

The background image shows a fire station. On the left, the front of a red fire truck is visible. In the center, a fire truck is parked with its rear door open. To the right, a fire station building with a ramp and stairs is visible. Several firefighters in yellow gear are standing in the background. The scene is dimly lit, suggesting dusk or dawn.

Emergency - 911 Calls

Montgomery County, PA

Data Science Bootcamp Project

Budour Alshehri

Background

Montgomery County

is the third-most populous county in the Commonwealth of Pennsylvania. locally referred to as Montco.

Montgomery County is very diverse, ranging from farms and open land in Upper Hanover to densely populated rowhouse streets in Cheltenham.

911 Calls

is an emergency telephone number created by Congress in 2004 as the 911 Implementation and Coordination Office (ICO), the National 911 Program is housed within the National Highway Traffic Safety Administration at the U.S. Department of Transportation and is a joint program with the National Telecommunication and Information Administration in the Department of Commerce.

Data Description

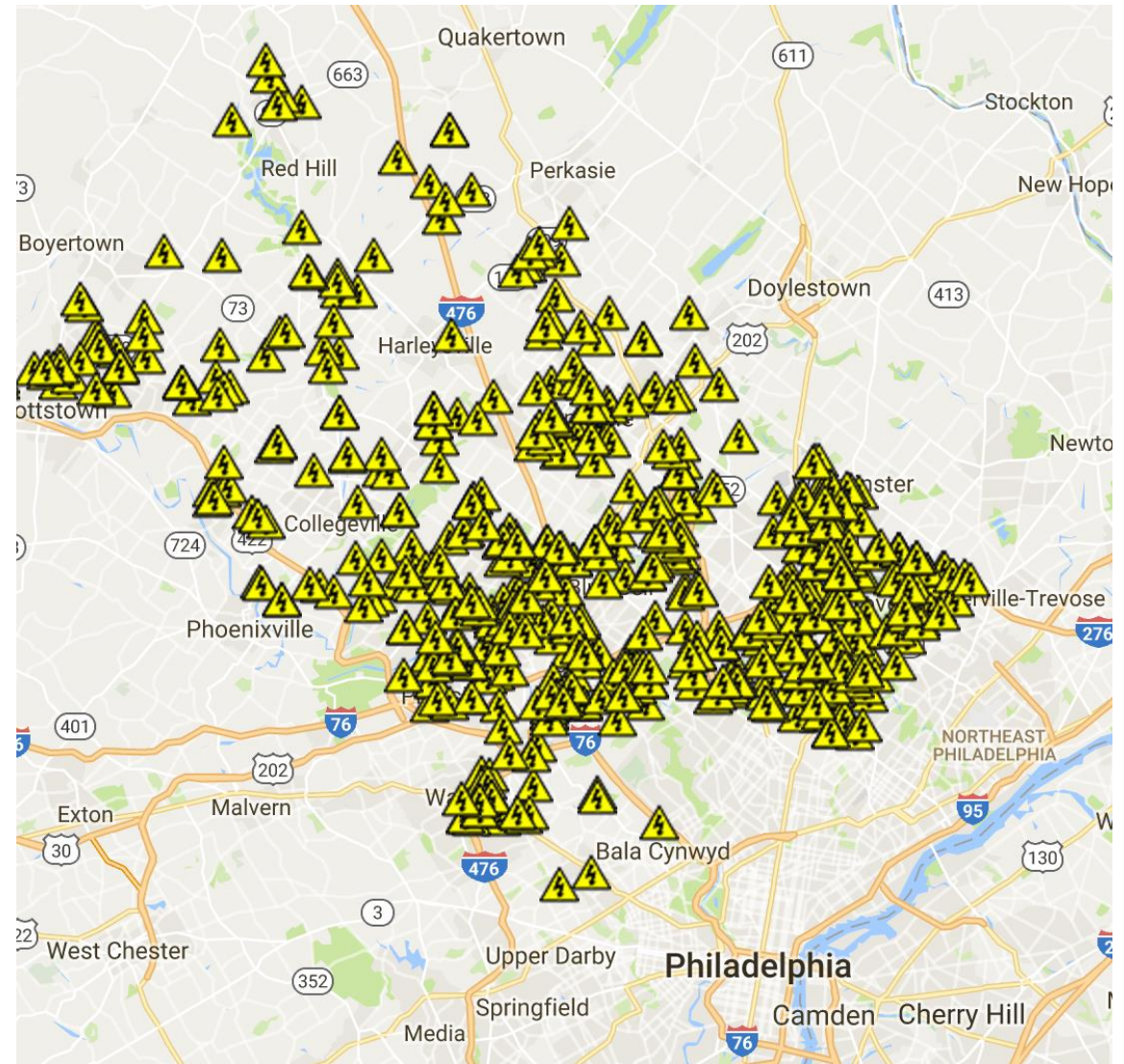
- This dataset contains emergency calls from Montgomery County, PA.
- It includes 663,522 calls records from 2015 to 2020 and 9 Features.
- Link : [Emergency - 911 Calls | Kaggle](#)

Feature Columns:

- lat: String variable, Latitude
- lng: String variable, Longitude
- desc: String variable, Description of the Emergency Call
- zip: String variable, ZIP Code
- title: String variable, Title of Emergency
- timeStamp: String variable, Date and time of the call, YYYY-MM-DD HH:MM:SS
- twp: String variable, Township
- addr: String variable, General Address
- e: String variable, Dummy variable, Index column (always 1)

Questions

- What are the top 5 zip codes for 911 calls?
- How many unique title of emergency codes are there?
- What is the most common Reason for a 911 call?
- What are the top 10 emergency calls from all the categories?
- Can we predict the type reason of the next call?





Tools



Python and Jupyter Notebook



Pandas and Numpy for Data Processing



Seaborn for Visualization



skLearn & Tensorflow for classification

Imports & read data

```
import numpy as np
import pandas as pd

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

```
df = pd.read_csv('C:/Users/BUDUR/Desktop/Data science Bootcamp/project/archive/911.csv')
```

Clean data

Dropping column e

```
In [6]: df = df.drop('e',axis=1)
```

```
In [7]: print(df.columns.values)
```

```
['lat' 'lng' 'desc' 'zip' 'title' 'timeStamp' 'twp' 'addr']
```

missing values

```
In [8]: print('Missing values:',df.isnull().values.sum())
df.isnull().sum()
```

Missing values: 80492

```
Out[8]: lat          0
        lng          0
        desc         0
        zip        80199
        title        0
        timeStamp   0
        twp         293
        addr         0
        dtype: int64
```

Type *Markdown* and LaTeX: α^2

```
In [9]: df['zip'].isnull().sum()/df.shape[0]
```

```
Out[9]: 0.12086863736243862
```

```
In [10]: df['twp'].isnull().sum()/df.shape[0]
```

```
Out[10]: 0.0004415829467598663
```

zip code contains 12% Nan


Clean data

```
df[df['twp'].isnull()]
```

	lat	lng	desc	zip	title	timeStamp	twp	addr
1635	40.162804	-75.097848	TURNPIKE OVERPASS; ; 2015-12-14 @ 21:36:52-Sta...	19040.0	Fire: VEHICLE ACCIDENT	2015-12-14 21:36:52	NaN	TURNPIKE OVERPASS
1821	40.099265	-75.175706	CHURCH RD; ; Station 322; 2015-12-15 @ 11:31:36;	NaN	EMS: UNKNOWN MEDICAL EMERGENCY	2015-12-15 11:31:36	NaN	CHURCH RD
5455	40.222272	-75.138302	GIANT; ; 2015-12-24 @ 17:30:07-Station:STA98;	18976.0	Fire: VEHICLE ACCIDENT	2015-12-24 17:30:07	NaN	GIANT
7281	40.113517	-75.332257	HIGH ST; ; Station 329; 2015-12-30 @ 03:32:49;	19401.0	EMS: VEHICLE ACCIDENT	2015-12-30 03:32:49	NaN	HIGH ST
7282	40.113517	-75.332257	HIGH ST; ; 2015-12-30 @ 03:32:28-Station:STA58;	19401.0	Fire: VEHICLE ACCIDENT	2015-12-30 03:32:28	NaN	HIGH ST
...
659226	40.229008	-75.387852	NO LOCATION - NEIGHBORING COUNTY; ; Station 3...	NaN	EMS: UNKNOWN MEDICAL EMERGENCY	2020-07-17 12:33:06	NaN	NO LOCATION - NEIGHBORING COUNTY
660100	40.157506	-75.072525	RAILS TO TRAILS CONNECTOR TRL; ; 2020-07-19 @ ...	19006.0	Fire: RESCUE - WATER	2020-07-19 21:12:52	NaN	RAILS TO TRAILS CONNECTOR TRL
660102	40.157506	-75.072525	RAILS TO TRAILS CONNECTOR TRL; ; Station 322A...	19006.0	EMS: RESCUE - WATER	2020-07-19 21:13:08	NaN	RAILS TO TRAILS CONNECTOR TRL
660154	40.229008	-75.387852	NO LOCATION - NEIGHBORING COUNTY; ; Station 3...	NaN	EMS: RESPIRATORY EMERGENCY	2020-07-20 03:34:03	NaN	NO LOCATION - NEIGHBORING COUNTY
661678	40.164183	-75.208340	ROSE; ; Station 351; 2020-07-24 @ 06:06:18;	19002.0	EMS: OVERDOSE	2020-07-24 06:06:18	NaN	ROSE

293 rows × 8 columns

Records with no townships are mostly dead ends. Lets skip them




What are the top 5 zip codes for 911 calls?

```
df_zip = pd.DataFrame(df['zip'].value_counts().head(5))  
df_zip.rename(columns = {'zip': 'Top 5'}, inplace = True)  
df_zip.style.background_gradient(cmap='Blues')
```

Top 5	
19401.0	45606
19464.0	43910
19403.0	34888
19446.0	32270
19406.0	22464

These are the top 5 zip codes for 911 calls



How many
unique title of
emergency
codes are
there?

```
df['title'].nunique()
```

```
148
```

There are 148 unique title of emergency codes

Adding new feature - Reason feature & Title_code

Reason feature

In the titles column there are "Reasons/Departments" specified before the title code. These are EMS, Fire, and Traffic.

```
] df['reason'] = df['title'].apply(lambda title: title.split(':')[0])
```

Title_code feature

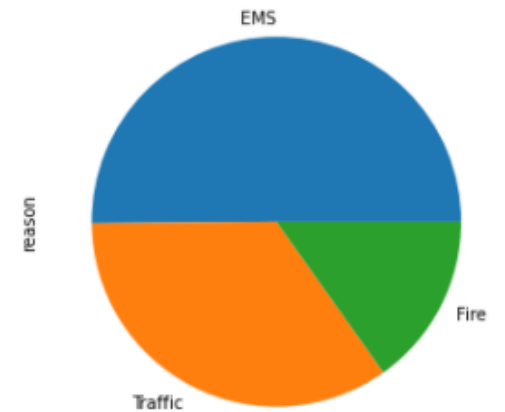
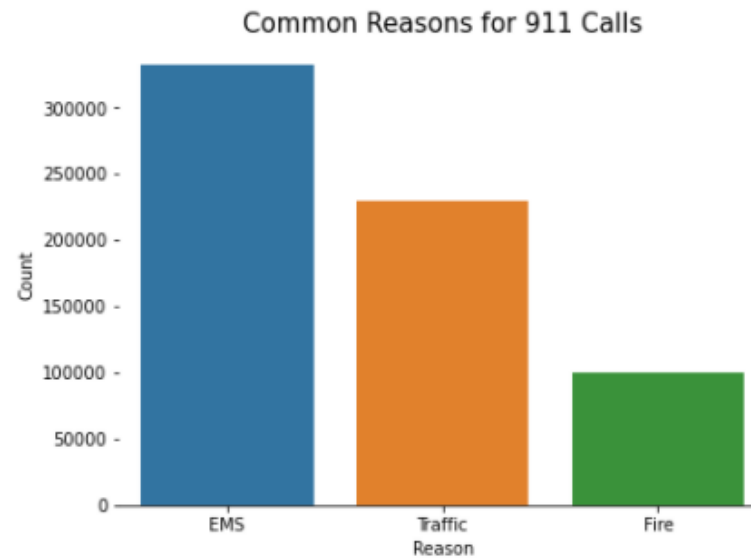
Using the same method from above, we are going to create a column with just the title code.

```
] df['title_code'] = df['title'].apply(lambda title: title.split(':')[1])
```

Exploratory Data Analysis (EDA)

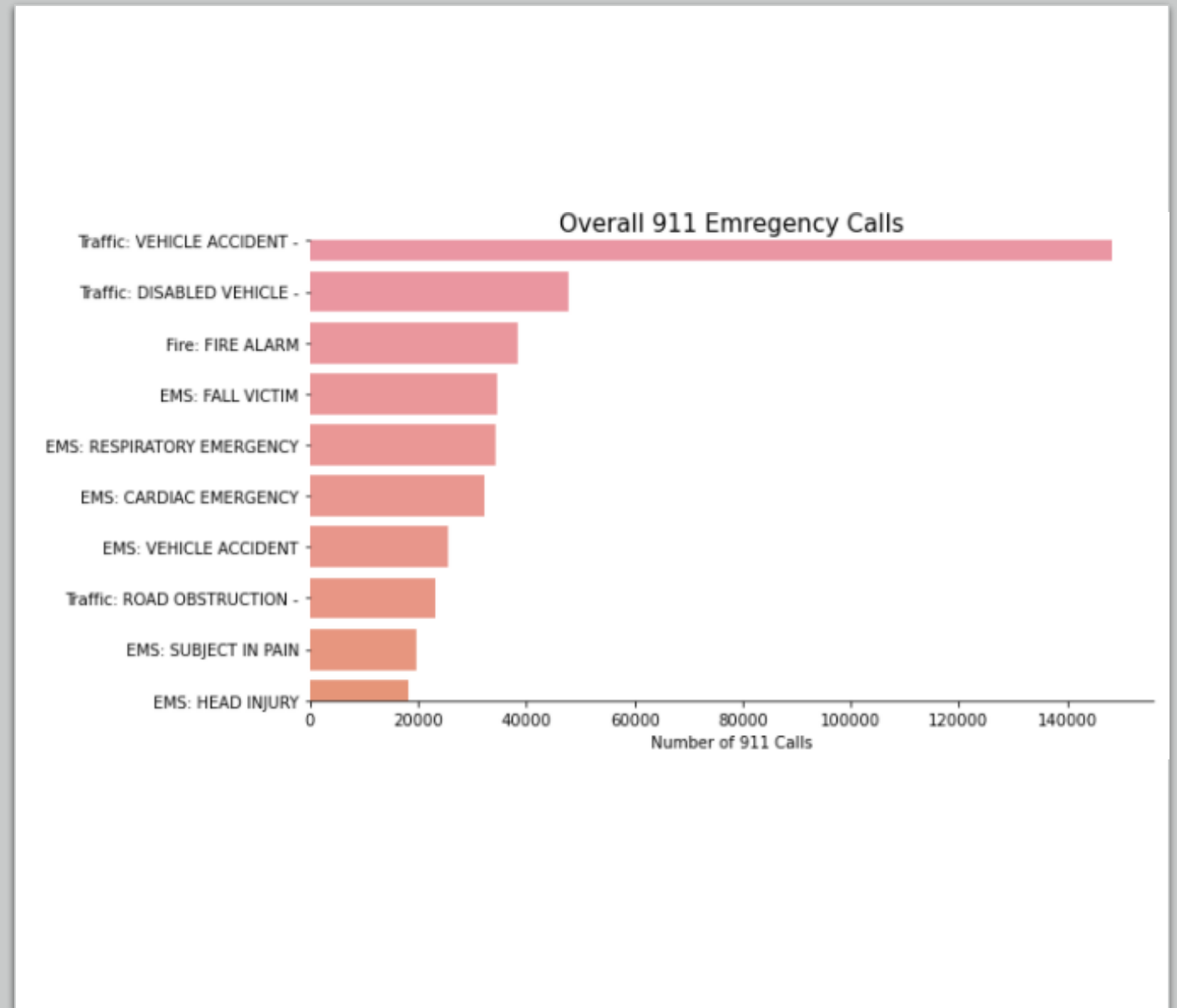
What is the most common Reason for a 911 call?

- The number one reason for 911 calls are Emergency Medical Services.
- Almost half of the reasons are for EMS.



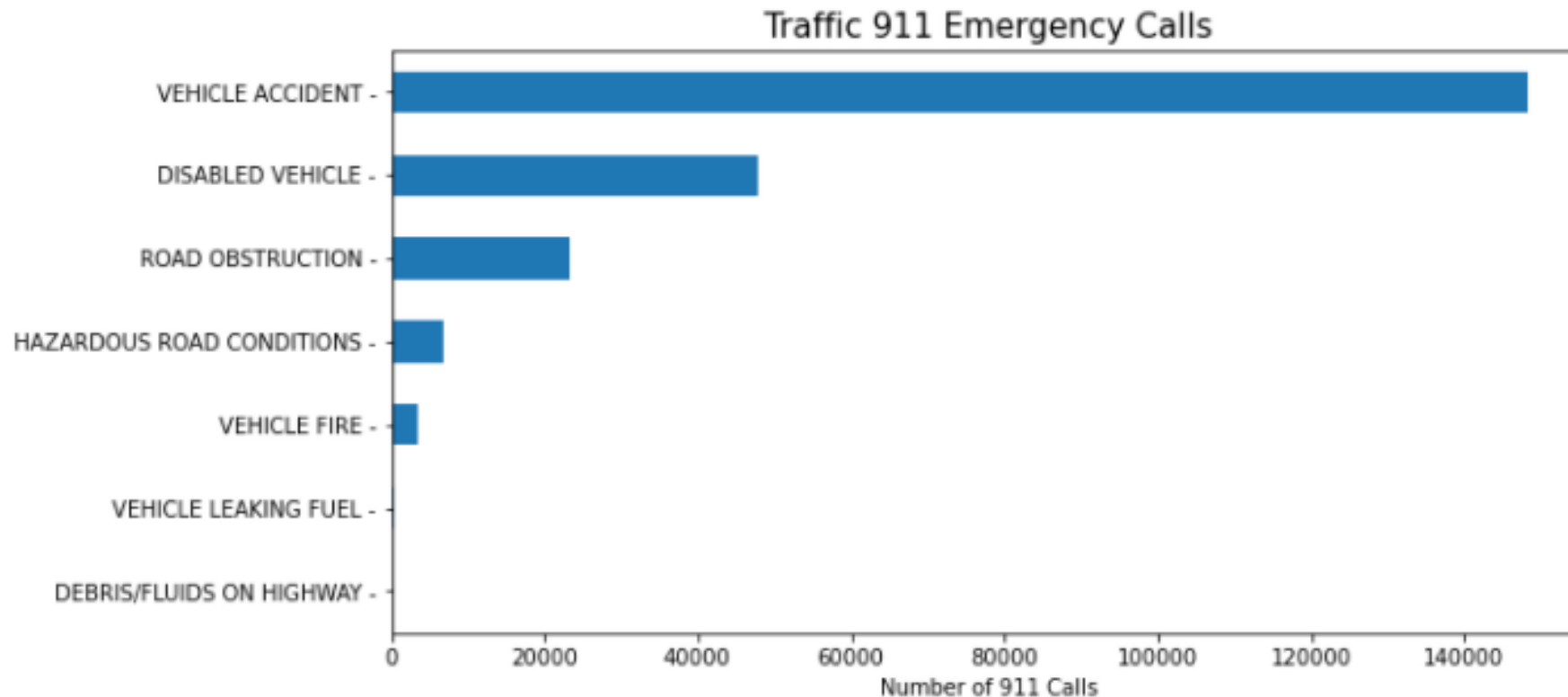
What are the top 10 emergency calls from all the categories?

- Vehicle accidents are the number one reason people call 911.
- Disabled vehicle and fire alarm are in second and third place.



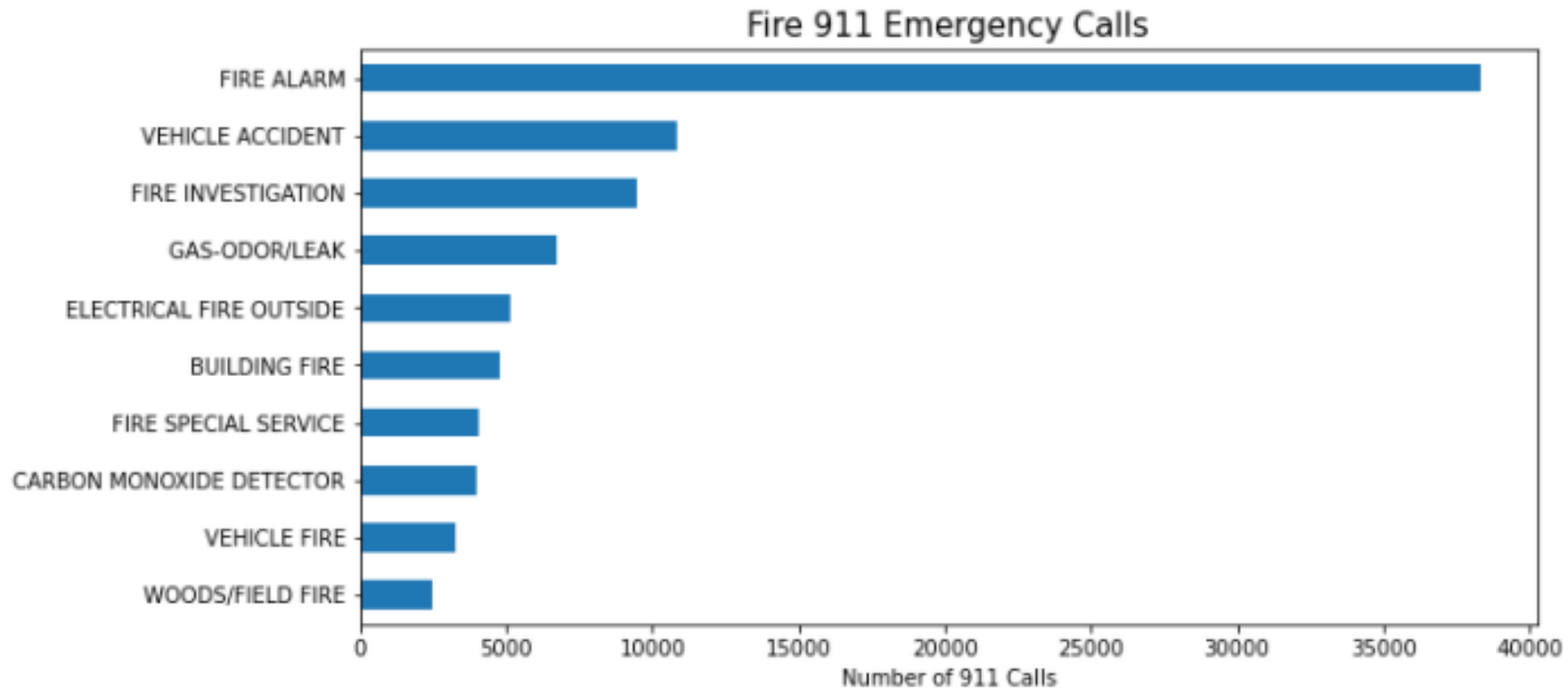
Traffic 911 Emergency Calls

- The most common emergency titles are vehicle accident, disable vehicle and road obstruction.



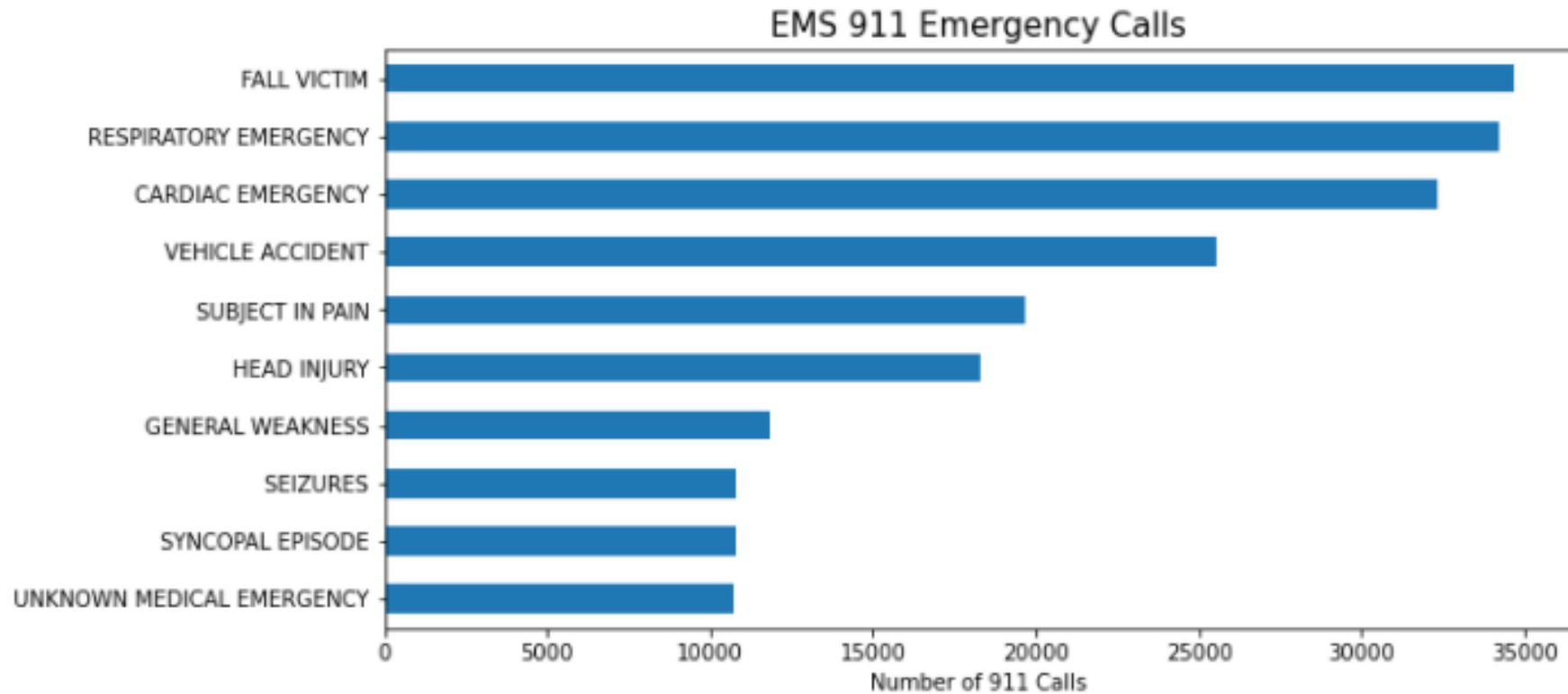
Fire 911 Emergency Calls

- The most common emergency titles are fire alarm, vehicle accident and fire investigation.



EMS 911 Emergency Calls

- The most common emergency titles are fall victim, respiratory emergency and cardiac emergency.



Feature Engineering

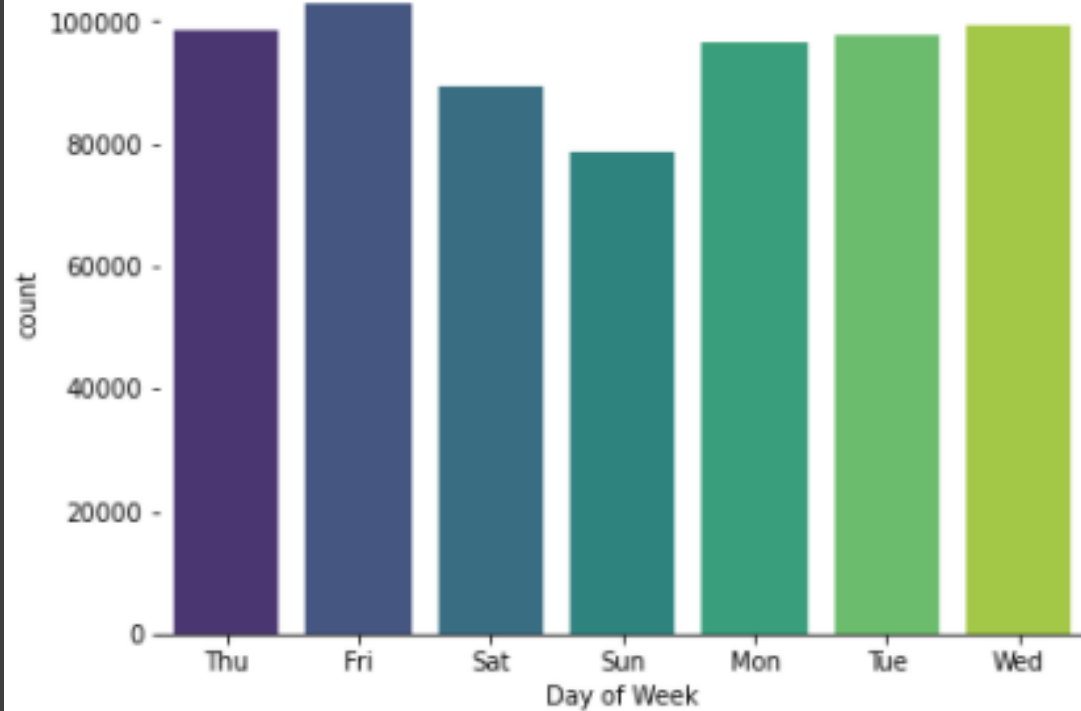
- Convert the timeStamp column from string to DateTime object to create 3 new columns called Hour, Month, and Day of Week.

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

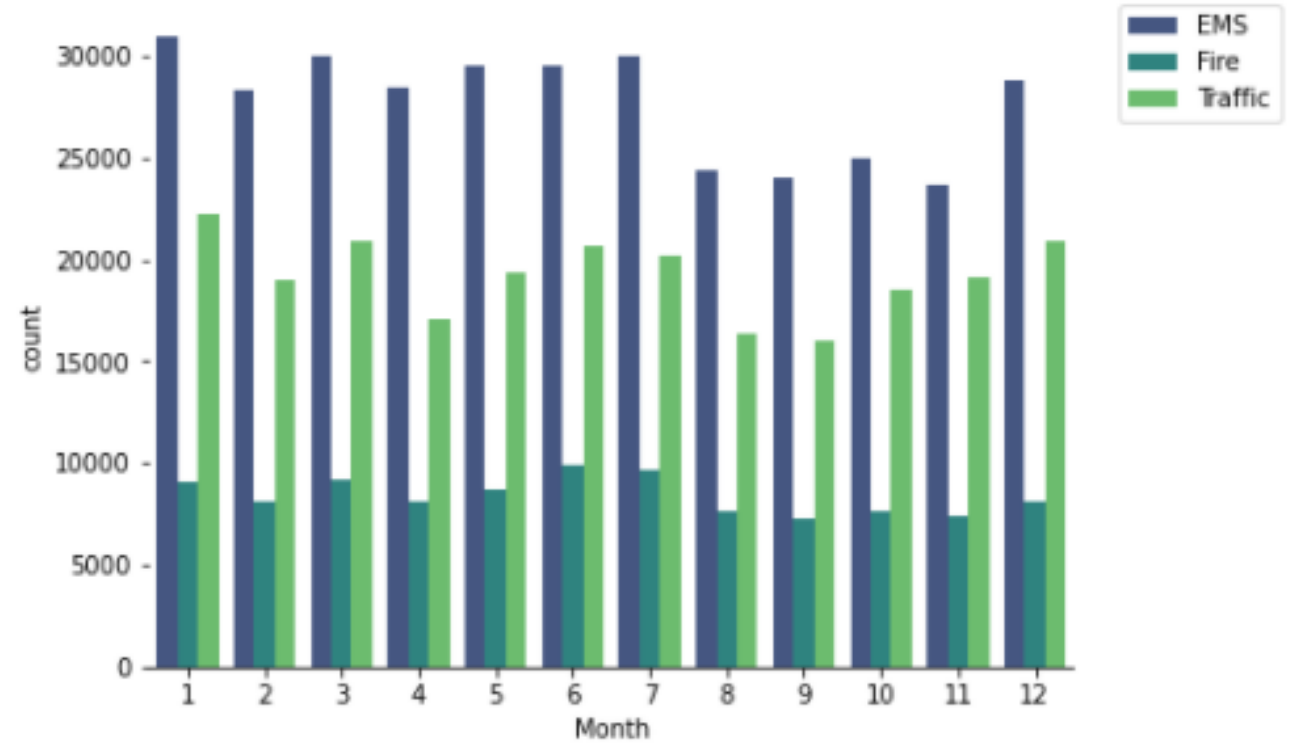
df['Hour'] = df['timeStamp'].apply(lambda time: time.hour)
df['Month'] = df['timeStamp'].apply(lambda time: time.month)
df['Day of Week'] = df['timeStamp'].apply(lambda time: time.dayofweek)
dmap = {0: 'Mon', 1: 'Tue', 2: 'Wed', 3: 'Thu', 4: 'Fri', 5: 'Sat', 6: 'Sun'}

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

Weekly Calls



Monthly Calls



Weekly and monthly calls

- It looks like friday is the day with more calls during the week.
- Regarding the monthly calls, looks like during the first semester there are more calls.

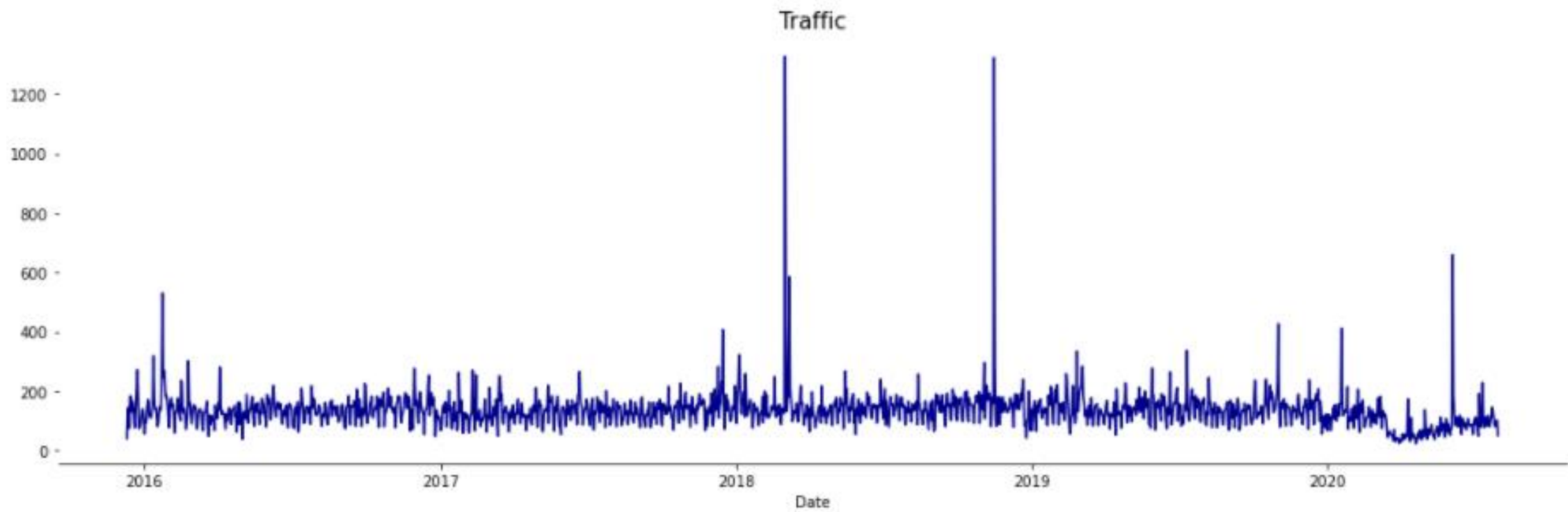


Date feature

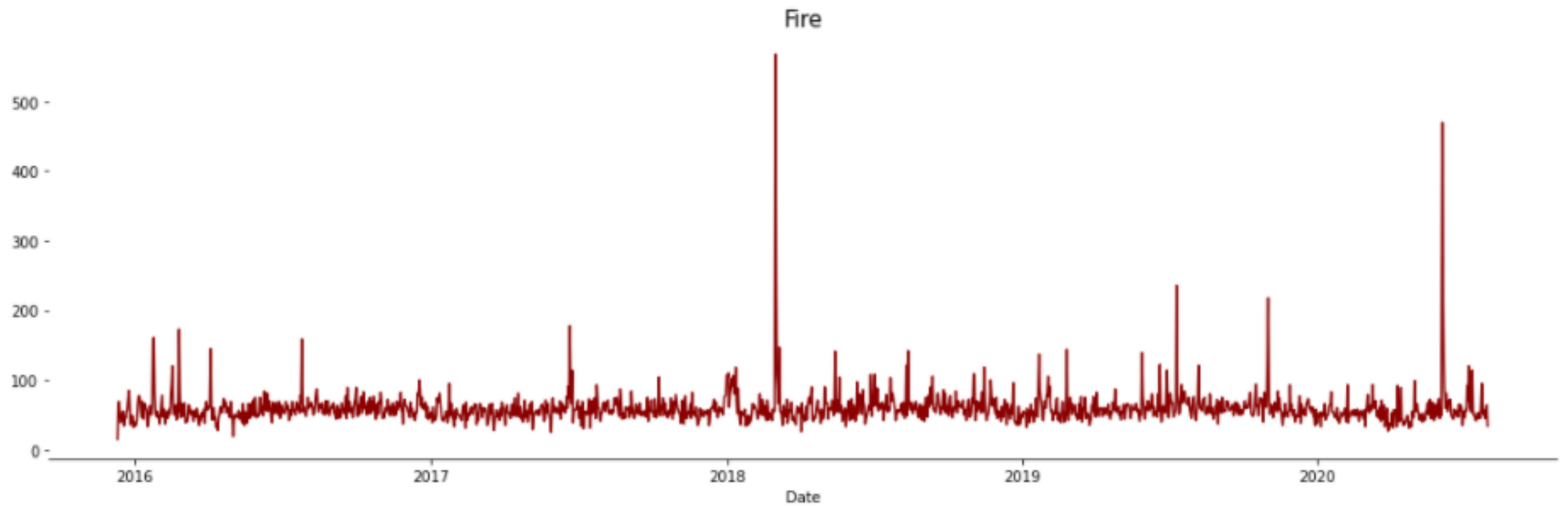
```
df['Date'] = df['timeStamp'].apply(lambda t: t.date())
```

- Create a new column called 'Date' that contains the date from the timeStamp column.
- groupby this Date column with the count() aggregate and create a plot of counts of 911 calls by reason.

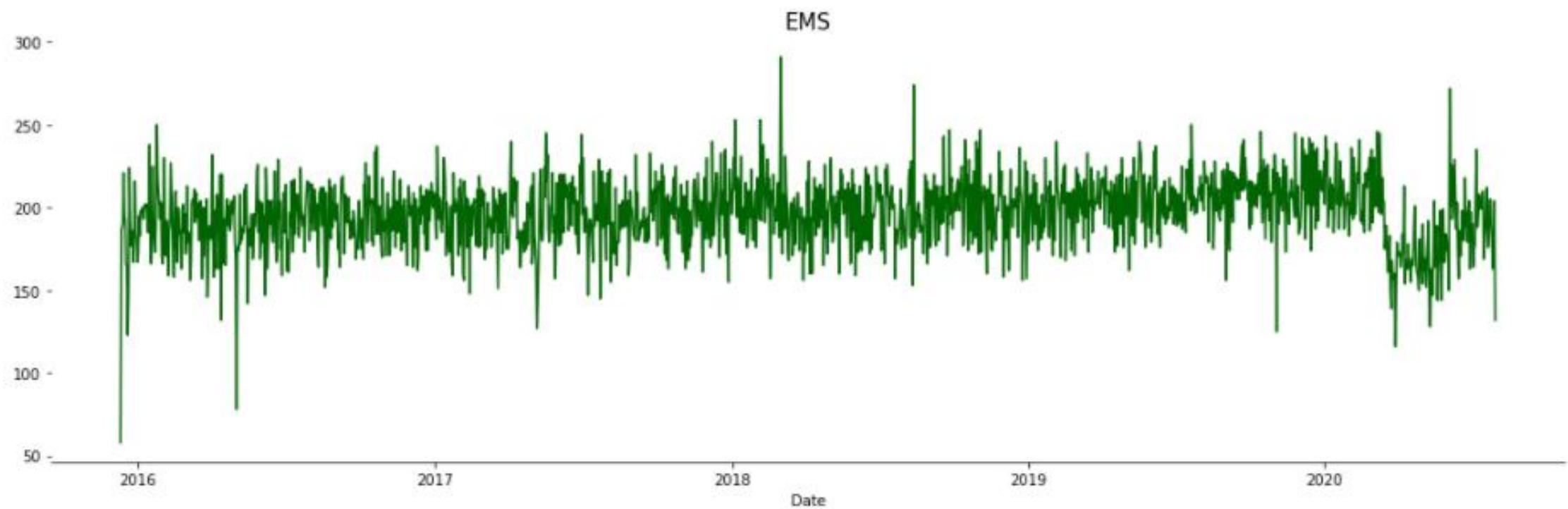
Traffic



Fire

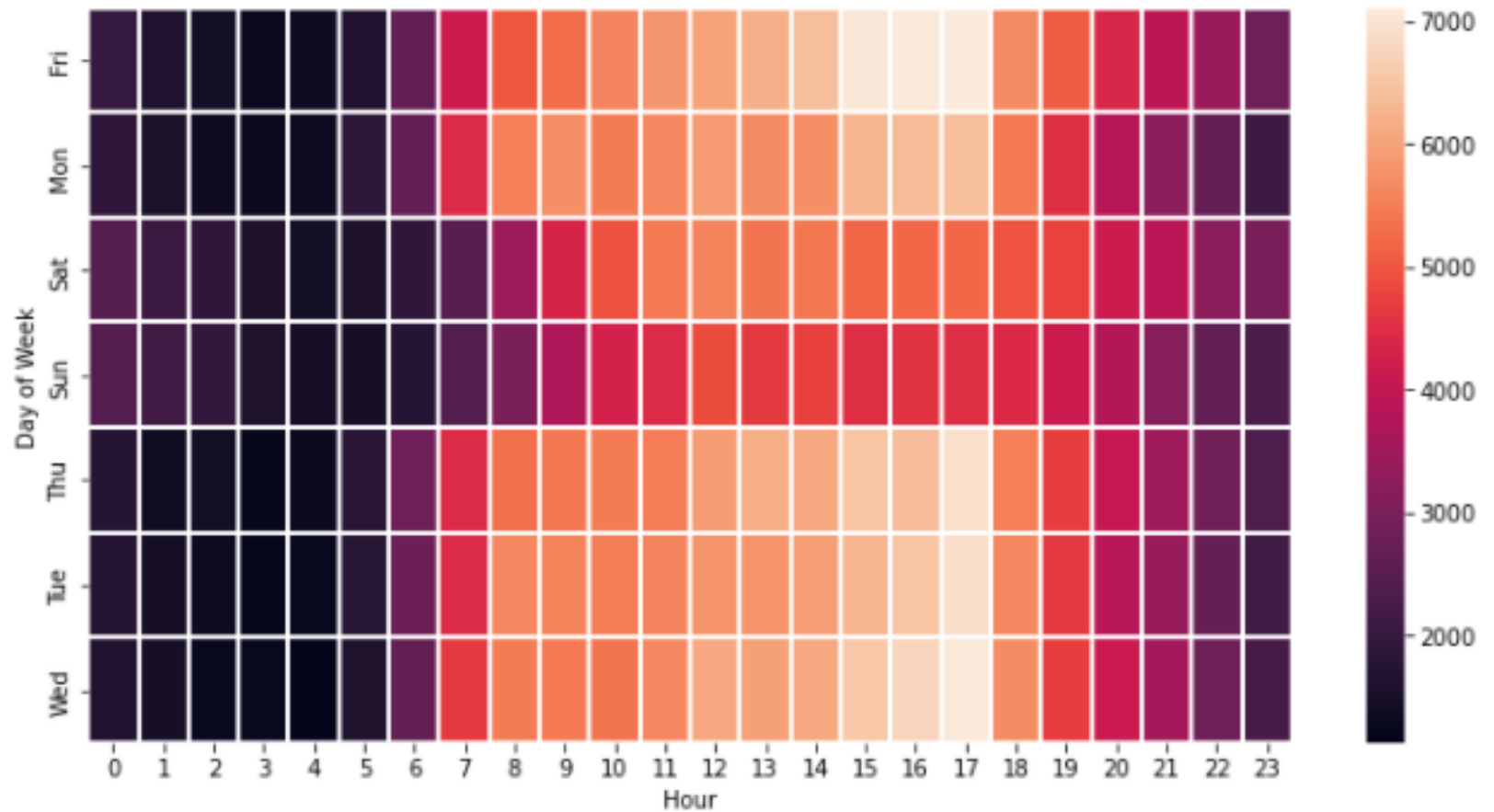



EMS



Heatmap

- In the heatmap we can see that during 14:00 and 17:00 hours there are more calls.
- Friday and Wednesday have more calls.
- Apparently during Sunday, the calls drop.





Modeling - 911 Call Type Prediction

Can we predict the type reason of the next call?

Modelling – imports

```
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
import re
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split

import tensorflow as tf
```

```
data = pd.read_csv('C:/Users/BUDUR/Desktop/Data scinsce Bootcamp/project/archive/911.csv', nrows=50000)
```

	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 POTTS GROVE; S...	NaN	EMS: DIZZINESS	2015-12-10 16:56:52	LOWER POTTS GROVE	CHERRYWOOD CT & DEAD END	1
...
49995	40.052428	-75.120794	CHELTENHAM AVE & WALNUT PARK DR; CHELTENHAM; ...	19012.0	EMS: HEMORRHAGING	2016-04-18 13:21:55	CHELTENHAM	CHELTENHAM AVE & WALNUT PARK DR	1
49996	40.254768	-75.660459	SHOEMAKER RD & ROBINSON ST; POTTSTOWN; 2016-04...	19464.0	Traffic: VEHICLE ACCIDENT -	2016-04-18 13:21:27	POTTSTOWN	SHOEMAKER RD & ROBINSON ST	1
49997	40.069832	-75.316295	CONSHOHOCKEN STATE RD & SCHUYLKILL EXPY OVERPA...	NaN	Fire: RESCUE - ELEVATOR	2016-04-18 13:30:04	WEST CONSHOHOCKEN	CONSHOHOCKEN STATE RD & SCHUYLKILL EXPY OVERPASS	1
49998	40.289027	-75.399590	HARLEYSVILLE PIKE & MAIN ST; LOWER SALFORD; S...	19438.0	EMS: RESPIRATORY EMERGENCY	2016-04-18 13:35:02	LOWER SALFORD	HARLEYSVILLE PIKE & MAIN ST	1
49999	40.100441	-75.107253	MEETINGHOUSE RD & DELENE RD; ABINGTON; 2016-04...	19046.0	Fire: WOODS/FIELD FIRE	2016-04-18 13:44:24	ABINGTON	MEETINGHOUSE RD & DELENE RD	1

Modelling

```
X_inputs = tf.keras.Input(shape=(X_train.shape[1],))
desc_inputs = tf.keras.Input(shape=(desc_train.shape[1],))
addr_inputs = tf.keras.Input(shape=(addr_train.shape[1],))

# X_inputs
X_dense1 = tf.keras.layers.Dense(128, activation='relu')(X_inputs)
X_dense2 = tf.keras.layers.Dense(128, activation='relu')(X_dense1)

# desc_inputs
desc_embedding = tf.keras.layers.Embedding(
    input_dim=10000,
    output_dim=64,
    input_length=desc_train.shape[1]
)(desc_inputs)
desc_flatten = tf.keras.layers.Flatten()(desc_embedding)

# addr_inputs
addr_embedding = tf.keras.layers.Embedding(
    input_dim=10000,
    output_dim=64,
    input_length=addr_train.shape[1]
)(addr_inputs)
addr_flatten = tf.keras.layers.Flatten()(addr_embedding)

# Concatenate results
concat = tf.keras.layers.concatenate([X_dense2, desc_flatten, addr_flatten])

# Make predictions
outputs = tf.keras.layers.Dense(3, activation='softmax')(concat)

model = tf.keras.Model(inputs=[X_inputs, desc_inputs, addr_inputs], outputs=outputs)

print(model.summary())
tf.keras.utils.plot_model(model)
```

Layer (type)	Output Shape	Param #	Connected to
=====			
input_1 (InputLayer)	[(None, 174)]	0	
input_2 (InputLayer)	[(None, 24)]	0	
input_3 (InputLayer)	[(None, 14)]	0	
dense (Dense)	(None, 128)	22400	input_1[0][0]
embedding (Embedding)	(None, 24, 64)	640000	input_2[0][0]
embedding_1 (Embedding)	(None, 14, 64)	640000	input_3[0][0]
dense_1 (Dense)	(None, 128)	16512	dense[0][0]
flatten (Flatten)	(None, 1536)	0	embedding[0][0]
flatten_1 (Flatten)	(None, 896)	0	embedding_1[0][0]
concatenate (Concatenate)	(None, 2560)	0	dense_1[0][0] flatten[0][0] flatten_1[0][0]
dense_2 (Dense)	(None, 3)	7683	concatenate[0][0]
=====			
Total params: 1,326,595			
Trainable params: 1,326,595			
Non-trainable params: 0			
None			

Training

```
model.compile(  
    optimizer='adam',  
    loss='sparse_categorical_crossentropy',  
    metrics=['accuracy']  
)  
  
history = model.fit(  
    [X_train, desc_train, addr_train],  
    y_train,  
    validation_split=0.2,  
    batch_size=32,  
    epochs=20,  
    callbacks=[  
        tf.keras.callbacks.ReduceLROnPlateau()  
    ]  
)
```

```
875/875 [=====] - 8s 9ms/step - loss: 0.1309 - accuracy: 0.9581 - val_loss: 0.0118 - val_accuracy: 0.9976  
Epoch 2/20  
875/875 [=====] - 8s 9ms/step - loss: 0.0053 - accuracy: 0.9992 - val_loss: 0.0068 - val_accuracy: 0.9990  
Epoch 3/20  
875/875 [=====] - 8s 9ms/step - loss: 0.0019 - accuracy: 0.9997 - val_loss: 0.0042 - val_accuracy: 0.9994  
Epoch 4/20  
875/875 [=====] - 8s 9ms/step - loss: 8.2584e-04 - accuracy: 0.9998 - val_loss: 0.0038 - val_accuracy: 0.9993  
Epoch 5/20  
875/875 [=====] - 8s 9ms/step - loss: 2.7862e-04 - accuracy: 1.0000 - val_loss: 0.0037 - val_accuracy: 0.9991  
Epoch 6/20  
875/875 [=====] - 7s 8ms/step - loss: 1.2712e-04 - accuracy: 1.0000 - val_loss: 0.0035 - val_accuracy: 0.9993  
Epoch 7/20  
875/875 [=====] - 8s 9ms/step - loss: 7.1434e-05 - accuracy: 1.0000 - val_loss: 0.0036 - val_accuracy: 0.9993  
Epoch 8/20  
875/875 [=====] - 7s 9ms/step - loss: 4.3320e-05 - accuracy: 1.0000 - val_loss: 0.0037 - val_accuracy: 0.9993  
Epoch 9/20  
875/875 [=====] - 8s 9ms/step - loss: 2.4316e-05 - accuracy: 1.0000 - val_loss: 0.0038 - val_accuracy: 0.9993  
Epoch 10/20  
875/875 [=====] - 8s 9ms/step - loss: 1.4416e-05 - accuracy: 1.0000 - val_loss: 0.0034 - val_accuracy: 0.9994  
Epoch 11/20  
875/875 [=====] - 8s 9ms/step - loss: 8.5148e-06 - accuracy: 1.0000 - val_loss: 0.0040 - val_accuracy: 0.9993  
Epoch 12/20  
875/875 [=====] - 7s 9ms/step - loss: 5.4619e-06 - accuracy: 1.0000 - val_loss: 0.0041 - val_accuracy: 0.9993  
Epoch 13/20  
875/875 [=====] - 8s 9ms/step - loss: 3.3203e-06 - accuracy: 1.0000 - val_loss: 0.0040 - val_accuracy: 0.9994  
Epoch 14/20  
875/875 [=====] - 8s 9ms/step - loss: 2.4549e-06 - accuracy: 1.0000 - val_loss: 0.0040 - val_accuracy: 0.9994  
Epoch 15/20  
875/875 [=====] - 8s 9ms/step - loss: 1.3315e-06 - accuracy: 1.0000 - val_loss: 0.0043 - val_accuracy: 0.9994  
Epoch 16/20  
875/875 [=====] - 7s 8ms/step - loss: 8.3403e-07 - accuracy: 1.0000 - val_loss: 0.0042 - val_accuracy: 0.9994  
Epoch 17/20  
875/875 [=====] - 8s 9ms/step - loss: 4.9513e-07 - accuracy: 1.0000 - val_loss: 0.0045 - val_accuracy: 0.9994  
Epoch 18/20  
875/875 [=====] - 8s 9ms/step - loss: 3.1701e-07 - accuracy: 1.0000 - val_loss: 0.0045 - val_accuracy: 0.9994  
Epoch 19/20  
875/875 [=====] - 7s 8ms/step - loss: 1.9979e-07 - accuracy: 1.0000 - val_loss: 0.0046 - val_accuracy: 0.9994  
Epoch 20/20  
875/875 [=====] - 7s 8ms/step - loss: 1.2660e-07 - accuracy: 1.0000 - val_loss: 0.0046 - val_accuracy: 0.9994
```

Results

```
results = model.evaluate([X_test, desc_test, addr_test], y_test, verbose=0)
```

```
print("Model loss: {:.5f}".format(results[0]))  
print("Model accuracy: {:.2f}%".format(results[1] * 100))
```

```
Model loss: 0.00773  
Model accuracy: 99.95%
```