911 Calls Capstone Project

This following part of this exercise can be done and delivered untill Saturday, 22/06/2024 up to 12:10PM (middle of day).

For this capstone project we will be analyzing some 911 call data. The data contains the following fields:

- lat : String variable, Latitude
- lng: String variable, Longitude
- desc: String variable, Description of the Emergency Call
- zip: String variable, Zipcode
- title: String variable, Title
- timeStamp: String variable, YYYY-MM-DD HH:MM:SS
- twp: String variable, Township
- addr: String variable, Address
- e: String variable, Dummy variable (always 1)

Just go along with this notebook and try to complete the instructions or answer the questions in bold using your Python and Data Science skills!

Data and Setup

** Import numpy and pandas **

In [ ]:
```
import numpy as np
import pandas as pd
```

** Import visualization libraries and set %matplotlib inline. **

In [ ]:
```
import matplotlib.pyplot as plt
%matplotlib inline
```

In [ ]:
```
import seaborn as sns
```

** Read in the csv file as a dataframe called df **

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

** Check the info() of the df **

In [ ]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 99492 entries, 0 to 99491
Data columns (total 9 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   lat        99492 non-null  float64
 1   lng        99492 non-null  float64
 2   desc       99492 non-null  object 
 3   zip        86637 non-null  float64
 4   title      99492 non-null  object 
 5   timeStamp  99492 non-null  object 
 6   twp        99449 non-null  object 
 7   addr       98973 non-null  object 
 8   e          99492 non-null  int64  
dtypes: float64(3), int64(1), object(5)
memory usage: 6.8+ MB
```

** Check the head of df **

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

Out[ ]:

	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 HANOV
1	40.258061	-75.264680	BRIAR PATH & WHITEMARSH LN; HATFIELD TOWNSHIP...	19446.0	EMS: DIABETIC EMERGENCY	2015-12-10 17:40:00	HATFIE 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	NORRISTOV
3	40.116153	-75.343513	AIRY ST & SWEDE ST; NORRISTOWN; Station 308A;...	19401.0	EMS: CARDIAC EMERGENCY	2015-12-10 17:40:01	NORRISTOV
4	40.251492	-75.603350	CHERRYWOOD CT & DEAD END; LOWER POTTSGROVE; S...	NaN	EMS: DIZZINESS	2015-12-10 17:40:01	LOW POTTSGRO

Basic Questions

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

In [ ]:
```
df.zip.value_counts().head(5)
```

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

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

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

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

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

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

Out[ ]:
```
110
```

Creating new features

** In the titles column there are "Reasons/Departments" specified before the title code. These are EMS, Fire, and Traffic. Use .apply() with a custom lambda expression to create a new column called "Reason" that contains this string value.**

For example, if the title column value is EMS: BACK PAINS/INJURY , the Reason column value would be EMS.

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

Out[ ]:

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

99492 rows × 10 columns

In [ ]:
```
df.Reason
```

Out[ ]:
```
0           EMS
1           EMS
2          Fire
3           EMS
4           EMS
          ...
99487    Traffic
99488    Traffic
99489        EMS
99490        EMS
99491    Traffic
Name: Reason, Length: 99492, dtype: object
```

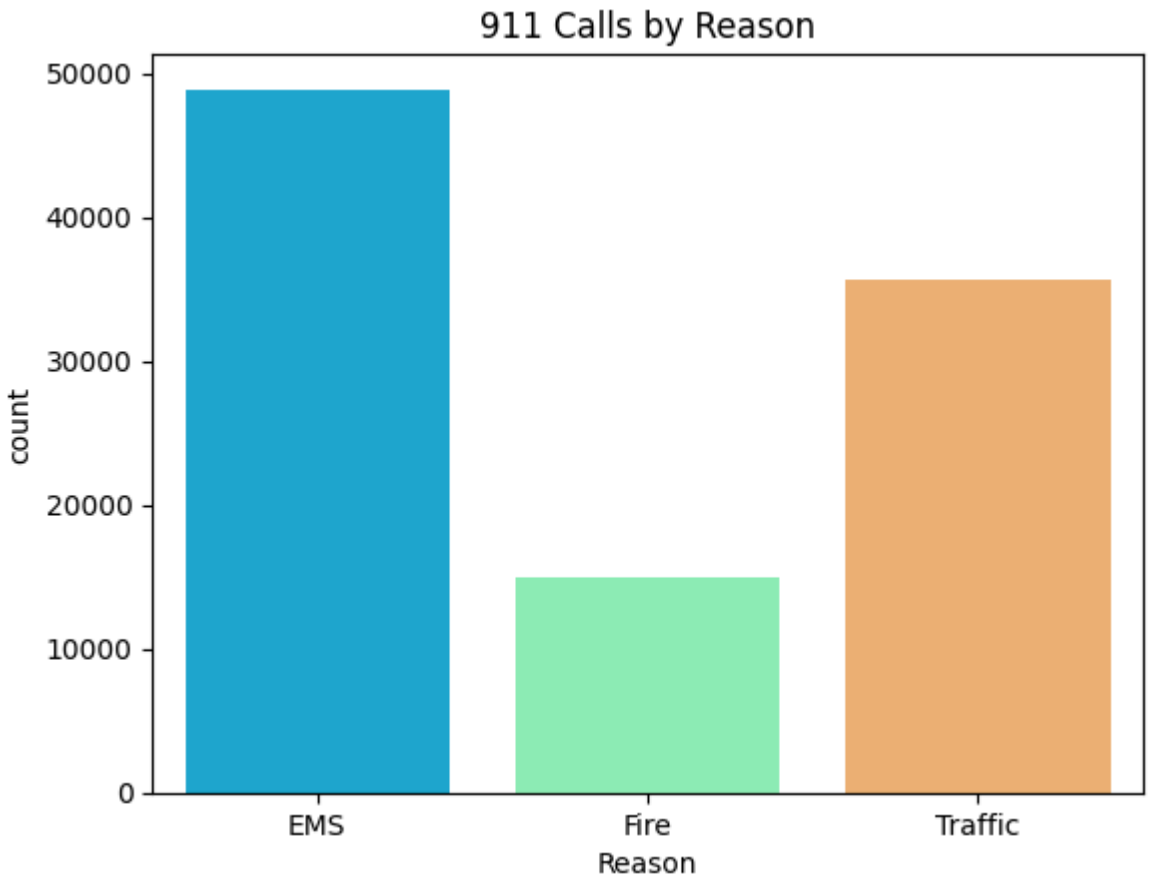
** What is the most common Reason for a 911 call based off of this new column? **

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

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

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

In [ ]:
```
gp = sns.countplot(x='Reason', data=df,palette = 'rainbow', hue='Reason').set_ti
```



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

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

Out[ ]:
```
str
```

** You should have seen that these timestamps are still strings. You must convert the column from strings to DateTime objects. **

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

In [ ]:
```
df.dtypes
```

Out[ ]:
```
lat                  float64
lng                  float64
desc                  object
zip                  float64
title                 object
timeStamp    datetime64[ns]
twp                   object
addr                  object
e                      int64
Reason                object
dtype: object
```

** You can now grab specific attributes from a Datetime object by calling them. For example:**

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

You can use Jupyter's tab method to explore the various attributes you can call. Now that the timestamp column are actually DateTime objects, use .apply() to create 3 new columns called Hour, Month, and Day of Week. You will create these columns based off of the timeStamp column, reference the solutions if you get stuck on this step.

In [ ]:
```
df['hour'] = df.apply(lambda x: x['timeStamp'].hour, axis=1)
df['month'] = df.apply(lambda x: x['timeStamp'].month, axis=1)
df['day of week'] = df.apply(lambda x: x['timeStamp'].dayofweek, axis=1)
df
```

Out[ ]:

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

99492 rows × 13 columns

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

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

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

In [ ]:
```
df
```

Out[ ]:

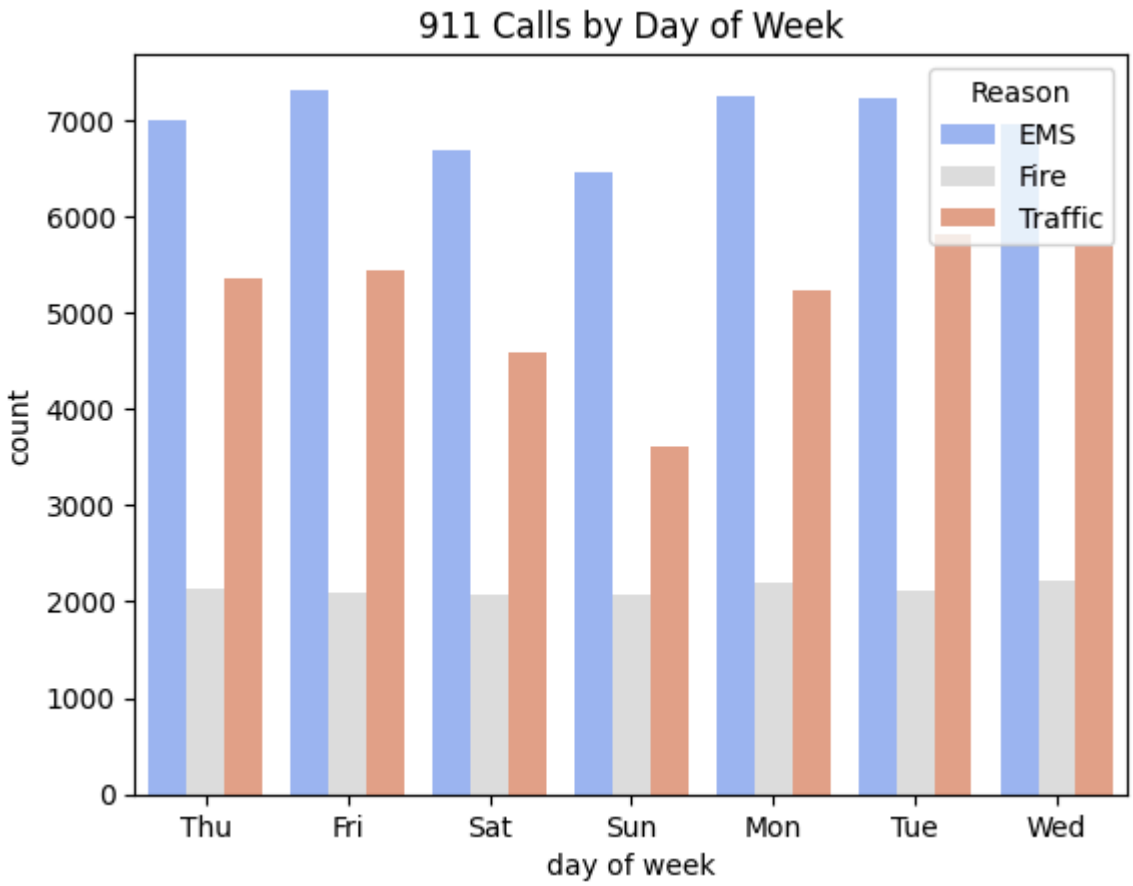
	lat	lng	desc	zip	title	timeStamp	
0	40.297876	-75.581294	REINDEER CT & DEAD END; NEW HANOVER; Station ...	19525.0	EMS: BACK PAINS/INJURY	2015-12-10 17:40:00	
1	40.258061	-75.264680	BRIAR PATH & WHITEMARSH LN; HATFIELD TOWNSHIP...	19446.0	EMS: DIABETIC EMERGENCY	2015-12-10 17:40:00	
2	40.121182	-75.351975	HAWS AVE; NORRISTOWN; 2015-12-10 @ 14:39:21-St...	19401.0	Fire: GAS-ODOR/LEAK	2015-12-10 17:40:00	N
3	40.116153	-75.343513	AIRY ST & SWEDE ST; NORRISTOWN; Station 308A;...	19401.0	EMS: CARDIAC EMERGENCY	2015-12-10 17:40:01	N
4	40.251492	-75.603350	CHERRYWOOD CT & DEAD END; LOWER POTTSGROVE; S...	NaN	EMS: DIZZINESS	2015-12-10 17:40:01	
...	...	...	...	...	...	...	
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	N
99488	40.006974	-75.289080	LANCASTER AVE & RITTENHOUSE PL; LOWER MERION; ...	19003.0	Traffic: VEHICLE ACCIDENT -	2016-08-24 11:07:02	
99489	40.115429	-75.334679	CHESTNUT ST & WALNUT ST; NORRISTOWN; Station ...	19401.0	EMS: FALL VICTIM	2016-08-24 11:12:00	N
99490	40.186431	-75.192555	WELSH RD & WEBSTER LN; HORSHAM; Station 352; ...	19002.0	EMS: NAUSEA/VOMITING	2016-08-24 11:17:01	
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	

99492 rows × 13 columns

** Now use seaborn to create a countplot of the Day of Week column with the hue based off of the Reason column. **

In [ ]:
```
sns.countplot(x='day of week',data=df,hue='Reason',palette='coolwarm').set_title
```

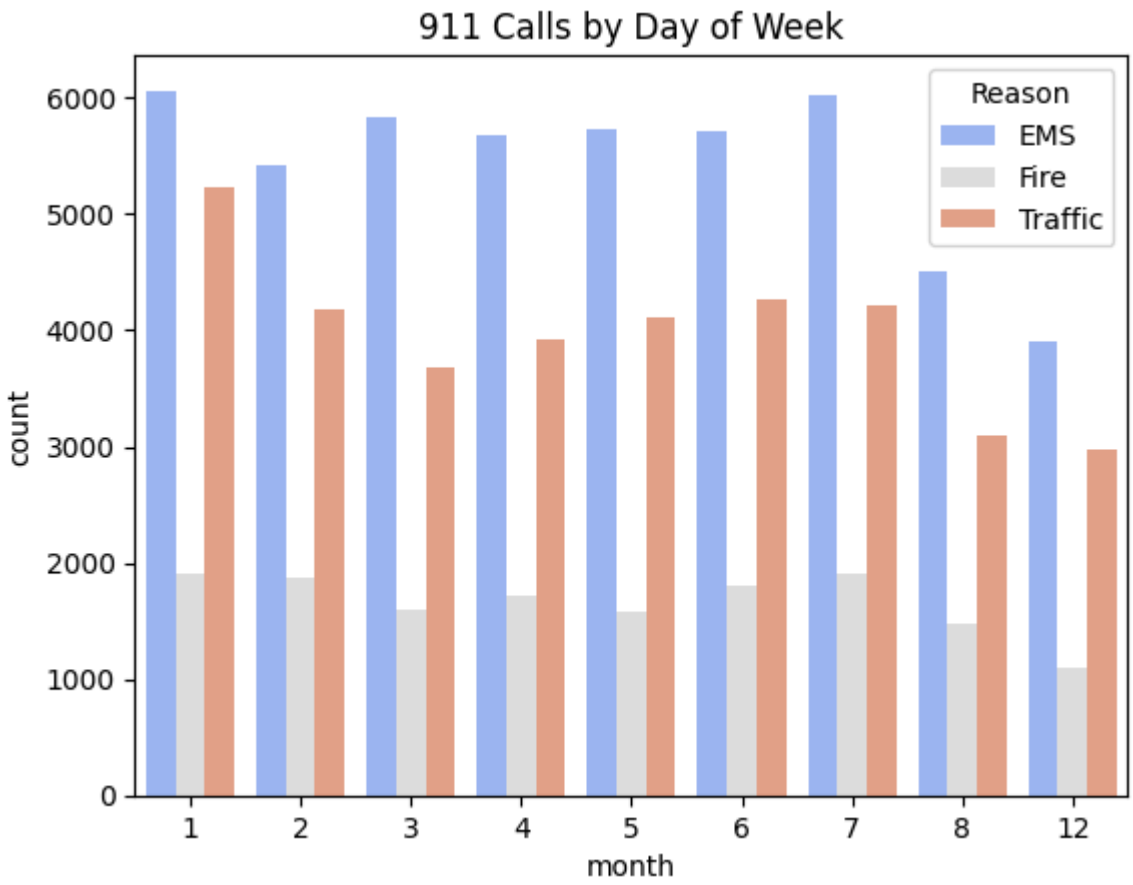
Out[ ]:
```
Text(0.5, 1.0, '911 Calls by Day of Week')
```



Now do the same for Month:

In [ ]:
```
sns.countplot(x='month',data=df,hue='Reason',palette='coolwarm').set_title('911
```

Out[ ]:
```
Text(0.5, 1.0, '911 Calls by Day of Week')
```



Did you notice something strange about the Plot?

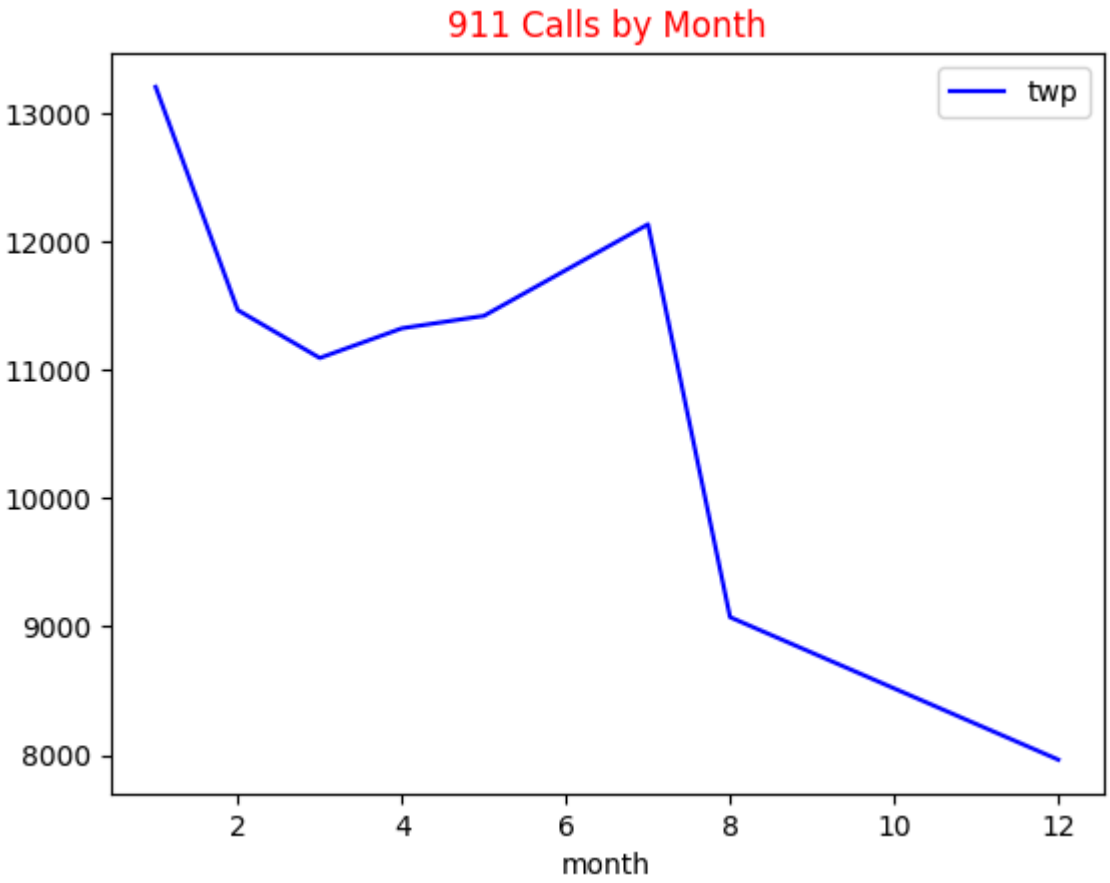
** You should have noticed it was missing some Months, let's see if we can maybe fill in this information by plotting the information in another way, possibly a simple line plot that fills in the missing months, in order to do this, we'll need to do some work with pandas... **

** Now create a gropuby object called byMonth, where you group the DataFrame by the month column and use the count() method for aggregation. Use the head() method on this returned DataFrame. **

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

In [ ]:
```
byMonth.plot(y='twp',color='blue').set_title('911 Calls by Month',color='red')
```

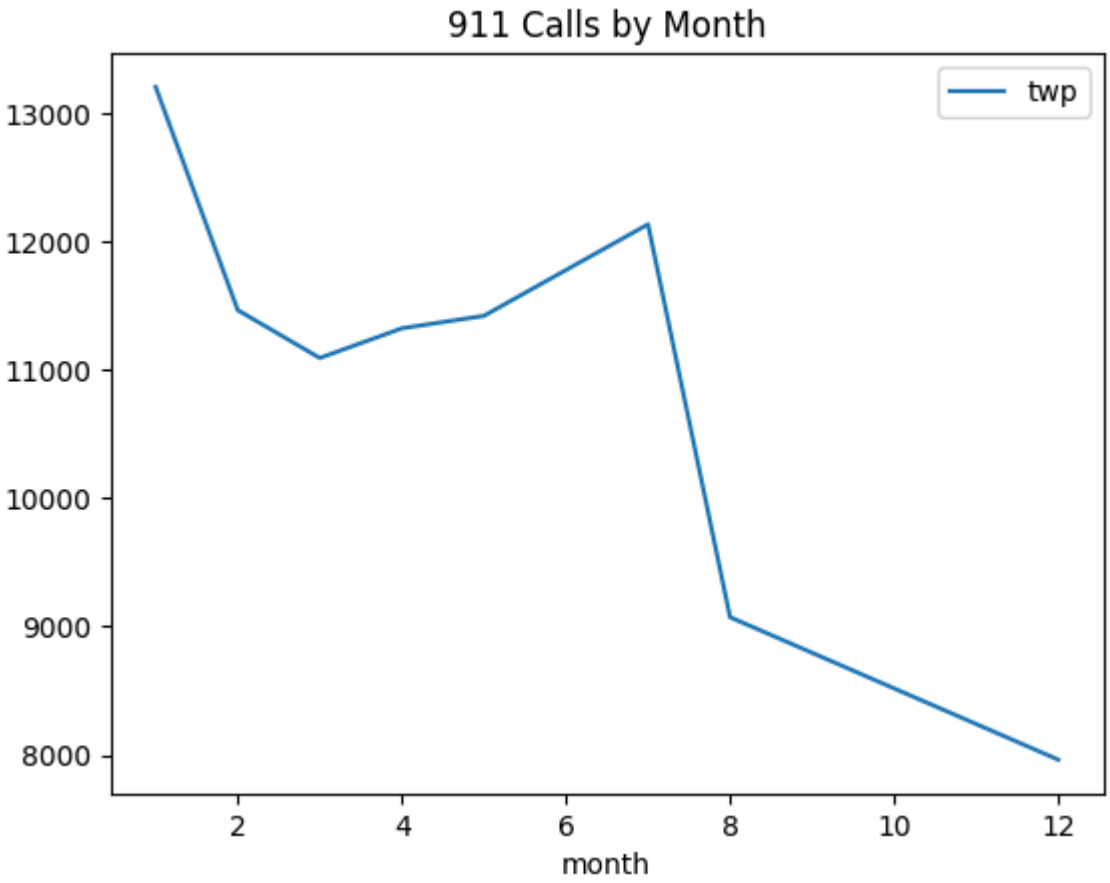
Out[ ]:
```
Text(0.5, 1.0, '911 Calls by Month')
```



** Now create a simple plot off of the dataframe indicating the count of calls per month. **

In [ ]:
```
byMonth.plot.line(y='twp').set_title('911 Calls by Month')
```

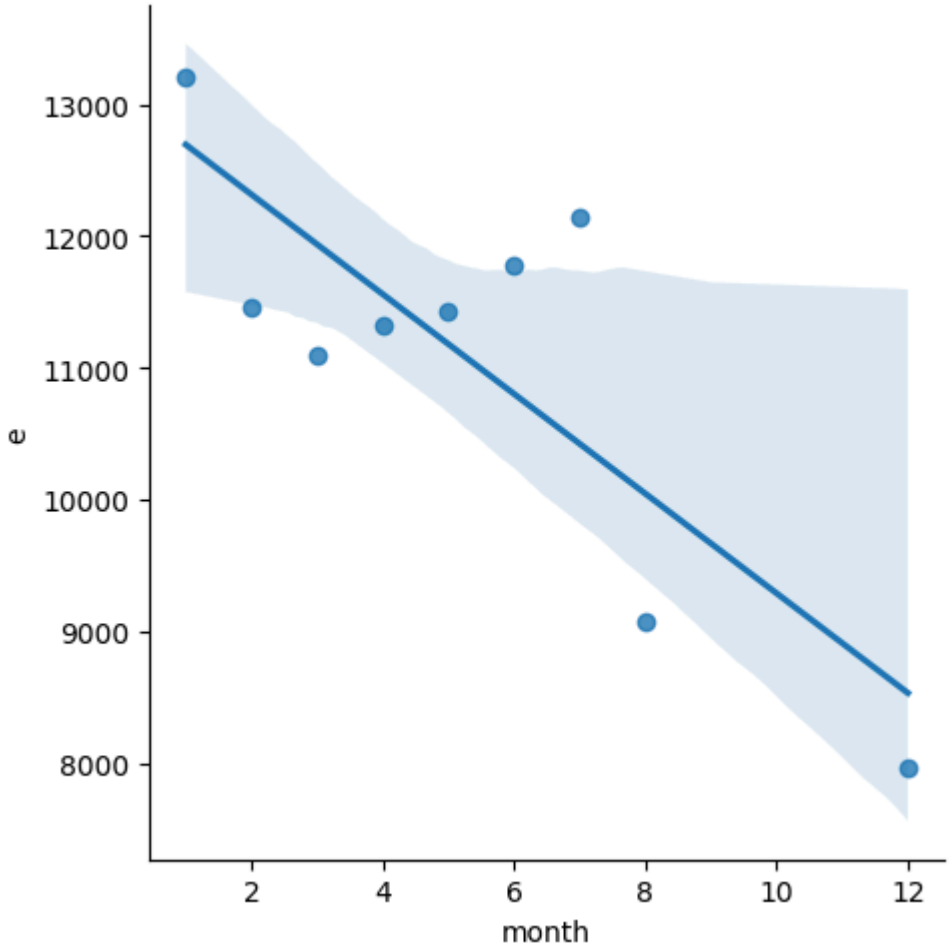
Out[ ]:
```
Text(0.5, 1.0, '911 Calls by Month')
```



** Now see if you can use seaborn's lmplot() to create a linear fit on the number of calls per month. Keep in mind you may need to reset the index to a column. **

In [ ]:
```
byMonth
sns.lmplot(x='month',y='e',data=byMonth.reset_index())
```

Out[ ]:
```
<seaborn.axisgrid.FacetGrid at 0x1a5217206e0>
```



Create a new column called 'Date' that contains the date from the timeStamp column. You'll need to use apply along with the .date() method.

In [ ]:
```
df['date'] = df.apply(lambda x: x['timeStamp'].date(), axis=1)
df
```

Out[ ]:

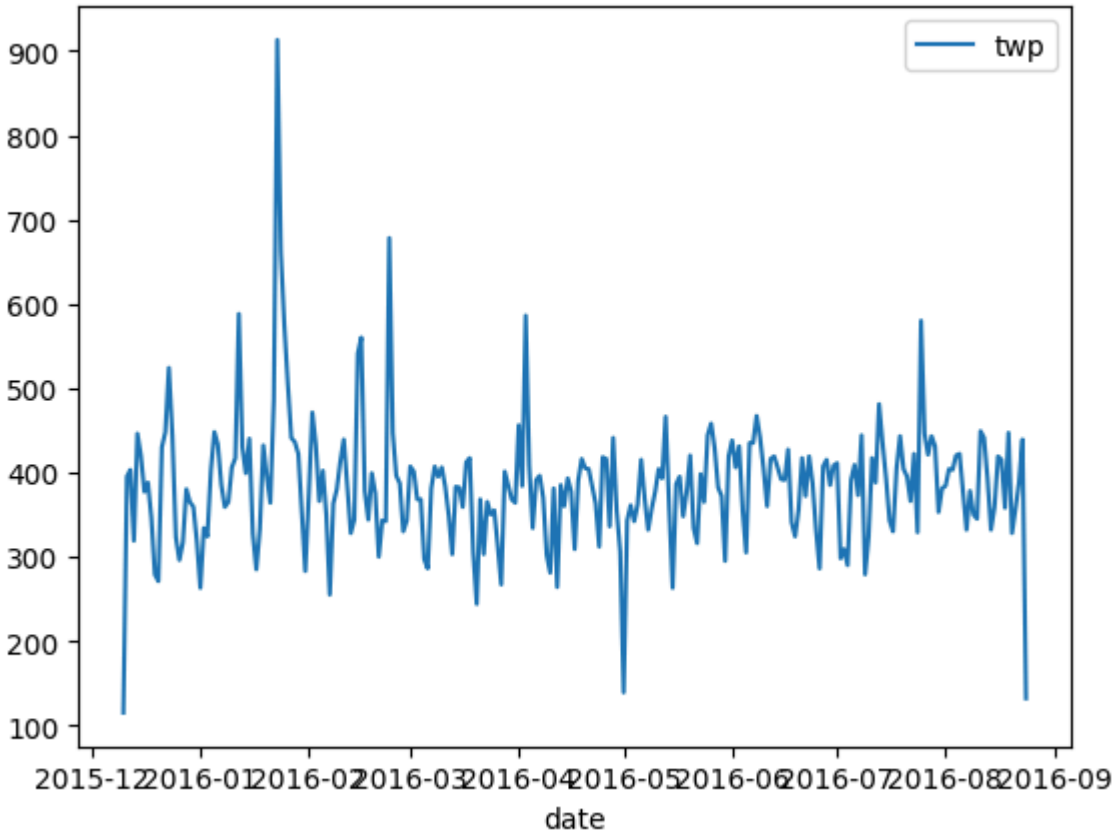
	lat	lng	desc	zip	title	timeStamp	
0	40.297876	-75.581294	REINDEER CT & DEAD END; NEW HANOVER; Station ...	19525.0	EMS: BACK PAINS/INJURY	2015-12-10 17:40:00	
1	40.258061	-75.264680	BRIAR PATH & WHITEMARSH LN; HATFIELD TOWNSHIP...	19446.0	EMS: DIABETIC EMERGENCY	2015-12-10 17:40:00	
2	40.121182	-75.351975	HAWS AVE; NORRISTOWN; 2015-12-10 @ 14:39:21-St...	19401.0	Fire: GAS-ODOR/LEAK	2015-12-10 17:40:00	N
3	40.116153	-75.343513	AIRY ST & SWEDE ST; NORRISTOWN; Station 308A;...	19401.0	EMS: CARDIAC EMERGENCY	2015-12-10 17:40:01	N
4	40.251492	-75.603350	CHERRYWOOD CT & DEAD END; LOWER POTTSGROVE; S...	NaN	EMS: DIZZINESS	2015-12-10 17:40:01	
...	...	...	...	...	...	...	
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	N
99488	40.006974	-75.289080	LANCASTER AVE & RITTENHOUSE PL; LOWER MERION; ...	19003.0	Traffic: VEHICLE ACCIDENT -	2016-08-24 11:07:02	
99489	40.115429	-75.334679	CHESTNUT ST & WALNUT ST; NORRISTOWN; Station ...	19401.0	EMS: FALL VICTIM	2016-08-24 11:12:00	N
99490	40.186431	-75.192555	WELSH RD & WEBSTER LN; HORSHAM; Station 352; ...	19002.0	EMS: NAUSEA/VOMITING	2016-08-24 11:17:01	
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	

99492 rows × 14 columns

** Now groupby this Date column with the count() aggregate and create a plot of counts of 911 calls.**

In [ ]:
```
byDate = df.groupby('date').count()
byDate.plot(y='twp',)
```

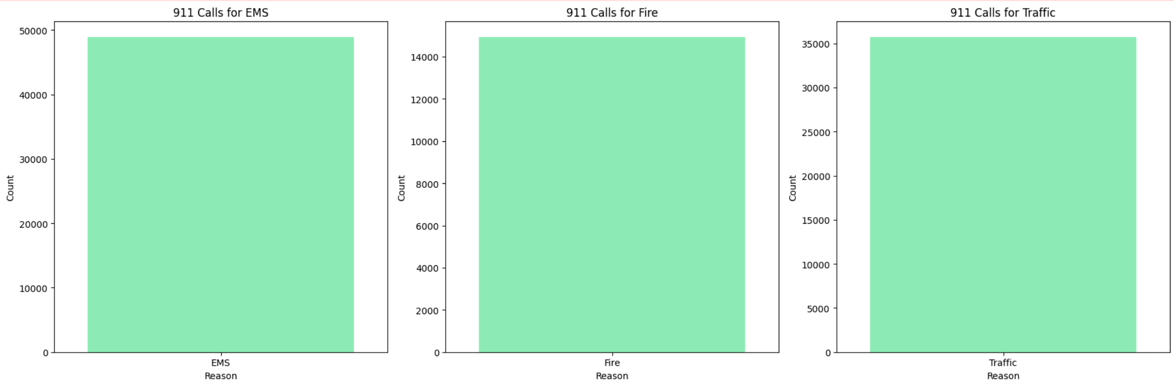
Out[ ]: <Axes: xlabel='date'>



** Now recreate this plot but create 3 separate plots with each plot representing a Reason for the 911 call**

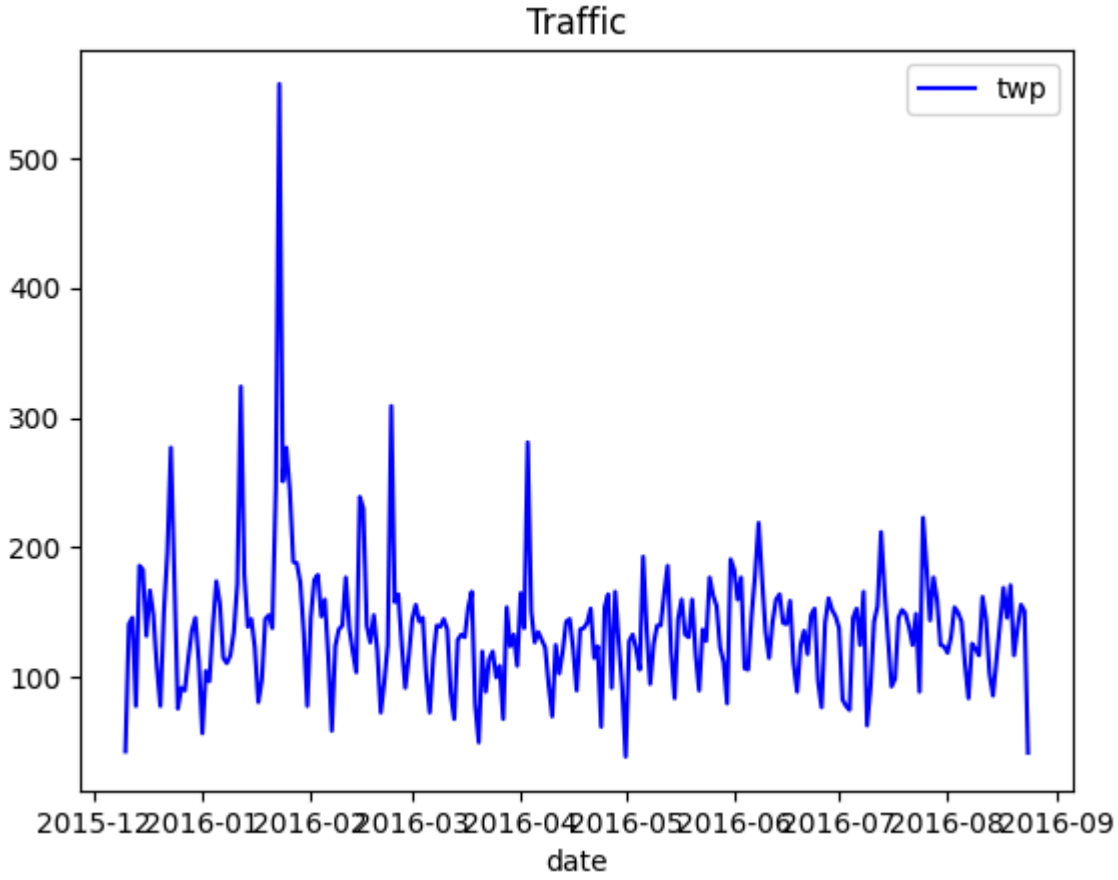
In [ ]:
```
unique_reasons = df['Reason'].unique()
fig, axes = plt.subplots(1, 3, figsize=(18, 6))
for ax, reason in zip(axes, unique_reasons):
    sns.countplot(x='Reason', data=df[df['Reason'] == reason], palette='rainbow'
    ax.set_title(f'911 Calls for {reason}')
    ax.set_xlabel('Reason')
    ax.set_ylabel('Count')
    plt.tight_layout()
plt.show()
```

```
C:\Users\Enzo\AppData\Local\Temp\ipykernel_13796\888554296.py:4: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v
0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effe
ct.

  sns.countplot(x='Reason', data=df[df['Reason'] == reason], palette='rainbow', a
x=ax)
C:\Users\Enzo\AppData\Local\Temp\ipykernel_13796\888554296.py:4: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v
0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effe
ct.

  sns.countplot(x='Reason', data=df[df['Reason'] == reason], palette='rainbow', a
x=ax)
C:\Users\Enzo\AppData\Local\Temp\ipykernel_13796\888554296.py:4: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v
0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effe
ct.

  sns.countplot(x='Reason', data=df[df['Reason'] == reason], palette='rainbow', a
x=ax)
```



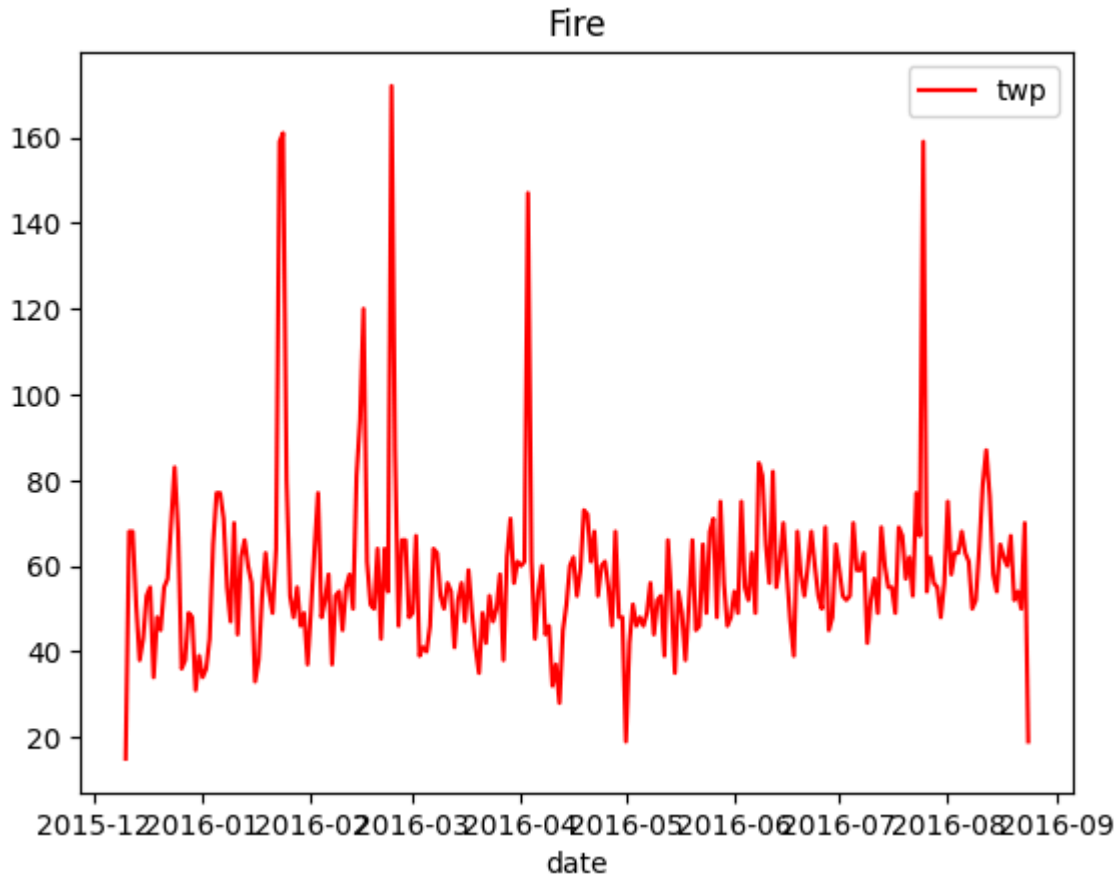
In [ ]:
```
df[df['Reason']=='Traffic'].groupby('date').count().plot(y='twp',color='blue').s
```

Out[ ]:
```
Text(0.5, 1.0, 'Traffic')
```



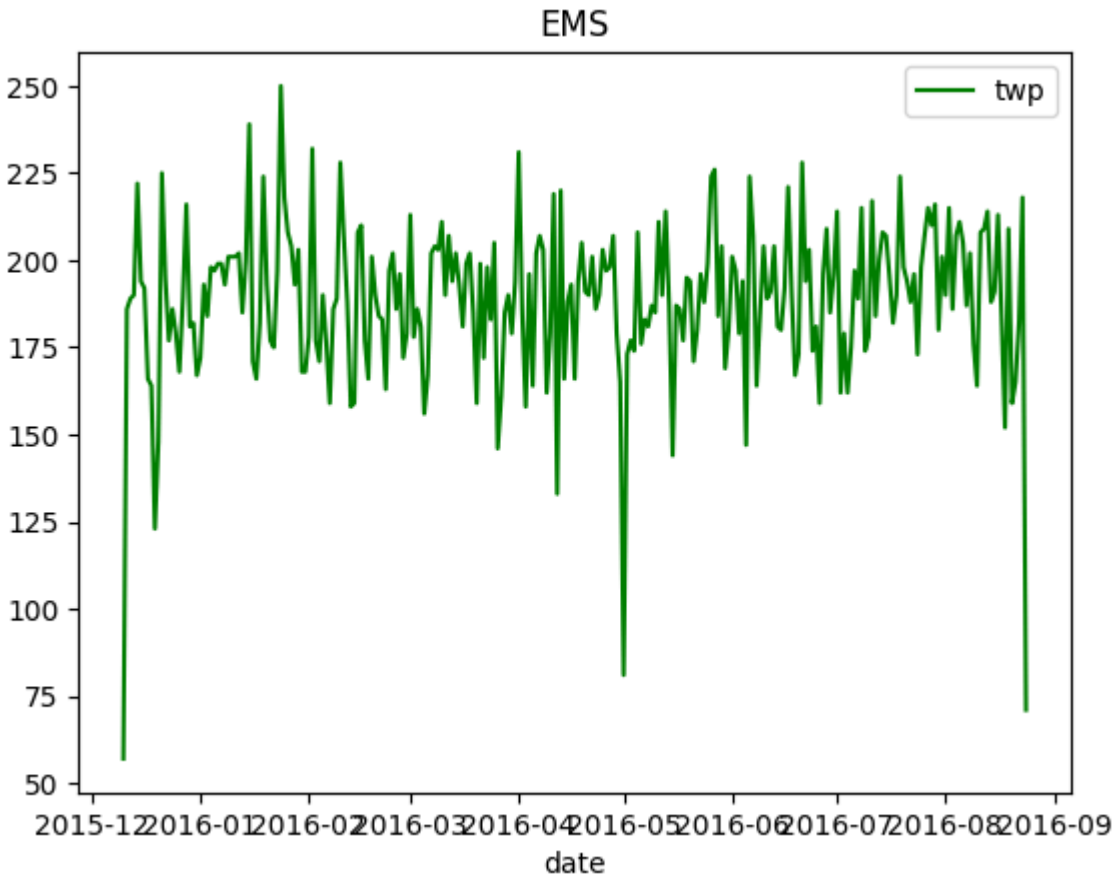
In [ ]:
```
df[df['Reason']=='Fire'].groupby('date').count().plot(y='twp',color='red').set_t
```

Out[ ]:
```
Text(0.5, 1.0, 'Fire')
```



In [ ]:
```
df[df['Reason']=='EMS'].groupby('date').count().plot(y='twp',color='green').set_
```

Out[ ]:
```
Text(0.5, 1.0, 'EMS')
```



** Now let's move on to creating heatmaps with seaborn and our data. We'll first need to restructure the dataframe so that the columns become the Hours and the Index becomes the Day of the Week. There are lots of ways to do this, but I would recommend trying to combine groupby with an unstack method. Reference the solutions if you get stuck on this!**

In [ ]:
```
byhourbydow = df.groupby(by=['day of week','hour']).count()['Reason'].unstack()
byhourbydow
```

Out[ ]:

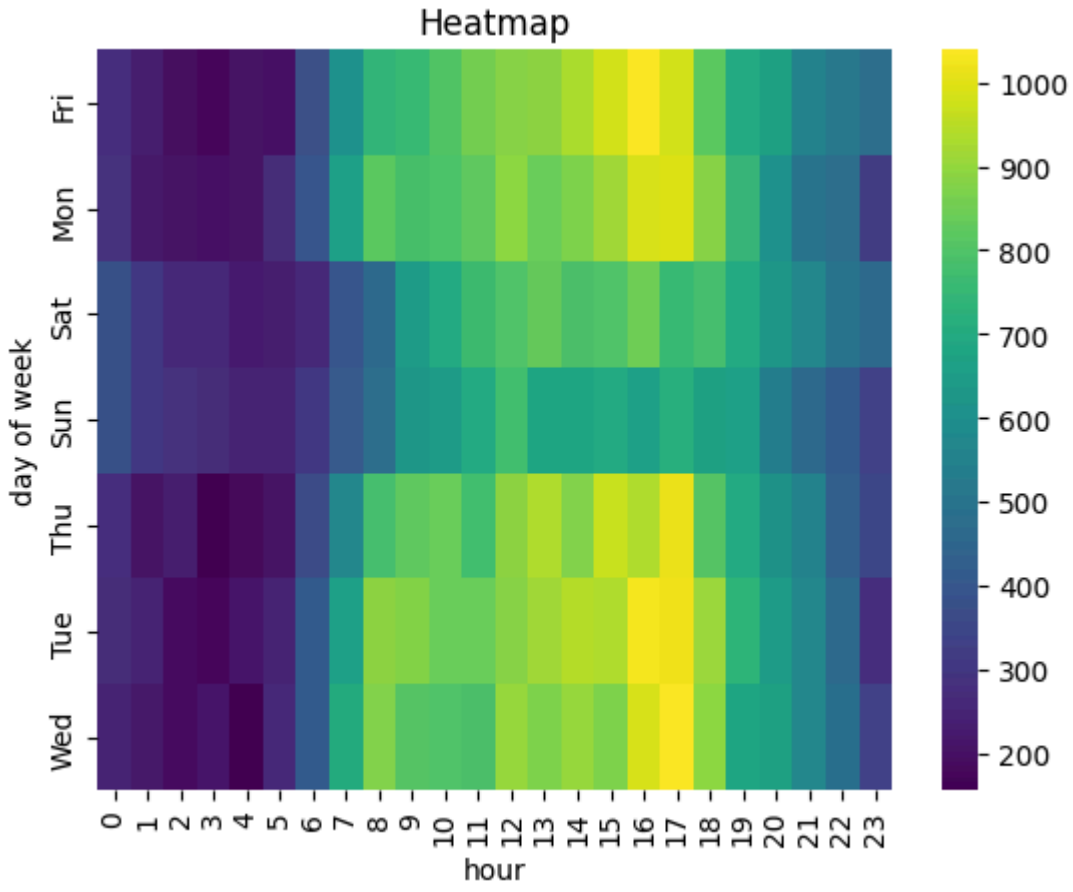
hour	0	1	2	3	4	5	6	7	8	9	...	14	15	16	17	1
day of week																
Fri	275	235	191	175	201	194	372	598	742	752	...	932	980	1039	980	82
Mon	282	221	201	194	204	267	397	653	819	786	...	869	913	989	997	88
Sat	375	301	263	260	224	231	257	391	459	640	...	789	796	848	757	77
Sun	383	306	286	268	242	240	300	402	483	620	...	684	691	663	714	67
Thu	278	202	233	159	182	203	362	570	777	828	...	876	969	935	1013	81
Tue	269	240	186	170	209	239	415	655	889	880	...	943	938	1026	1019	90
Wed	250	216	189	209	156	255	410	701	875	808	...	904	867	990	1037	89

7 rows × 24 columns

** Now create a HeatMap using this new DataFrame. **

In [ ]:
```
sns.heatmap(byhourbydow,cmap='viridis').set_title('Heatmap')
```

Out[ ]:
```
Text(0.5, 1.0, 'Heatmap')
```



** Now create a clustermap using this DataFrame. **

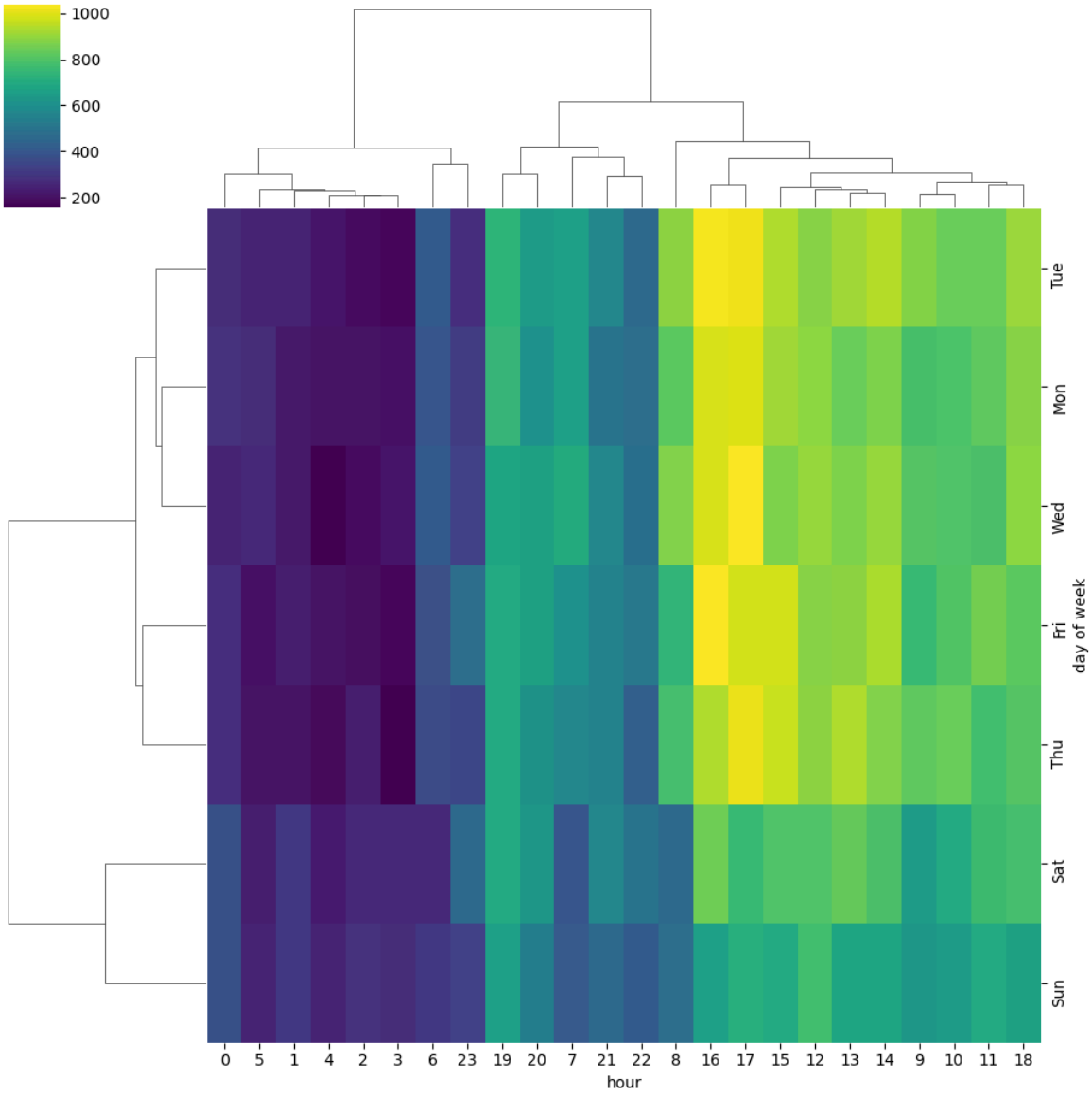
In [ ]:
```
%pip install scipy
```

```
Requirement already satisfied: scipy in c:\users\enzo\appdata\local\programs\pyth
on\python312\lib\site-packages (1.13.1)
Requirement already satisfied: numpy<2.3,>=1.22.4 in c:\users\enzo\appdata\local
\programs\python\python312\lib\site-packages (from scipy) (1.26.4)
Note: you may need to restart the kernel to use updated packages.
```

In [ ]:
```
import scipy
```

In [ ]:
```
sns.clustermap(byhourbydow,cmap='viridis')
```

Out[ ]:
```
<seaborn.matrix.ClusterGrid at 0x1a5245bfd70>
```



** Now repeat these same plots and operations, for a DataFrame that shows the Month as the column. **

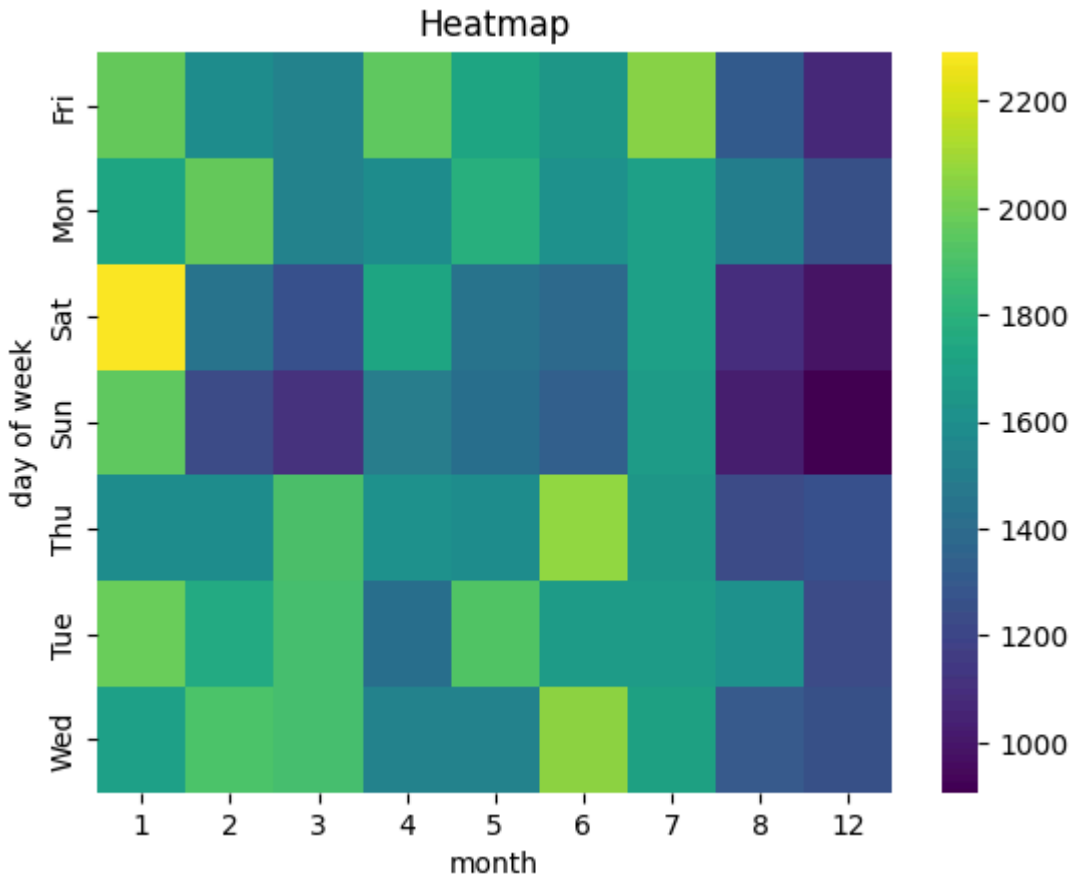
In [ ]:
```
byMonthbydow = df.groupby(by=['day of week','month']).count()['Reason'].unstack(
byMonthbydow
```

Out[ ]:

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
Tue	1973	1753	1884	1430	1918	1676	1670	1612	1234
Wed	1700	1903	1889	1517	1538	2058	1717	1295	1262

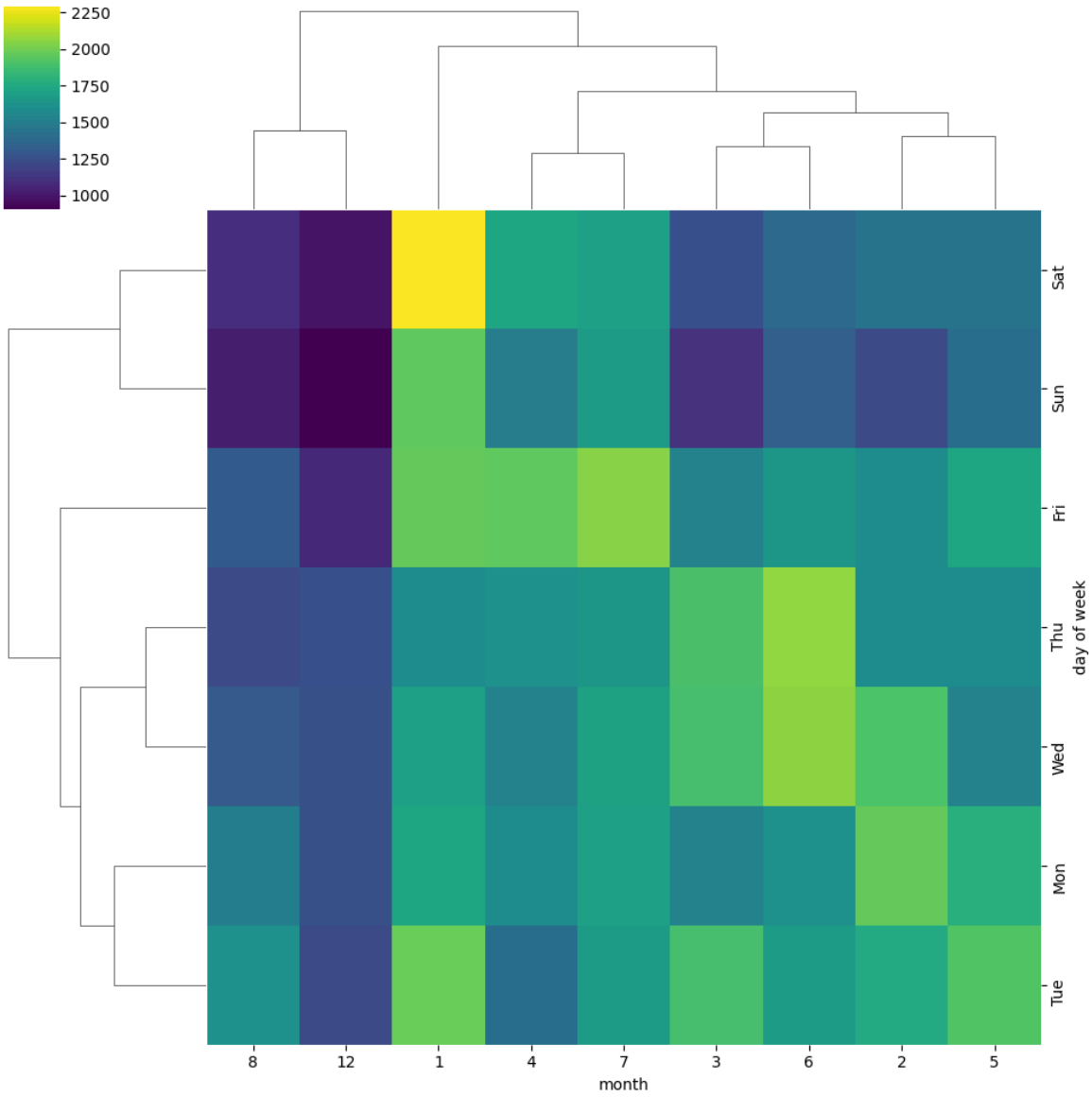
In [ ]:
```
sns.heatmap(byMonthbydow,cmap='viridis').set_title('Heatmap')
```

Out[ ]:
```
Text(0.5, 1.0, 'Heatmap')
```



In [ ]:
```
sns.clustermap(byMonthbydow,cmap='viridis')
```

Out[ ]:
```
<seaborn.matrix.ClusterGrid at 0x1a521a66de0>
```



PART 2

This following part of this exercise can be done and delivered untill Sunday, 23/06/2024 up to 23:59 (11:59 PM).

Exercise: Analyzing Students.csv or whatever data base you may want to

(Data.gov, EU Open Data Portal, Kaggle Datasets) work through -

Performance Data

***You are provided with a dataset containing information about student performance in exams. Your task is to perform data analysis and visualization using Python libraries. Here are the steps to follow:

1) Load the Data:

1.1) Use pandas to read the dataset from a CSV file (students.csv). Data Exploration:

1.2) Display the first few rows of the dataset to understand its structure.

1.3) Check for missing values and handle them appropriately if necessary.

2) Data Analysis:

2.1) Calculate basic statistics of the dataset (mean, median, min, max, etc.).

Explore the distribution of scores using histograms and box plots.

3) Data Visualization:

3.1) Use matplotlib and seaborn to create visualizations such as:

a) Histograms of scores in different subjects.

b) Box plots to compare scores across different categories (e.g., gender, parental ##### level of education).

c) Scatter plots to explore relationships between variables (e.g., math vs. reading ##### scores).

4) Advanced Analysis:

4.1) Calculate correlations between different variables (e.g., scores in different

subjects).

4.2) Create a heatmap using seaborn to visualize correlations.

Conclusion:

Summarize - State your findings from the analysis.

Provide insights or conclusions based on the visualizations and ### analyses performed.

Send these two exercises to

fischer.stefan@academico.domhelder.edu.br

Subeject: Project Capstone

Save versions in .py or ipynb and .pdf

Do not forget to write down your name!!

In [ ]:
```
Enzo Rocha Leite Diniz Ribas
D24642
```

```
  Cell In[39], line 1
    Enzo Rocha Leite Diniz Ribas
         ^
SyntaxError: invalid syntax
```