MUKESH PATEL SCHOOL OF TECHNOLOGY MANAGEMENT AND ENGINEERING

(Affiliated to NMIMS Deemed to be University, Mumbai)

Data Extraction and Processing

Project Report

on

"Analysis of the 911 Emergency Calls from Montgomery County"

Submitted by:

Group 7	
Name	**Roll No.**
Vansh Dhoka	C042
Amogh Jambaulikar	C043
Amishi Desai	C044
Sparsh Panchori	C045
Pratham Vasa	C048
Semester: VII	
Year: IV	

B. Tech Integrated Program

Department of Computer Science Engineering

MPSTME, Mumbai

2023-2024

About the Dataset

Dataset Title: *Emergency – 911 Calls, Montgomery Country*

URL for Dataset Download:

https://www.kaggle.com/datasets/mchirico/montcoalert

The 911 emergency call dataset contains collection of information related to the emergency calls made on 911 Montgomery County, PA, due to various reasons as this serves as a crucial lifeline for residents and visitors in times of distress. This dataset contains details about the exact location of the emergency stations that are contacted, description of emergency, detail about the emergency, timestamp, etc. which are used to perform various analysis and visualization.

Description about the attributes in the dataset:

Attribute Name	Description
Lat	Latitude of the station
Lng	Longitude of the station
Desc	Description of the Emergency Call
Zip	Zipcode
title	Emergency Reason
Timestamp	Timestamp of Call (YYYY-MM-DD HH:MM:SS)
Twp	Township
Addr	Address
e	Dummy Variable (always 1)

DATA EXPLORATION

- Importing pandas, numpy, seaborn, matplotlib.pyplot, geopandas and plotly.express libraries

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

```
[ ] import seaborn as sns
    import matplotlib.pyplot as plt
    %matplotlib inline
```

```
[ ] import geopandas as gpd
    import plotly.express as px
```

- Reading the dataset and displaying the first 5 values

```
[ ] dataframe=pd.read_csv('/content/911.csv')
```

```
[ ] (dataframe.head())
```

	lat	lng	desc	zip	title	timeStamp	twp	addr	e
0	40.297876	-75.581294	REINDEER CT & DEAD END; NEW HANOVER; Station ...	19525.0	EMS: BACK PAINS/INJURY	2015-12-10 17:10:52	NEW HANOVER	REINDEER CT & DEAD END	1
1	40.258061	-75.264680	BRIAR PATH & WHITEMARSH LN; HATFIELD TOWNSHIP...	19446.0	EMS: DIABETIC EMERGENCY	2015-12-10 17:29:21	HATFIELD TOWNSHIP	BRIAR PATH & WHITEMARSH LN	1
2	40.121182	-75.351975	HAWS AVE; NORRISTOWN; 2015-12-10 @ 14:39:21-St...	19401.0	Fire: GAS-ODOR/LEAK	2015-12-10 14:39:21	NORRISTOWN	HAWS AVE	1
3	40.116153	-75.343513	AIRY ST & SWEDE ST; NORRISTOWN; Station 308A;...	19401.0	EMS: CARDIAC EMERGENCY	2015-12-10 16:47:36	NORRISTOWN	AIRY ST & SWEDE ST	1
4	40.251492	-75.603350	CHERRYWOOD CT & DEAD END; LOWER POTTSGROVE; S...	NaN	EMS: DIZZINESS	2015-12-10 16:56:52	LOWER POTTSGROVE	CHERRYWOOD CT & DEAD END	1

- Printing all the columns of the dataframe

```
[ ] print(dataframe.columns)

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

- .shape -> shows the total number of rows and columns in the dataset

```
[ ] print(dataframe.shape)

    (663522, 9)
```

There are 663522 rows and 9 columns

- .dtypes-> gives the data type of the attributes used in the dataset

```
[ ] print(dataframe.dtypes)

    lat          float64
    lng          float64
    desc          object
    zip          float64
    title         object
    timeStamp     object
    twp           object
    addr          object
    e              int64
    dtype: object
```

- .isnull.sum() -> checks the null values and returns the total null values present for all attributes.

```
[ ] dataframe.isnull().sum()

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

Zip and twp comprises of null values

- .describe() -> generates descriptive statistics summarizing the distribution of numerical data.

```
[ ] print(dataframe.describe())

                   lat            lng            zip         e
    count  663522.000000  663522.000000  583323.000000  663522.0
    mean       40.158162     -75.300105   19236.055791       1.0
    std         0.220641       1.672884     298.222637       0.0
    min         0.000000    -119.698206    1104.000000       1.0
    25%        40.100344     -75.392735   19038.000000       1.0
    50%        40.143927     -75.305143   19401.000000       1.0
    75%        40.229008     -75.211865   19446.000000       1.0
    max        51.335390      87.854975   77316.000000       1.0
```

- Converts the 'timeStamp' column to a datetime format, sorts the DataFrame based on the 'timeStamp' column, and prints the 'timeStamp' column without any further modifications

```
[ ] #sorting of timeStamp in ascending order
    #dataframe['timeStamp'] = dataframe['timeStamp']

    dataframe['timeStamp'] = pd.to_datetime(dataframe['timeStamp'], format='%Y-%m-%d %H:%M')

    sorted_dataframe = dataframe.sort_values(by='timeStamp', ascending=False)

    print(dataframe['timeStamp'])

    0        2015-12-10 17:10:52
    1        2015-12-10 17:29:21
    2        2015-12-10 14:39:21
    3        2015-12-10 16:47:36
    4        2015-12-10 16:56:52
                     ...
    663517   2020-07-29 15:46:51
    663518   2020-07-29 15:52:19
    663519   2020-07-29 15:52:52
    663520   2020-07-29 15:54:08
    663521   2020-07-29 15:52:46
    Name: timeStamp, Length: 663522, dtype: datetime64[ns]
```

- Sorting the titles in ascending alphabetical order

```
[ ] # Sort the incident titles in ascending alphabetical order
    sorted_dataframe = dataframe.sort_values(by='title', ascending=True)

    dataframe['title'].head(25)
```

```
0          EMS: BACK PAINS/INJURY
1         EMS: DIABETIC EMERGENCY
2             Fire: GAS-ODOR/LEAK
3          EMS: CARDIAC EMERGENCY
4                  EMS: DIZZINESS
5                EMS: HEAD INJURY
6            EMS: NAUSEA/VOMITING
7      EMS: RESPIRATORY EMERGENCY
8           EMS: SYNCOPAL EPISODE
9     Traffic: VEHICLE ACCIDENT -
10    Traffic: VEHICLE ACCIDENT -
11    Traffic: VEHICLE ACCIDENT -
12    Traffic: VEHICLE ACCIDENT -
13    Traffic: VEHICLE ACCIDENT -
14    Traffic: VEHICLE ACCIDENT -
15    Traffic: VEHICLE ACCIDENT -
16     EMS: RESPIRATORY EMERGENCY
17                 EMS: DIZZINESS
18          EMS: VEHICLE ACCIDENT
19    Traffic: DISABLED VEHICLE -
20    Traffic: VEHICLE ACCIDENT -
21    Traffic: DISABLED VEHICLE -
22           Fire: APPLIANCE FIRE
23    Traffic: DISABLED VEHICLE -
24    Traffic: VEHICLE ACCIDENT -
Name: title, dtype: object
```

- Printing the dataframe

```
[ ] dataframe
```

	lat	lng	desc	zip	title	timeStamp	twp	addr	e
0	40.297876	-75.581294	REINDEER CT & DEAD END; NEW HANOVER; Station ...	19525.0	EMS: BACK PAINS/INJURY	2015-12-10 17:10:52	NEW HANOVER	REINDEER CT & DEAD END	1
1	40.258061	-75.264680	BRIAR PATH & WHITEMARSH LN; HATFIELD TOWNSHIP...	19446.0	EMS: DIABETIC EMERGENCY	2015-12-10 17:29:21	HATFIELD TOWNSHIP	BRIAR PATH & WHITEMARSH LN	1
2	40.121182	-75.351975	HAWS AVE; NORRISTOWN; 2015-12-10 @ 14:39:21-St...	19401.0	Fire: GAS-ODOR/LEAK	2015-12-10 14:39:21	NORRISTOWN	HAWS AVE	1
3	40.116153	-75.343513	AIRY ST & SWEDE ST; NORRISTOWN; Station 308A;...	19401.0	EMS: CARDIAC EMERGENCY	2015-12-10 16:47:36	NORRISTOWN	AIRY ST & SWEDE ST	1
4	40.251492	-75.603350	CHERRYWOOD CT & DEAD END; LOWER POTTSGROVE; S...	NaN	EMS: DIZZINESS	2015-12-10 16:56:52	LOWER POTTSGROVE	CHERRYWOOD CT & DEAD END	1
...	...	...	...	...	...	...	...	...	...
663517	40.157956	-75.348060	SUNSET AVE & WOODLAND AVE; EAST NORRITON; 2020...	19403.0	Traffic: VEHICLE ACCIDENT -	2020-07-29 15:46:51	EAST NORRITON	SUNSET AVE & WOODLAND AVE	1
663518	40.136306	-75.428697	EAGLEVILLE RD & BUNTING CIR; LOWER PROVIDENCE...	19403.0	EMS: GENERAL WEAKNESS	2020-07-29 15:52:19	LOWER PROVIDENCE	EAGLEVILLE RD & BUNTING CIR	1
663519	40.013779	-75.300835	HAVERFORD STATION RD; LOWER MERION; Station 3...	19041.0	EMS: VEHICLE ACCIDENT	2020-07-29 15:52:52	LOWER MERION	HAVERFORD STATION RD	1
663520	40.121603	-75.351437	MARSHALL ST & HAWS AVE; NORRISTOWN; 2020-07-29...	19401.0	Fire: BUILDING FIRE	2020-07-29 15:54:08	NORRISTOWN	MARSHALL ST & HAWS AVE	1
663521	40.015046	-75.299674	HAVERFORD STATION RD & W MONTGOMERY AVE; LOWER...	19041.0	Traffic: VEHICLE ACCIDENT -	2020-07-29 15:52:46	LOWER MERION	HAVERFORD STATION RD & W MONTGOMERY AVE	1

663522 rows × 9 columns

```
[ ] dataframe.shape

    (663522, 9)
```

DATA PREPROCESSING

```
dataframe.isna().sum()
```

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

- Dropping all the null values

```
[ ] df=dataframe.dropna()
```

- converts the 'timeStamp' column to a string type and then splits it into separate 'Date' and 'Time' columns based on the space character

```
df['timeStamp'] = df['timeStamp'].astype(str)

df[['Date','Time']]=df['timeStamp'].str.split(' ', expand=True)
```

```
[19] df
```

	lat	lng	desc	zip	title	timeStamp	twp	addr	e	Date	Time
0	40.297876	-75.581294	REINDEER CT & DEAD END; NEW HANOVER; Station ...	19525.0	EMS: BACK PAINS/INJURY	2015-12-10 17:10:52	NEW HANOVER	REINDEER CT & DEAD END	1.0	2015-12-10	17:10:52
1	40.258061	-75.264680	BRIAR PATH & WHITEMARSH LN; HATFIELD TOWNSHIP...	19446.0	EMS: DIABETIC EMERGENCY	2015-12-10 17:29:21	HATFIELD TOWNSHIP	BRIAR PATH & WHITEMARSH LN	1.0	2015-12-10	17:29:21
2	40.121182	-75.351975	HAWS AVE; NORRISTOWN; 2015-12-10 @ 14:39:21-St...	19401.0	Fire: GAS-ODOR/LEAK	2015-12-10 14:39:21	NORRISTOWN	HAWS AVE	1.0	2015-12-10	14:39:21
3	40.116153	-75.343513	AIRY ST & SWEDE ST; NORRISTOWN; Station 308A;...	19401.0	EMS: CARDIAC EMERGENCY	2015-12-10 16:47:36	NORRISTOWN	AIRY ST & SWEDE ST	1.0	2015-12-10	16:47:36
5	40.253473	-75.283245	CANNON AVE & W 9TH ST; LANSDALE; Station 345;...	19446.0	EMS: HEAD INJURY	2015-12-10 15:39:04	LANSDALE	CANNON AVE & W 9TH ST	1.0	2015-12-10	15:39:04
...	...	...	...	...	...	...	...	...	...	...	...
33968	40.376450	-75.481847	5TH ST & BITTING ALY; RED HILL; Station 369; ...	18076.0	EMS: CARDIAC EMERGENCY	2016-03-05 01:58:13	RED HILL	5TH ST & BITTING ALY	1.0	2016-03-05	01:58:13
33969	40.093076	-75.161915	LIMEKILN PIKE & W CHURCH RD; CHELTENHAM; 2016-...	19038.0	Traffic: DISABLED VEHICLE -	2016-03-05 02:03:00	CHELTENHAM	LIMEKILN PIKE & W CHURCH RD	1.0	2016-03-05	02:03:00
33970	40.118372	-75.351878	MAIN ST & GEORGE ST; NORRISTOWN; 2016-03-05 @ ...	19401.0	Traffic: DISABLED VEHICLE -	2016-03-05 02:06:45	NORRISTOWN	MAIN ST & GEORGE ST	1.0	2016-03-05	02:06:45
33971	40.276304	-75.565024	SWAMP PIKE & SANATOGA RD; NEW HANOVER; 2016-03...	19525.0	Traffic: VEHICLE ACCIDENT -	2016-03-05 02:19:08	NEW HANOVER	SWAMP PIKE & SANATOGA RD	1.0	2016-03-05	02:19:08
33972	40.252800	-75.713390	BENJAMIN FRANKLIN HWY & DOUGLASS DR; BERKS CO...	19518.0	EMS: UNCONSCIOUS SUBJECT	2016-03-05 02:25:56	BERKS COUNTY	BENJAMIN FRANKLIN HWY & DOUGLASS DR	1.0	2016-03-05	02:25:56

29714 rows × 11 columns

- dropping the entire column ‘e’ which has dummy values

```
[20] df = df.drop(['e'],axis=1)
```

```
[21] df.isna().sum()

lat          0
lng          0
desc         0
zip          0
title        0
timeStamp    0
twp          0
addr         0
Date         0
Time         0
dtype: int64
```

Shows that the dataset has no null values and the column 'e' has been dropped out.

- Dropping the missing values from the DataFrame 'df' and subsequently counts the non-null values for each column.

```
[22] df.dropna(inplace=True)
     df.count()

lat          29714
lng          29714
desc         29714
zip          29714
title        29714
timeStamp    29714
twp          29714
addr         29714
Date         29714
Time         29714
dtype: int64
```

- the 'desc' column in the DataFrame 'df' is split on the string 'Station', and the resulting second part is further split on ';' to extract the first part.

```
[23] df['desc'].str.split('Station', expand=True)[1].str.split(';', expand=True)[0]

0           332
1           345
2        :STA27
3          308A
5           345
          ...
33968       369
33969      None
33970      None
33971      None
33972       EMS
Name: 0, Length: 29714, dtype: object
```

- Extracting the substring after the second occurrence of ';' in the 'desc' column, then extracts the text after 'Station' and assigns it to the 'Station_num' column in the DataFrame 'df'

```
[24] #station from description
     df_station = pd.DataFrame()
     df_station = df['desc'].str.split(';', expand=False)
     df_station

     df_station = df_station.str[2]
     df_station = df_station.str.extract(rf'Station\s+(.+)')
     df_station

     df['Station_num'] = df_station
```

- Checking for null values in the dataframe

```
[26] df.isna().sum()

lat                0
lng                0
desc               0
zip                0
title              0
timeStamp          0
twp                0
addr               0
Date               0
Time               0
Station_num    15055
dtype: int64
```

- Creating a subset of the data frame that contains only the records of all EMS calls

```
[28] dataframe_Ems = df
     dataframe_Ems = dataframe_Ems.dropna()
     dataframe_Ems
```

	lat	lng	desc	zip	title	timeStamp	twp	addr	Date	Time	Station_num
0	40.297876	-75.581294	REINDEER CT & DEAD END; NEW HANOVER; Station ...	19525.0	EMS: BACK PAINS/INJURY	2015-12-10 17:10:52	NEW HANOVER	REINDEER CT & DEAD END	2015-12-10	17:10:52	332
1	40.258061	-75.264680	BRIAR PATH & WHITEMARSH LN; HATFIELD TOWNSHIP...	19446.0	EMS: DIABETIC EMERGENCY	2015-12-10 17:29:21	HATFIELD TOWNSHIP	BRIAR PATH & WHITEMARSH LN	2015-12-10	17:29:21	345
3	40.116153	-75.343513	AIRY ST & SWEDE ST; NORRISTOWN; Station 308A;...	19401.0	EMS: CARDIAC EMERGENCY	2015-12-10 16:47:36	NORRISTOWN	AIRY ST & SWEDE ST	2015-12-10	16:47:36	308A
5	40.253473	-75.283245	CANNON AVE & W 9TH ST; LANSDALE; Station 345;...	19446.0	EMS: HEAD INJURY	2015-12-10 15:39:04	LANSDALE	CANNON AVE & W 9TH ST	2015-12-10	15:39:04	345
6	40.182111	-75.127795	LAUREL AVE & OAKDALE AVE; HORSHAM; Station 35...	19044.0	EMS: NAUSEA/VOMITING	2015-12-10 16:46:48	HORSHAM	LAUREL AVE & OAKDALE AVE	2015-12-10	16:46:48	352
...	...	...	...	...	...	...	...	...	...	...	...
33965	40.205310	-75.174410	HORSHAM RD & KEITH VALLEY RD; HORSHAM; Statio...	19002.0	EMS: SUBJECT IN PAIN	2016-03-05 01:24:51	HORSHAM	HORSHAM RD & KEITH VALLEY RD	2016-03-05	01:24:51	352
33966	40.244636	-75.642183	HIGH ST & S WASHINGTON ST; POTTSTOWN; Station...	19464.0	EMS: ABDOMINAL PAINS	2016-03-05 01:27:29	POTTSTOWN	HIGH ST & S WASHINGTON ST	2016-03-05	01:27:29	329
33967	40.245673	-75.279667	6TH ST & N BROAD ST; LANSDALE; Station 345; 2...	19446.0	EMS: DIZZINESS	2016-03-05 01:26:12	LANSDALE	6TH ST & N BROAD ST	2016-03-05	01:26:12	345
33968	40.376450	-75.481847	5TH ST & BITTING ALY; RED HILL; Station 369; ...	18076.0	EMS: CARDIAC EMERGENCY	2016-03-05 01:58:13	RED HILL	5TH ST & BITTING ALY	2016-03-05	01:58:13	369
33972	40.252800	-75.713390	BENJAMIN FRANKLIN HWY & DOUGLASS DR; BERKS CO...	19518.0	EMS: UNCONSCIOUS SUBJECT	2016-03-05 02:25:56	BERKS COUNTY	BENJAMIN FRANKLIN HWY & DOUGLASS DR	2016-03-05	02:25:56	EMS

14659 rows × 11 columns

- Creating a subset of the data frame that contains only the records of all non-EMS calls

```
dataframe_nems = df
dataframe_nems = dataframe_nems[dataframe_nems['Station_num'].isnull() == True]

dataframe_nems
```

	lat	lng	desc	zip	title	timeStamp	twp	addr	Date	Time	Station_num
2	40.121182	-75.351975	HAWS AVE; NORRISTOWN; 2015-12-10 @ 14:39:21-St...	19401.0	Fire: GAS-ODOR/LEAK	2015-12-10 14:39:21	NORRISTOWN	HAWS AVE	2015-12-10	14:39:21	NaN
9	40.102398	-75.291458	BLUEROUTE & RAMP I476 NB TO CHEMICAL RD; PLYM...	19462.0	Traffic: VEHICLE ACCIDENT -	2015-12-10 17:35:41	PLYMOUTH	BLUEROUTE & RAMP I476 NB TO CHEMICAL RD	2015-12-10	17:35:41	NaN
11	40.084161	-75.308386	BROOK RD & COLWELL LN; PLYMOUTH; 2015-12-10 @ ...	19428.0	Traffic: VEHICLE ACCIDENT -	2015-12-10 16:32:10	PLYMOUTH	BROOK RD & COLWELL LN	2015-12-10	16:32:10	NaN
12	40.174131	-75.098491	BYBERRY AVE & S WARMINSTER RD; UPPER MORELAND;...	19040.0	Traffic: VEHICLE ACCIDENT -	2015-12-10 17:15:49	UPPER MORELAND	BYBERRY AVE & S WARMINSTER RD	2015-12-10	17:15:49	NaN
13	40.062974	-75.135914	OLD YORK RD & VALLEY RD; CHELTENHAM; 2015-12-1...	19027.0	Traffic: VEHICLE ACCIDENT -	2015-12-10 17:12:47	CHELTENHAM	OLD YORK RD & VALLEY RD	2015-12-10	17:12:47	NaN
...	...	...	...	...	...	...	...	...	...	...	...
33961	40.115249	-75.341379	DEKALB ST & E AIRY ST; NORRISTOWN; 2016-03-05 ...	19401.0	Traffic: VEHICLE ACCIDENT -	2016-03-05 01:15:49	NORRISTOWN	DEKALB ST & E AIRY ST	2016-03-05	01:15:49	NaN
33963	40.144953	-75.116750	EASTON RD & PARK AVE; UPPER MORELAND; 2016-03-...	19090.0	Traffic: DISABLED VEHICLE -	2016-03-05 01:20:01	UPPER MORELAND	EASTON RD & PARK AVE	2016-03-05	01:20:01	NaN
33969	40.093076	-75.161915	LIMEKILN PIKE & W CHURCH RD; CHELTENHAM; 2016-...	19038.0	Traffic: DISABLED VEHICLE -	2016-03-05 02:03:00	CHELTENHAM	LIMEKILN PIKE & W CHURCH RD	2016-03-05	02:03:00	NaN
33970	40.118372	-75.351878	MAIN ST & GEORGE ST; NORRISTOWN; 2016-03-05 @ ...	19401.0	Traffic: DISABLED VEHICLE -	2016-03-05 02:06:45	NORRISTOWN	MAIN ST & GEORGE ST	2016-03-05	02:06:45	NaN
33971	40.276304	-75.565024	SWAMP PIKE & SANATOGA RD; NEW HANOVER; 2016-03...	19525.0	Traffic: VEHICLE ACCIDENT -	2016-03-05 02:19:08	NEW HANOVER	SWAMP PIKE & SANATOGA RD	2016-03-05	02:19:08	NaN

15055 rows × 11 columns

- Converting given field to pandas timestamp format.
- Selecting hours using dt.hour from the timeStamp data field.
- Storing all the hours in a number format in a new column called 'Timing'

```
df['timeStamp']= pd.to_datetime(df['timeStamp'])
df['Timing'] = df['timeStamp'].dt.hour
```

- Creating a data map, containing string values corresponding to the time of call to categorize the data.

```
df['timeStamp']= pd.to_datetime(df['timeStamp'])
time = df['timeStamp'].iloc[0]
time.hour
df['Timing'] = df['timeStamp'].apply(lambda time: time.hour)
```

- Maping and replacing the values in the 'Timings' column.

```
df['Timing'] = df['Timing'].map(dmap)
```

- Storing data about hour, day and month when the call took place, by extracting the corresponding values using dt.hour,dt.month, dt.weekday.

```
[32] df['Hour'] = df.timeStamp.dt.hour
     df['Month'] = df.timeStamp.dt.month
     df['DayOfWeek'] = df.timeStamp.dt.weekday
```

- Here, the records stored in the title column are used to extract the category of call (EMS, Fire, Traffic) into a separate field labelled Call_Category.
- The extended reason such as head injury, gas leak or accident is separately stored in another field labelled Call_Reason.

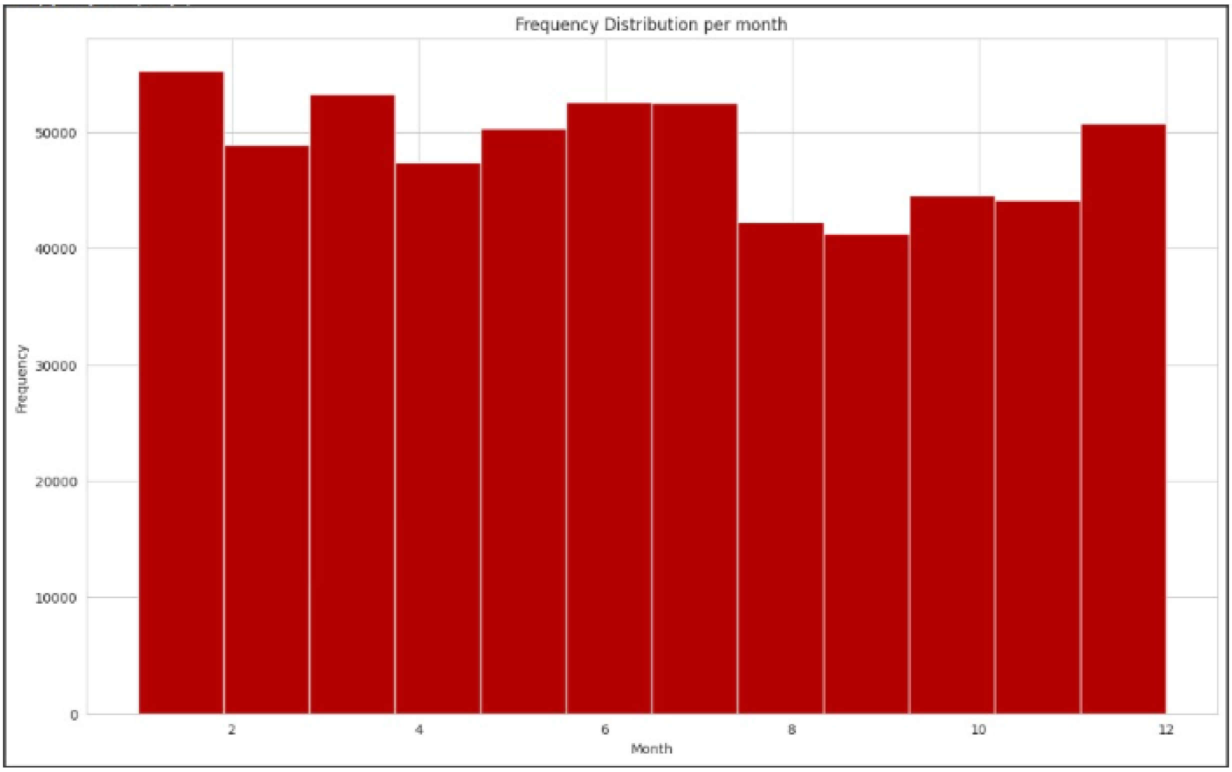
```
df['Call_Category'] = df.title.str.split(':', expand=True)[0]
df['Call_Reason'] = df.title.str.split(':', expand=True)[1].str.replace(' -', '')
```

DATA VISUALIZATION

- Histogram to indicate the number of calls made each month

```
plt.figure(figsize=(15,9))

plt.hist(df.Month,bins=12,color="#b30000")
plt.show()
```



The maximum number of calls were made in January followed by March and then June & July

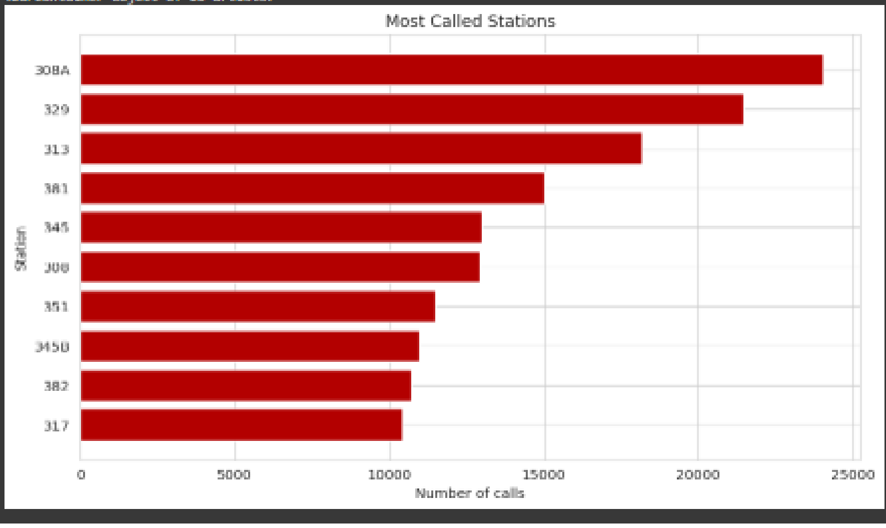
- Most Called Stations due to EMS reasons

```
most_called_ems = dataframe_Ems.Station_num.str.replace(':', '').value_counts()[:10]
plt.figure(figsize=(10, 6))
x = list(most_called_ems.index)
y = list(most_called_ems.values)
x.reverse()
y.reverse()

plt.title("Most Called Stations")
plt.ylabel("Station")
plt.xlabel("Number of calls")

plt.barh(x, y, color="#b30000")
```

```
<BarContainer object of 10 artists>
```



The most called station number is 308A.

- Top 10 Townships who have called 911

```
dataframe.twp.value_counts().head(10)
```

```
LOWER MERION         55490
ABINGTON             39947
NORRISTOWN           37633
UPPER MERION         36010
CHELTENHAM           30574
POTTSTOWN            27387
UPPER MORELAND       22932
LOWER PROVIDENCE     22476
PLYMOUTH             20116
UPPER DUBLIN         18862
Name: twp, dtype: int64
```

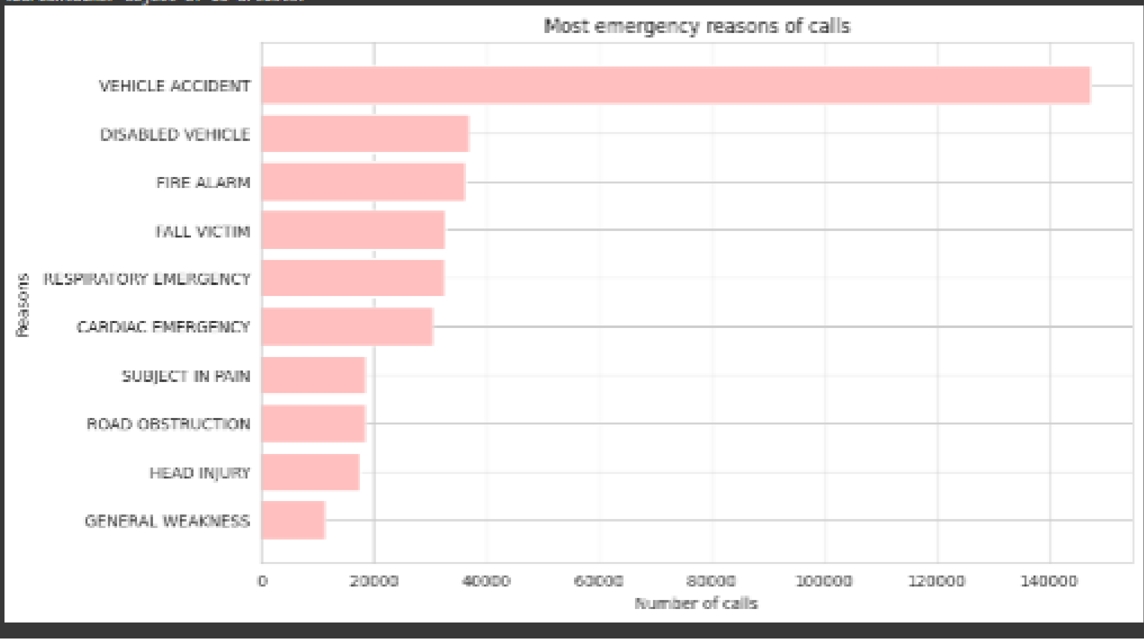
- Most number calls made with respect to reasons

```
most_reasons_call = df.Call_Reason.value_counts()[:10]
plt.figure(figsize=(10,6))
x = list(most_reasons_call.index)
y = list(most_reasons_call.values)
x.reverse()
y.reverse()

plt.title("Most emergency reasons of calls")
plt.ylabel("Reasons")
plt.xlabel("Number of calls")

plt.barh(x, y, color="#ffbfbf")
```

```
<BarContainer object of 10 artists>
```



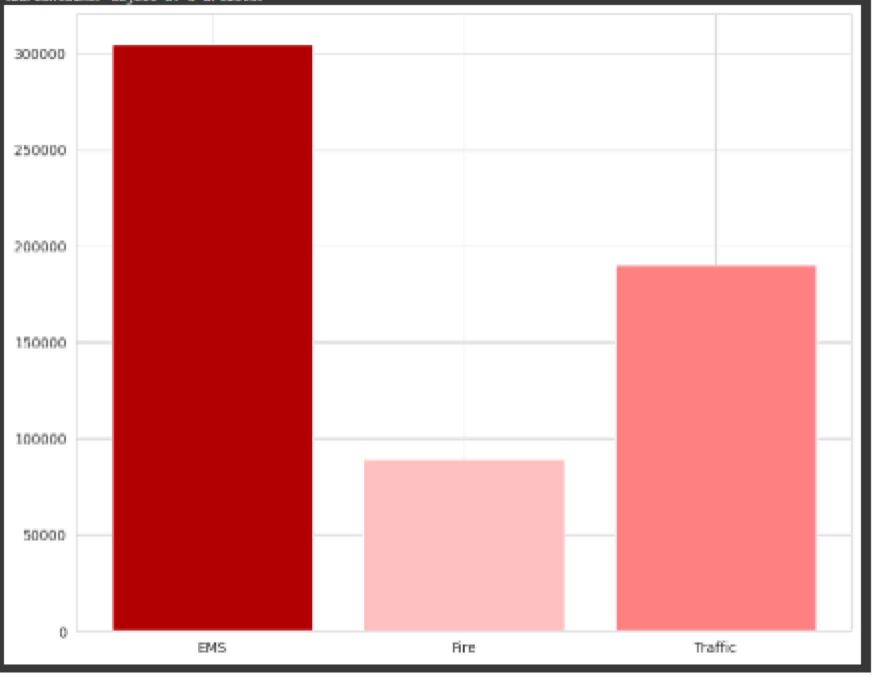
The most number of emergency calls are made because of Vehicle Accidents

- Bar Plot indicating the frequency of calls made per category

```
plt.figure(figsize = (10,8))
cs=["#b30000","#ffbfbf","#ff8080"]
listOfCategories = df['Call_Category'].unique()
listOfCountByCategory = []
for i in listOfCategories:
  listOfCountByCategory.append(df['Call_Category'].value_counts()[i])

plt.bar(listOfCategories,listOfCountByCategory, color=cs)
```

```
<BarContainer object of 3 artists>
```



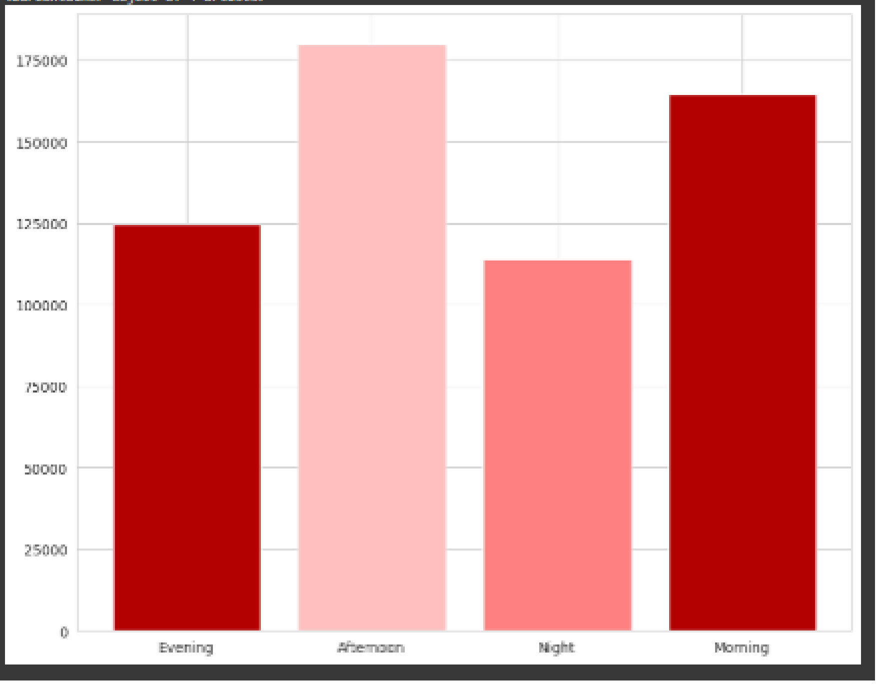
The most number of 911 calls are made because of EMS followed by traffic and Fire reasons.

- Bar Plot indicating frequency of calls by time of the day

```python
plt.figure(figsize = (10,8))
cs=["#b30000","#ffbfbf","#ff8080"]
listOfHours = df['Timing'].unique()
listOfCountByHours = []
for i in listOfHours:
  listOfCountByHours.append(df['Timing'].value_counts()[i])

plt.bar(listOfHours,listOfCountByHours, color=cs)
```

```
<BarContainer object of 4 artists>
```

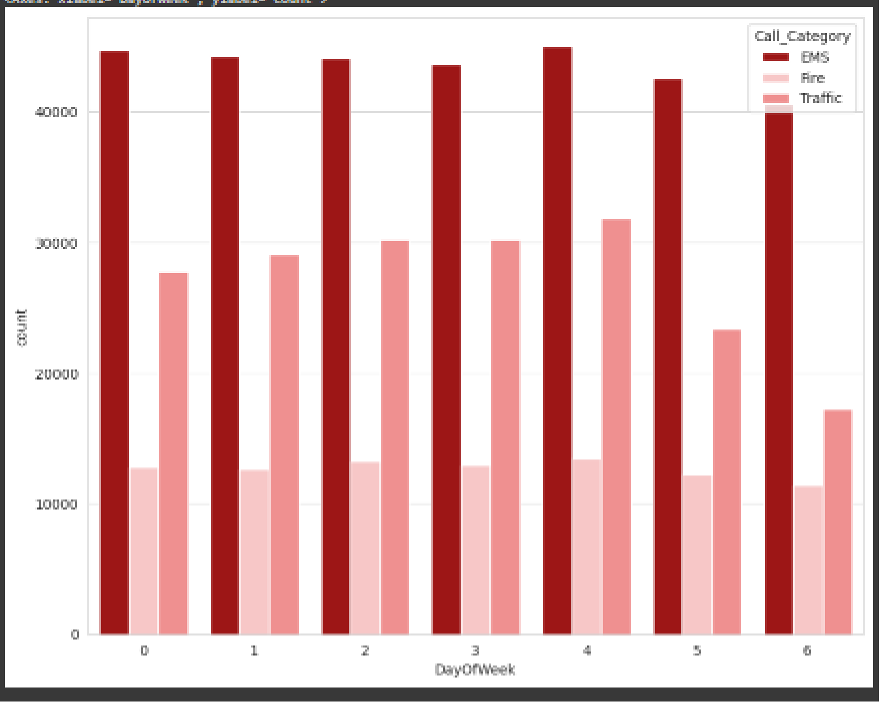


The most number of calls were made in the Afternoon

- Bar Plot for number of calls made on each day of the week

```python
cs=["#b30000","#ffbfbf","#ff8080"]
plt.figure(figsize=(10, 8))
sns.countplot(x=df.DayOfWeek, hue=df.Call_Category,palette=cs)
```

```
<Axes: xlabel='DayOfWeek', ylabel='count'>
```



- Grouping the count of occurrences for each group and storing in the 'byWeek' DataFrame

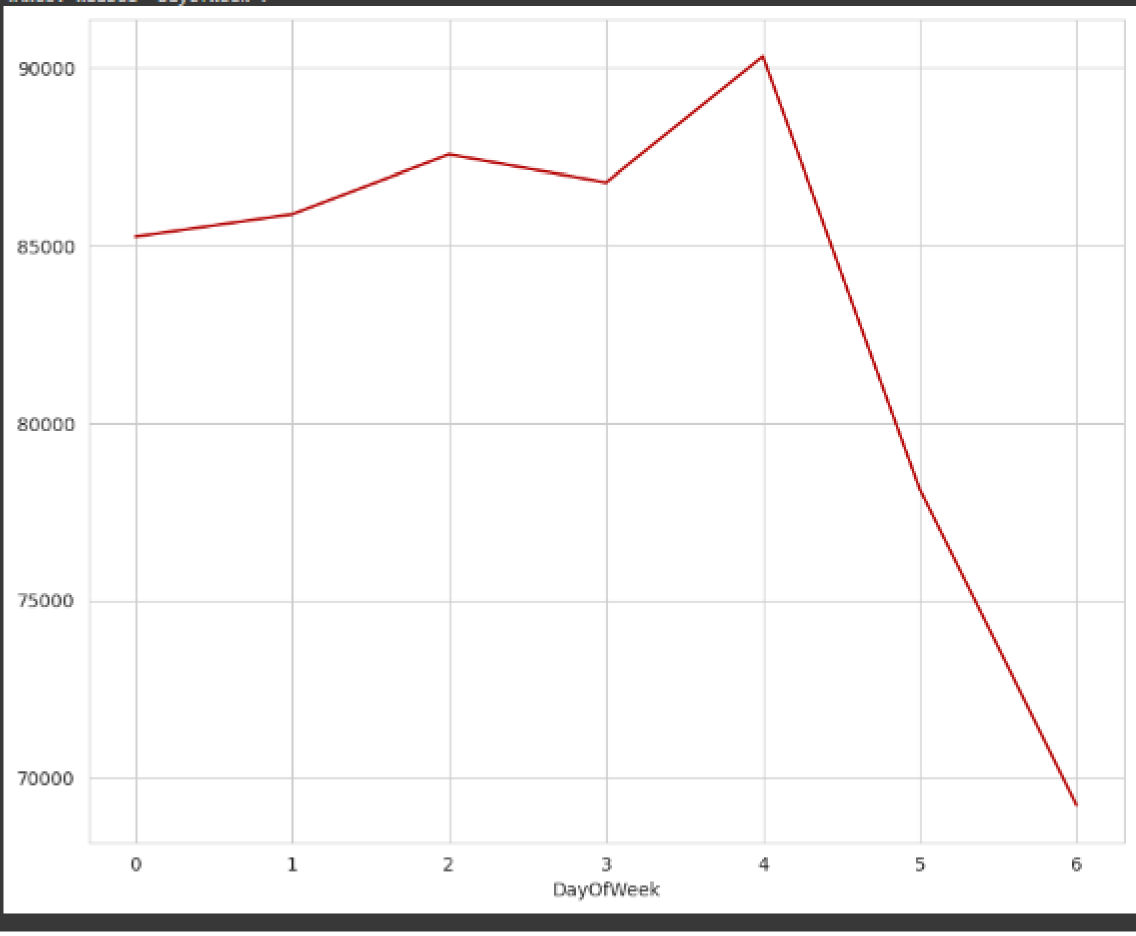
```
[ ] byWeek = df.groupby('DayOfWeek').count()
    byWeek
```

DayOfWeek	lat	lng	desc	zip	title	timeStamp	twp	addr	Date	Time	Station_num	Timing	Hour	Month	Call_Category	Call_Reason
0	85260	85260	85260	85260	85260	85260	85260	85260	85260	85260	44700	85260	85260	85260	85260	85260
1	85890	85890	85890	85890	85890	85890	85890	85890	85890	85890	44237	85890	85890	85890	85890	85890
2	87563	87563	87563	87563	87563	87563	87563	87563	87563	87563	44074	87563	87563	87563	87563	87563
3	86780	86780	86780	86780	86780	86780	86780	86780	86780	86780	43633	86780	86780	86780	86780	86780
4	90328	90328	90328	90328	90328	90328	90328	90328	90328	90328	45019	90328	90328	90328	90328	90328
5	78144	78144	78144	78144	78144	78144	78144	78144	78144	78144	42548	78144	78144	78144	78144	78144
6	69234	69234	69234	69234	69234	69234	69234	69234	69234	69234	40574	69234	69234	69234	69234	69234

- Line plot of the 'twp' column from the 'byWeek' DataFrame

```
[ ] byWeek.twp.plot(figsize=(10, 8), color="#b30000")
```

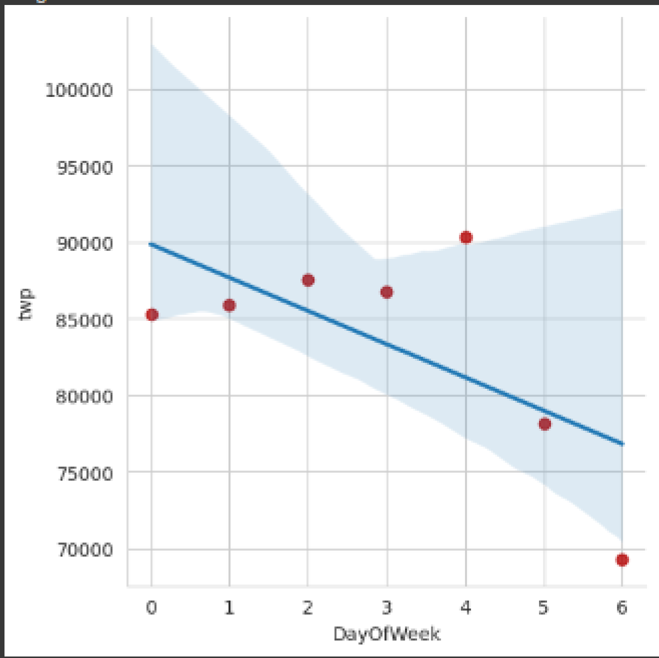
<Axes: xlabel='DayOfWeek'>



Maximum calls are made on the 5th Day of the Week

```
[ ] plt.figure(figsize=(12, 8))
    sns.lmplot(x='DayOfWeek',y='twp',data=byWeek.reset_index(), scatter_kws={'color':
    "#b30000"})
```

```
<seaborn.axisgrid.FacetGrid at 0x782894cb43d0>
<Figure size 1200x800 with 0 Axes>
```

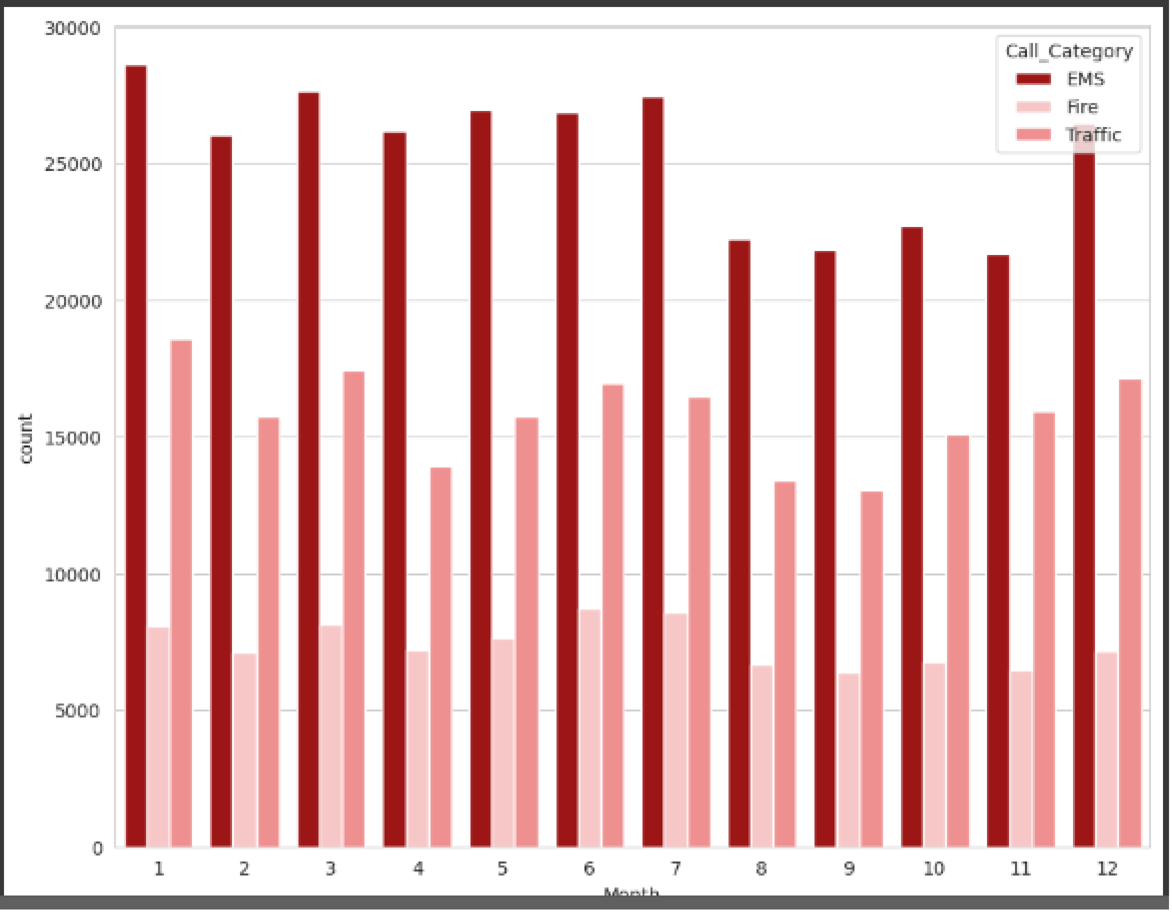


The provided code generates a scatter plot with a linear regression model (lmplot) using Seaborn. The plot visualizes the relationship between the 'DayOfWeek' and 'twp' columns from the 'byWeek' DataFrame. There is negative co-relation.

- Bar Plot for number of calls made per Month

```
cs=["#b30000","#ffbfbf","#ff8080"]
plt.figure(figsize=(10, 8))
sns.countplot(x=df.Month, hue=df.Call_Category,palette=cs)

%matplotlib inline
sns.set_style("whitegrid")
```



- Grouping the count of occurrences for each group and storing in the 'byMonth' DataFrame

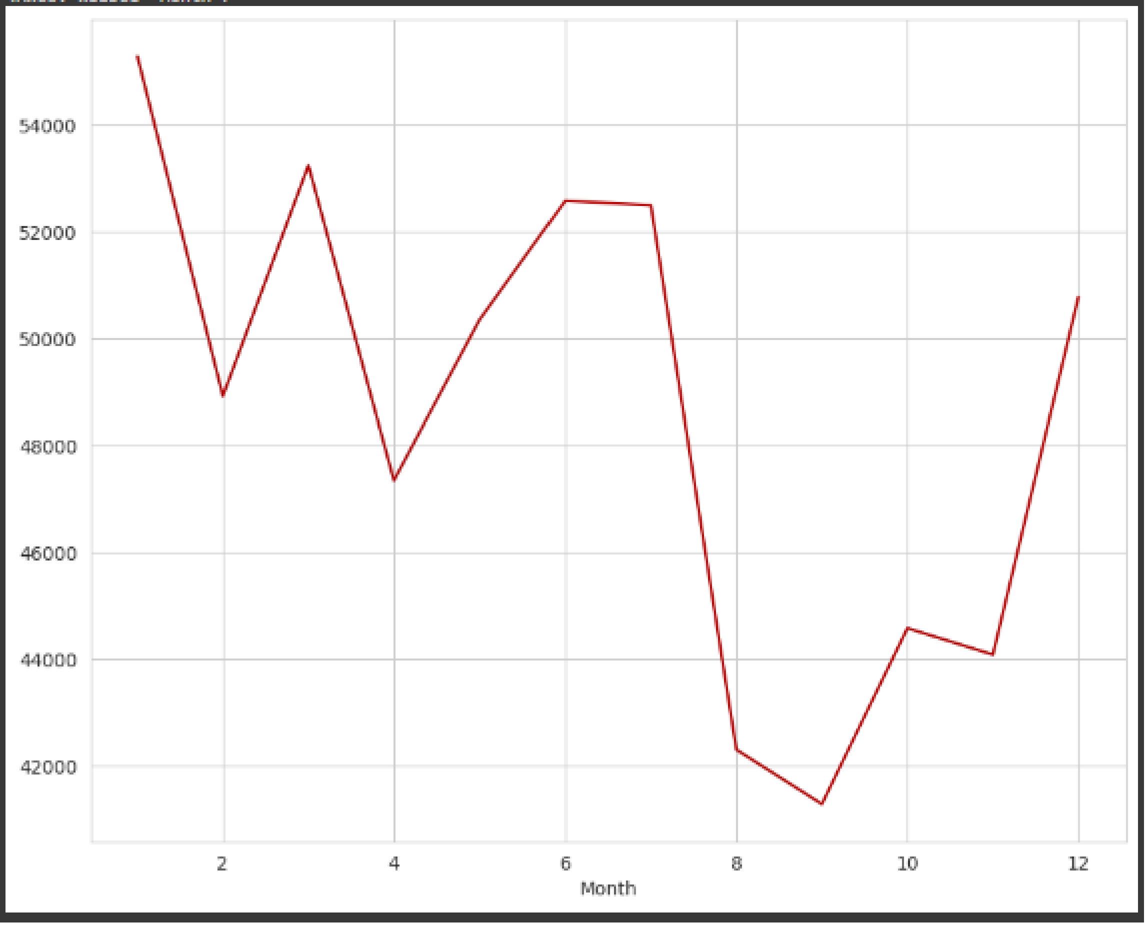
```
byMonth = df.groupby('Month').count()
byMonth
```

Month	lat	lng	desc	zip	title	timeStamp	twp	addr	Date	Time	Station_num	Timing	Hour	DayOfWeek	Call_Category	Call_Reason
1	55282	55282	55282	55282	55282	55282	55282	55282	55282	55282	28623	55282	55282	55282	55282	55282
2	48916	48916	48916	48916	48916	48916	48916	48916	48916	48916	26042	48916	48916	48916	48916	48916
3	53240	53240	53240	53240	53240	53240	53240	53240	53240	53240	27666	53240	53240	53240	53240	53240
4	47344	47344	47344	47344	47344	47344	47344	47344	47344	47344	26188	47344	47344	47344	47344	47344
5	50342	50342	50342	50342	50342	50342	50342	50342	50342	50342	26950	50342	50342	50342	50342	50342
6	52577	52577	52577	52577	52577	52577	52577	52577	52577	52577	26875	52577	52577	52577	52577	52577
7	52491	52491	52491	52491	52491	52491	52491	52491	52491	52491	27457	52491	52491	52491	52491	52491
8	42299	42299	42299	42299	42299	42299	42299	42299	42299	42299	22245	42299	42299	42299	42299	42299
9	41280	41280	41280	41280	41280	41280	41280	41280	41280	41280	21846	41280	41280	41280	41280	41280
10	44576	44576	44576	44576	44576	44576	44576	44576	44576	44576	22716	44576	44576	44576	44576	44576
11	44080	44080	44080	44080	44080	44080	44080	44080	44080	44080	21698	44080	44080	44080	44080	44080
12	50772	50772	50772	50772	50772	50772	50772	50772	50772	50772	26479	50772	50772	50772	50772	50772

- Line plot of the 'twp' column from the 'byMonth' DataFrame

```
byMonth.twp.plot(figsize=(10, 8), color="#b30000")
```

```
<Axes: xlabel='Month'>
```



There is a sudden drop in the number of emergency calls made in the month of September, a steady number of calls were made from June to July and the most of calls were made in January.

- The code groups the DataFrame 'df' by both the 'DayOfWeek' and 'Hour' columns, counts the occurrences of 'Call_Category' for each group, and aggregates the counts and .unstack() reshapes the grouped data, pivoting the inner level of the hierarchical index (Hour) to the columns and the outer level index, creating a new DataFrame 'dayHour' where the rows represent 'DayOfWeek' and the columns represent 'Hour'.

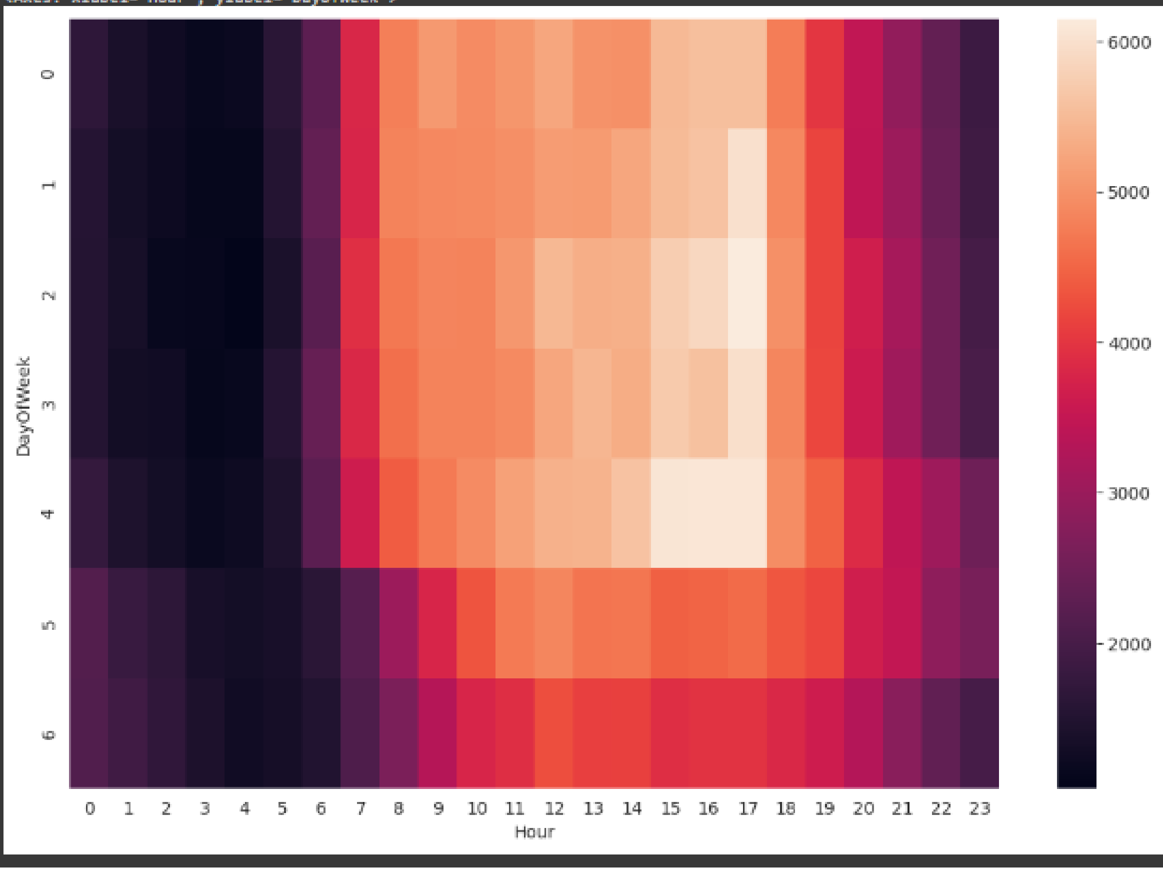
```
dayHour = df.groupby(by=['DayOfWeek','Hour']).count()['Call_Category'].unstack()
dayHour.head()
```

Hour	0	1	2	3	4	5	6	7	8	9	...	14	15	16	17	18	19	20	21	22	23
DayOfWeek																					
0	1661	1390	1238	1140	1184	1604	2265	3808	4785	5079	...	4985	5484	5561	5557	4765	3997	3460	2902	2346	1848
1	1547	1300	1202	1102	1111	1521	2350	3777	4821	4875	...	5248	5507	5611	6007	4887	4156	3445	3030	2408	1878
2	1518	1356	1150	1134	1038	1410	2218	3927	4713	4854	...	5358	5749	5880	6153	4978	4164	3661	3137	2511	1964
3	1530	1284	1276	1101	1156	1550	2394	3818	4599	4815	...	5331	5695	5577	5974	4855	4176	3609	3070	2507	2022
4	1751	1451	1311	1173	1226	1440	2248	3621	4411	4724	...	5596	6060	6088	6079	4945	4463	3856	3449	3048	2494

5 rows × 24 columns

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

```
<Axes: xlabel='Hour', ylabel='DayOfWeek'>
```



From this heatmap, we can conclude that the maximum number of calls were made in the middle of the week from 15:00hrs – 17:00hrs. The least amount of calls are made at the start of almost all days of the week. Also, as observed in the previous bar plots, our inference about the maximum calls being made during the afternoon can be verified.

CONCLUSION

In this project, we analysed the data collected in the Montgomery County, PA that contains records of emergency calls made to Montgomery County in Pennsylvania. We aimed to understand and analyse any trends present across emergency calls made in the county & aimed to visualize the data that spanned across multiple years.
We also wanted to understand what the majority of emergency calls were, pertaining to the type of emergency that are broadly classified in three categories: EMS, Fire & Traffic.

While performing, we started with an original dataset from Kaggle (sourced from www.montgomerycountypa.gov), containing 663522 records, each containing 9 attributes.

Throughout the project we carried out the following tasks:

- understanding the shape of the dataset
- understanding the nature of the attributes stored in the dataset
- indetifying any anamolies/inconcistencies in the dataset
- data cleaning such as removal of null values from the dataset
- extracting critical information about the emergency call into seperate attributes
- visualizing the a-forementioned data accurately, such that it can be used for inference and analysis

In conclusion, after performing data extraction & processing on the given dataset, we were able to gain a comprehensive understanding of the nature of the emergency calls made.

PYTHON CODE LINK

https://colab.research.google.com/drive/10G9i33NFULJnEiKyez69jsfJX-aOHpgy?usp=sharing