STUDY ON RESPONSE TIME

San Francisco Fire Department Calls for service

ABSTRACT

Using the San Francisco Fire Department call logs to extract the 90th percentile Response Times each month by Battalion for the past 12 months and visualize them accordingly.

INTRODUCTION

The Fire Department plays a vital role in helping the Community rapidly return to life in critical moments by reacting immediately to incidents and resolving the crisis accordingly. Fire Calls-For-Service covers the response of fire units to requests.

OBJECTIVE

The key goal in this analysis is to explain Fire Department's response time between the moment they receive a call at the 911 Watch Center until the time they appear on the scene. This timeframe is known to be crucial so knowing this statistic from this point of view will allow San Francisco Fire Department officials to coordinate and respond efficiently. In specific, we would concentrate on 90th percentile Response Time depending on Month and their respective Battalion in detail.

ENVIRONMENT

Scripting language: Python

Version used: 3.7.3 (preferred 3.6 or above)

External Libraries: Pandas, Matplotlib

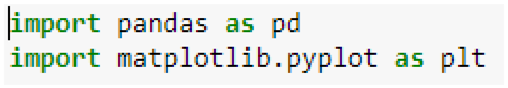
Input filename: Fire_Department_Calls_for_Service.csv

Execution command: python ResponseTime.py (or) python3 ResponseTime.py

DATA ANALYSIS

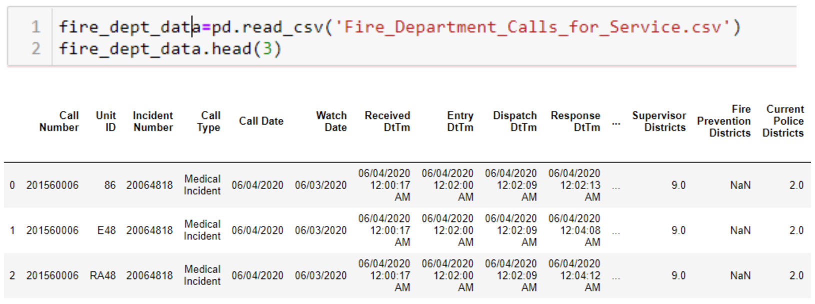
Data analysis is critical in business to identify the problems facing an enterprise and to analyze data in practical ways. Data is nothing more than statistics and figures. Data processing organizes, interprets, integrates and introduces data to relevant knowledge that provides a context for the data.

- To work with our data, we will first import necessary libraries



```python
import pandas as pd
import matplotlib.pyplot as plt
```

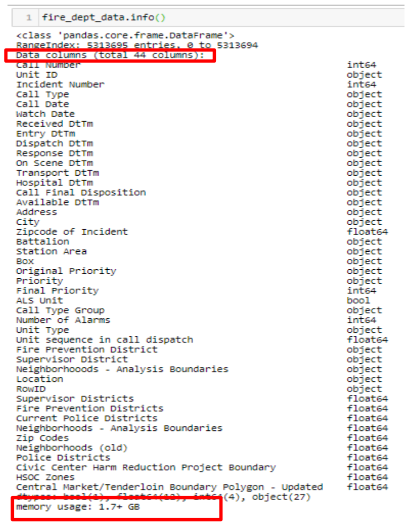
- Initially, we start by reading the data using read_csv function followed by analyzing the raw data and understanding the big picture before we jump into the study itself. Pandas has been selected since it is simple, efficient, versatile and easy-to-use open source data analysis and manipulation tool that can be used with Python programming language. We start our analysis with raw data and sample few rows to understand them.



```python
fire_dept_data=pd.read_csv('Fire_Department_Calls_for_Service.csv')
fire_dept_data.head(3)
```

	Call Number	Unit ID	Incident Number	Call Type	Call Date	Watch Date	Received DtTm	Entry DtTm	Dispatch DtTm	Response DtTm	...	Supervisor Districts	Fire Prevention Districts	Current Police Districts
0	201560006	86	20064818	Medical Incident	06/04/2020	06/03/2020	06/04/2020 12:00:17 AM	06/04/2020 12:02:00 AM	06/04/2020 12:02:09 AM	06/04/2020 12:02:13 AM	...	9.0	NaN	2.0
1	201560006	E48	20064818	Medical Incident	06/04/2020	06/03/2020	06/04/2020 12:00:17 AM	06/04/2020 12:02:00 AM	06/04/2020 12:02:09 AM	06/04/2020 12:04:08 AM	...	9.0	NaN	2.0
2	201560006	RA48	20064818	Medical Incident	06/04/2020	06/03/2020	06/04/2020 12:00:17 AM	06/04/2020 12:02:00 AM	06/04/2020 12:02:09 AM	06/04/2020 12:04:12 AM	...	9.0	NaN	2.0

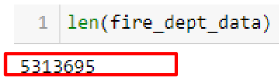
- In summary, we have around 5,313,695 records out of which we are only interested in last twelve months. In addition, there are about 44 columns in which only a few columns, such as 'Call Date', 'Received DtTm', 'On Scene DtTm', 'Battalion', would be needed for our analysis. So instead of loading the entire csv file in our DataFrame, we can restrict the columns thereby reducing the memory consumption and speeding up the performance time.



```
fire_dept_data.info()
```

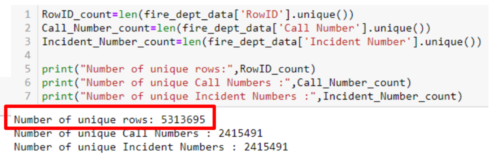
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5313695 entries, 0 to 5313694
Data columns (total 44 columns):
Call Number                                                 int64
Unit ID                                                     object
Incident Number                                             int64
Call Type                                                   object
Call Date                                                   object
Watch Date                                                  object
Received DtTm                                               object
Entry DtTm                                                  object
Dispatch DtTm                                               object
Response DtTm                                               object
On Scene DtTm                                               object
Transport DtTm                                              object
Hospital DtTm                                               object
Call Final Disposition                                      object
Available DtTm                                              object
Address                                                     object
City                                                        object
Zipcode of Incident                                         float64
Battalion                                                   object
Station Area                                                object
Box                                                         object
Original Priority                                           object
Priority                                                    object
Final Priority                                              int64
ALS Unit                                                    bool
Call Type Group                                             object
Number of Alarms                                            int64
Unit Type                                                   object
Unit sequence in call dispatch                              float64
Fire Prevention District                                    object
Supervisor District                                         object
Neighborhooods - Analysis Boundaries                        object
Location                                                    object
RowID                                                       object
Supervisor Districts                                        float64
Fire Prevention Districts                                   float64
Current Police Districts                                    float64
Neighborhoods - Analysis Boundaries                         float64
Zip Codes                                                   float64
Neighborhoods (old)                                         float64
Police Districts                                            float64
Civic Center Harm Reduction Project Boundary                float64
HSOC Zones                                                  float64
Central Market/Tenderloin Boundary Polygon - Updated        float64
dtypes: bool(1), float64(12), int64(4), object(27)
memory usage: 1.7+ GB
```

- From below output we can find that 'RowID' which is a unique combination of Call Number and Unit ID matches the total number of records. In addition, we can figure out there are duplicate records for Call Number and Incident Number.



```
len(fire_dept_data)
```

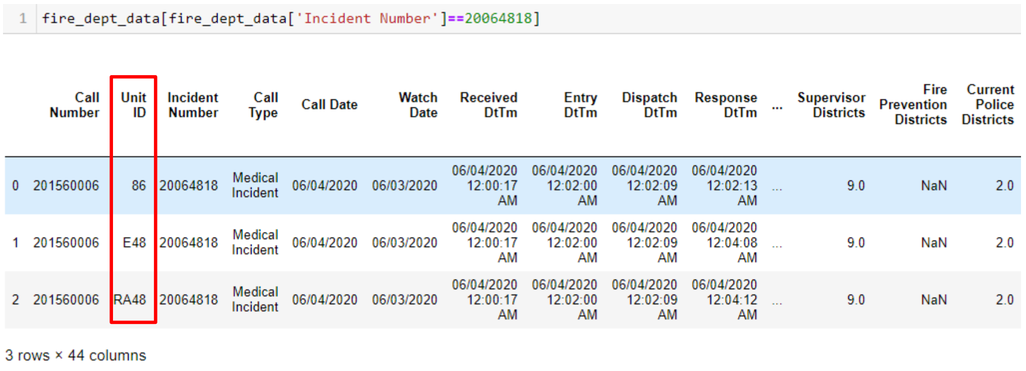
```
5313695
```



```
RowID_count=len(fire_dept_data['RowID'].unique())
Call_Number_count=len(fire_dept_data['Call Number'].unique())
Incident_Number_count=len(fire_dept_data['Incident Number'].unique())

print("Number of unique rows:",RowID_count)
print("Number of unique Call Numbers :",Call_Number_count)
print("Number of unique Incident Numbers :",Incident_Number_count)
```

```
Number of unique rows: 5313695
Number of unique Call Numbers : 2415491
Number of unique Incident Numbers : 2415491
```

- Now, let 's review one of the Incident Numbers and figure out more about the duplicate data. From below results, we can realize that even if the 'Call Number' and 'Incident Number' values are duplicated, the 'Unit ID' remains distinct for each record. This makes sense because we may have the same Incident Number and Call Number for any reported accidents, but different units will be dispatched to deal with the situation.

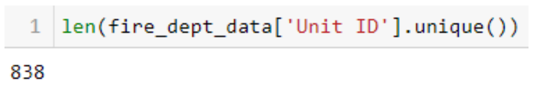


```
fire_dept_data[fire_dept_data['Incident Number']==20064818]
```

	Call Number	Unit ID	Incident Number	Call Type	Call Date	Watch Date	Received DtTm	Entry DtTm	Dispatch DtTm	Response DtTm	...	Supervisor Districts	Fire Prevention Districts	Current Police Districts
0	201560006	86	20064818	Medical Incident	06/04/2020	06/03/2020	06/04/2020 12:00:17 AM	06/04/2020 12:02:00 AM	06/04/2020 12:02:09 AM	06/04/2020 12:02:13 AM	...	9.0	NaN	2.0
1	201560006	E48	20064818	Medical Incident	06/04/2020	06/03/2020	06/04/2020 12:00:17 AM	06/04/2020 12:02:00 AM	06/04/2020 12:02:09 AM	06/04/2020 12:04:08 AM	...	9.0	NaN	2.0
2	201560006	RA48	20064818	Medical Incident	06/04/2020	06/03/2020	06/04/2020 12:00:17 AM	06/04/2020 12:02:00 AM	06/04/2020 12:02:09 AM	06/04/2020 12:04:12 AM	...	9.0	NaN	2.0

3 rows × 44 columns

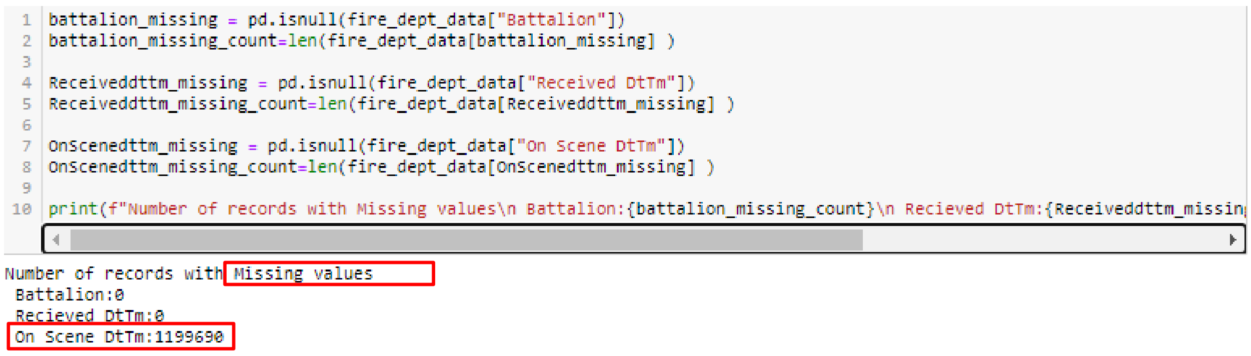
In general, there are 838 different units available for service. It can refer to Engine(E01), Truck(T01) etc.



```
len(fire_dept_data['Unit ID'].unique())
```

```
838
```

- We will now check whether there are any missing values in our data for the necessary columns. As per result, we can find that only ‘On Scene DtTm’ column has missing values and therefore these records should be dropped during Data Manipulation since we will calculate Response Time by taking difference between ‘On Scene DtTm’ and ‘Received DtTm’.

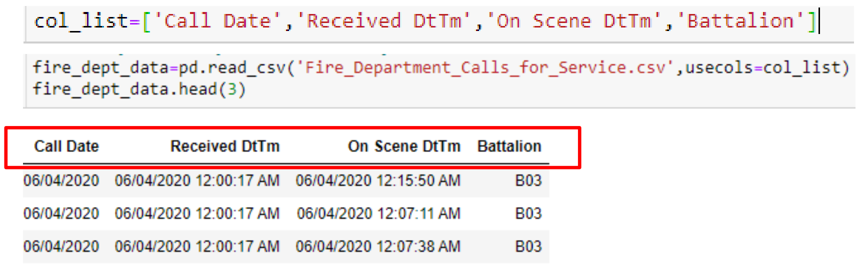


```
battalion_missing = pd.isnull(fire_dept_data["Battalion"])
battalion_missing_count=len(fire_dept_data[battalion_missing] )

Receiveddttm_missing = pd.isnull(fire_dept_data["Received DtTm"])
Receiveddttm_missing_count=len(fire_dept_data[Receiveddttm_missing] )

OnScenedttm_missing = pd.isnull(fire_dept_data["On Scene DtTm"])
OnScenedttm_missing_count=len(fire_dept_data[OnScenedttm_missing] )

print(f"Number of records with Missing values\n Battalion:{battalion_missing_count}\n Recieved DtTm:{Receiveddttm_missin
```

```
Number of records with Missing values
 Battalion:0
 Recieved DtTm:0
 On Scene DtTm:1199690
```

DATA MANIPULATION

- Based on the use-case, we will import the csv file with necessary columns to save memory and accelerate processing time.

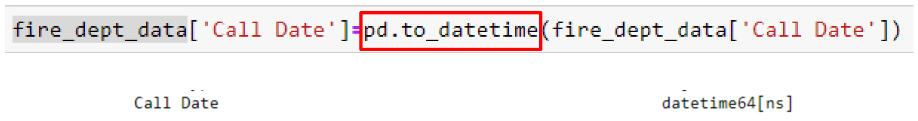


```
col_list=['Call Date','Received DtTm','On Scene DtTm','Battalion']
```

```
fire_dept_data=pd.read_csv('Fire_Department_Calls_for_Service.csv',usecols=col_list)
fire_dept_data.head(3)
```

Call Date	Received DtTm	On Scene DtTm	Battalion
06/04/2020	06/04/2020 12:00:17 AM	06/04/2020 12:15:50 AM	B03
06/04/2020	06/04/2020 12:00:17 AM	06/04/2020 12:07:11 AM	B03
06/04/2020	06/04/2020 12:00:17 AM	06/04/2020 12:07:38 AM	B03

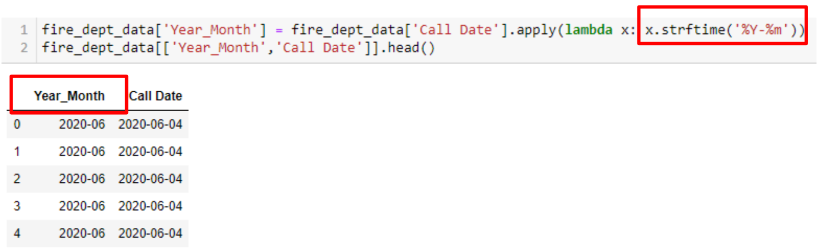
- When a csv file is imported and a DataFrame is generated, the Date Time objects in the file are interpreted as a string object rather than a Date Time object and thus it is very difficult to perform operations such as last(“12M”) to get recent twelve months data. The Pandas to_datetime method helps to translate the Date time string to the Python Date time object.



```
fire_dept_data['Call Date']=pd.to_datetime(fire_dept_data['Call Date'])
```

```
Call Date                          datetime64[ns]
```

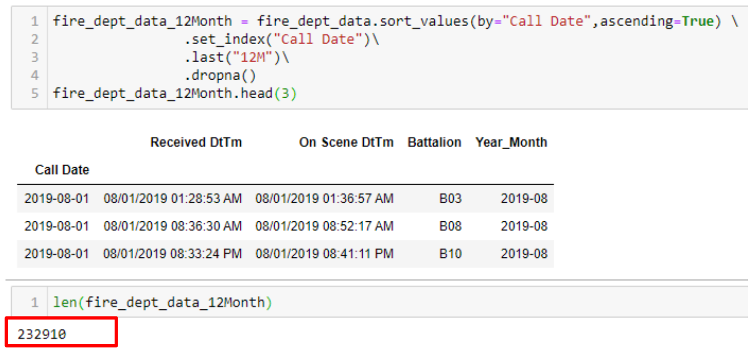
- We will need a new column ‘Year_Month’ for further analysis on grouping data based on Month. So, we will create one using existing column ‘Call Date’ using strftime method.



```
fire_dept_data['Year_Month'] = fire_dept_data['Call Date'].apply(lambda x: x.strftime('%Y-%m'))
fire_dept_data[['Year_Month','Call Date']].head()
```

	Year_Month	Call Date
0	2020-06	2020-06-04
1	2020-06	2020-06-04
2	2020-06	2020-06-04
3	2020-06	2020-06-04
4	2020-06	2020-06-04

- We will now sort and index the data based on ‘Call Date’ column, filter the last 12-month records and finally drop any records with missing values since it will affect Response time calculation. In conclusion, we have 232910 out of 5313695 records for our case study.



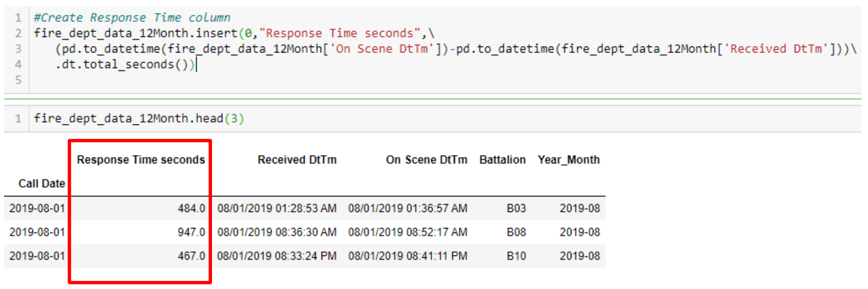
```
fire_dept_data_12Month = fire_dept_data.sort_values(by="Call Date",ascending=True) \
                    .set_index("Call Date")\
                    .last("12M")\
                    .dropna()
fire_dept_data_12Month.head(3)
```

Call Date	Received DtTm	On Scene DtTm	Battalion	Year_Month
2019-08-01	08/01/2019 01:28:53 AM	08/01/2019 01:36:57 AM	B03	2019-08
2019-08-01	08/01/2019 08:36:30 AM	08/01/2019 08:52:17 AM	B08	2019-08
2019-08-01	08/01/2019 08:33:24 PM	08/01/2019 08:41:11 PM	B10	2019-08

```
len(fire_dept_data_12Month)
```

```
232910
```

- For the filtered records, we will calculate ‘Response Time seconds’ column from ‘On Scene DtTm’ field and ‘Received DtTm’ field.

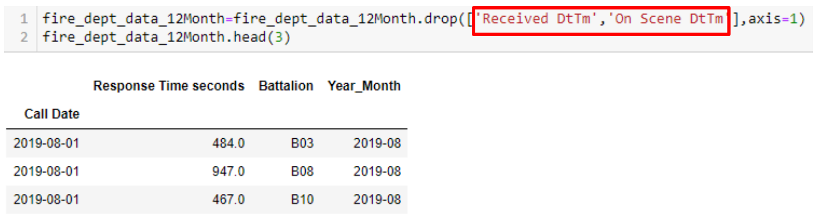


```
#Create Response Time column
fire_dept_data_12Month.insert(0,"Response Time seconds",\
    (pd.to_datetime(fire_dept_data_12Month['On Scene DtTm'])-pd.to_datetime(fire_dept_data_12Month['Received DtTm']))\
    .dt.total_seconds())
```

```
fire_dept_data_12Month.head(3)
```

Call Date	Response Time seconds	Received DtTm	On Scene DtTm	Battalion	Year_Month
2019-08-01	484.0	08/01/2019 01:28:53 AM	08/01/2019 01:36:57 AM	B03	2019-08
2019-08-01	947.0	08/01/2019 08:36:30 AM	08/01/2019 08:52:17 AM	B08	2019-08
2019-08-01	467.0	08/01/2019 08:33:24 PM	08/01/2019 08:41:11 PM	B10	2019-08

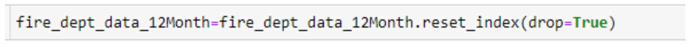
- Since ‘Received DtTm’ and ‘On Scene DtTm’ is no longer needed we will drop these columns for reducing DataFrame size and faster processing in next steps. This will also help us achieve on retaining only required columns in our DataFrame.



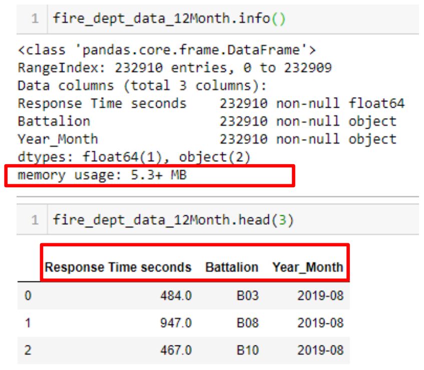
```
fire_dept_data_12Month=fire_dept_data_12Month.drop(['Received DtTm','On Scene DtTm'],axis=1)
fire_dept_data_12Month.head(3)
```

Call Date	Response Time seconds	Battalion	Year_Month
2019-08-01	484.0	B03	2019-08
2019-08-01	947.0	B08	2019-08
2019-08-01	467.0	B10	2019-08

- In addition, we will also remove the index on ‘Call Date’ since it does not serve any purpose anymore. Finally, we were able to reduce from 1.7GB of raw data to 5.3 MB of relevant data and just the required columns needed for analysis.



```
fire_dept_data_12Month=fire_dept_data_12Month.reset_index(drop=True)
```



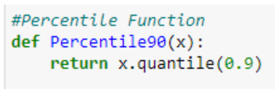
```
fire_dept_data_12Month.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 232910 entries, 0 to 232909
Data columns (total 3 columns):
Response Time seconds    232910 non-null float64
Battalion                232910 non-null object
Year_Month               232910 non-null object
dtypes: float64(1), object(2)
memory usage: 5.3+ MB
```

```
fire_dept_data_12Month.head(3)
```

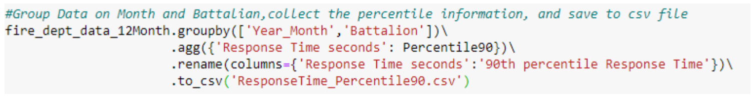
	Response Time seconds	Battalion	Year_Month
0	484.0	B03	2019-08
1	947.0	B08	2019-08
2	467.0	B10	2019-08

- quantile function could be used for calculating 90th Percentile value for Response time as shown below

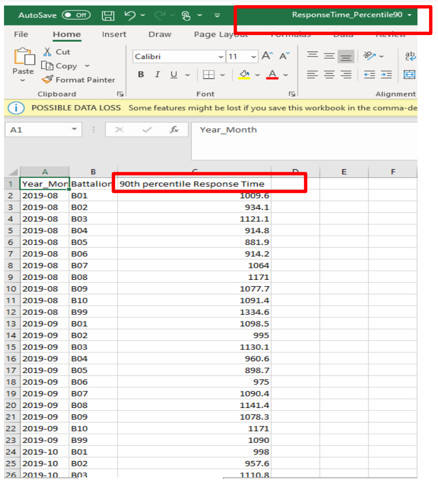


```
#Percentile Function
def Percentile90(x):
    return x.quantile(0.9)
```

- Finally, we group the data using groupby method on 'Year_Month' and 'Battalion' column, calculate 90th Percentile value for the 'Response time seconds' field, and then save the results to a table named as 'ResponseTime_Percentile90.csv'(snapshot added below for reference)



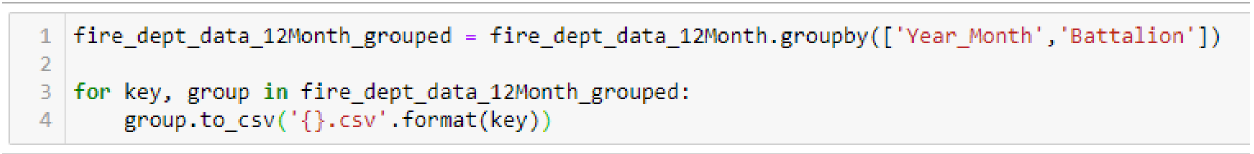
```
#Group Data on Month and Battalian,collect the percentile information, and save to csv file
fire_dept_data_12Month.groupby(['Year_Month','Battalion'])\
                       .agg({'Response Time seconds': Percentile90})\
                       .rename(columns={'Response Time seconds':'90th percentile Response Time'})\
                       .to_csv('ResponseTime_Percentile90.csv')
```


AutoSave
ResponseTime_Percentile90
File
Home
Insert
Draw
Page Layout
Cut
Copy
Format Painter
Paste
Clipboard
Calibri
11
Font
Alignment
POSSIBLE DATA LOSS Some features might be lost if you save this workbook in the comma-de
A1
Year_Month

	A	B	C
1	Year_Mon	Battalion	90th percentile Response Time
2	2019-08	B01	1009.6
3	2019-08	B02	934.1
4	2019-08	B03	1121.1
5	2019-08	B04	914.8
6	2019-08	B05	881.9
7	2019-08	B06	914.2
8	2019-08	B07	1064
9	2019-08	B08	1171
10	2019-08	B09	1077.7
11	2019-08	B10	1091.4
12	2019-08	B99	1334.6
13	2019-09	B01	1098.5
14	2019-09	B02	995
15	2019-09	B03	1130.1
16	2019-09	B04	960.6
17	2019-09	B05	898.7
18	2019-09	B06	975
19	2019-09	B07	1090.4
20	2019-09	B08	1141.4
21	2019-09	B09	1078.3
22	2019-09	B10	1171
23	2019-09	B99	1090
24	2019-10	B01	998
25	2019-10	B02	957.6
26	2019-10	B03	1110.8

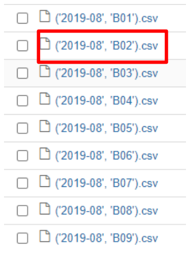
DATA VALIDATION

- To validate the data that is grouped by 'Year_Month' and 'Battalion' we will generate excel file for each unique group using the below code.



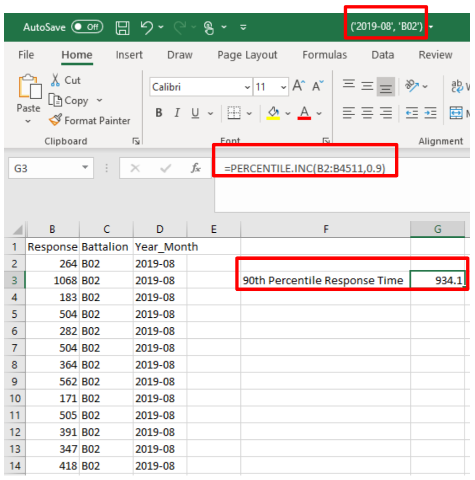
```
fire_dept_data_12Month_grouped = fire_dept_data_12Month.groupby(['Year_Month','Battalion'])

for key, group in fire_dept_data_12Month_grouped:
    group.to_csv('{}.csv'.format(key))
```

For instance, in case of 2019 August we have below list of unique combinations with Battalion



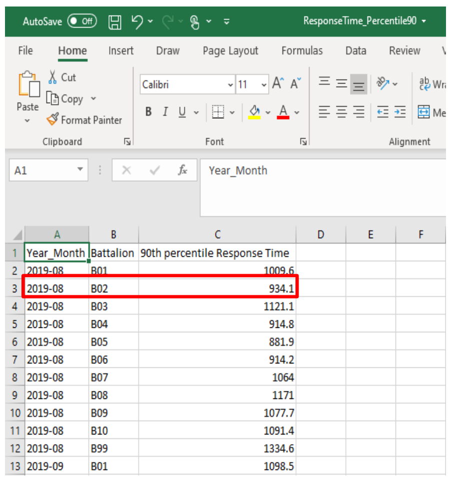
```
('2019-08', 'B01').csv
('2019-08', 'B02').csv
('2019-08', 'B03').csv
('2019-08', 'B04').csv
('2019-08', 'B05').csv
('2019-08', 'B06').csv
('2019-08', 'B07').csv
('2019-08', 'B08').csv
('2019-08', 'B09').csv
```

Let us sample one of the excel file that is for 2019_08 and 'B02' Battalion combination and find the 90th percentile using Excel through the formula("=PERCENTILE.EXC(A1:AX,k)") .Thus from below output, we have validated our results that is obtained programmatically and saved in 'ResponseTime_Percentile90' csv file.



G3: `=PERCENTILE.INC(B2:B4511,0.9)`

	B	C	D	E	F	G
1	Response	Battalion	Year_Month			
2	264	B02	2019-08			
3	1068	B02	2019-08		90th Percentile Response Time	934.1
4	183	B02	2019-08			
5	504	B02	2019-08			
6	282	B02	2019-08			
7	504	B02	2019-08			
8	364	B02	2019-08			
9	562	B02	2019-08			
10	171	B02	2019-08			
11	505	B02	2019-08			
12	391	B02	2019-08			
13	347	B02	2019-08			
14	418	B02	2019-08			

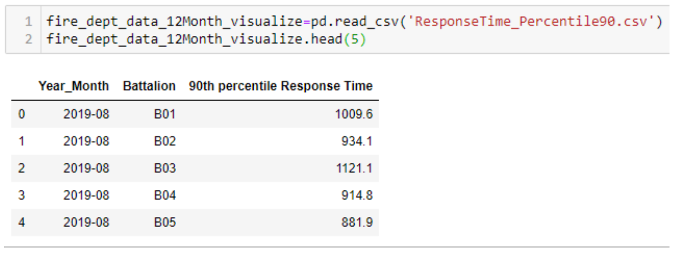


A1: Year_Month

	A	B	C
1	Year_Month	Battalion	90th percentile Response Time
2	2019-08	B01	1009.6
3	2019-08	B02	934.1
4	2019-08	B03	1121.1
5	2019-08	B04	914.8
6	2019-08	B05	881.9
7	2019-08	B06	914.2
8	2019-08	B07	1064
9	2019-08	B08	1171
10	2019-08	B09	1077.7
11	2019-08	B10	1091.4
12	2019-08	B99	1334.6
13	2019-09	B01	1098.5

DATA VISUALIZATION

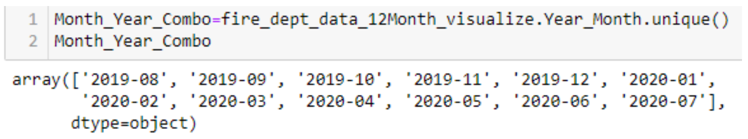
- To visualize the data, we will first import the prepared table saved in 'ResponseTime_Percentile90.csv' to a DataFrame named 'fire_dept_data_12Month_visualize'



```
fire_dept_data_12Month_visualize=pd.read_csv('ResponseTime_Percentile90.csv')
fire_dept_data_12Month_visualize.head(5)
```

	Year_Month	Battalion	90th percentile Response Time
0	2019-08	B01	1009.6
1	2019-08	B02	934.1
2	2019-08	B03	1121.1
3	2019-08	B04	914.8
4	2019-08	B05	881.9

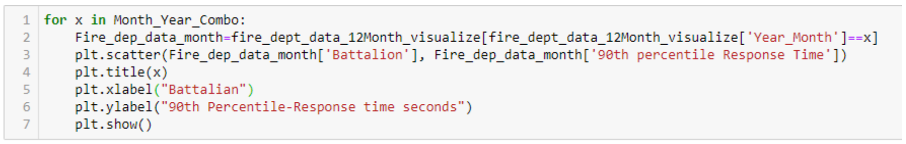
- We will now extract unique Month_Year combinations from the DataFrame so we can generate scatter plots individually based on Month indicating the 90th Percentile Response time with respect to the 'Battalion'



```
Month_Year_Combo=fire_dept_data_12Month_visualize.Year_Month.unique()
Month_Year_Combo
```

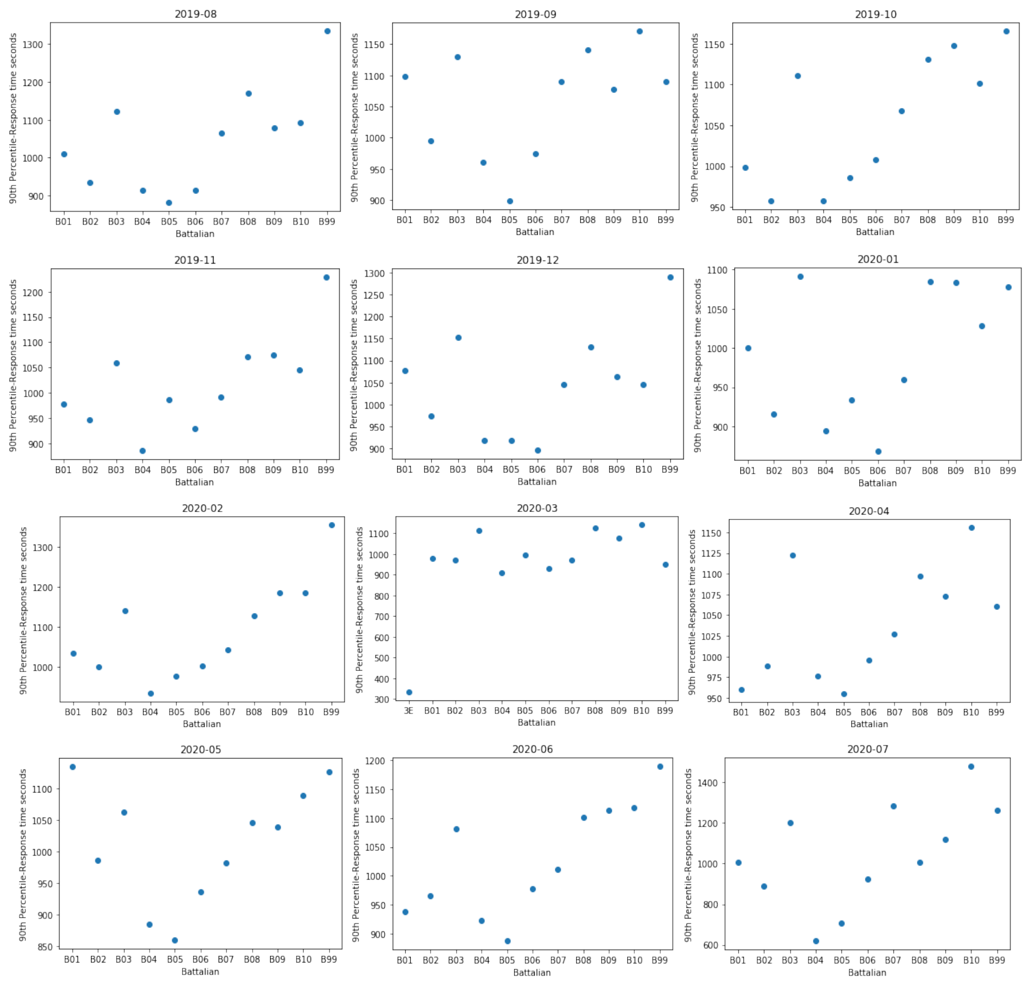
```
array(['2019-08', '2019-09', '2019-10', '2019-11', '2019-12', '2020-01',
       '2020-02', '2020-03', '2020-04', '2020-05', '2020-06', '2020-07'],
      dtype=object)
```

- Initially, line plot was created but the visual representation was not depicting the actual relationship. So I have used matplotlib library for scatter plot between 'Battalion' and their respective 90th Percentile data. It is to be noted that Pandas require x column to be numeric for scatter plot. However, as per our data 'Battalion' is categorical so matplotlib has been used for this purpose. Based on unique combination we will run through the loop and obtain monthly data and plot 90th percentile Response time with respect to the Battalion.



```
for x in Month_Year_Combo:
    Fire_dep_data_month=fire_dept_data_12Month_visualize[fire_dept_data_12Month_visualize['Year_Month']==x]
    plt.scatter(Fire_dep_data_month['Battalion'], Fire_dep_data_month['90th percentile Response Time'])
    plt.title(x)
    plt.xlabel("Battalian")
    plt.ylabel("90th Percentile-Response time seconds")
    plt.show()
```

Below visualization has been obtained for months from August 2019 to July 2020.



<u>Observation:</u>

From above scatter plot, we can find that the Battalion with the least 90th Percentile Response time could be best for the respective Month. For example, 'B05' Battalion have surpassed all other Battalion teams for the month of August 2019 with the least value of 889.1 seconds.

In fact, ‘B05’ Battalion have been the best for 5 months(2019_08, 2019_09, 2020_04, 2020_05, 2020_06, 2020_07).In contrast, B99 Battalion have the highest 90th Response time for 6 months. This suggest that there might be some improvement to be considered by ‘B99’ for better performance in the future.

CONCLUSION

To conclude, we focused on Fire Department Calls for Service data to determine the 90th percentile Response Time for Month and Battalion. While the raw data is about 1.7 GB, our data analysis supplemented by data engineering has helped to minimize the bulk of the data to just 5.3 MB. In fact, this has helped us to achieve better performance and memory usage. Finally, we made use of visualization to recognize the bigger picture of the data and discussed few observations that would help future performance.