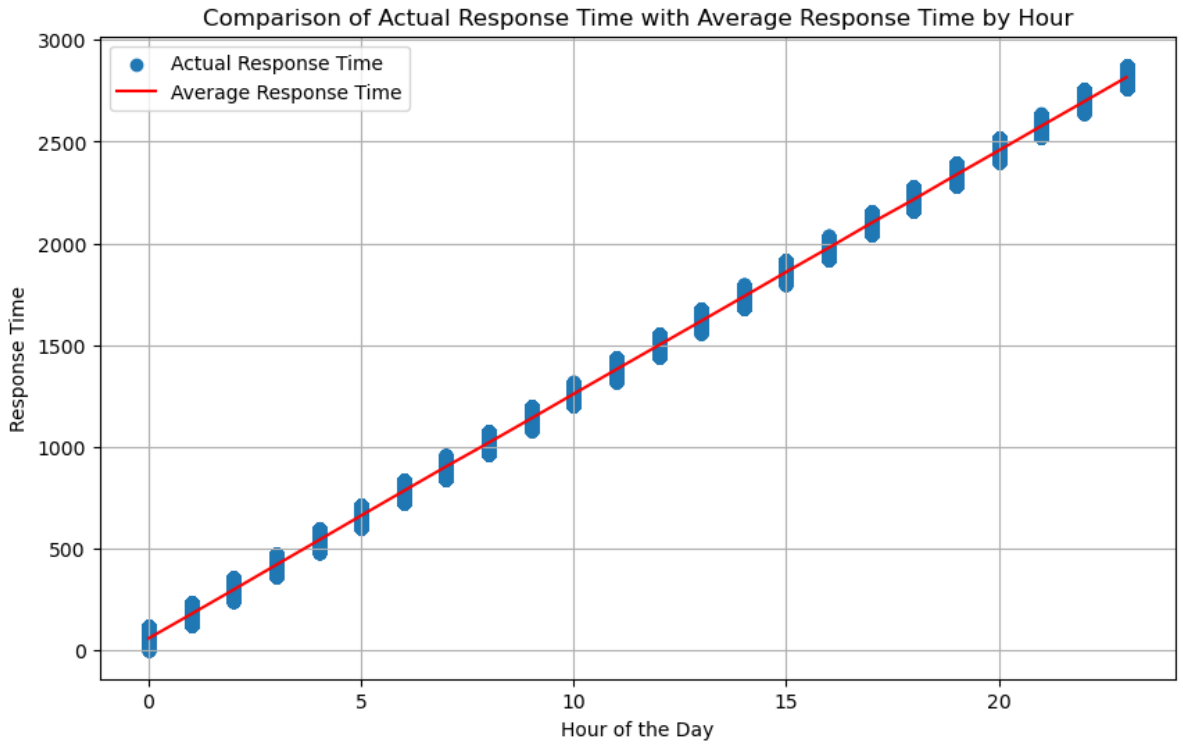


In [72]:
```
###Response Time Patterns by Hour
avg_response_time_by_hour = df.groupby('Hour')['ResponseTime'].mean().round(2)
```

In [73]:
```
avg_response_time_by_hour
```

Out[73]:
```
Hour
0       57.84
1      176.90
2      297.31
3      418.98
4      540.05
5      661.51
6      782.51
7      903.00
8     1019.67
9     1139.01
10    1259.67
11    1379.83
12    1498.94
13    1618.20
14    1738.29
15    1859.36
16    1978.33
17    2099.77
18    2215.90
19    2337.48
20    2456.39
21    2576.80
22    2696.24
23    2816.08
Name: ResponseTime, dtype: float64
```

In [74]:
```
plt.figure(figsize=(10, 6))
plt.scatter(df['Hour'], df['ResponseTime'], label='Actual Response Time')
plt.plot(avg_response_time_by_hour.index, avg_response_time_by_hour.values, color=
plt.xlabel('Hour of the Day')
plt.ylabel('Response Time')
plt.title('Comparison of Actual Response Time with Average Response Time by Hour')
plt.legend()
plt.grid(True)
plt.show()
```



Response Time Analysis based on Township (twp) Column

This analysis offers valuable insights into the efficiency of emergency services in various townships, aiding resource allocation and optimization efforts for improving emergency response management.

In [75]:
```
avg_response_by_twp = df.groupby('twp')['ResponseTime'].mean().reset_index().round
```

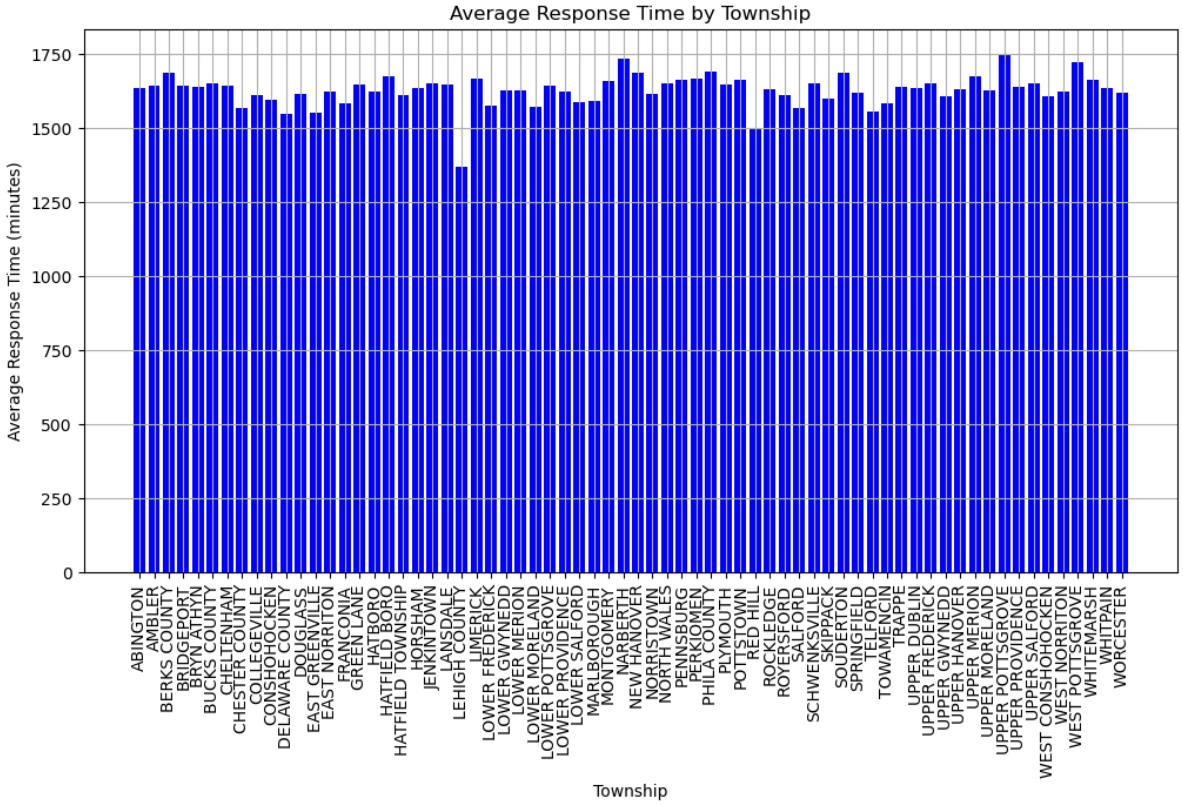
In [76]:
```
avg_response_by_twp
```

Out[76]:

	twp	ResponseTime
0	ABINGTON	1634.26
1	AMBLER	1641.57
2	BERKS COUNTY	1686.44
3	BRIDGEPORT	1643.83
4	BRYN ATHYN	1638.87
...	...	...
63	WEST NORRITON	1625.08
64	WEST POTTSGROVE	1723.38
65	WHITEMARSH	1662.63
66	WHITPAIN	1634.60
67	WORCESTER	1618.83

68 rows × 2 columns

In [77]:
```
plt.figure(figsize=(12, 6))
plt.bar(avg_response_by_twp['twp'], avg_response_by_twp['ResponseTime'], color='blu
plt.title('Average Response Time by Township')
plt.xlabel('Township')
plt.ylabel('Average Response Time (minutes)')
plt.xticks(rotation=90)
plt.grid(True)
plt.show()
```



In [78]:
```
from scipy.stats import chi2_contingency
```

In [79]:
```
#chi-squared test
contingency_table_response = pd.crosstab(df['twp'], df['ResponseTime'])
chi2_response, p_response, dof_response, expected_response = chi2_contingency(cont
```

In [80]:
```
print("Chi-squared:", round(chi2_response, 1))
print("p-value:", round(p_response, 1))
print("Degrees of Freedom:", dof_response)
print("Expected Frequencies:\n", np.round(expected_response, 1))
```

```
Chi-squared: 215548.0
p-value: 0.0
Degrees of Freedom: 201536
Expected Frequencies:
 [[1.1 0.4 0.1 ... 0.1 0.1 0.1]
 [0.1 0.  0.  ... 0.  0.  0. ]
 [0.1 0.  0.  ... 0.  0.  0. ]
 ...
 [0.5 0.2 0.  ... 0.1 0.1 0. ]
 [0.4 0.2 0.  ... 0.  0.  0. ]
 [0.1 0.1 0.  ... 0.  0.  0. ]]
```

The data reveals a strong link between township and response time, with a substantial chi-squared statistic and very low p-value highlighting a significant departure from independence. Expected frequencies further contextualize observed counts, offering insight into the robustness of this township-based Response Time Analysis.

Time Series Analysis

Explore trends in 911 calls over time by analyzing call volumes across different hours, days, months, or even years. Time series analysis Can reveal patterns and trends in emergency call

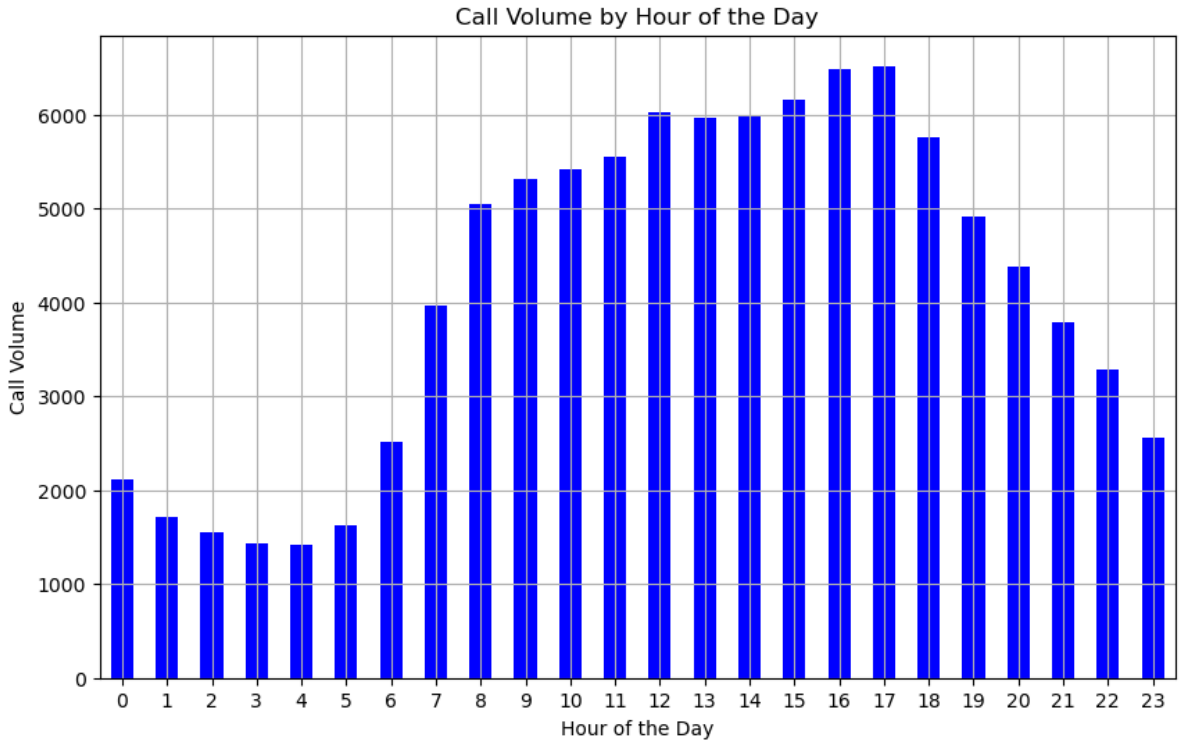
volumes and response times. Here are a few time-based analysis:

In [81]:
```
###Call Volume by Hour of the Day df['Hour'] = df['timeStamp'].dt.hour
call_volume_by_Hour = df.groupby('Hour').size()
```

In [82]:
```
call_volume_by_Hour
```

Out[82]:
```
Hour
0     2112
1     1721
2     1549
3     1435
4     1418
5     1629
6     2513
7     3970
8     5044
9     5314
10    5413
11    5543
12    6029
13    5967
14    5997
15    6154
16    6490
17    6517
18    5762
19    4908
20    4377
21    3788
22    3283
23    2559
dtype: int64
```

In [83]:
```
plt.figure(figsize=(10, 6))
call_volume_by_Hour.plot(kind='bar', color='blue')
plt.title('Call Volume by Hour of the Day')
plt.xlabel('Hour of the Day')
plt.ylabel('Call Volume')
plt.grid(True)
plt.xticks(rotation=0)
plt.show()
```



In [84]:
```
df = df.merge(call_volume_by_Hour.rename('call_volume_by_Hour'), on='Hour', how='l
```

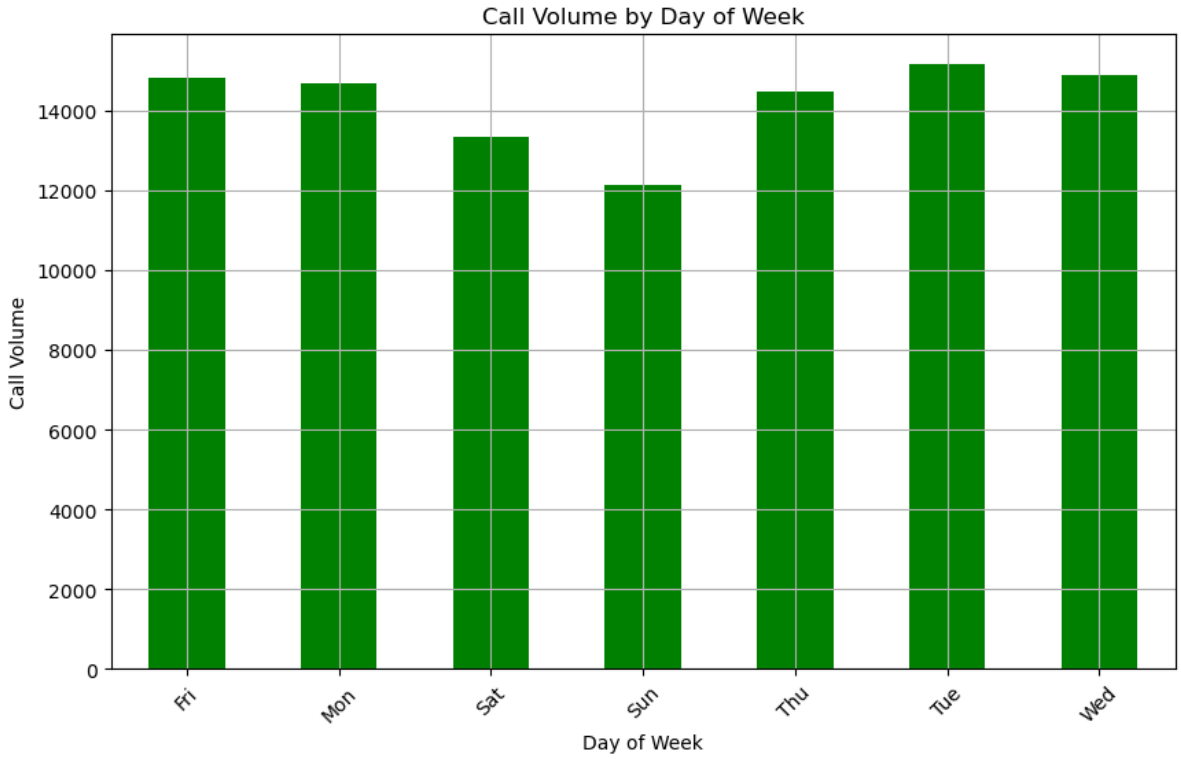
In [85]:
```
###Call Volume by Day of the Week
call_volume_by_day = df.groupby('Day of Week').size()
```

In [86]:
```
call_volume_by_day
```

Out[86]:
```
Day of Week
Fri    14833
Mon    14680
Sat    13336
Sun    12136
Thu    14478
Tue    15150
Wed    14879
dtype: int64
```

In [87]:
```
df = df.merge(call_volume_by_day.rename('call_volume_by_day'), how='left', left_on
```

In [88]:
```
plt.figure(figsize=(10, 6))
call_volume_by_day.plot(kind='bar', color='green')
plt.title('Call Volume by Day of Week')
plt.xlabel('Day of Week')
plt.ylabel('Call Volume')
plt.grid(True)
plt.xticks(range(7), rotation=45)
plt.show()
```

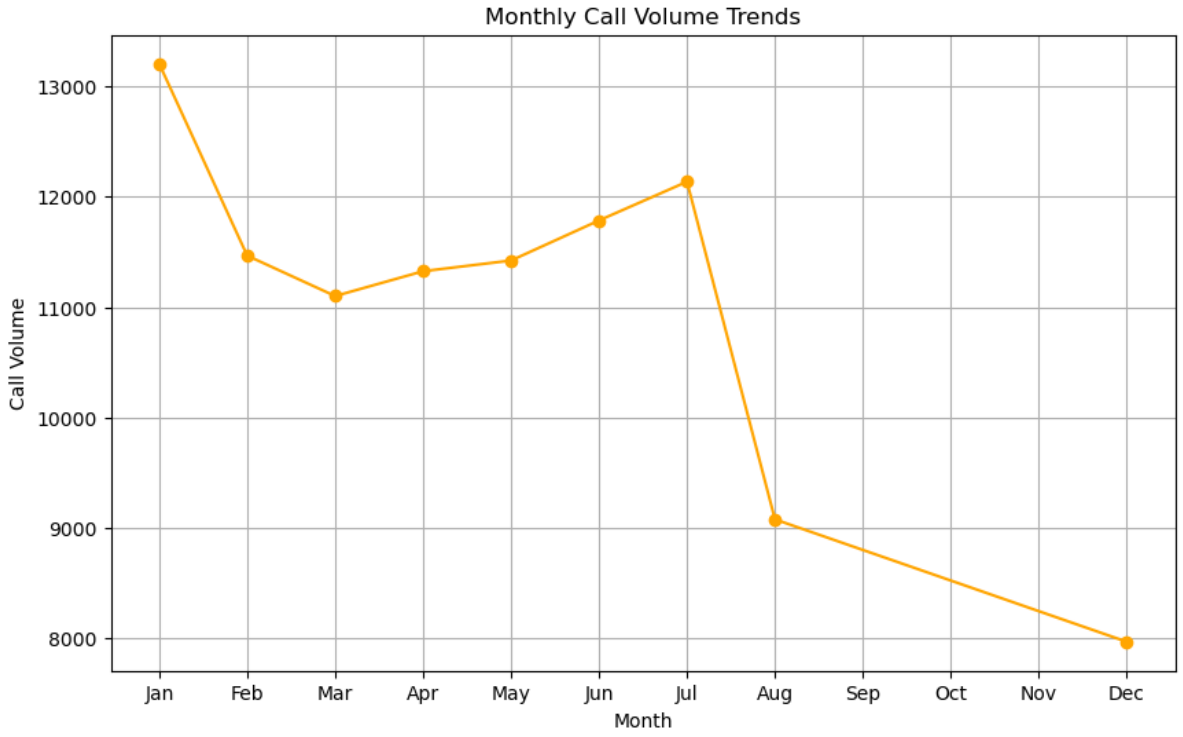


In [89]:
```
###Call Voulume_Monthly Call Trends
call_volume_by_month = df.groupby('Month').size()
```

In [90]:
```
call_volume_by_month
```

Out[90]:
```
Month
1     13205
2     11467
3     11101
4     11326
5     11423
6     11786
7     12137
8      9078
12     7969
dtype: int64
```

In [91]:
```
plt.figure(figsize=(10, 6))
call_volume_by_month.plot(kind='line', marker='o', color='orange')
plt.title('Monthly Call Volume Trends')
plt.xlabel('Month')
plt.ylabel('Call Volume')
plt.grid(True)
plt.xticks(range(1, 13), ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug',
plt.show()
```



In [92]:
```
df = df.merge(call_volume_by_month.rename('call_volume_by_month'), how='left', lef
```

In [93]:
```
###Call Volume_Peak Call Times
call_volume_by_Hour_day = df.groupby(['Day of Week', 'Hour']).size().unstack()
```

In [94]:
```
call_volume_by_Hour_day
```

Out[94]:

Hour	0	1	2	3	4	5	6	7	8	9	...	14	15	16	17	18	19	2
Day of Week																		
Fri	275	235	191	175	201	194	372	598	742	752	...	932	980	1039	980	820	696	6(
Mon	282	221	201	194	204	267	397	653	819	786	...	869	913	989	997	885	746	6
Sat	375	301	263	260	224	231	257	391	459	640	...	789	796	848	757	778	696	6
Sun	383	306	286	268	242	240	300	402	483	620	...	684	691	663	714	670	655	5
Thu	278	202	233	159	182	203	362	570	777	828	...	876	969	935	1013	810	698	6
Tue	269	240	186	170	209	239	415	655	889	880	...	943	938	1026	1019	905	731	6
Wed	250	216	189	209	156	255	410	701	875	808	...	904	867	990	1037	894	686	6(

7 rows × 24 columns

In [95]:
```
call_volume_by_hour_day = df.groupby(['Day of Week', 'Hour']).size().reset_index(n
```

In [96]:
```
df = df.merge(call_volume_by_hour_day, how='left', on=['Day of Week', 'Hour'])
```

In [97]:
```
###Call Volume by Reasons
call_volume_by_reasons = df.groupby('title').size()
```

In [98]:
```
call_volume_by_reasons
```

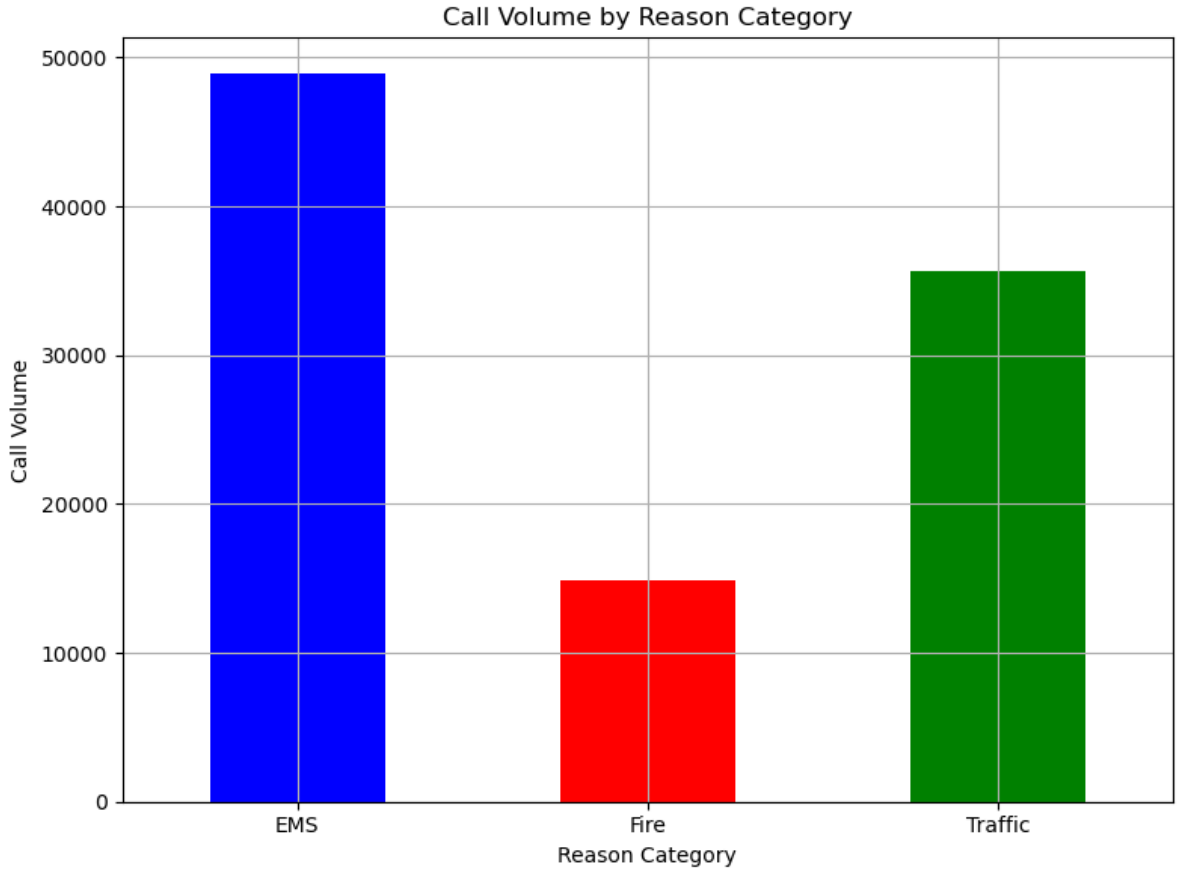
Out[98]:
```
title
EMS: ABDOMINAL PAINS                       1436
EMS: ACTIVE SHOOTER                           2
EMS: ALLERGIC REACTION                      438
EMS: ALTERED MENTAL STATUS                 1386
EMS: AMPUTATION                              14
                                           ...
Traffic: HAZARDOUS ROAD CONDITIONS -       1086
Traffic: ROAD OBSTRUCTION -                3144
Traffic: VEHICLE ACCIDENT -               23066
Traffic: VEHICLE FIRE -                     553
Traffic: VEHICLE LEAKING FUEL -              77
Length: 110, dtype: int64
```

In [99]:
```
call_volume_by_reasons = df.groupby('Reasons').size()
```

In [100…
```
call_volume_by_reasons
```

Out[100]:
```
Reasons
EMS        48877
Fire       14920
Traffic    35695
dtype: int64
```

In [101…
```
plt.figure(figsize=(8, 6))
ax = call_volume_by_reasons.plot(kind='bar', color=['blue', 'red', 'green'])
plt.title('Call Volume by Reason Category')
plt.xlabel('Reason Category')
plt.ylabel('Call Volume')
plt.grid(True)
plt.xticks(rotation=0)
x_labels = call_volume_by_reasons.index
ax.set_xticks(range(len(x_labels)))
ax.set_xticklabels(x_labels, fontsize=10)

plt.tight_layout()
plt.show()
```



In [102…
```
df = df.merge(call_volume_by_day.rename('call_volume_by_reasons'), how='left', lef
```

In [103…
```
### Call counts by per reasons[EMS,Fire,Traffic]
df['timeStamp'] = pd.to_datetime(df['timeStamp'])
df['Date'] = pd.to_datetime(df['Date'])
```

In [104…
```
reasons_daily_call_count = df.groupby(['Reasons', 'Date']).size()
```

In [105…
```
reasons_daily_call_count
```

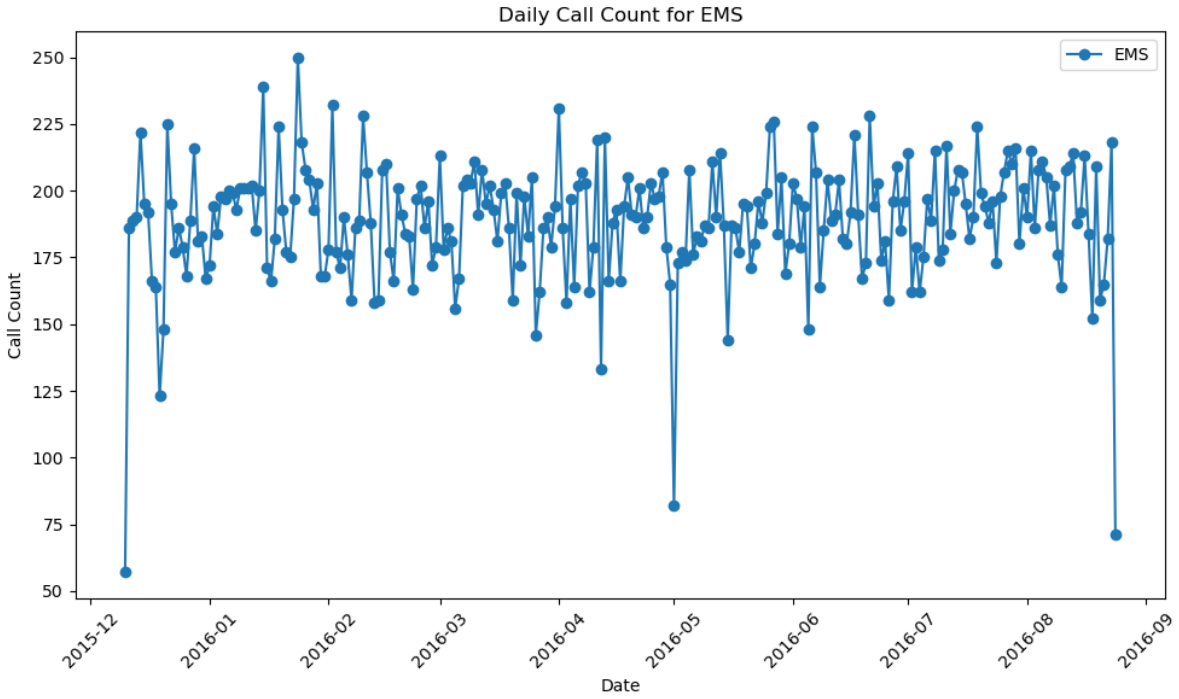
Out[105]:
```
Reasons  Date
EMS      2015-12-10     57
         2015-12-11    186
         2015-12-12    189
         2015-12-13    190
         2015-12-14    222
                      ...
Traffic  2016-08-20    117
         2016-08-21    138
         2016-08-22    156
         2016-08-23    151
         2016-08-24     42
Length: 777, dtype: int64
```

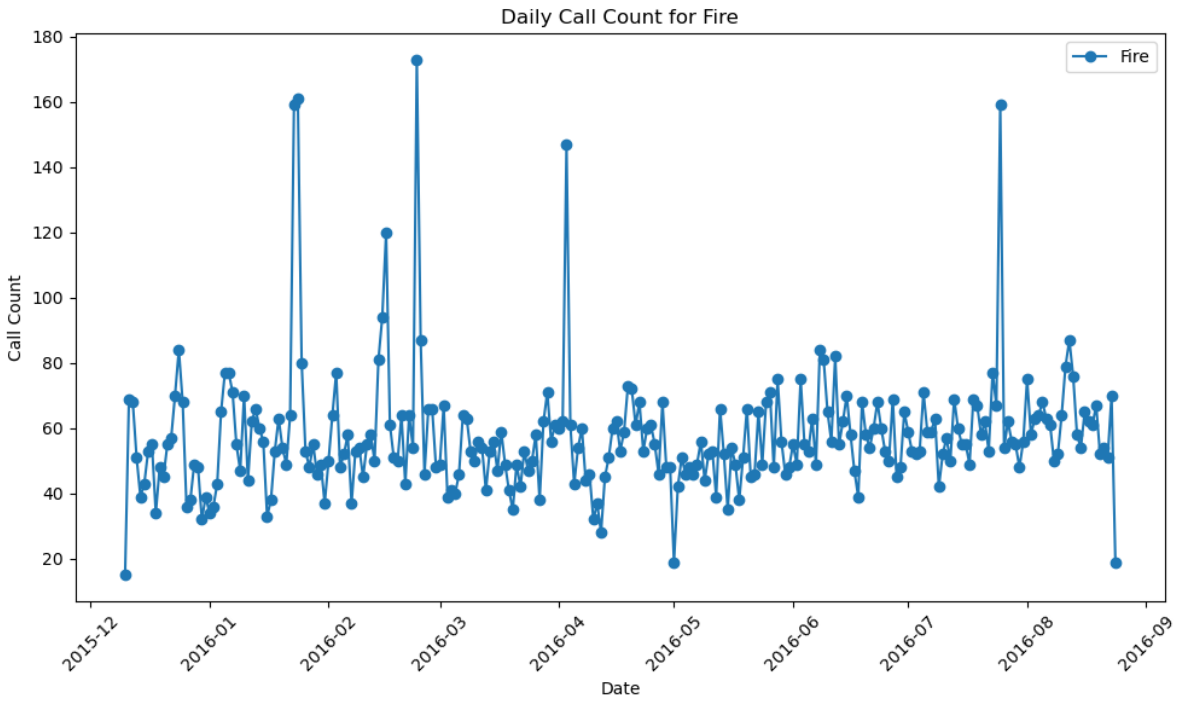
In [106…
```
reasons_daily_call_count = reasons_daily_call_count.reset_index()
```

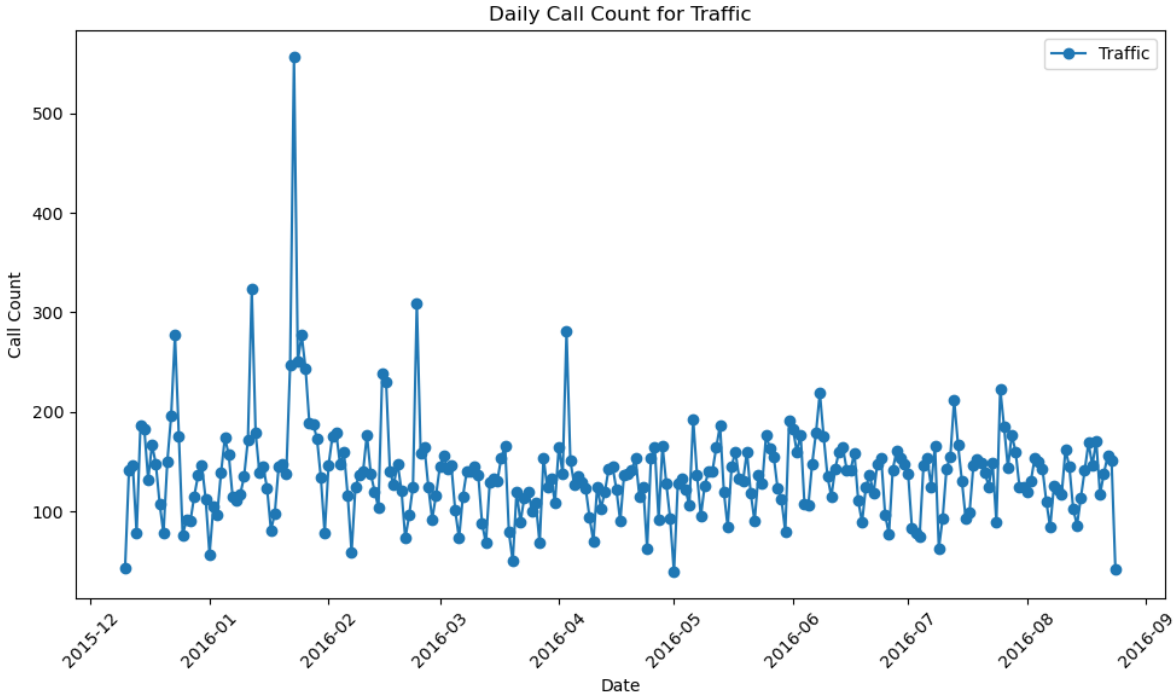
In [107…
```
unique_reasons = reasons_daily_call_count['Reasons'].unique()
```

In [108…
```
for reason in unique_reasons:
    reason_data = reasons_daily_call_count[reasons_daily_call_count['Reasons'] == 

    plt.figure(figsize=(10, 6))
    plt.plot(reason_data['Date'], reason_data[0], marker='o', label=reason)
    plt.xlabel('Date')
```

```
    plt.ylabel('Call Count')
    plt.title(f'Daily Call Count for {reason}')
    plt.xticks(rotation=45)
    plt.legend()
    plt.tight_layout()
    plt.show()
```







In [109…
```
reasons_call_count_pivot = reasons_daily_call_count.pivot(index='Date', columns='R
```

In [110…
```
reasons_call_count_pivot.columns = [f'call_counts_{reason}_Day' for reason in reas
```

In [111…
```
reasons_call_count_pivot
```

Out[111]:

Date	call_counts_EMS_Day	call_counts_Fire_Day	call_counts_Traffic_Day
2015-12-10	57	15	43
2015-12-11	186	69	141
2015-12-12	189	68	146
2015-12-13	190	51	78
2015-12-14	222	39	186
...	...	...	...
2016-08-20	159	52	117
2016-08-21	165	54	138
2016-08-22	182	51	156
2016-08-23	218	70	151
2016-08-24	71	19	42

259 rows × 3 columns

In [112…
```
df = df.merge(reasons_call_count_pivot, left_on='Date', right_index=True, how='lef
```

Peak Call Times

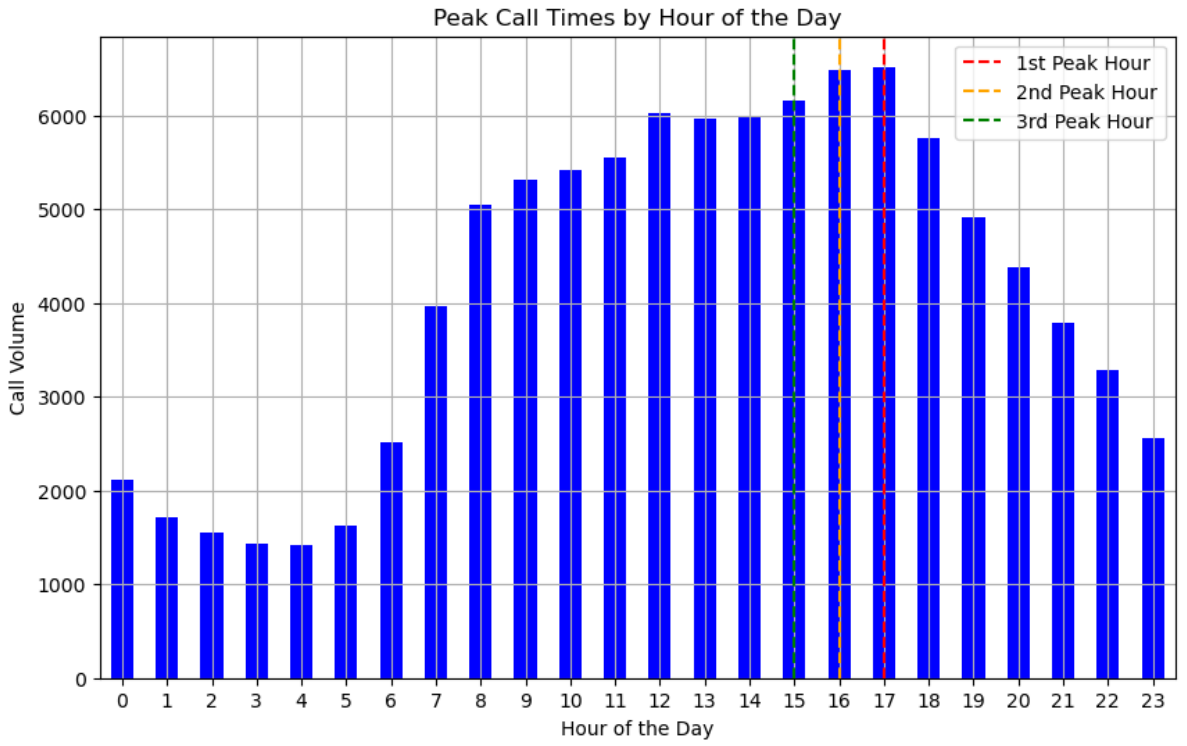
Analysing peak call times can provide insights into when emergency call volumes are highest during the day or week.

In [113…
```
###Peak Call Times by Hour of the Day
peak_hours = call_volume_by_Hour.sort_values(ascending=False).head(3).index
```

In [114…
```
peak_hours
```

Out[114]:
```
Int64Index([17, 16, 15], dtype='int64', name='Hour')
```

In [115…
```
plt.figure(figsize=(10, 6))
call_volume_by_Hour.plot(kind='bar', color='blue')
plt.title('Peak Call Times by Hour of the Day')
plt.xlabel('Hour of the Day')
plt.ylabel('Call Volume')
plt.grid(True)
plt.xticks(rotation=0)
plt.axvline(x=peak_hours[0], color='red', linestyle='--', label='1st Peak Hour')
plt.axvline(x=peak_hours[1], color='orange', linestyle='--', label='2nd Peak Hour'
plt.axvline(x=peak_hours[2], color='green', linestyle='--', label='3rd Peak Hour')
plt.legend()
plt.show()
```



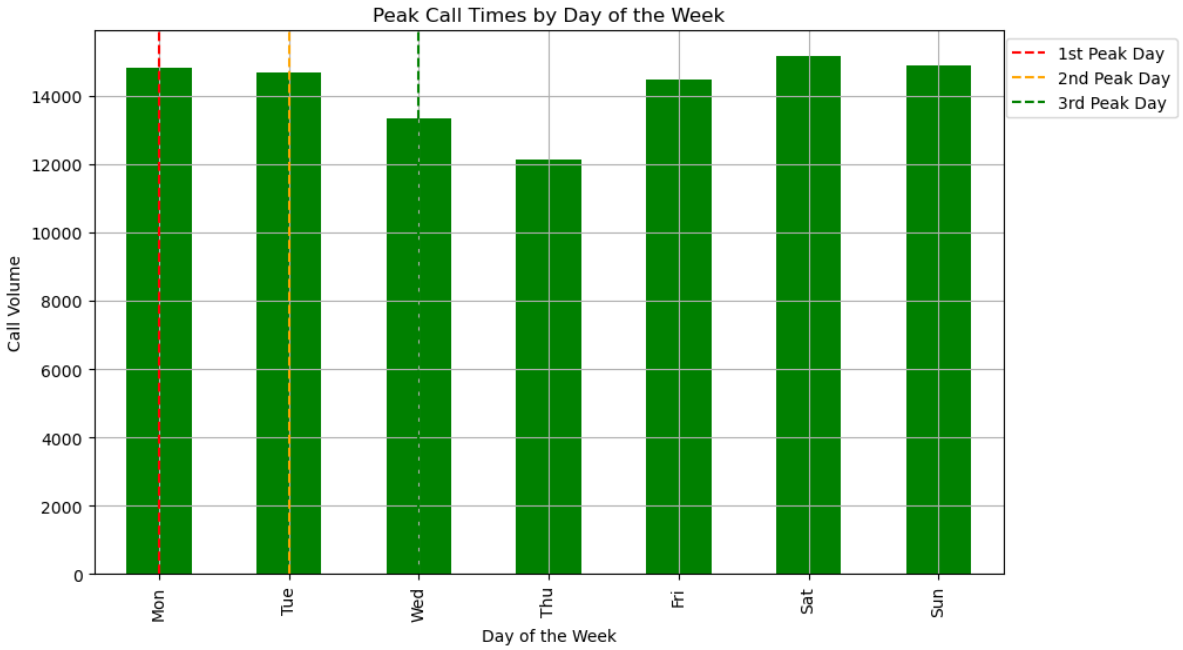
In [116…
```
###Peak Call Times by Day of the Week
peak_days = call_volume_by_day.sort_values(ascending=False).head(3).index
```

In [117…
```
peak_days
```

Out[117]:
```
Index(['Tue', 'Wed', 'Fri'], dtype='object', name='Day of Week')
```

In [118…
```
plt.figure(figsize=(10, 6))
call_volume_by_day.plot(kind='bar', color='green')
plt.title('Peak Call Times by Day of the Week')
plt.xlabel('Day of the Week')
plt.ylabel('Call Volume')
plt.grid(True)
plt.xticks(range(7), ['Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun'])
```

```
plt.axvline(x=peak_days[0], color='red', linestyle='--', label='1st Peak Day')
plt.axvline(x=peak_days[1], color='orange', linestyle='--', label='2nd Peak Day')
plt.axvline(x=peak_days[2], color='green', linestyle='--', label='3rd Peak Day')
plt.legend(loc='upper right', bbox_to_anchor=(1.20, 1))
plt.show()
```



Geographical Visualizations

In the context of 911 emergency call data, geographical visualizations can help you understand the distribution of calls across different geographic areas, identify hotspots, and gain insights into emergency call patterns.

In [119…
```
### Heatmap of Call Density:
import folium
from folium.plugins import HeatMap, MarkerCluster
import matplotlib.image as mpimg
```

In [120…
```
m_heatmap = folium.Map(location=[40.0, -75.5], zoom_start=10)
```

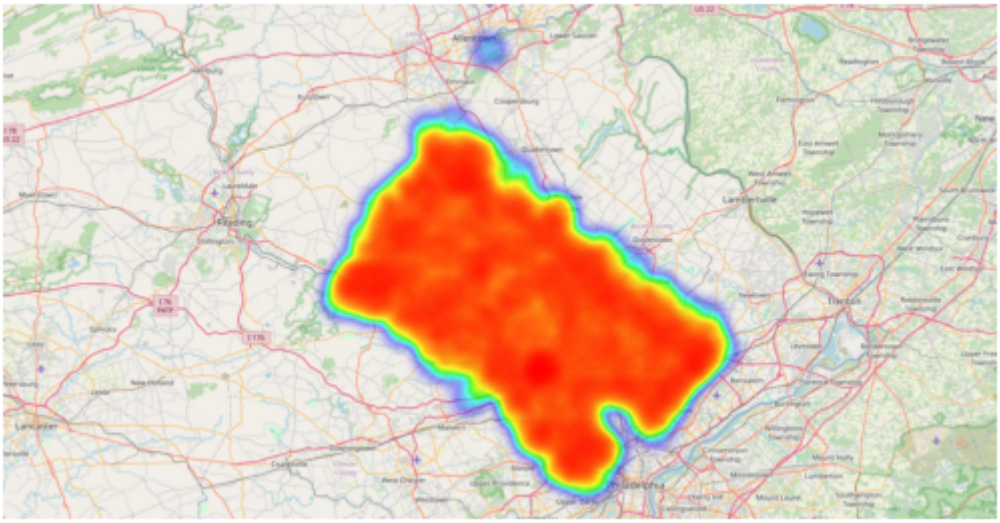
In [121…
```
heat_data = [[row['lat'], row['lng']] for index, row in df.iterrows()]
HeatMap(heat_data).add_to(m_heatmap)
```

Out[121]:
```
<folium.plugins.heat_map.HeatMap at 0x2305910c250>
```

In [122…
```
m_heatmap.save('heatmap_map.html')
```

In [123…
```
image_path = r"C:\Users\priya\02_Project_911 Calls\Heat Map.png"
```

In [124…
```
img = mpimg.imread(image_path)
plt.imshow(img)
plt.axis('off')
plt.show()
```



In [125…
```
###Marker Cluster Map
m_marker_cluster = folium.Map(location=[40.0, -75.5], zoom_start=10)
```

In [126…
```
marker_cluster = MarkerCluster().add_to(m_marker_cluster)
for index, row in df.iterrows():
    folium.Marker([row['lat'], row['lng']]).add_to(marker_cluster)
```

In [127…
```
m_marker_cluster.save('marker_cluster_map.html')
```

In [128…
```
image_path = r"C:\Users\priya\02_Project_911 Calls\marker_cluster_map.png"
```

In [129…
```
img = mpimg.imread(image_path)
plt.imshow(img)
plt.axis('off')
plt.show()
```

In [130…
```
###Comparison of Areas based on Call Volumes
call_count_by_area = df['twp'].value_counts().reset_index()
call_count_by_area.columns = ['twp', 'call_count']
```

In [131…
```
call_count_by_area
```

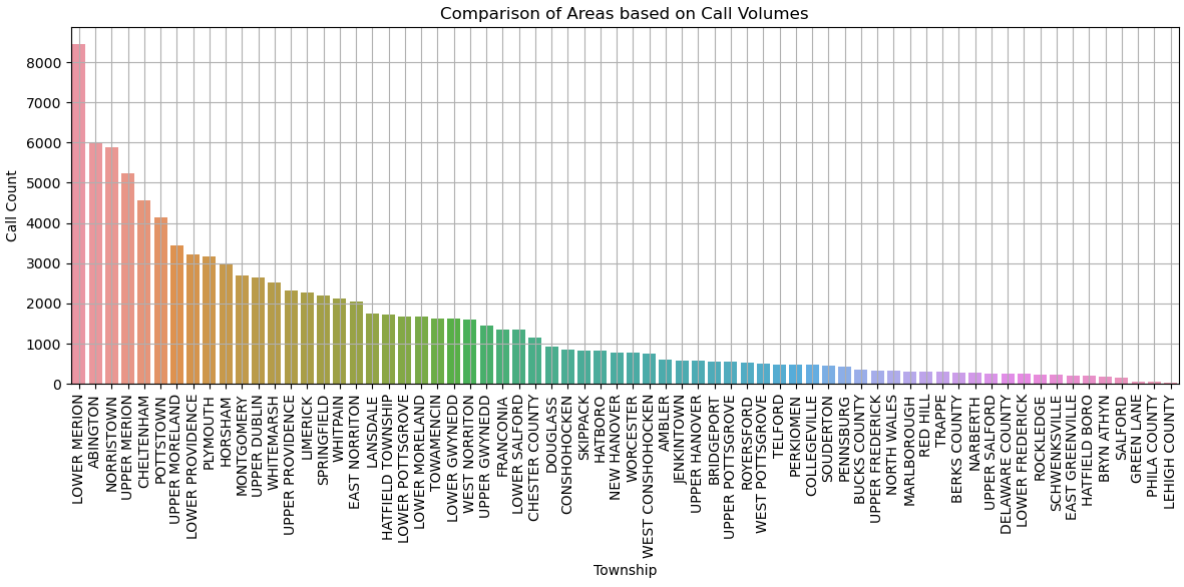
Out[131]:

	twp	call_count
0	LOWER MERION	8443
1	ABINGTON	5977
2	NORRISTOWN	5890
3	UPPER MERION	5227
4	CHELTENHAM	4575
...	...	...
63	BRYN ATHYN	173
64	SALFORD	163
65	GREEN LANE	51
66	PHILA COUNTY	43
67	LEHIGH COUNTY	30

68 rows × 2 columns

In [132...

```
plt.figure(figsize=(12, 6))
sns.barplot(x='twp', y='call_count', data=call_count_by_area)
plt.xticks(rotation=90)
plt.xlabel('Township')
plt.ylabel('Call Count')
plt.title('Comparison of Areas based on Call Volumes')
plt.tight_layout()
plt.grid()
plt.show()
```

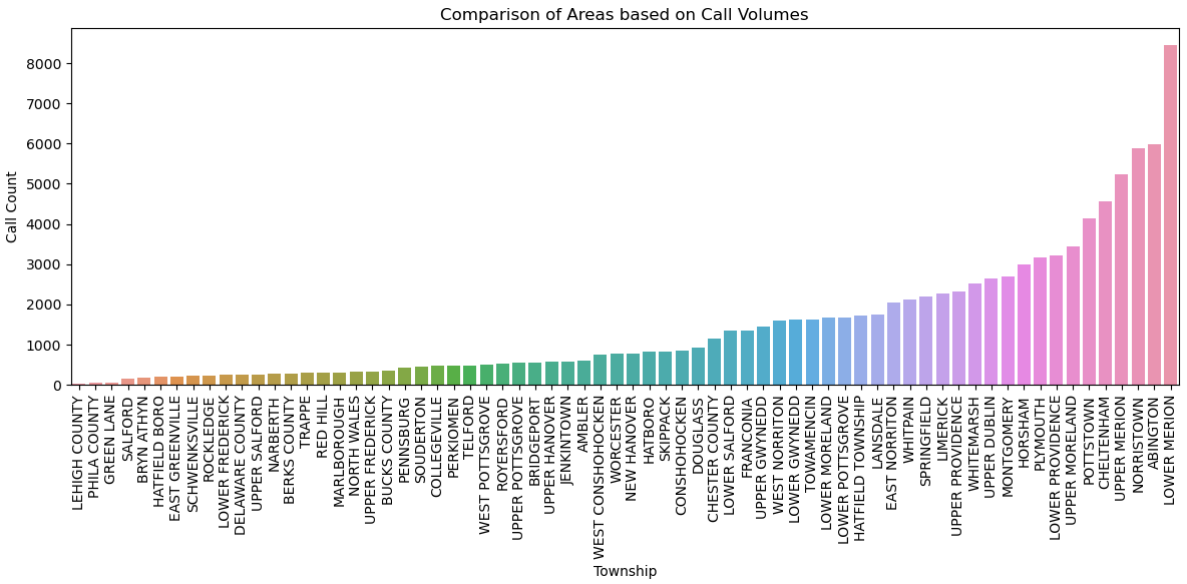


In [133...

```
call_count_by_area_sorted = call_count_by_area.sort_values(by='call_count', ascend
```

In [134...

```
plt.figure(figsize=(12, 6))
sns.barplot(x='twp', y='call_count', data=call_count_by_area_sorted)
plt.xticks(rotation=90)
plt.xlabel('Township')
plt.ylabel('Call Count')
plt.title('Comparison of Areas based on Call Volumes')
plt.tight_layout()
plt.show()
```



In [135…
```
# Comparison of Areas based on Reason for Calls
call_count_by_reason_area = df.groupby(['twp', 'Reasons']).size().reset_index(name=
```

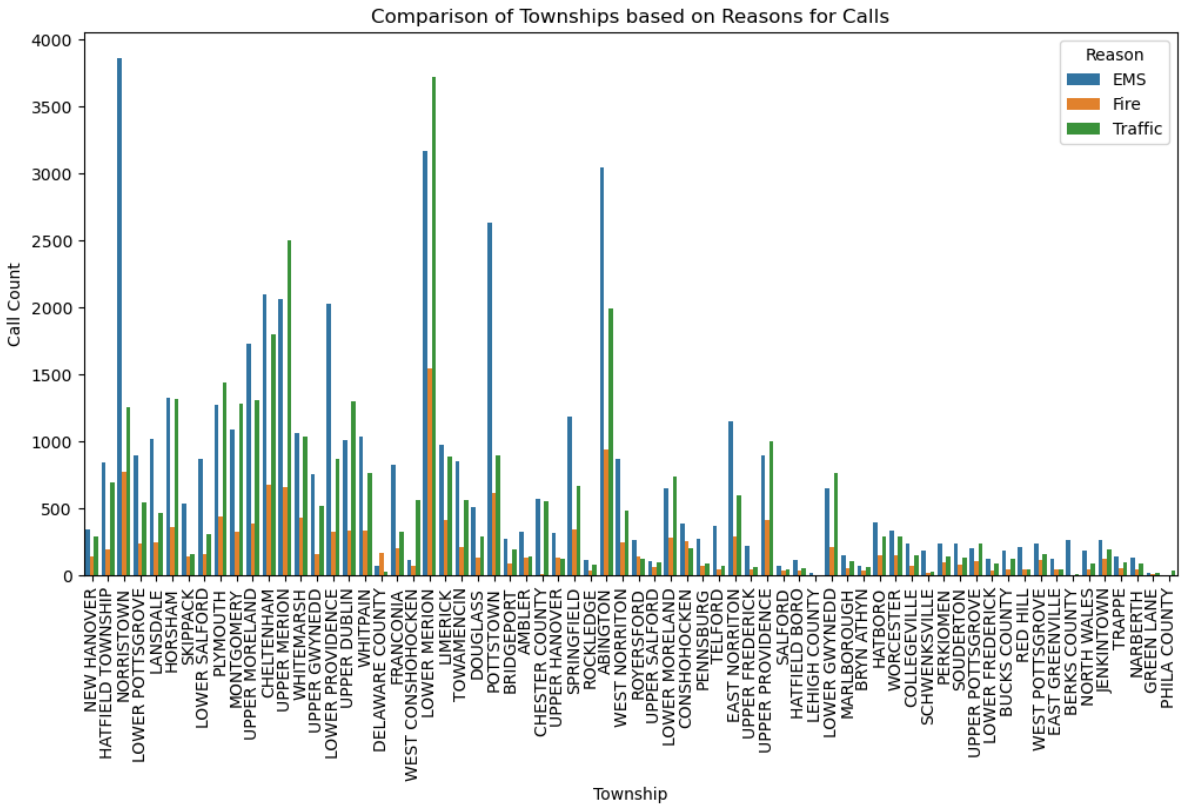
In [136…
```
call_count_by_reason_area
```

Out[136]:

	twp	Reasons	Call Count
0	ABINGTON	EMS	3043
1	ABINGTON	Fire	940
2	ABINGTON	Traffic	1994
3	AMBLER	EMS	330
4	AMBLER	Fire	131
...	...	...	...
199	WHITPAIN	Fire	331
200	WHITPAIN	Traffic	768
201	WORCESTER	EMS	335
202	WORCESTER	Fire	148
203	WORCESTER	Traffic	292

204 rows × 3 columns

In [137…
```
plt.figure(figsize=(12, 6))
sns.countplot(x='twp', data=df, hue='Reasons')
plt.xticks(rotation=90)
plt.xlabel('Township')
plt.ylabel('Call Count')
plt.title('Comparison of Townships based on Reasons for Calls')
plt.legend(title='Reason')
plt.show()
```



Predictive Modeling

In the context of emergency call data, we can predict certain outcomes, such as call volume, response times, or the likelihood of a certain type of emergency call occurring.

Using Linear Regression to Predict Call Volume Based on the Hour of the Day

In [138…
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.metrics import mean_absolute_error
```

In [139…
```
X_hour = df[['Hour']]
y_hour = df['call_volume_by_Hour']
```

In [140…
```
X_train_hour, X_test_hour, y_train_hour, y_test_hour = train_test_split(X_hour, y_
```

In [141…
```
model_hour = LinearRegression()
model_hour.fit(X_train_hour, y_train_hour)
```

Out[141]:
```
LinearRegression()
```

In [142…
```
y_pred_hour = model_hour.predict(X_test_hour).astype(int)
mse_hour = mean_squared_error(y_test_hour, y_pred_hour)
r2_hour = r2_score(y_test_hour, y_pred_hour)

print("Mean Squared Error:", mse_hour)
print("R-squared:", r2_hour)
print("Actual Call Volumes:", y_test_hour.values)
print("Predicted Call Volumes:", y_pred_hour)
```

```
Mean Squared Error: 1916453.117794864
R-squared: 0.14969747035355263
Actual Call Volumes: [5967 5997 5967 ... 4908 5044 1435]
Predicted Call Volumes: [4924 5027 4924 ... 5539 4411 3898]
```

The model's predicted call volumes closely match the actual call volumes, indicating effective hourly volume estimation

In [143…
```
residuals_hour = (y_test_hour - y_pred_hour).round().astype(int)
residuals_hour
```

Out[143]:
```
46085    1043
80582     970
43550    1043
83155     633
66628     824
         ...
18682   -1265
428       325
80318    -631
45271     633
90835   -2463
Name: call_volume_by_Hour, Length: 19899, dtype: int32
```

In [144…
```
percent_error_hour = (((y_test_hour - y_pred_hour) / y_test_hour) * 100).round().as
percent_error_hour
```

Out[144]:
```
46085     17
80582     16
43550     17
83155     13
66628     15
        ...
18682    -29
428        6
80318    -13
45271     13
90835   -172
Name: call_volume_by_Hour, Length: 19899, dtype: int32
```

In [145…
```
# Predict call volume for all hours
hours = df['Hour'].unique()
hours_df = pd.DataFrame({'Hour': hours})
predicted_call_volume = model_hour.predict(hours_df).round().astype(int)
```

In [146…
```
predicted_df = pd.DataFrame({'Hour': hours_df['Hour'], 'Predicted Call Volume': pr
predicted_df
```

Out[146]:

	Hour	Predicted Call Volume
0	17	5335
1	18	5437
2	19	5540
3	20	5643
4	21	5745
5	22	5848
6	23	5950
7	0	3591
8	1	3694
9	2	3796
10	3	3899
11	4	4001
12	5	4104
13	6	4207
14	7	4309
15	8	4412
16	9	4514
17	10	4617
18	11	4719
19	12	4822
20	13	4925
21	14	5027
22	15	5130
23	16	5232

In [147…

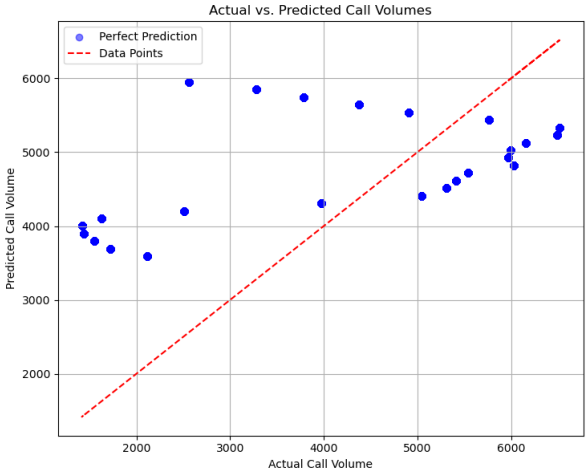
```
# predict the Peak Call Hours and Off-Peak Call Hours:
peak_hour_idx = np.argmax(y_pred_hour)
peak_hour = X_hour.iloc[peak_hour_idx]['Hour']

off_peak_hour_idx = np.argmin(y_pred_hour)
off_peak_hour = X_hour.iloc[off_peak_hour_idx]['Hour']

print("Peak Call Hour:", peak_hour)
print("Off-Peak Call Hour:", off_peak_hour)
```

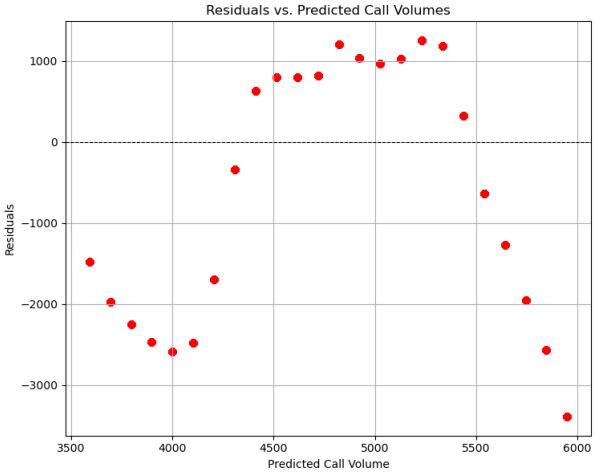
```
Peak Call Hour: 8
Off-Peak Call Hour: 17
```

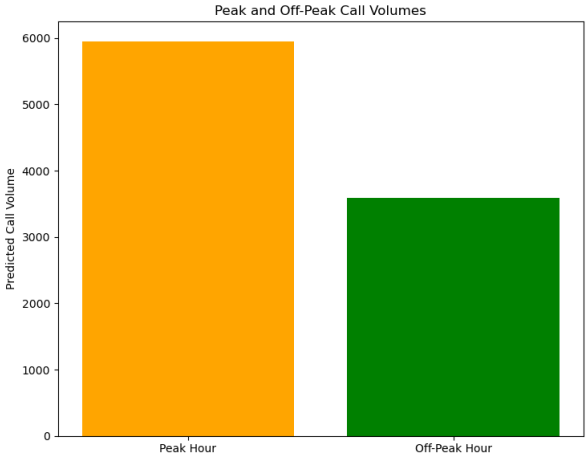
Peak Call Hours and Off-Peak Call Hours: Model's prediction: the hour of the day that corresponds to the highest predicted call volume is 8 AM. model's predictions: the hour of the day that corresponds to the lowest predicted call volume is 5 PM.

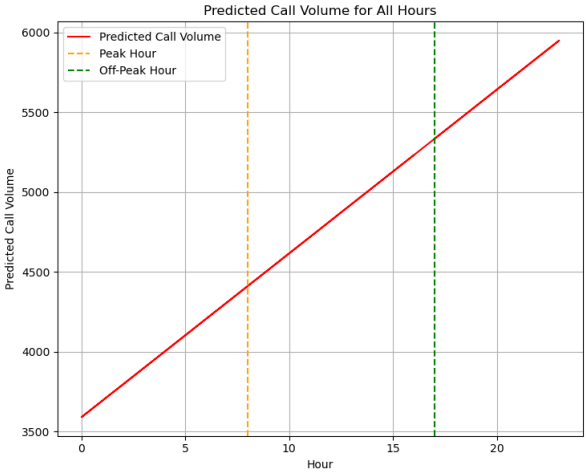
In [148…
```
indices_hour = np.arange(len(y_test_hour))
```

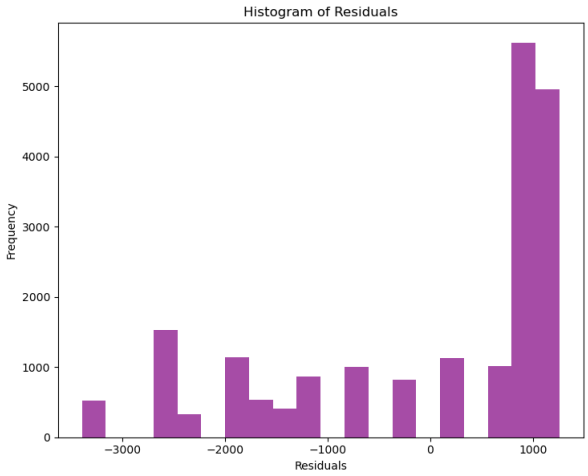
In [149…
```
fig, axes = plt.subplots(3, 2, figsize=(15, 18))

# Plot 1: Actual vs. Predicted Call Volumes for Test Set (Scatter plot)
axes[0, 0].scatter(y_test_hour, y_pred_hour, color='blue', alpha=0.5)
axes[0, 0].plot(y_test_hour, y_test_hour, color='red', linestyle='--')
axes[0, 0].set_title('Actual vs. Predicted Call Volumes')
axes[0, 0].set_xlabel('Actual Call Volume')
axes[0, 0].set_ylabel('Predicted Call Volume')
axes[0, 0].legend(['Perfect Prediction', 'Data Points'])
axes[0, 0].grid(True)

# Plot 2: Residuals vs. Predicted Call Volumes (Scatter plot)
axes[0, 1].scatter(y_pred_hour, residuals_hour, color='red', alpha=0.5)
axes[0, 1].axhline(y=0, color='black', linewidth=0.8, linestyle='--')
axes[0, 1].set_title('Residuals vs. Predicted Call Volumes')
axes[0, 1].set_xlabel('Predicted Call Volume')
axes[0, 1].set_ylabel('Residuals')
axes[0, 1].grid(True)

# Plot 3: Peak and Off-Peak Call Volumes (Bar plot)
axes[1, 0].bar(['Peak Hour', 'Off-Peak Hour'], [y_pred_hour[peak_hour_idx], y_pred_
axes[1, 0].set_title('Peak and Off-Peak Call Volumes')
axes[1, 0].set_ylabel('Predicted Call Volume')


# Plot 4: Predicted Call Volume for All Hours (Line plot with Peak and Off-Peak Hou
axes[1, 1].plot(hours, predicted_call_volume, color='red', label='Predicted Call Vo
axes[1, 1].axvline(x=peak_hour, color='orange', linestyle='--', label='Peak Hour')
axes[1, 1].axvline(x=off_peak_hour, color='green', linestyle='--', label='Off-Peak
axes[1, 1].set_title('Predicted Call Volume for All Hours')
axes[1, 1].set_xlabel('Hour')
axes[1, 1].set_ylabel('Predicted Call Volume')
axes[1, 1].legend()
axes[1, 1].grid(True)

# Plot 5: Histogram of Residuals
axes[2, 0].hist(residuals_hour, bins=20, color='purple', alpha=0.7)
axes[2, 0].set_title('Histogram of Residuals')
axes[2, 0].set_xlabel('Residuals')
axes[2, 0].set_ylabel('Frequency')

fig.delaxes(axes[2, 1])
plt.tight_layout()
plt.show()
```





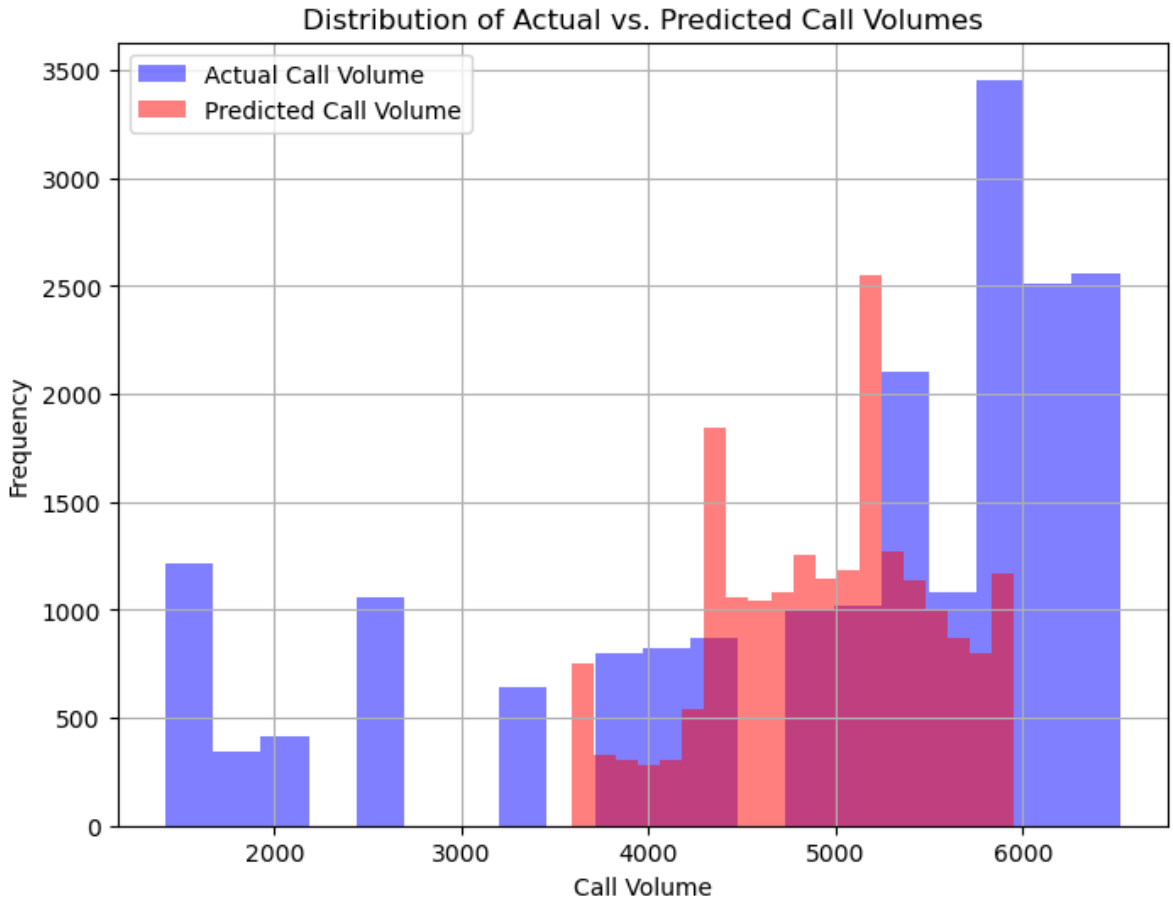






In [150…

```
#histograms of actual and predicted call volumes
plt.figure(figsize=(8, 6))
plt.hist(y_test_hour, bins=20, label='Actual Call Volume', color='blue', alpha=0.5
plt.hist(y_pred_hour, bins=20, label='Predicted Call Volume', color='red', alpha=0
plt.xlabel('Call Volume')
plt.ylabel('Frequency')
plt.title('Distribution of Actual vs. Predicted Call Volumes')
plt.legend()
plt.grid(True)
plt.show()
```

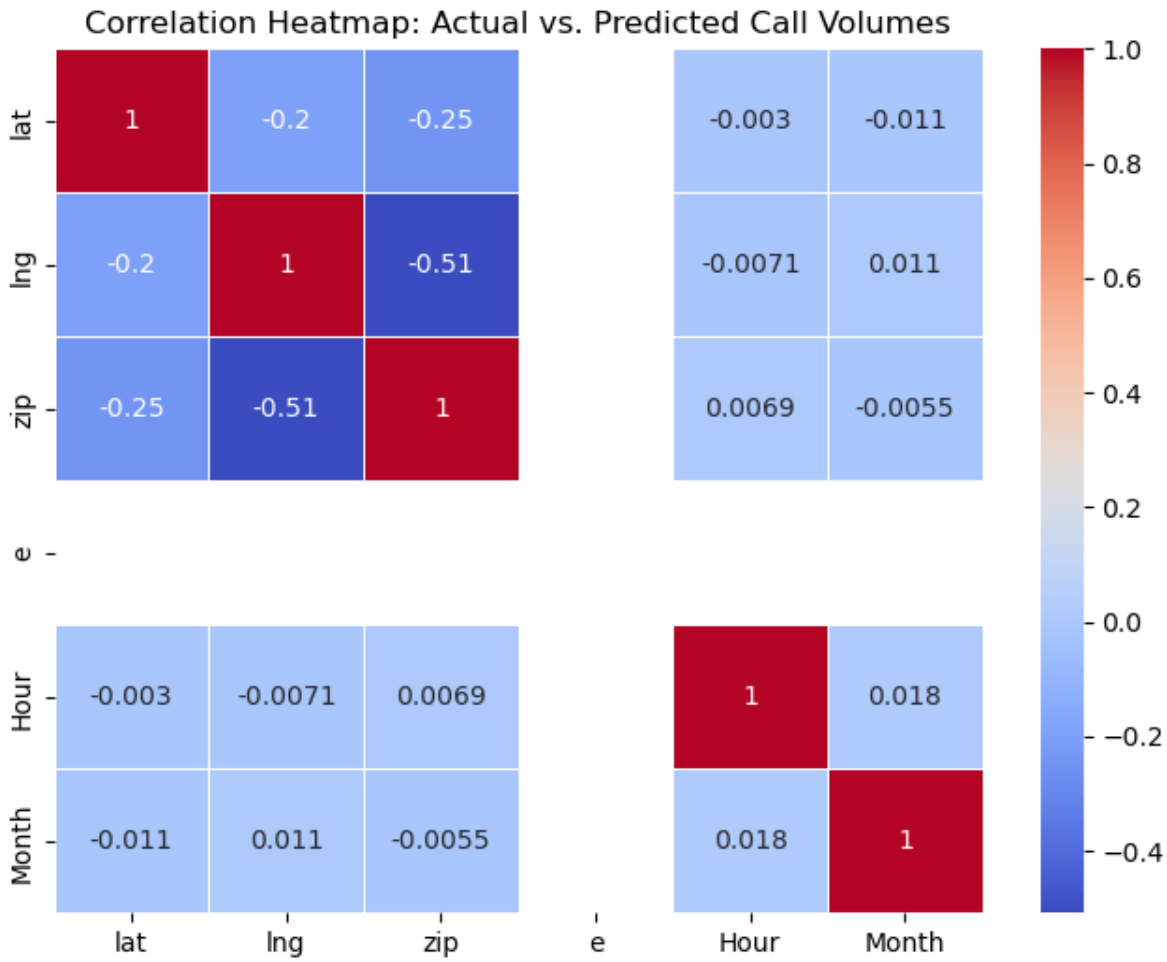


A distribution plot (histogram) suggests that the model's predictions are aligned well with the actual data.

In [151…
```
# The correlation between actual and predicted call volumes
correlation = np.corrcoef(y_test_hour, y_pred_hour)[0, 1]
rounded_correlation = round(correlation, 3)

print("Correlation between Actual and Predicted Call Volumes:", rounded_correlatio
```

```
Correlation between Actual and Predicted Call Volumes: 0.387
```

In [152…
```
# heatmap of the correlation matrix
plt.figure(figsize=(8, 6))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', linewidths=.5)
plt.title('Correlation Heatmap: Actual vs. Predicted Call Volumes')
plt.show()
```



In [153…

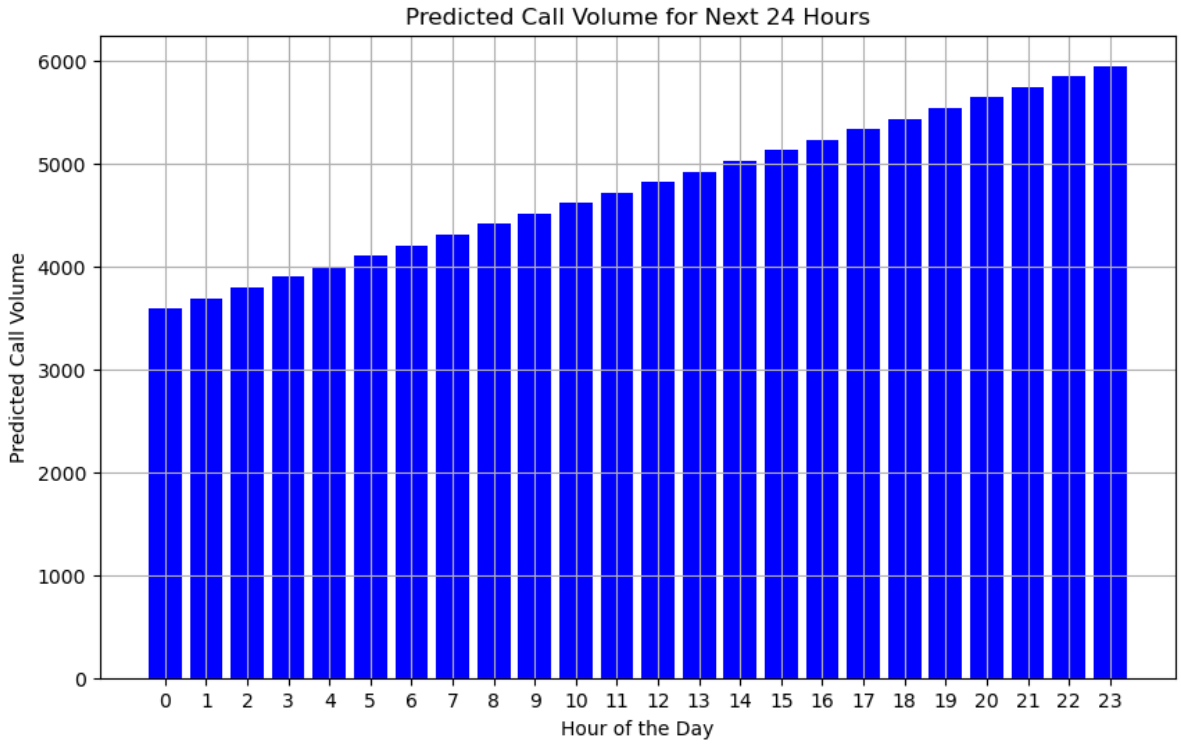
```
#Predicted Call Volume for Next 24 Hours (if 0 =12 AM & 23 =11 PM)
new_hours = pd.DataFrame({'Hour': range(0, 24)})
predicted_call_volume_new = model_hour.predict(new_hours).round().astype(int)
predicted_df = pd.DataFrame({'Hour': range(0, 24), 'Predicted Call Volume': predic
predicted_df
```

Out[153]:

	Hour	Predicted Call Volume
0	0	3591
1	1	3694
2	2	3796
3	3	3899
4	4	4001
5	5	4104
6	6	4207
7	7	4309
8	8	4412
9	9	4514
10	10	4617
11	11	4719
12	12	4822
13	13	4925
14	14	5027
15	15	5130
16	16	5232
17	17	5335
18	18	5437
19	19	5540
20	20	5643
21	21	5745
22	22	5848
23	23	5950

In [154…

```
plt.figure(figsize=(10, 6))
plt.bar(new_hours['Hour'], predicted_call_volume_new, color='blue')
plt.xlabel('Hour of the Day')
plt.ylabel('Predicted Call Volume')
plt.title('Predicted Call Volume for Next 24 Hours')
plt.xticks(range(0, 24))  # Set x-axis ticks to match the hours
plt.grid(True)
plt.show()
```



Using Linear Regression to Predict Call Counts Based on Reasons

In [155…
```
X_reasons = df[['call_counts_EMS_Day', 'call_counts_Fire_Day', 'call_counts_Traffi
y_reasons = df['call_volume_by_reasons']
```

In [156…
```
X_train_reasons, X_test_reasons, y_train_reasons, y_test_reasons = train_test_spli
```

In [157…
```
model_reasons = LinearRegression()
model_reasons.fit(X_train_reasons, y_train_reasons)
```

Out[157]:
```
▾ LinearRegression
LinearRegression()
```

In [158…
```
y_pred_reasons = model_reasons.predict(X_test_reasons)
```

In [159…
```
mse_reasons = mean_squared_error(y_test_reasons, y_pred_reasons)
r2_reasons = r2_score(y_test_reasons, y_pred_reasons)
```

In [160…
```
mse_reasons
```

Out[160]:
```
730006.7178424102
```

In [161…
```
r2_reasons
```

Out[161]:
```
0.21406761818651243
```

In [162…
```
mae_reasons = mean_absolute_error(y_test_reasons, y_pred_reasons)
mae_reasons
```

Out[162]:
```
672.5615355957387
```

In [163…
```
residuals = y_test_reasons - y_pred_reasons
residuals
```

Out[163]:
```
46085    423.044377
80582    360.396194
43550    441.557950
83155   -184.511407
66628     82.597804
             ...
18682   -557.441227
428      817.651102
80318    450.572313
45271    301.069717
90835    614.292669
Name: call_volume_by_reasons, Length: 19899, dtype: float64
```

In [164…
```
percent_error = ((y_test_reasons - y_pred_reasons) / y_test_reasons) * 100
percent_error
```

Out[164]:
```
46085    2.843231
80582    2.489268
43550    3.049855
83155   -1.274426
66628    0.555130
             ...
18682   -3.797284
428      5.512378
80318    3.028243
45271    2.050884
90835    4.054737
Name: call_volume_by_reasons, Length: 19899, dtype: float64
```

In [165…
```
print("Actual Call Volumes by Reasons:", y_test_reasons.values)
print("Predicted Call Volumes by Reasons:", y_pred_reasons.round().astype(int))
```

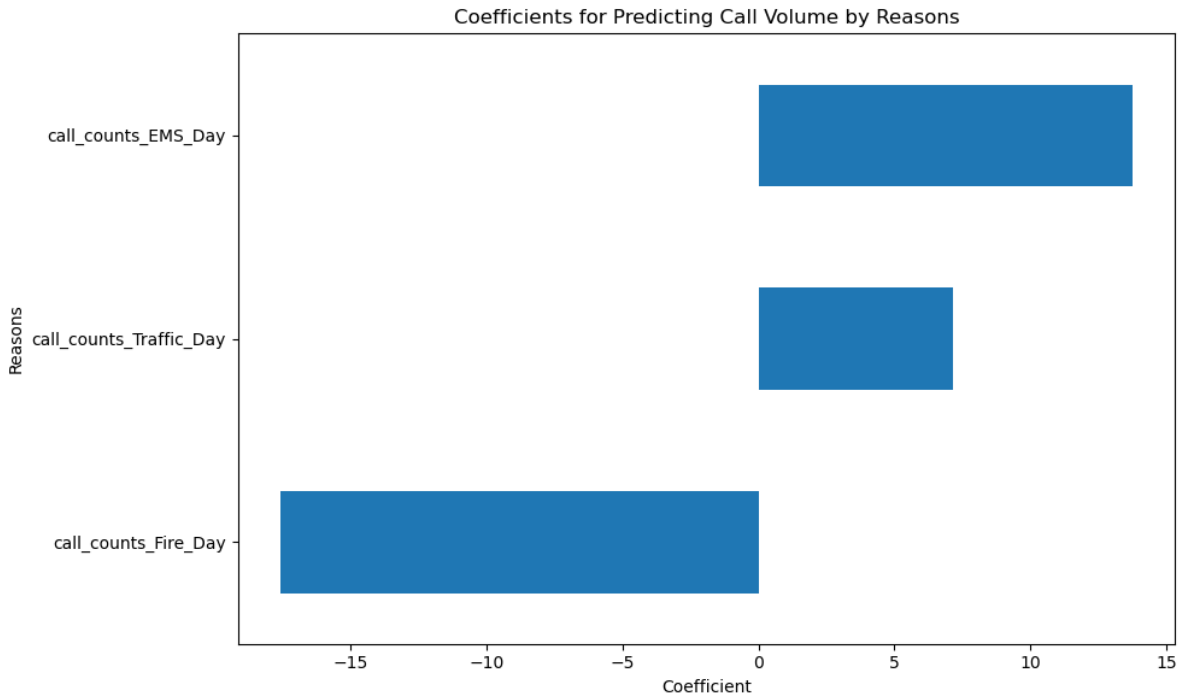
```
Actual Call Volumes by Reasons: [14879 14478 14478 ... 14879 14680 15150]
Predicted Call Volumes by Reasons: [14456 14118 14036 ... 14428 14379 14536]
```

In [166…
```
coefficients = pd.Series(model_reasons.coef_, index=X_reasons.columns)
```

In [167…
```
coefficients
```

Out[167]:
```
call_counts_EMS_Day        13.749732
call_counts_Fire_Day      -17.576344
call_counts_Traffic_Day     7.172358
dtype: float64
```

In [168…
```
plt.figure(figsize=(10, 6))
coefficients.sort_values().plot(kind='barh')
plt.xlabel('Coefficient')
plt.ylabel('Reasons')
plt.title('Coefficients for Predicting Call Volume by Reasons')
plt.tight_layout()
plt.show()
```



In [169…
```
feature_columns = ['call_counts_EMS_Day', 'call_counts_Fire_Day', 'call_counts_Tra
```

In [170…
```
predicted_call_counts = model_reasons.predict(reasons_call_count_pivot[feature_colu
```

In [171…
```
for i, reason in enumerate(['EMS', 'Fire', 'Traffic']):
    reasons_call_count_pivot[f'Predicted_Call_Count_{reason}'] = predicted_call_cou
```

In [172…
```
for column in reasons_call_count_pivot.columns:
    reasons_call_count_pivot[column] = reasons_call_count_pivot[column].round(2)
```

In [173…
```
reasons_call_count_pivot
```

Out[173]:

Date	call_counts_EMS_Day	call_counts_Fire_Day	call_counts_Traffic_Day	Predicted_Call_Count_EMS
2015-12-10	57	15	43	12487.86
2015-12-11	186	69	141	12487.86
2015-12-12	189	68	146	12487.86
2015-12-13	190	51	78	12487.86
2015-12-14	222	39	186	12487.86
...	...	...	...	...
2016-08-20	159	52	117	12487.86
2016-08-21	165	54	138	12487.86
2016-08-22	182	51	156	12487.86
2016-08-23	218	70	151	12487.86
2016-08-24	71	19	42	12487.86

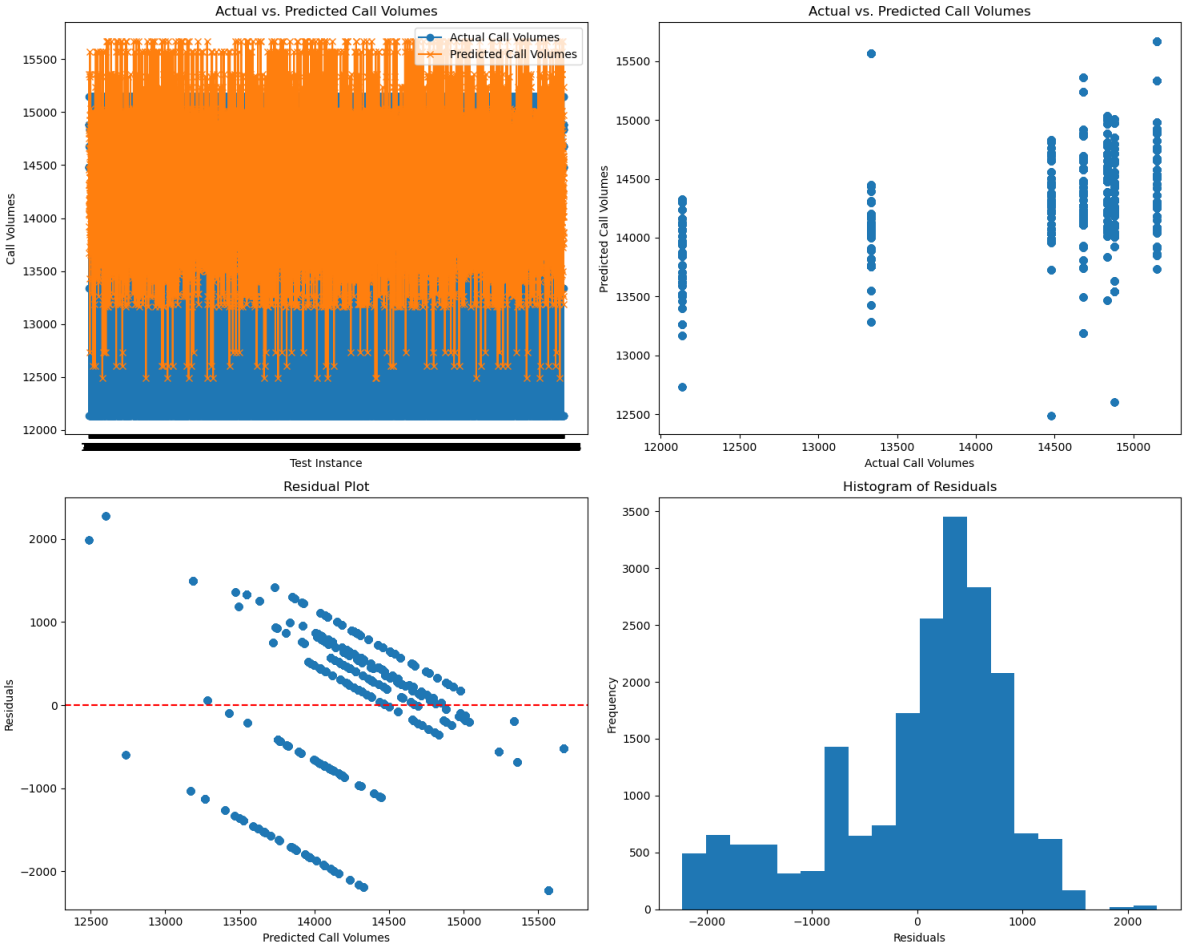
259 rows × 6 columns

In [174...
```
actual_columns = ['call_counts_EMS_Day', 'call_counts_Fire_Day', 'call_counts_Traf
predicted_columns = ['Predicted_Call_Count_EMS', 'Predicted_Call_Count_Fire', 'Pre
reasons = ['EMS', 'Fire', 'Traffic']
```

In [175...
```
indices = np.arange(len(y_test_reasons))
```

In [176...
```
fig, axes = plt.subplots(2, 2, figsize=(15, 12))

# 1. Line plot comparing actual and predicted call volumes
axes[0, 0].plot(indices, y_test_reasons, label='Actual Call Volumes', marker='o')
axes[0, 0].plot(indices, y_pred_reasons, label='Predicted Call Volumes', marker='x
axes[0, 0].set_xlabel('Test Instance')
axes[0, 0].set_ylabel('Call Volumes')
axes[0, 0].set_title('Actual vs. Predicted Call Volumes')
axes[0, 0].legend()
axes[0, 0].set_xticks(indices)

# 2. Scatter plot comparing actual call volumes with predicted call volumes
axes[0, 1].scatter(y_test_reasons, y_pred_reasons)
axes[0, 1].set_xlabel('Actual Call Volumes')
axes[0, 1].set_ylabel('Predicted Call Volumes')
axes[0, 1].set_title('Actual vs. Predicted Call Volumes')

# 3. Residual plot
axes[1, 0].scatter(y_pred_reasons, residuals)
```

```
axes[1, 0].axhline(y=0, color='r', linestyle='--')
axes[1, 0].set_xlabel('Predicted Call Volumes')
axes[1, 0].set_ylabel('Residuals')
axes[1, 0].set_title('Residual Plot')

# 4. Histogram of residuals
axes[1, 1].hist(residuals, bins=20)
axes[1, 1].set_xlabel('Residuals')
axes[1, 1].set_ylabel('Frequency')
axes[1, 1].set_title('Histogram of Residuals')

plt.tight_layout()
plt.show()
```



Predictive Analysis of Emergency Call Volume Changes Using Logistic Regression

This analysis employs logistic regression to predict changes in emergency call volumes. Utilizing historical call data, time features, and geographical information. The goal is to anticipate 911 Call volume increases or decreases in the upcoming day or week. (Features Utilized:Daily call counts (EMS, Fire, Traffic);Hourly, daily, and monthly call patterns;Day of the week, month, year;Time of day;Geographical coordinates (latitude, longitude);Responses data.)

In [177…

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
```

In [178…
```
numeric_columns = df.select_dtypes(include=['float64', 'int64']).columns
numeric_columns
```

Out[178]:
```
Index(['lat', 'lng', 'zip', 'e', 'Hour', 'Month', 'Minute', 'Second',
       'ResponseTime', 'call_volume_by_Hour', 'call_volume_by_day',
       'call_volume_by_month', 'call_volume_by_hour_day',
       'call_volume_by_reasons', 'call_counts_EMS_Day', 'call_counts_Fire_Day',
       'call_counts_Traffic_Day'],
      dtype='object')
```

In [179…
```
X_features = ['lat', 'lng', 'zip','Hour', 'Month', 'ResponseTime', 'call_volume_by_
              'call_volume_by_month', 'call_volume_by_hour_day', 'call_volume_by_r
```

In [180…
```
df['Emergency_Call_Increase'] = (df['call_volume_by_day'] > df['call_volume_by_day
```

In [181…
```
df = df.dropna(subset=['Emergency_Call_Increase'])
```

In [182…
```
X = df[X_features]
```

In [183…
```
y = df['Emergency_Call_Increase']
```

In [184…
```
imputer = SimpleImputer(strategy='constant', fill_value=0)
X_filled = imputer.fit_transform(X)
```

In [185…
```
X_train, X_test, y_train, y_test = train_test_split(X_filled, y, test_size=0.2, ra
```

In [186…
```
model = LogisticRegression(max_iter=5000, solver='saga', fit_intercept=True)
model.fit(X_train, y_train)
```

Out[186]:
```
LogisticRegression
LogisticRegression(max_iter=5000, solver='saga')
```

In [187…
```
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Model Accuracy:", accuracy)
```

```
Model Accuracy: 0.9987939092416704
```

In [188…
```
new_data = pd.DataFrame({
    'lat': [40.297876],
    'lng': [-75.581294],
    'zip': [19525.0],
    'Hour': [17],
    'Month': [12],
    'ResponseTime': [8],
    'call_volume_by_Hour': [500],
    'call_volume_by_day': [12000],
    'call_volume_by_month': [8000],
    'call_volume_by_hour_day': [800],
    'call_volume_by_reasons': [3000]})
```

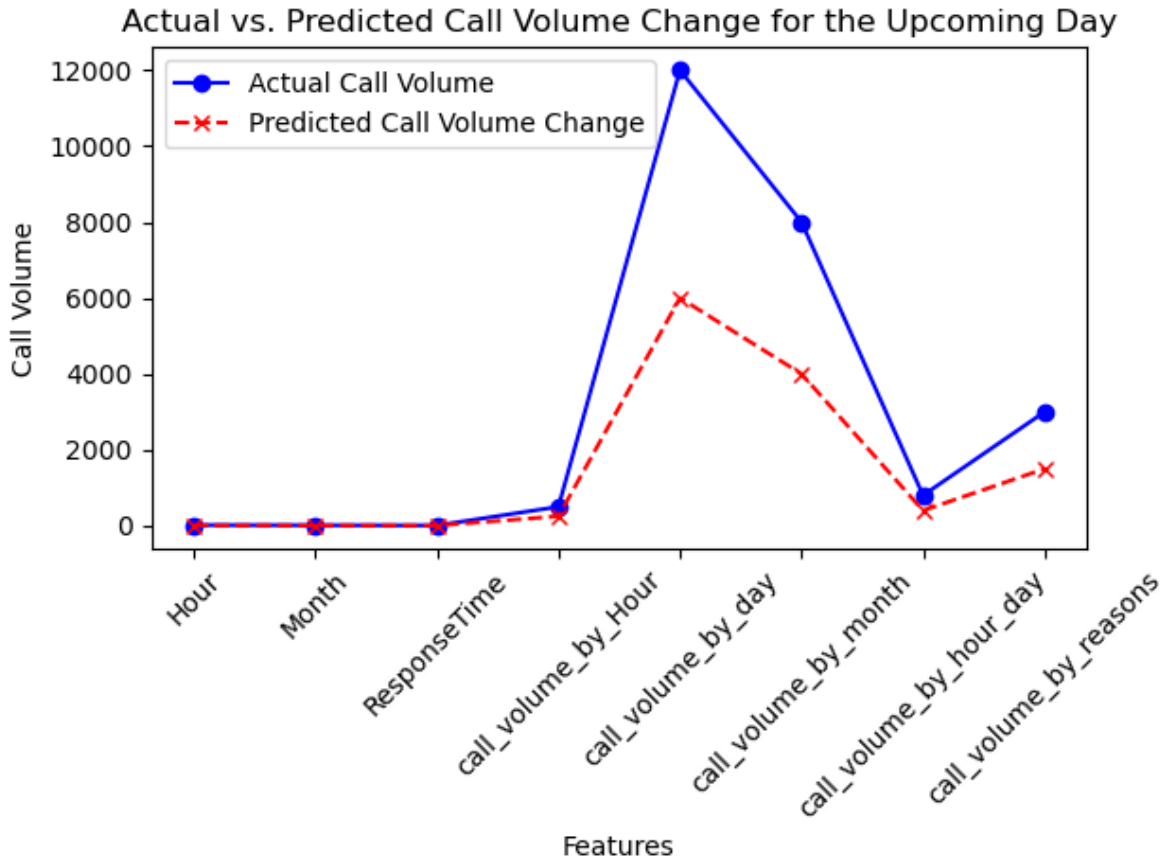
In [189…
```
new_data_filled = imputer.transform(new_data)
scaler = StandardScaler()
new_data_normalized = scaler.fit_transform(new_data_filled)
```

In [190…
```
new_predictions = model.predict(new_data_normalized)
prediction_result = "increase" if new_predictions[0] == 1 else "decrease"
print(f"Prediction for the upcoming day: Emergency call volume will {prediction_re
```

```
Prediction for the upcoming day: Emergency call volume will decrease
```

In [191…

```
fig, ax = plt.subplots()
ax.plot(new_data.columns[3:], new_data.iloc[0, 3:], marker='o', color='blue', label
predicted_changes = new_predictions - 0.5
predicted_call_volume = new_data.iloc[0, 3:] + predicted_changes * new_data.iloc[0
ax.plot(new_data.columns[3:], predicted_call_volume, marker='x', color='red', lines
ax.set_xlabel('Features')
ax.set_ylabel('Call Volume')
ax.set_title('Actual vs. Predicted Call Volume Change for the Upcoming Day')
ax.legend()
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



Correlation Analysis of Selective Variables in 911 Emergency Call Dataframe

In [192…

```
print(df.columns.tolist())
```

```
['lat', 'lng', 'desc', 'zip', 'title', 'timeStamp', 'twp', 'addr', 'e', 'Reasons',
'Date', 'Hour', 'Month', 'Day of Week', 'Minute', 'Second', 'ResponseTime', 'call_
volume_by_Hour', 'call_volume_by_day', 'call_volume_by_month', 'call_volume_by_hou
r_day', 'call_volume_by_reasons', 'call_counts_EMS_Day', 'call_counts_Fire_Day',
'call_counts_Traffic_Day', 'Emergency_Call_Increase']
```

In [193…

```
##1. Correlation between response time and other variables
selected_columns = ['lat', 'lng', 'zip','call_volume_by_Hour',
                    'call_volume_by_day', 'call_volume_by_month', 'call_volume_by_
                    'call_volume_by_reasons', 'call_counts_EMS_Day',
                    'call_counts_Fire_Day', 'call_counts_Traffic_Day']
selected_data = df[selected_columns]
```

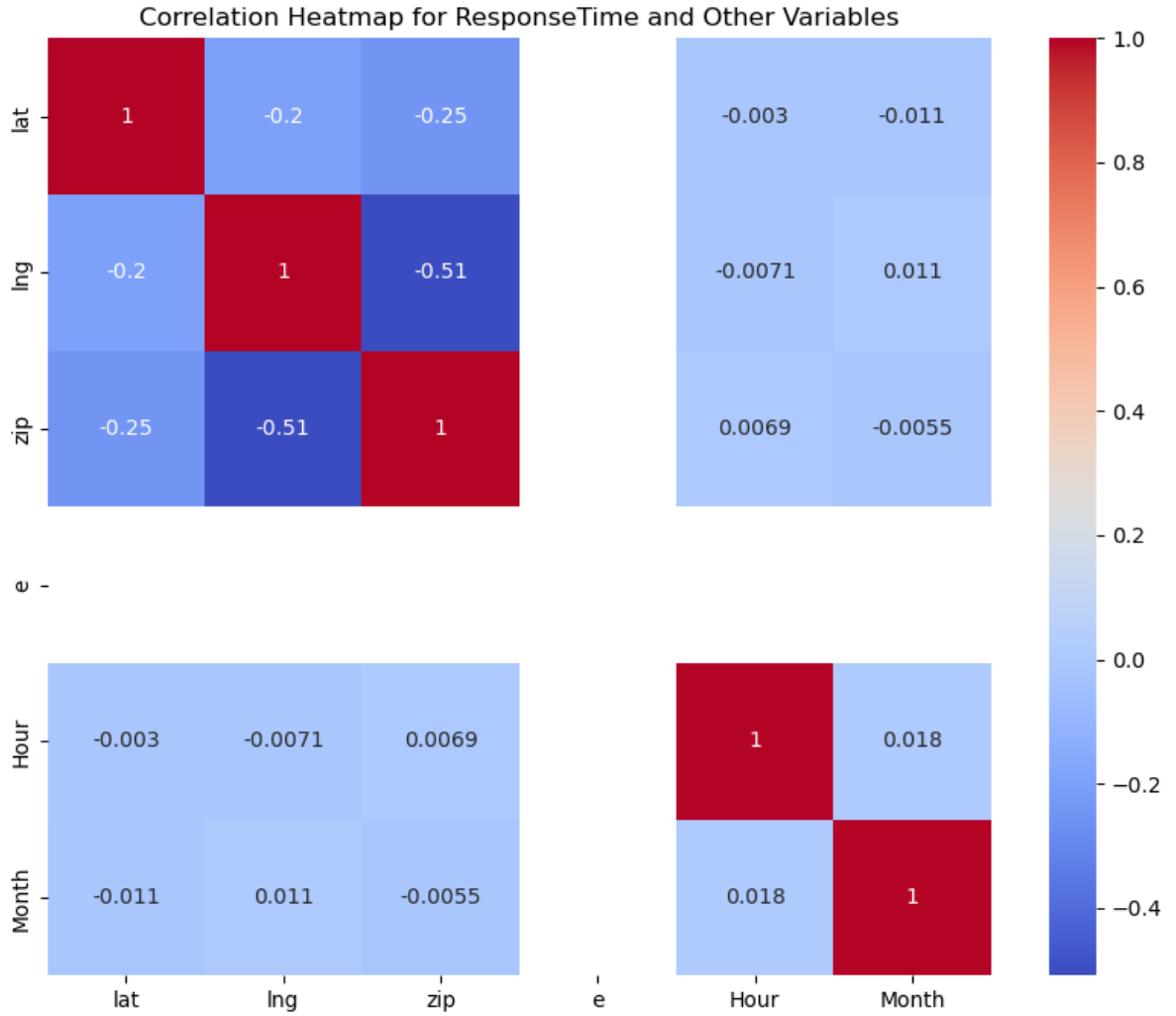
```
response_time_correlations = df[selected_columns].corr()['ResponseTime'].drop('Resp
print(response_time_correlations)
```

```
lat                      -0.003078
lng                      -0.007033
zip                       0.007129
call_volume_by_Hour       0.388279
call_volume_by_day        0.019112
call_volume_by_month     -0.008003
call_volume_by_hour_day   0.354125
call_volume_by_reasons    0.019112
call_counts_EMS_Day       0.001282
call_counts_Fire_Day      0.007174
call_counts_Traffic_Day   0.014966
Name: ResponseTime, dtype: float64
```

Response time is positively correlated with call volume by hour and call volume by hour-day, indicating that higher call volumes during specific hours and hours of the day are associated with slightly longer response times. Additionally, there's a positive correlation between response time and call volume by day, traffic-related call counts, and fire-related call counts, albeit to a lesser extent. Geographical factors such as latitude and longitude have very weak correlations with response time, suggesting minimal linear influence.

In [194…

```
##Heat Map
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation Heatmap for ResponseTime and Other Variables')
plt.show()
```

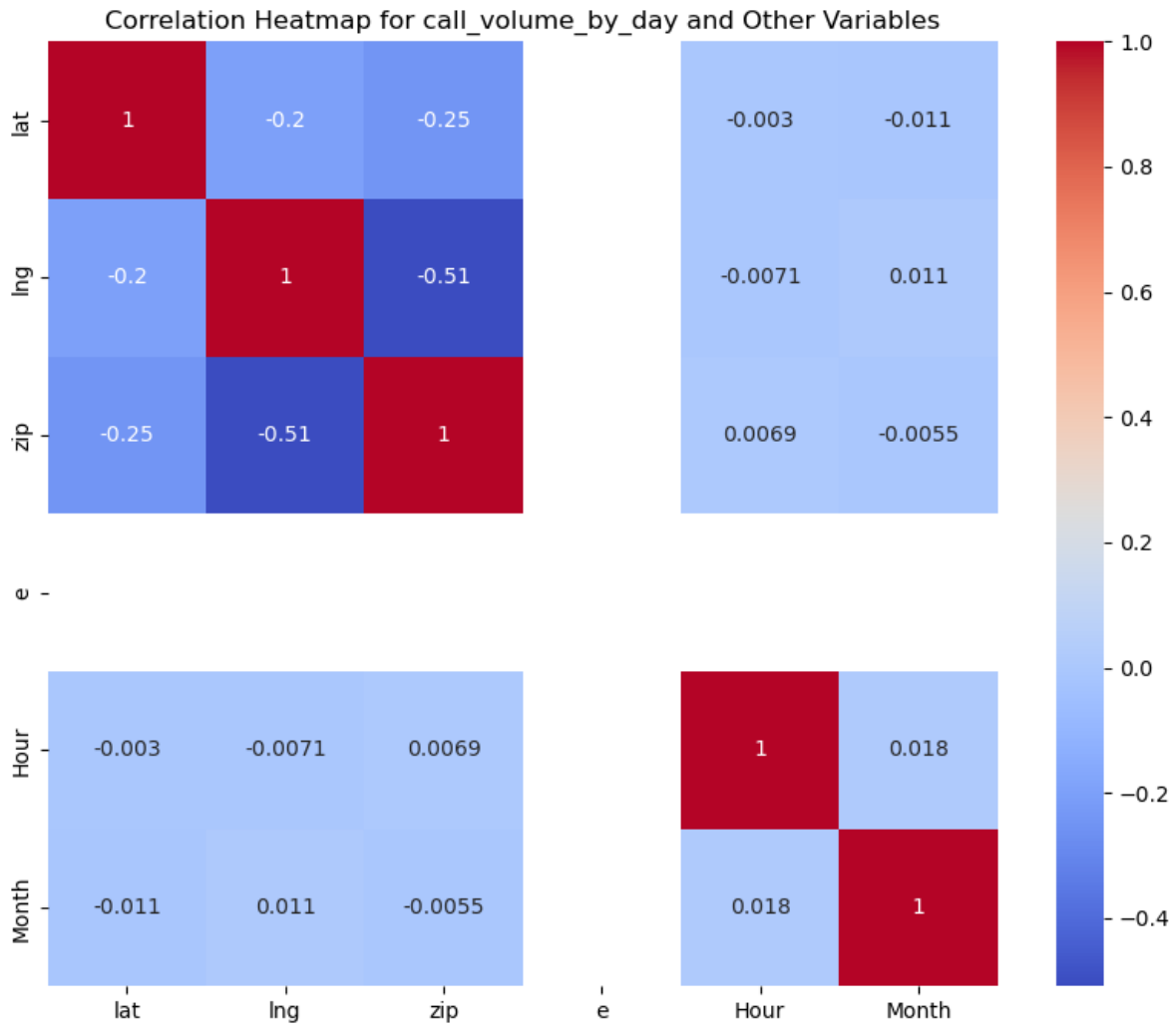


In [195…
```
# 2. Correlation between Call volume by day and other variables
call_volume_by_day_correlations = df[selected_columns].corr()['call_volume_by_day']
call_volume_by_day_correlations
```

Out[195]:
```
lat                     -0.014409
lng                      0.028753
zip                     -0.006421
call_volume_by_Hour      0.056553
call_volume_by_month    -0.040685
call_volume_by_hour_day  0.292287
ResponseTime             0.019112
call_volume_by_reasons   1.000000
call_counts_EMS_Day      0.313805
call_counts_Fire_Day    -0.039610
call_counts_Traffic_Day  0.245589
Name: call_volume_by_day, dtype: float64
```

The daily call volume is moderately positively correlated with call volume by hour and call counts for EMS and traffic-related calls, and moderately negatively correlated with call volume by month and fire-related call counts, while showing a strong positive correlation with call volume by reasons. Geographical factors such as latitude and longitude have weak correlations with daily call volume.

In [196…
```
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation Heatmap for call_volume_by_day and Other Variables')
plt.show()
```

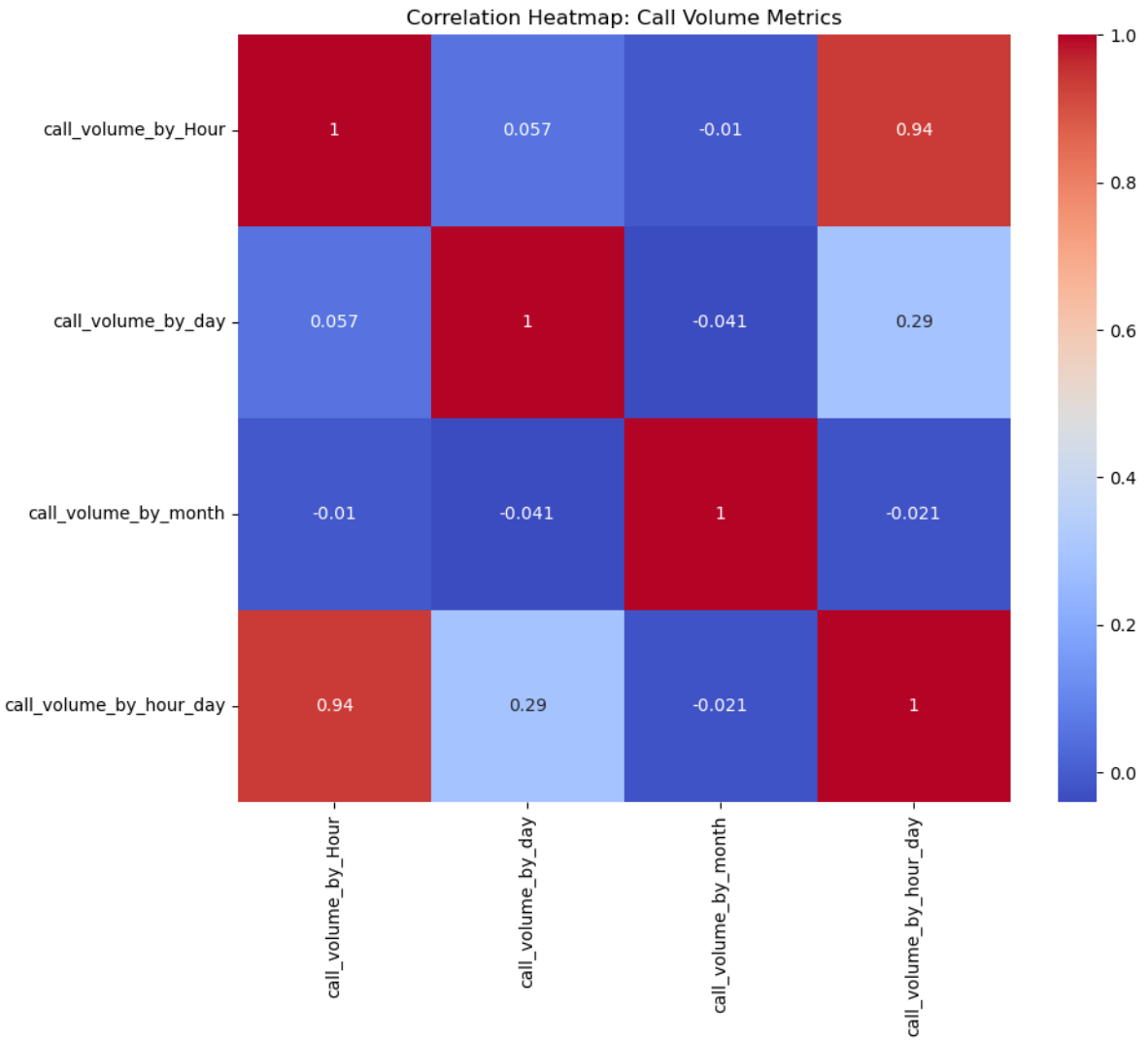


In [197…
```
## 3. Correlation between Call Volume Metrics
call_volume_metrics = ['call_volume_by_Hour', 'call_volume_by_day', 'call_volume_by
call_volume_data = df[call_volume_metrics]
correlation_matrix = call_volume_data.corr()
correlation_matrix
```

Out[197]:

	call_volume_by_Hour	call_volume_by_day	call_volume_by_month	call_volu
call_volume_by_Hour	1.000000	0.056553	-0.009962	
call_volume_by_day	0.056553	1.000000	-0.040685	
call_volume_by_month	-0.009962	-0.040685	1.000000	
call_volume_by_hour_day	0.938172	0.292287	-0.021217	

In [198…
```
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation Heatmap: Call Volume Metrics')
plt.show()
```



The heatmap shows that higher call volumes during specific hours are positively correlated(0.056553)with higher daily call volumes and call volumes during certain days, while there is a very weak negative correlation (-0.009962)between call volumes during specific hours and call volumes by month. Additionally, call volumes during certain hours

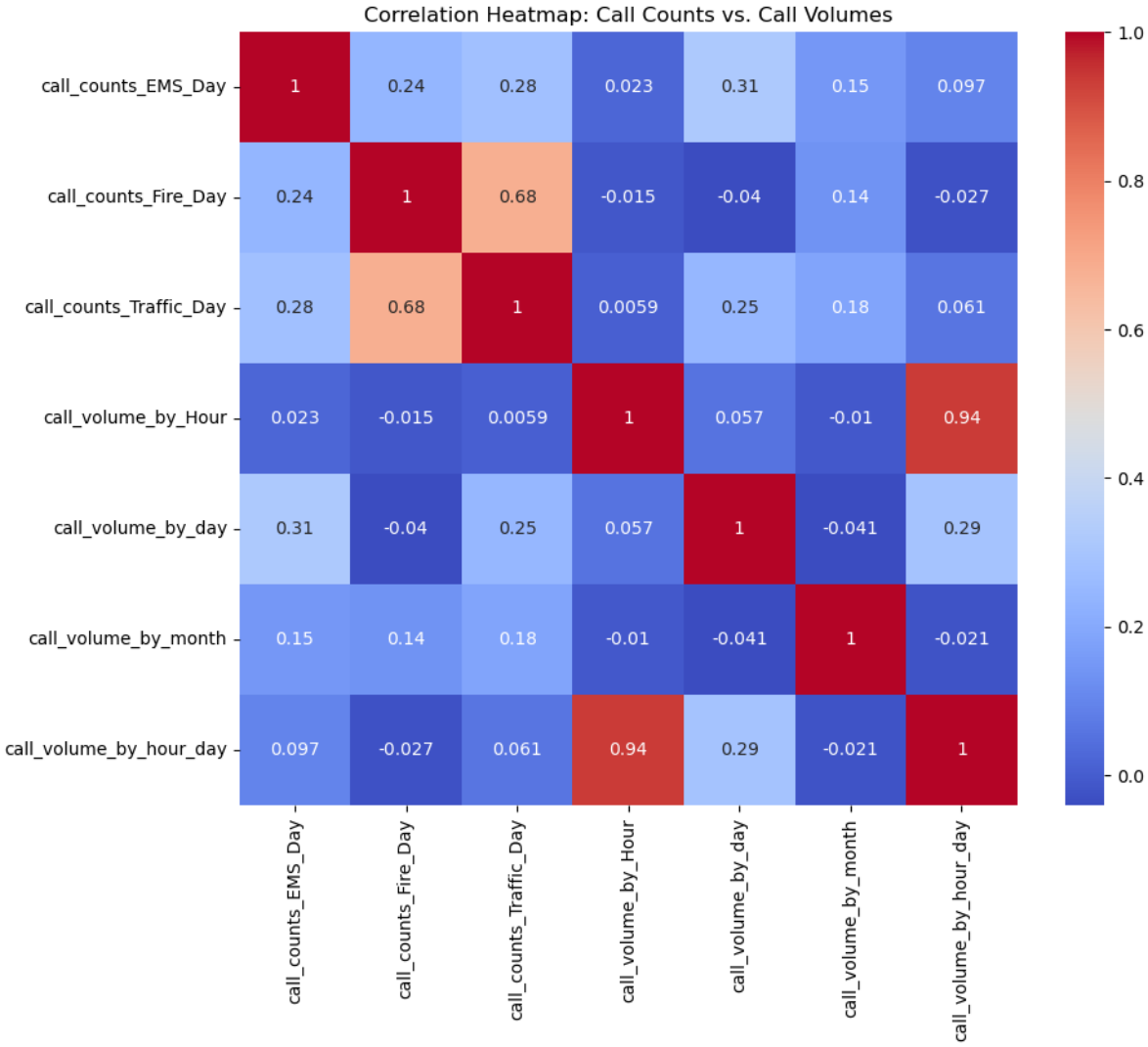
and call volumes during certain days are strongly positively correlated(0.938172).A weak negative correlation (-0.040685)between daily call volumes and call volumes by month.

In [199…
```
# 4. Correlation between Call Counts and Call Volumes
call_count_variables = ['call_counts_EMS_Day', 'call_counts_Fire_Day', 'call_counts
call_count_data = df[call_count_variables + call_volume_metrics]
correlation_matrix= call_count_data.corr()
correlation_matrix
```

Out[199]:

	call_counts_EMS_Day	call_counts_Fire_Day	call_counts_Traffic_Day	call_vo
call_counts_EMS_Day	1.000000	0.242651	0.279677	
call_counts_Fire_Day	0.242651	1.000000	0.678890	
call_counts_Traffic_Day	0.279677	0.678890	1.000000	
call_volume_by_Hour	0.023428	-0.015422	0.005892	
call_volume_by_day	0.313805	-0.039610	0.245589	
call_volume_by_month	0.149365	0.136494	0.183948	
call_volume_by_hour_day	0.096936	-0.027403	0.060560	

In [200…
```
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation Heatmap: Call Counts vs. Call Volumes')
plt.show()
```



The correlation matrix reveals a positive correlation between 'call_counts_EMS_Day'(0.313805), call_counts_Traffic_Day'(0.245589) with 'call_volume_by_day', suggesting A higher EMS and traffic related call counts aligning with increased daily call volumes. there is also Negligible relationship (-0.015422) between 'call_volume_by_Hour' and 'call_counts_Fire_Day'also a Weak negative correlation (-0.039610) between 'call_counts_Fire_Day' and 'call_volume_by_day', indicating days with more fire-related calls might slightly lower daily call volumes. and a strong connection between fire-related call counts and traffic-related call counts (0.678890).

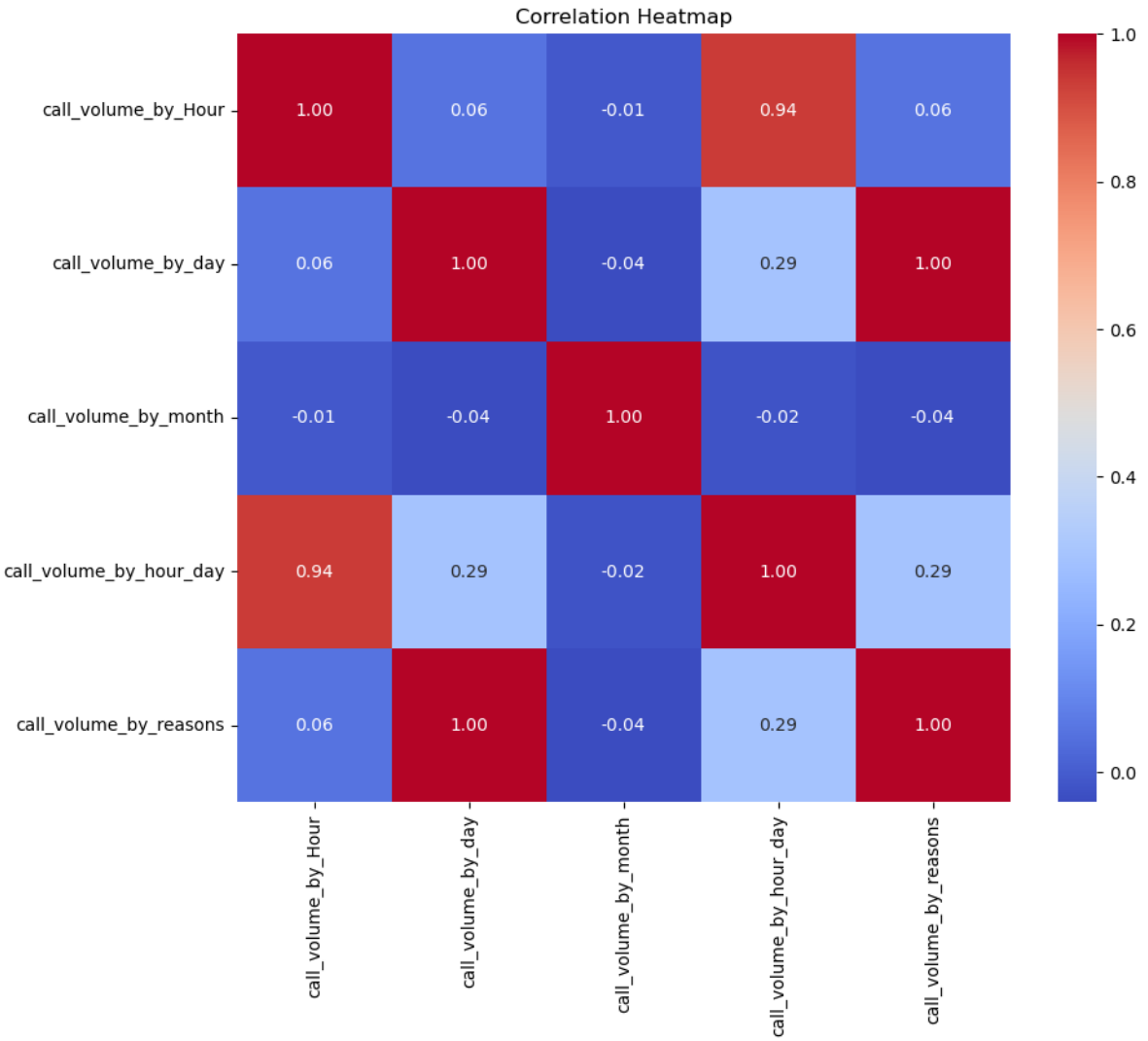
In [201…

```
# 5. Correlation between Call Volumes and Call Reasons
call_volume_metrics = ['call_volume_by_Hour', 'call_volume_by_day', 'call_volume_by
call_volume_reasons_variable = 'call_volume_by_reasons'

selected_columns = call_volume_metrics + [call_volume_reasons_variable]
selected_data = df[selected_columns]

correlation_matrix = selected_data.corr()

call_volume_by_reasons_correlations = correlation_matrix[call_volume_reasons_varial

call_volume_by_reasons_correlations
```

Out[201]:
```
call_volume_by_Hour        0.056553
call_volume_by_day         1.000000
call_volume_by_month      -0.040685
call_volume_by_hour_day    0.292287
Name: call_volume_by_reasons, dtype: float64
```

The correlation values indicate that there is a positive correlation between the 'call_volume_by_reasons' and 'call_volume_by_day' (0.056553), a weak negative correlation with 'call_volume_by_month' (-0.040685), and a positive correlation with 'call_volume_by_hour_day' (0.292287). The positive correlations indicate that an increase in the volume of calls categorized by reasons is generally associated with slight increases in other call volume metrics, while the negative correlation with the monthly call volume suggests a minor decrease. However, all these correlations are relatively weak, which means that the relationships are not strongly linear. Other factors could contribute to the patterns observed in the data

In [202…
```
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Heatmap')
plt.show()
```



In [203…
```
df
```

Out[203]:

	lat	lng	desc	zip	title	timeStamp	tw
0	40.297876	-75.581294	REINDEER CT & DEAD END; NEW HANOVER; Station ...	19525.0	EMS: BACK PAINS/INJURY	2015-12-10 17:40:00	NE\ HANOVE
1	40.258061	-75.264680	BRIAR PATH & WHITEMARSH LN; HATFIELD TOWNSHIP...	19446.0	EMS: DIABETIC EMERGENCY	2015-12-10 17:40:00	HATFIEL TOWNSH
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	NORRISTOW
3	40.116153	-75.343513	AIRY ST & SWEDE ST; NORRISTOWN; Station 308A;...	19401.0	EMS: CARDIAC EMERGENCY	2015-12-10 17:40:01	NORRISTOW
4	40.251492	-75.603350	CHERRYWOOD CT & DEAD END; LOWER POTTSGROVE; S...	NaN	EMS: DIZZINESS	2015-12-10 17:40:01	LOWE POTTSGRO\
...	...	...	...	...	...	...	
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	NORRISTOW
99488	40.006974	-75.289080	LANCASTER AVE & RITTENHOUSE PL; LOWER MERION; ...	19003.0	Traffic: VEHICLE ACCIDENT -	2016-08-24 11:07:02	LOWE MERIO
99489	40.115429	-75.334679	CHESTNUT ST & WALNUT ST; NORRISTOWN; Station ...	19401.0	EMS: FALL VICTIM	2016-08-24 11:12:00	NORRISTOW
99490	40.186431	-75.192555	WELSH RD & WEBSTER LN; HORSHAM; Station 352; ...	19002.0	EMS: NAUSEA/VOMITING	2016-08-24 11:17:01	HORSHA
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	UPPE GWYNED

99492 rows × 26 columns