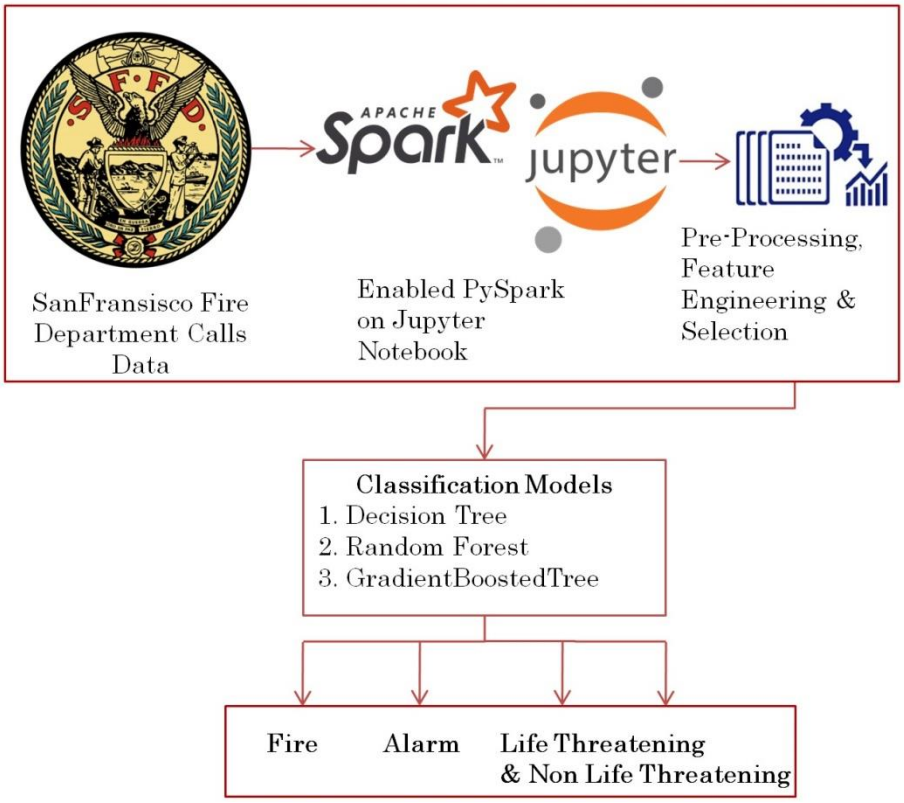
Analysis and Prediction of Fire Department Calls using Spark Big Data Platform

Rekha Pasupuleti

I. INTRODUCTION

In both urban and rural regions, fire, whether caused by people or nature, is a major hazard that poses a danger to life and property. There were 1,319,500 fires recorded in the United States in 2017, resulting in 3400 fire fatalities, 14,670 fire injuries, and billions in property damages [1]. While it is hard to totally prevent fires, fire management based on big data analysis may help to lower the danger of fire. Spatial assessments of fire events may provide additional information to planners and decision makers when developing fire safety policies [2]. Basic tactics include upgrading or constructing new fire stations closer to problem regions, moving present fire personnel and equipment into smaller and more widely distributed stations, and so on. Cluster analysis approaches such as spatial have received a lot of attention in recent years. It has been used in a variety of catastrophe analyses, including fires, road accidents, and so on [3, 4]. The use of spatial analytic approaches for fire occurrences, according to [5] may give information for decision makers in terms of responsiveness and resource allocation, such as geographical targeting of preventative actions. [6] conducted temporal, geographical, and spatiotemporal assessments of structure fire events in Toronto, Ontario, Canada from 2000 to 2006. And the findings revealed that there are considerable changes in fire causation throughout time and place.

II. DATASET & IMPLEMENTATION



The data for this project comes from San Francisco's official open data site [7]. Since 2000, the database has recorded hundreds of thousands of fires in San Francisco. Each occurrence is documented which covers the incidence number, time of fires, fire locations, fire injuries, and so on. It includes all fire units' responses to calls. Each record includes the call number, incident number, address, unit identifier, call type, and disposition. All relevant time intervals are also included. Because this dataset is based on responses, and since most calls involved multiple units, there are multiple records for each call number. Addresses are associated with a block number, intersection or call box, not a specific address.

This Dataset of 55Million records is analysed on Jupyter notebook by connecting to spark. PySpark links Python to Apache Spark, allowing us to develop Spark applications using the Python API and allowing us to interact with Apache Spark's Resilient Distributed Datasets (RDDs). Applying different data cleaning procedures to address faults in the original gathered data is a key step before data analysis. Missing data is filled in, noise is smoothed down, outliers are identified, and inconsistent data is corrected.

III. MOTIVATION

We are trying to improve operational efficiencies and enable smart decision making by analysing the data of incoming calls for firefighters. Spark is an open-source distributed analytics engine that can rapidly handle enormous volumes of data. PySpark is a python API for Apache Spark that allows us to use the capabilities of Apache Spark while using a programming language like python. Our motivation for putting this project together is to learn how spark can help in the analysis of large amounts of data, as well as how we can include machine learning libraries into the PySpark pipeline.

IV. CODE

```
import pandas as pd
import seaborn as sns
sns.set()
import matplotlib.pyplot as plt
import warnings
import os
```

```
from pyspark.sql import Row
from pyspark.sql import SparkSession
from pyspark.sql.types import *
from pyspark.sql.functions import to_date, unix_timestamp, year, month, hour, to_timestamp, udf
from pyspark.sql.functions import concat, col, lit, udf, split
from pyspark.ml.clustering import KMeans
from pyspark.ml.evaluation import ClusteringEvaluator
from pyspark.ml.feature import VectorAssembler
```

```
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.tsa.stattools import adfuller
import statsmodels.api as sm
```

```
from pyspark.sql import SparkSession
spark = SparkSession \
    .builder \
    .appName("fire analysis") \
    .config("spark.some.config.option", "some-value") \
    .getOrCreate()

```

```
df_fire = spark.read.format("csv").option("header", "true").load("C:\\Users\\manoja\\Downloads\\FireDepartmentCalls1.csv
df_fire.createOrReplaceTempView("sf_fire")
```

```
1 df_fire.printSchema()
```

```
root
 |-- _c0: string (nullable = true)
 |-- Call Number: string (nullable = true)
 |-- Unit ID: string (nullable = true)
 |-- Incident Number: string (nullable = true)
 |-- Call Type: string (nullable = true)
 |-- Call Date: string (nullable = true)
 |-- Watch Date: string (nullable = true)
 |-- Received DtTm: string (nullable = true)
 |-- Entry DtTm: string (nullable = true)
 |-- Dispatch DtTm: string (nullable = true)
 |-- Response DtTm: string (nullable = true)
 |-- On Scene DtTm: string (nullable = true)
 |-- Transport DtTm: string (nullable = true)
 |-- Hospital DtTm: string (nullable = true)
 |-- Call Final Disposition: string (nullable = true)
 |-- Available DtTm: string (nullable = true)
 |-- Address: string (nullable = true)
 |-- City: string (nullable = true)
 |-- Zipcode of Incident: string (nullable = true)
 |-- Battalion: string (nullable = true)
 |-- Station Area: string (nullable = true)
 |-- Box: string (nullable = true)
 |-- Original Priority: string (nullable = true)
 |-- Priority: string (nullable = true)
 |-- Final Priority: string (nullable = true)
 |-- ALS Unit: string (nullable = true)
 |-- Call Type Group: string (nullable = true)
 |-- Number of Alarms: string (nullable = true)
 |-- Unit Type: string (nullable = true)
 |-- Unit sequence in call dispatch: string (nullable = true)
 |-- Fire Prevention District: string (nullable = true)
 |-- Supervisor District: string (nullable = true)
 |-- Neighborhooods - Analysis Boundaries: string (nullable = true)
 |-- RowID: string (nullable = true)
 |-- Analysis Neighborhoods: string (nullable = true)
 |-- X: string (nullable = true)
 |-- Y: string (nullable = true)
```

```
1 df_fire.count()
```

```
5500000
```

```
1 df_fire.head(3)
```

```
[Row(_c0='0', Call Number='211220229', Unit ID='54', Incident Number='21052265', Call Type='Medical Incident', Call Date='0
5/02/2021', Watch Date='05/01/2021', Received DtTm='05/02/2021 02:52:22 AM', Entry DtTm='05/02/2021 02:54:09 AM', Dispatch D
tTm='05/02/2021 02:54:17 AM', Response DtTm='05/02/2021 02:54:22 AM', On Scene DtTm='05/02/2021 03:03:37 AM', Transport DtTm
='05/02/2021 03:25:27 AM', Hospital DtTm='05/02/2021 03:50:51 AM', Call Final Disposition='Code 2 Transport', Available DtTm
='05/02/2021 04:22:26 AM', Address='600 Block of HURON AVE', City='San Francisco', Zipcode of Incident='94112.0', Battalion
='B09', Station Area='33.0', Box='8347', Original Priority='3', Priority='2', Final Priority='2', ALS Unit='True', Call Type
Group='Non Life-threatening', Number of Alarms='1', Unit Type='MEDIC', Unit sequence in call dispatch='2.0', Fire Prevention
District='9', Supervisor District='11', Neighborhooods - Analysis Boundaries='Outer Mission', RowID='211220229-54', Analysis
Neighborhoods='28.0', X='-122.449165', Y='37.712452'),
 Row(_c0='1', Call Number='213150159', Unit ID='T13', Incident Number='21138532', Call Type='Alarms', Call Date='11/11/202
1', Watch Date='11/10/2021', Received DtTm='11/11/2021 01:45:25 AM', Entry DtTm='11/11/2021 01:46:46 AM', Dispatch DtTm='11/
11/2021 01:47:12 AM', Response DtTm='11/11/2021 01:49:56 AM', On Scene DtTm='11/11/2021 01:54:21 AM', Transport DtTm=None, H
ospital DtTm=None, Call Final Disposition='Fire', Available DtTm='11/11/2021 02:21:37 AM', Address='200 Block of SUTTER ST',
City='San Francisco', Zipcode of Incident='94108.0', Battalion='B01', Station Area='13.0', Box='1242', Original Priority
='3', Priority='3', Final Priority='3', ALS Unit='False', Call Type Group='Alarm', Number of Alarms='1', Unit Type='TRUCK',
Unit sequence in call dispatch='1.0', Fire Prevention District='1', Supervisor District='3', Neighborhooods - Analysis Bound
aries='Financial District/South Beach', RowID='213150159-T13', Analysis Neighborhoods='8.0', X='-122.404816', Y='37.78976'),
 Row(_c0='2', Call Number='210690030', Unit ID='T03', Incident Number='21030278', Call Type='Alarms', Call Date='03/10/202
1', Watch Date='03/09/2021', Received DtTm='03/10/2021 12:16:03 AM', Entry DtTm='03/10/2021 12:18:36 AM', Dispatch DtTm='03/
10/2021 12:19:01 AM', Response DtTm='03/10/2021 12:20:15 AM', On Scene DtTm=None, Transport DtTm=None, Hospital DtTm=None, C
all Final Disposition='Fire', Available DtTm='03/10/2021 12:28:42 AM', Address='1300 Block of MARKET ST', City='San Francisc
o', Zipcode of Incident='94102.0', Battalion='B02', Station Area='36.0', Box='3111', Original Priority='3', Priority='3', Fi
nal Priority='3', ALS Unit='False', Call Type Group='Alarm', Number of Alarms='1', Unit Type='TRUCK', Unit sequence in call
dispatch='3.0', Fire Prevention District='2', Supervisor District='6', Neighborhooods - Analysis Boundaries='Tenderloin', Ro
wID='210690030-T03', Analysis Neighborhoods='36.0', X='-122.41698', Y='37.777084')]
```

```
1 df_fire.describe().toPandas().transpose()
```

	0	1	2	3	4
summary	count	mean	stddev	min	max
_c0	5500000	2749999.5	1587713.3846090317	0	999999
Call Number	5500000	1.1171316322598836E8	6.141330015101658E7	100010001	93650393
Unit ID	5500000	74.85264674903334	12.925072863751724	27	VAN9
Incident Number	5500000	1.1050175479470363E7	6151839.325678898	100000	99999
Call Type	5500000	None	None	Administrative	Watercraft in Distress
Call Date	5500000	None	None	01/01/2001	12/31/2021
Watch Date	5500000	None	None	01/01/2001	12/31/2021
Received DtTm	5500000	None	None	01/01/2001 01:00:26 PM	12/31/2021 12:59:27 AM
Entry DtTm	5500000	None	None	01/01/2001 01:01:14 PM	12/31/2021 12:58:59 PM
Dispatch DtTm	5500000	None	None	01/01/2001 01:01:27 PM	12/31/2021 12:59:12 PM
Response DtTm	5078759	None	None	01/01/2001 01:01:34 AM	12/31/2021 12:59:29 AM
On Scene DtTm	4265864	None	None	01/01/2001 01:00:12 AM	12/31/2021 12:59:29 AM
Transport DtTm	1445974	None	None	01/01/2001 01:04:56 PM	12/31/2021 12:59:36 AM
Hospital DtTm	1310526	None	None	01/01/2001 01:04:24 AM	12/31/2021 12:59:30 PM
Call Final Disposition	5500000	None	None	Against Medical Advice	Unable to Locate
Available DtTm	5422999	None	None	01/01/2001 01:00:20 PM	12/31/2021 12:59:21 PM
Address	5500000	None	None	*** 4563/ A2/ TCA7 ***	ZOE ST/WELSH ST
City	5491769	None	None	Al	Yerba Buena
Zipcode of Incident	5485528	94113.51350918271	10.230238092200716	94102.0	94158.0
Battalion	5500000	None	None	3E	B99
Station Area	5497613	18.11322276191097	14.521798458205932	01	F3
Box	5499545	4080.045146407931	2367.993415003039	0123	SS02
Original Priority	5474462	2.7603397581720324	0.5030778248373213	1	T
Priority	5499997	2.7400049409266387	0.5144285807931388	1	T
Final Priority	5500000	2.784960909090909	0.4108494991849251	2	3
ALS Unit	5500000	None	None	False	True
Call Type Group	2685056	None	None	Alarm	Potentially Life-Threatening
Number of Alarms	5500000	1.0052534545454546	0.09895105904681592	1	5
Unit Type	5500000	None	None	AIRPORT	TRUCK
Unit sequence in call dispatch	5499928	2.16049537375762	2.1665364189452347	1.0	99.0
Fire Prevention District	5500000	4.737265454951625	2.921815889747279	1	None
Supervisor District	5500000	5.978566609542313	2.7140632285139428	1	None
Neighborhooods - Analysis Boundaries	5500000	None	None	Bayview Hunters Point	Western Addition
RowID	5500000	None	None	001030101-E18	220140184-KM112
Analysis Neighborhoods	5482716	22.532300961786092	12.681710755951759	1.0	9.0
X	5499048	-122.42549044300623	0.02764388186048403	-122.33257	-122.51365
Y	5499048	37.7684561716221	0.02542008146496659	37.616882	37.854465

Stripped Year, Month from “Received DtTm”, converted X ,Y (Latitude, Longitude) values into float as they are in string format.

```
df = (df_fire.withColumn("Received DtTm", unix_timestamp("Received DtTm", "MM/dd/yyyy hh:mm:ss a").cast('timestamp'))
        .withColumn("Year", year(col("Received DtTm")))
        .withColumn("Month", month(col("Received DtTm"))).withColumn("hour", hour(col("Received DtTm"))))
```

```
df=df.withColumn('X',df['X'].cast("float").alias('X'))
df=df.withColumn('Y',df['Y'].cast("float").alias('Y'))
df=df.withColumn('Call Number',df['Call Number'].cast("int").alias('Call Number'))

```

```
df.printSchema()
```

```
root
 |-- _c0: string (nullable = true)
 |-- Call Number: integer (nullable = true)
 |-- Unit ID: string (nullable = true)
 |-- Incident Number: string (nullable = true)
 |-- Call Type: string (nullable = true)
 |-- Call Date: string (nullable = true)
 |-- Watch Date: string (nullable = true)
 |-- Received DtTm: timestamp (nullable = true)
 |-- Entry DtTm: string (nullable = true)
 |-- Dispatch DtTm: string (nullable = true)
 |-- Response DtTm: string (nullable = true)
 |-- On Scene DtTm: string (nullable = true)
 |-- Transport DtTm: string (nullable = true)
 |-- Hospital DtTm: string (nullable = true)
 |-- Call Final Disposition: string (nullable = true)
 |-- Available DtTm: string (nullable = true)
 |-- Address: string (nullable = true)
 |-- City: string (nullable = true)
 |-- Zipcode of Incident: string (nullable = true)
 |-- Battalion: string (nullable = true)
 |-- Station Area: string (nullable = true)
 |-- Box: string (nullable = true)
 |-- Original Priority: string (nullable = true)
 |-- Priority: string (nullable = true)
 |-- Final Priority: string (nullable = true)
 |-- ALS Unit: string (nullable = true)
 |-- Call Type Group: string (nullable = true)
 |-- Number of Alarms: string (nullable = true)
 |-- Unit Type: string (nullable = true)
 |-- Unit sequence in call dispatch: string (nullable = true)
 |-- Fire Prevention District: string (nullable = true)
 |-- Supervisor District: string (nullable = true)
 |-- Neighborhooods - Analysis Boundaries: string (nullable = true)
 |-- RowID: string (nullable = true)
 |-- Analysis Neighborhoods: string (nullable = true)
 |-- X: float (nullable = true)
 |-- Y: float (nullable = true)
 |-- Year: integer (nullable = true)
 |-- Month: integer (nullable = true)
 |-- hour: integer (nullable = true)
```

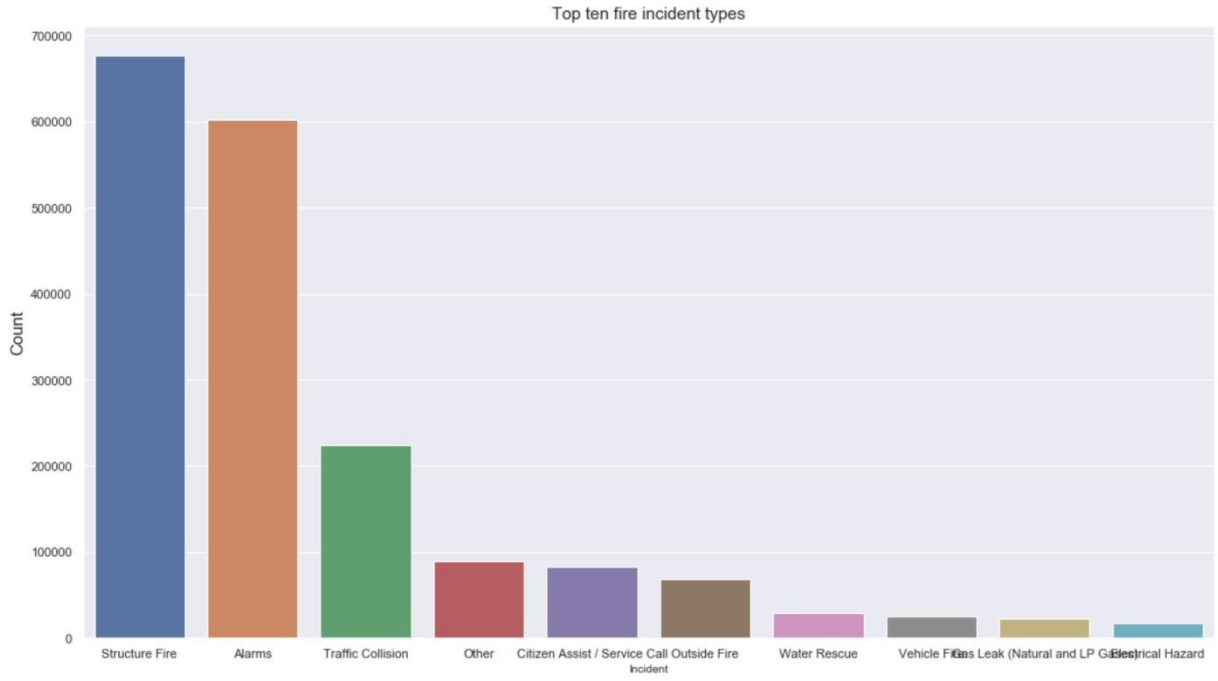
```
df = df.filter(df['Call Type'] != 'Medical Incident')
q1_result = df.groupBy('Call Type').count().orderBy('count', ascending=False)
q1_result.show()
```

```
+--------------------+------+
|           Call Type| count|
+--------------------+------+
|      Structure Fire|677523|
|              Alarms|601890|
|   Traffic Collision|224497|
|               Other| 89926|
|Citizen Assist / ...| 82344|
|        Outside Fire| 68383|
|        Water Rescue| 28625|
|        Vehicle Fire| 25555|
|Gas Leak (Natural...| 23022|
|   Electrical Hazard| 17298|
|Elevator / Escala...| 14742|
|Odor (Strange / U...| 12956|
|Smoke Investigati...| 12398|
|          Fuel Spill|  6193|
|              HazMat|  4129|
|Industrial Accidents|  3089|
|           Explosion|  2774|
|  Aircraft Emergency|  1512|
|Train / Rail Inci...|  1450|
|       Assist Police|  1438|
+--------------------+------+
only showing top 20 rows
```

```
# Show the number of incidents for different fire emergency call type
q1 = q1_result.toPandas()
print (q1)
```

```
                                Call Type   count
0                             Structure Fire  677523
1                                     Alarms  601890
2                           Traffic Collision  224497
3                                      Other   89926
4                  Citizen Assist / Service Call   82344
5                                Outside Fire   68383
6                                Water Rescue   28625
7                                Vehicle Fire   25555
8               Gas Leak (Natural and LP Gases)   23022
9                           Electrical Hazard   17298
10                 Elevator / Escalator Rescue   14742
11                    Odor (Strange / Unknown)   12956
12                 Smoke Investigation (Outside)   12398
13                               Fuel Spill    6193
14                                   HazMat    4129
15                         Industrial Accidents    3089
16                                 Explosion    2774
17                          Aircraft Emergency    1512
18                         Train / Rail Incident    1450
19                                 Assist Police    1438
20                             High Angle Rescue    1282
21                          Watercraft in Distress    1059
22  Extrication / Entrapped (Machinery, Vehicle)     804
23           Confined Space / Structure Collapse     602
24            Mutual Aid / Assist Outside Agency     542
25                                   Oil Spill     517
26                                Marine Fire     453
27                           Suspicious Package     351
28                               Administrative     294
29                            Train / Rail Fire      72
30               Lightning Strike (Investigation)       9
```

```
# Visualize top ten fire department incidents in San Francisco
import time
start_time = time.time()
fig, ax = plt.subplots(figsize=(18, 10))
sns.barplot(x='Call Type', y='count', data=q1.loc[:9, :], ax=ax)
ax.set_title('Top ten fire incident types', fontsize=15)
ax.set_xlabel('Incident', fontsize=10)
ax.set_ylabel('Count', fontsize=15)
display(fig)
print("--- %s seconds ---" % (time.time() - start_time))
```




```
--- 0.5058193206787109 seconds ---
```

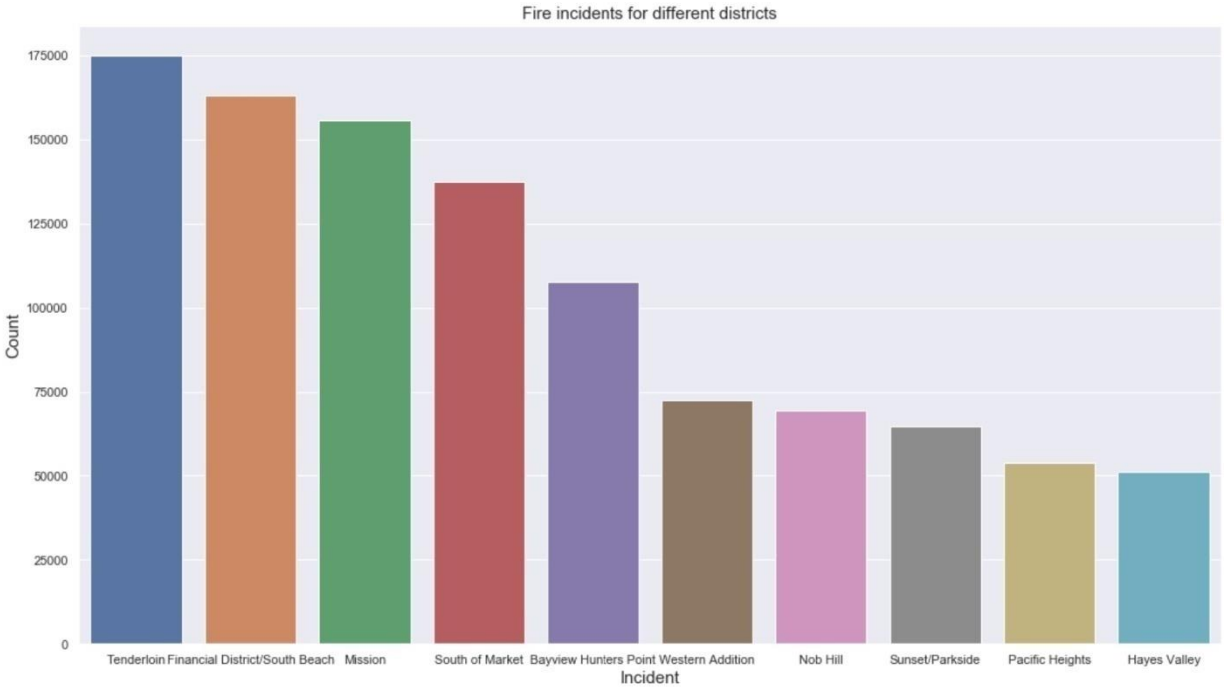
```
start_time = time.time()
q2_result = df.groupBy('Neighborhooods - Analysis Boundaries').count().orderBy('count', ascending=False)
q2_result.show()
```

```
+-----------------------------------+------+
|Neighborhooods - Analysis Boundaries| count|
+-----------------------------------+------+
|                         Tenderloin|175100|
|                Financial Distric...|163020|
|                            Mission|155679|
|                     South of Market|137541|
|                Bayview Hunters P...|107718|
|                    Western Addition| 72592|
|                           Nob Hill| 69539|
|                     Sunset/Parkside| 64556|
|                     Pacific Heights| 53743|
|                        Hayes Valley| 51032|
|                      Outer Richmond| 49847|
|                             Marina| 48679|
|                Castro/Upper Market| 45709|
|                 West of Twin Peaks| 44836|
|                           Chinatown| 44728|
|                         North Beach| 43838|
|                       Potrero Hill| 42072|
|                     Bernal Heights| 38910|
|                       Russian Hill| 38004|
|                       Inner Sunset| 35754|
+-----------------------------------+------+
only showing top 20 rows

--- 6.570775270462036 seconds ---
```

```
# The number of fire incidents for different neighborhoood
q2 = q2_result.toPandas()
print(q2)
```

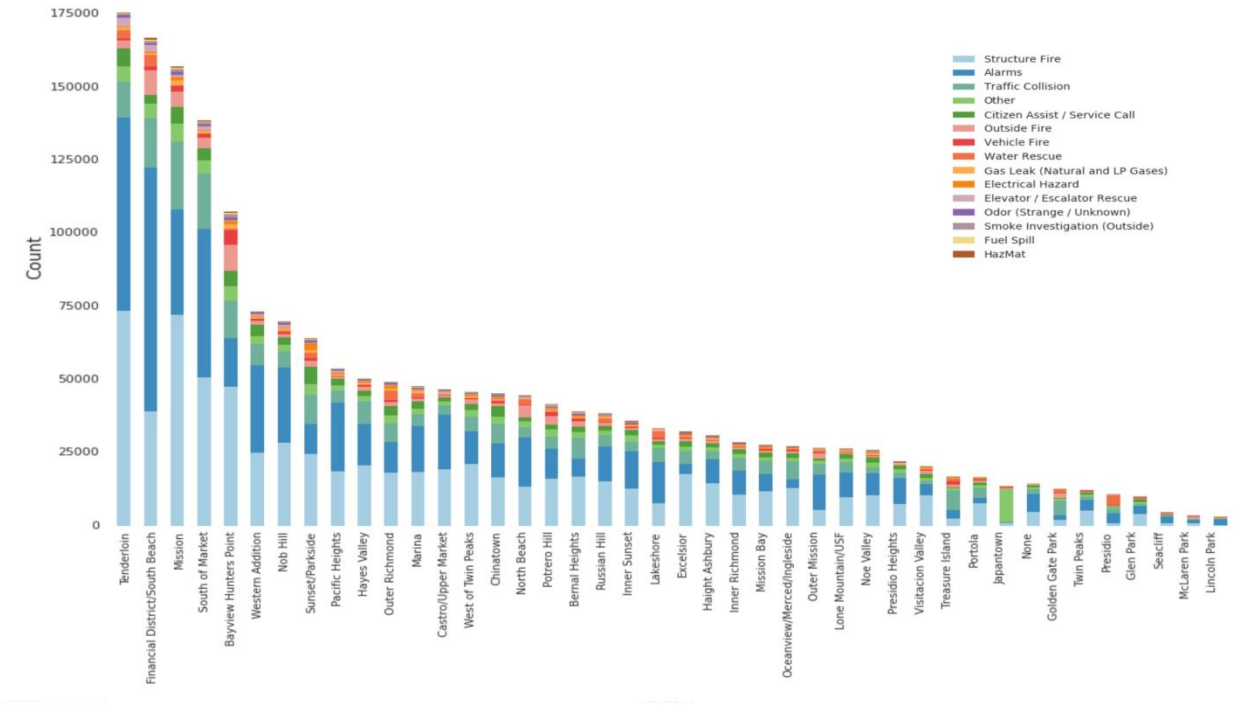
```
   Neighborhooods - Analysis Boundaries   count
0                             Tenderloin  175100
1          Financial District/South Beach  163020
2                                Mission  155679
3                         South of Market  137541
4                    Bayview Hunters Point  107718
5                         Western Addition   72592
6                                Nob Hill   69539
7                          Sunset/Parkside   64556
8                          Pacific Heights   53743
9                             Hayes Valley   51032
10                          Outer Richmond   49847
11                                  Marina   48679
12                     Castro/Upper Market   45709
13                      West of Twin Peaks   44836
14                                Chinatown   44728
15                              North Beach   43838
16                            Potrero Hill   42072
17                          Bernal Heights   38910
18                            Russian Hill   38004
19                            Inner Sunset   35754
20                               Lakeshore   34072
21                               Excelsior   31411
22                           Haight Ashbury   31064
23                           Inner Richmond   28303
24                              Mission Bay   27644
25              Oceanview/Merced/Ingleside   27227
26                            Outer Mission   26760
27                     Lone Mountain/USF   26380
28                                Noe Valley   26099
29                       Presidio Heights   21743
30                      Visitacion Valley   19876
31                        Treasure Island   17685
32                                Portola   16626
33                              Japantown   14502
34                                   None   13860
35                         Golden Gate Park   12917
36                               Twin Peaks   12316
37                                Presidio   12159
38                               Glen Park   10336
39                                Seacliff    4898
40                            McLaren Park    3543
41                            Lincoln Park    3411
--- 6.712604999542236 seconds ---
```

```
start_time = time.time()
# Visualize fire incidents for different neighborhoood
fig, ax = plt.subplots(figsize=(18, 10))
sns.barplot(x='Neighborhooods - Analysis Boundaries', y='count', data=q2.loc[:9, :], ax=ax)
ax.set_title('Fire incidents for different districts', fontsize=15)
ax.set_xlabel('Incident', fontsize=15)
ax.set_ylabel('Count', fontsize=15)
display(fig)

print("--- %s seconds ---" % (time.time() - start_time))
```



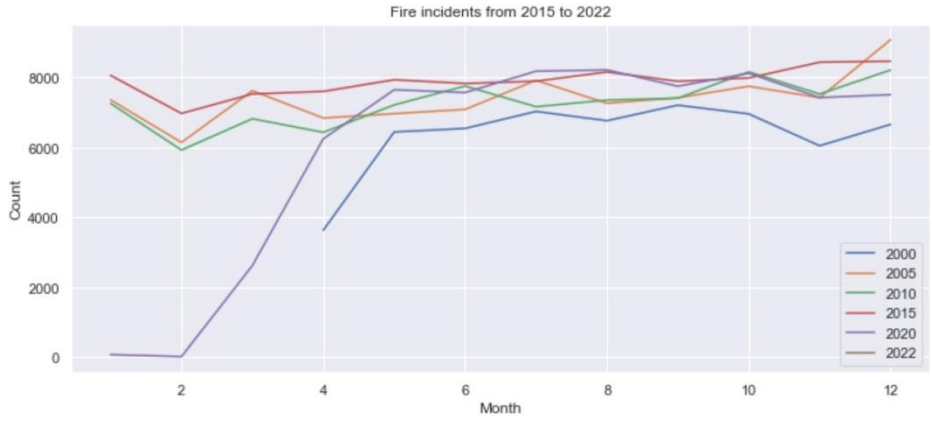

```
--- 0.5441725254058838 seconds ---
```

```
start_time = time.time()
# Visualize incidents of different categories for different district, here we take the top 15 fire emergnecy categories.
fig, ax = plt.subplots(figsize=(18, 10))
df_Pd = df.groupBy('Call Type', 'Neighborhooods - Analysis Boundaries').count()
q2_2 = df_Pd.toPandas()
table = pd.pivot_table(q2_2, values='count', index=['Neighborhooods - Analysis Boundaries'], columns=['Call Type'])
table = table[q1['Call Type'][:15].values]
order = q2['Neighborhooods - Analysis Boundaries'].values
table.loc[order].plot.bar(stacked=True, ax=ax, edgecolor = "none", cmap='tab10').legend(bbox_to_anchor=(0.95, 0.9))
ax.set_title('Fire incidents for different districts', fontsize=15)
ax.set_xlabel('Incident', fontsize=15)
ax.set_ylabel('Count', fontsize=15)
display(fig)
print("--- %s seconds ---" % (time.time() - start_time))
```

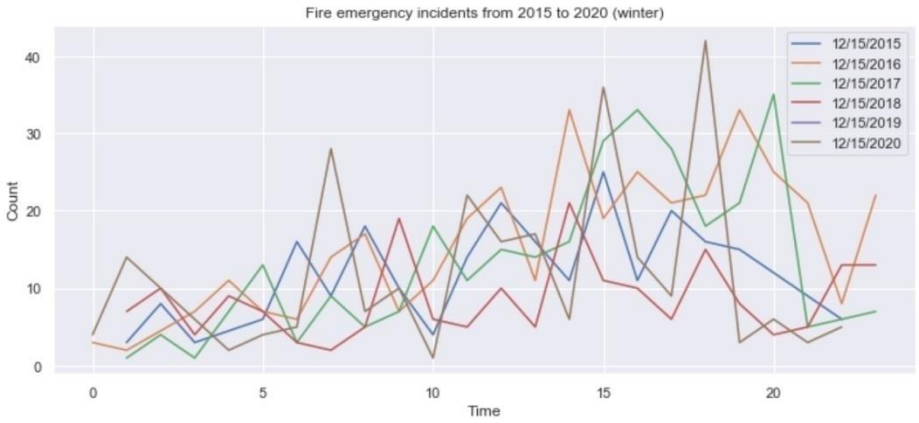



```
--- 9.23856234550476 seconds ---
```

```
start_time = time.time()
fig, ax = plt.subplots(figsize=(12, 5))
for year in ['2000', '2005', '2010', '2015','2020', '2022']:
    df_month = df.filter(df['Year'] == year).groupBy('Month').count().orderBy('Month', ascending=True)
    df_month = df_month.toPandas()
    ax.plot(df_month['Month'], df_month['count'],label = year)
    ax.set_title('Fire incidents from 2015 to 2022')
    ax.set_xlabel('Month')
    ax.set_ylabel('Count')
    ax.legend()
display(fig)

print("--- %s seconds ---" % (time.time() - start_time))
```

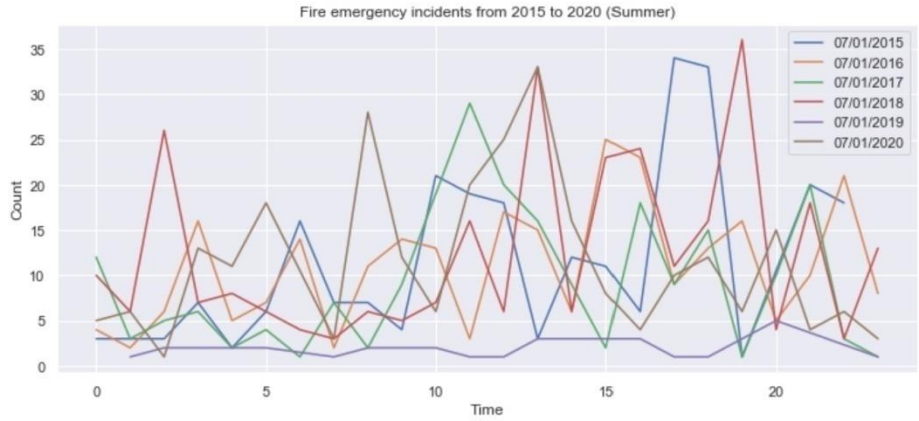


```
start_time = time.time()
fig, ax = plt.subplots(figsize=(12, 5))
for date in ['12/15/2015','12/15/2016','12/15/2017','12/15/2018', '12/15/2019', '12/15/2020']:
    df_hour = df.filter(df['Call Date'] == date).groupBy('Hour').count().orderBy('Hour', ascending=True)
    df_hour = df_hour.toPandas()
    ax.plot(df_hour['Hour'], df_hour['count'], label=date)
    ax.set_title('Fire emergency incidents from 2015 to 2020 (winter)')
    ax.set_xlabel('Time')
    ax.set_ylabel('Count')
    ax.legend()
display(fig)
print("--- %s seconds ---" % (time.time() - start_time))
```

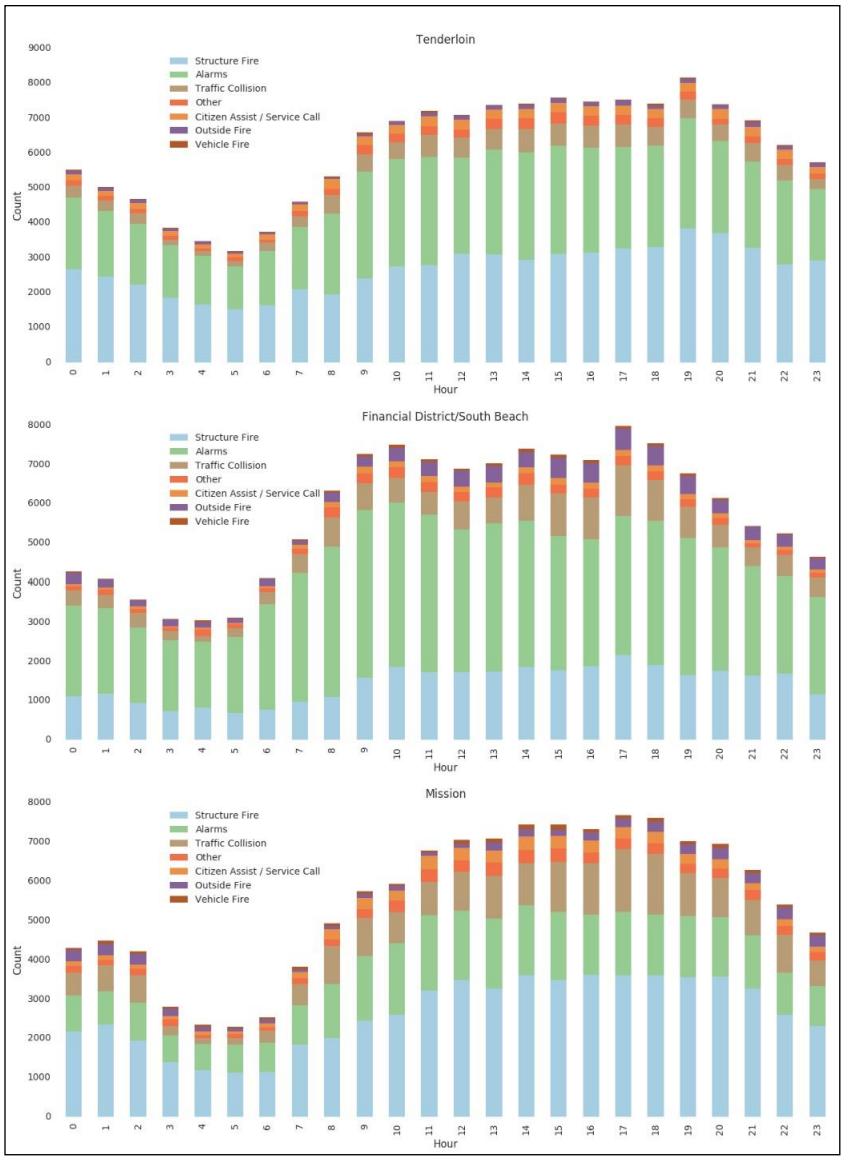



```
--- 32.33174514770508 seconds ---
```

```
start_time = time.time()
fig, ax = plt.subplots(figsize=(12, 5))
for date in ['07/01/2015','07/01/2016','07/01/2017','07/01/2018','07/01/2019','07/01/2020']:
    df_hour = df.filter(df['Call Date'] == date).groupBy('Hour').count().orderBy('Hour', ascending=True)
    df_hour = df_hour.toPandas()
    ax.plot(df_hour['Hour'], df_hour['count'], label=date)
    ax.set_title('Fire emergency incidents from 2015 to 2020 (Summer)')
    ax.set_xlabel('Time')
    ax.set_ylabel('Count')
    ax.legend()
display(fig)
print("--- %s seconds ---" % (time.time() - start_time))
```




```
start_time = time.time()
# Visualize fire related event w.r.t category and time (hour) top-3 district
fig, ax = plt.subplots(3, 1, figsize=(15, 20))
area = ['Tenderloin', 'Financial District/South Beach', 'Mission']
for i in range(3):
    df_area = df.filter(df['Neighborhooods - Analysis Boundaries'] == area[i]).groupBy('Call Type', 'Hour').count()
    q6 = df_area.toPandas()
    table = pd.pivot_table(q6, values='count', index=['Hour'], columns=['Call Type'])
    table = table[q1['Call Type'][:7].values]
    table.plot.bar(stacked=True, ax=ax[i], edgecolor = "none", cmap='Paired').legend(bbox_to_anchor=(0.35, 1))
    ax[i].set_title(area[i])
    ax[i].set_xlabel('Hour')
    ax[i].set_ylabel('Count')
display(fig)

print("--- %s seconds ---" % (time.time() - start_time))
```




```
start_time = time.time()
df_q7 = df.filter(df['Year'] != 2019).groupBy('Year', 'Call Type').count().orderBy('Call Type', 'Year', ascending=True)
df_q7 = df_q7.toPandas()
sum_year = df_q7.groupby('Year').sum()
df_q7 = df_q7.merge(sum_year, how='left', left_on='Year', right_index=True)
df_q7['percentage'] = df_q7['count_x'] / df_q7['count_y']
df_q7.head(10)
print("--- %s seconds ---" % (time.time() - start_time))
```

```
--- 10.543229341506958 seconds ---
```

```
df_q7.head(10)
```

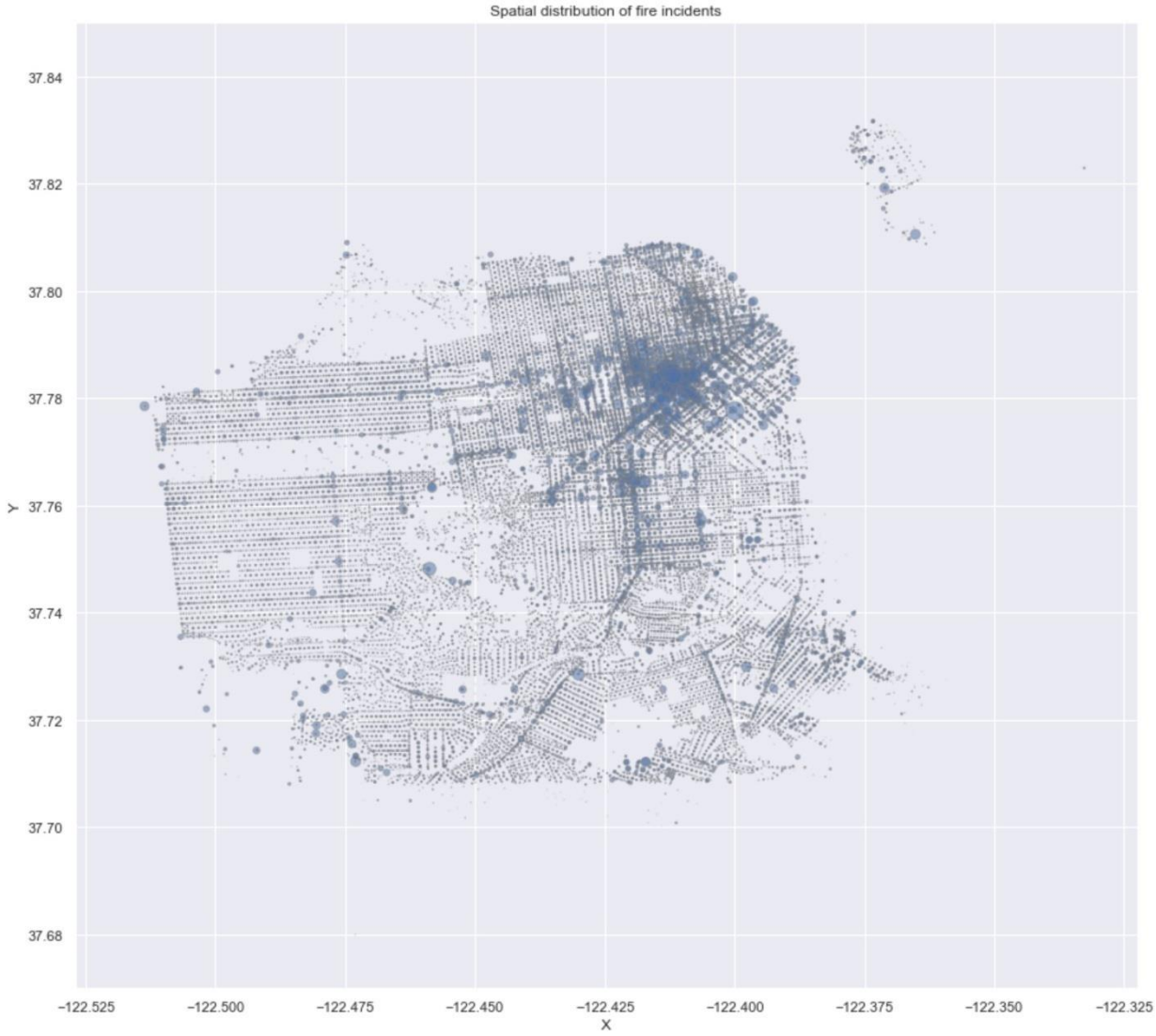
	Year	Call Type	count_x	count_y	percentage
0	2001	Administrative	4	220328	0.000018
1	2002	Administrative	9	225951	0.000040
2	2003	Administrative	2	240457	0.000008
3	2004	Administrative	2	235353	0.000008
4	2005	Administrative	93	232919	0.000399
5	2006	Administrative	15	235440	0.000064
6	2007	Administrative	6	235856	0.000025
7	2008	Administrative	29	249690	0.000116
8	2009	Administrative	16	244633	0.000065
9	2010	Administrative	6	256174	0.000023

```python
start_time = time.time()
# Visualize the percentage of different fire related incidents over time
fig, ax = plt.subplots(5, 1, figsize=(15, 20))
j = 0
for category in (q1['Call Type'][i] for i in [0, 1, 5, 6, 8]):
    df_temp = df_q7[df_q7['Call Type'] == category]
    ax[j].plot(df_temp['Year'], df_temp['percentage'],label = category)
    ax[j].set_xlabel('Year')
    ax[j].set_ylabel('Percentage')
    ax[j].legend()
    j += 1
display(fig)

print("--- %s seconds ---" % (time.time() - start_time))
```

```
start_time = time.time()
q8_result = df.groupBy('X', 'Y').count()
q8 = q8_result.toPandas()
print("--- %s seconds ---" % (time.time() - start_time))
```

```
--- 15.143091440200806 seconds ---
```

```
import time
start_time = time.time()
# Visualize the spatial distribution of fire related incidents
fig, ax = plt.subplots(figsize=(15, 14))
ax.scatter(q8['X'], q8['Y'], s=q8['count']/100, alpha=0.5, edgecolors='grey')
ax.set_title('Spatial distribution of fire incidents')
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_ylim([37.67, 37.85])
display(fig)
print("--- %s seconds ---" % (time.time() - start_time))
```




```
--- 2.7191519737243652 seconds ---
```

```
start_time = time.time()
# Perfrom clustering analysis based on the number of top 5 fire incident categories at different location.
df_cluster = df.groupBy('X', 'Y').pivot('Call Type').count()
df_cluster = df_cluster.na.fill(0)
print("--- %s seconds ---" % (time.time() - start_time))
```

```
--- 5.921852350234985 seconds ---
```

```
FEATURES_COL = ['Structure Fire', 'Alarms', 'Outside Fire', 'Vehicle Fire', 'Gas Leak (Natural and LP Gases)']
```

```
vecAssembler = VectorAssembler(inputCols=FEATURES_COL, outputCol="features")
df_kmeans = vecAssembler.transform(df_cluster).select('X', 'Y', 'features')
df_kmeans.show(20)
```

```
+-----------+---------+--------------------+
|          X|        Y|            features|
+-----------+---------+--------------------+
| -122.41486|37.782635|[287.0,836.0,3.0,...|
| -122.43595|37.795174|(5,[0,1],[25.0,44...|
|-122.398445|37.798004|           (5,[],[])|
| -122.40898|37.752003|[35.0,0.0,2.0,2.0...|
| -122.39881|37.730877|           (5,[],[])|
| -122.48355| 37.76422|(5,[0,1],[35.0,6.0])|
| -122.46798|37.786682|[10.0,27.0,0.0,2....|
|-122.444756|37.761967|      (5,[0],[10.0])|
| -122.42377|37.762375|[74.0,29.0,3.0,1....|
|-122.444756|37.788414|[35.0,40.0,1.0,0....|
| -122.40904|37.728874|           (5,[],[])|
|-122.494644|37.775894|(5,[0,2],[18.0,1.0])|
|-122.390366| 37.77106|(5,[1,4],[59.0,1.0])|
|-122.429924| 37.79041|[31.0,41.0,1.0,0....|
| -122.43959|37.796513|(5,[0,1],[25.0,44...|
| -122.44772| 37.71493|           (5,[],[])|
|  -122.4718| 37.76474|[54.0,68.0,0.0,0....|
|-122.467514|37.765877| (5,[1,2],[3.0,3.0])|
|-122.456665|37.745857|       (5,[1],[3.0])|
| -122.40976|37.763004|      (5,[1],[27.0])|
+-----------+---------+--------------------+
only showing top 20 rows
```

```
# Trains a k-means model
kmeans = KMeans().setK(5).setSeed(1).setFeaturesCol("features")
model = kmeans.fit(df_kmeans)
centers = model.clusterCenters()
```

```
transformed = model.transform(df_kmeans).select('X', 'Y', 'prediction')
rows = transformed.collect()
print(rows[:5])
```

```
[Row(X=-122.41486358642578, Y=37.78263473510742, prediction=3), Row(X=-122.
43595123291016, Y=37.79517364501953, prediction=0), Row(X=-122.398445129394
53, Y=37.798004150390625, prediction=0), Row(X=-122.40898132324219, Y=37.75
200271606445, prediction=0), Row(X=-122.39881134033203, Y=37.7308769226074
2, prediction=0)]
```

```
# Clustering result
start_time = time.time()
df_pred = sqlContext.createDataFrame(rows)
# Merge clustering results with fire events count
df_pred = df_pred.toPandas()
df_pred = df_pred.merge(q8, on=['X','Y'])
# Visulize clustering results (5 clusters)
fig, ax = plt.subplots(figsize=(6, 5))
ax.scatter(df_pred['X'], df_pred['Y'], s=df_pred['count']/100, c=df_pred['prediction'], cmap='rainbow', alpha=0.5, edgec
ax.set_title('Clustering of fire incidents')
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_ylim([37.67, 37.85])
display(fig)
print("--- %s seconds ---" % (time.time() - start_time))
```

The dot size imply the number of fire related incidents in that place, The most intensive areas of fire incidents are in the northeast of San Francisco. These areas include lots of blocks and a high density of people according to Dataset of San Fransisco, which may account for the frequent fires. They can come up with strategies strengthening or building new fire stations closer to these problematic areas, distributing current fire staff and equipment into smaller and widespread stations considering clusters in space.

V. ML CLASSIFICATION

A. Feature Engineering

Four categories of calls are identified in the dataset, null values are removed and transformed to numerical values to input into the model.

```
import pyspark.sql.functions as F
from pyspark.sql.window import Window

newdf= newdf.withColumn("Call_Type_num", F.dense_rank().over(Window.orderBy("Call Type Group")))
```

```
newdf.select('Call Type Group').distinct().show(35, False)
```

```
+---------------------------+
|Call Type Group            |
+---------------------------+
|Alarm                      |
|null                       |
|Potentially Life-Threatening|
|Non Life-threatening       |
|Fire                       |
+---------------------------+
```

```
newdf.select('Call_Type_num').distinct().show(35, False)
```

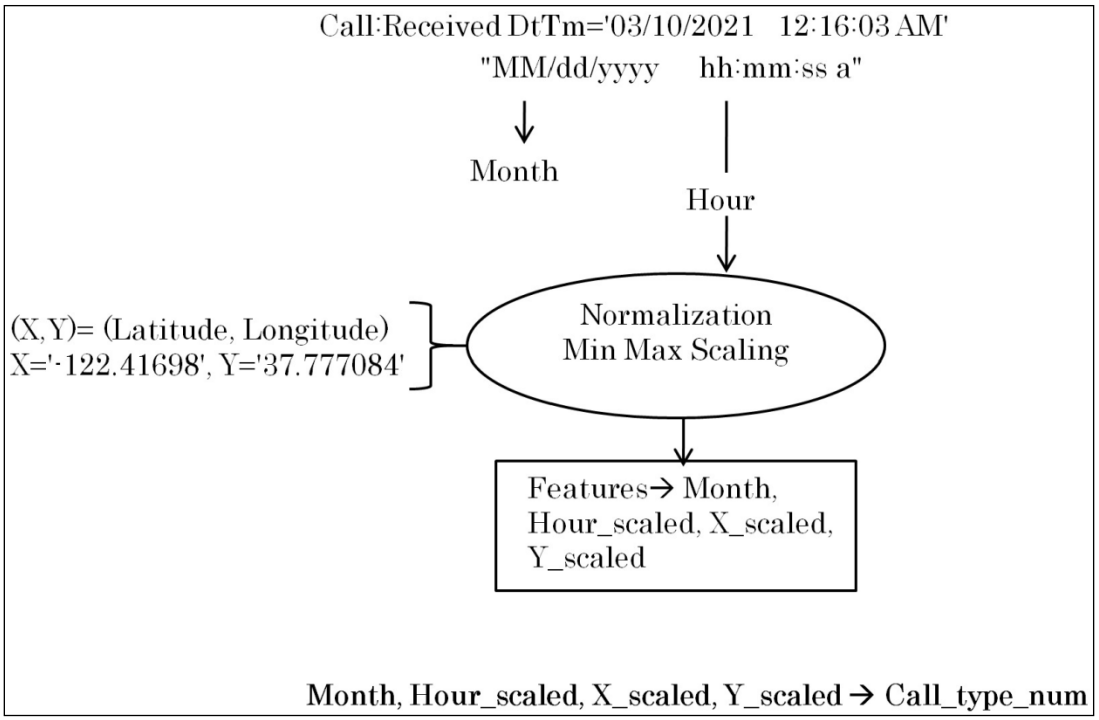
```
+-------------+
|Call_Type_num|
+-------------+
|1            |
|2            |
|3            |
|4            |
|5            |
+-------------+
```

```
import pyspark.sql.functions as F

newdf = newdf.filter(
               (F.col("Call Type Group") != "null")
               )
```

```
newdf.select('Call Type Group').distinct().show(35, False)
```

```
+---------------------------+
|Call Type Group            |
+---------------------------+
|Alarm                      |
|Potentially Life-Threatening|
|Non Life-threatening       |
|Fire                       |
+---------------------------+
```

The features used for categorising the type of calls are month, hour, lattitude, and longitude, as shown here. They are scaled using the min max normalisation technique before being fed into the model.




```
min = newdf.agg({"X": "min"}).collect()[0][0]
max = newdf.agg({"X": "max"}).collect()[0][0]

df_scaled = newdf.withColumn('X_scaled', (col('X') - min)/max)
```

```
min = newdf.agg({"Y": "min"}).collect()[0][0]
max = newdf.agg({"Y": "max"}).collect()[0][0]

df_scaled2 = df_scaled.withColumn('Y_scaled', (col('Y') - min)/max)
```

```
min = newdf.agg({"hour": "min"}).collect()[0][0]
max = newdf.agg({"hour": "max"}).collect()[0][0]

df_scaled3 = df_scaled2.withColumn('hour_scaled', (col('hour') - min)/max)
```

```
df_scaled3.printSchema()
```

```
root
 |-- _c0: string (nullable = true)
 |-- Call Number: integer (nullable = true)
 |-- Unit ID: string (nullable = true)
 |-- Incident Number: string (nullable = true)
 |-- Call Type: string (nullable = true)
 |-- Call Date: string (nullable = true)
 |-- Watch Date: string (nullable = true)
 |-- Received DtTm: timestamp (nullable = true)
 |-- Entry DtTm: string (nullable = true)
 |-- Dispatch DtTm: string (nullable = true)
 |-- Response DtTm: string (nullable = true)
 |-- On Scene DtTm: string (nullable = true)
 |-- Transport DtTm: string (nullable = true)
 |-- Hospital DtTm: string (nullable = true)
 |-- Call Final Disposition: string (nullable = true)
 |-- Available DtTm: string (nullable = true)
 |-- Address: string (nullable = true)
 |-- City: string (nullable = true)
 |-- Zipcode of Incident: string (nullable = true)
 |-- Battalion: string (nullable = true)
 |-- Station Area: string (nullable = true)
 |-- Box: string (nullable = true)
 |-- Original Priority: string (nullable = true)
 |-- Priority: string (nullable = true)
 |-- Final Priority: string (nullable = true)
 |-- ALS Unit: string (nullable = true)
 |-- Call Type Group: string (nullable = true)
 |-- Number of Alarms: string (nullable = true)
 |-- Unit Type: string (nullable = true)
 |-- Unit sequence in call dispatch: string (nullable = true)
 |-- Fire Prevention District: string (nullable = true)
 |-- Supervisor District: string (nullable = true)
 |-- Neighborhooods - Analysis Boundaries: string (nullable = true)
 |-- RowID: string (nullable = true)
 |-- Analysis Neighborhoods: string (nullable = true)
 |-- X: float (nullable = true)
 |-- Y: float (nullable = true)
 |-- Year: integer (nullable = true)
 |-- Month: integer (nullable = true)
 |-- hour: integer (nullable = true)
 |-- Call_Type_num: integer (nullable = false)
 |-- X_scaled: double (nullable = true)
 |-- Y_scaled: double (nullable = true)
 |-- hour_scaled: double (nullable = true)
```

```
from pyspark.ml.feature import VectorAssembler
vectorAssembler = VectorAssembler(inputCols = [ 'X_scaled','Y_scaled','hour_scaled','Month'], outputCol = 'features')
df_scaled= df_scaled3.fillna(0)
vfire_df = vectorAssembler.transform(df_scaled3)
vfire_df = vhouse_df.select(['features', 'Call_Type_num'])
vfire_df.show()

```

```
+--------------------+-------------+
|            features|Call_Type_num|
+--------------------+-------------+
|[-8.5435197817189...|            2|
|[-8.8965114012862...|            2|
|[-6.3675696745273...|            2|
|[-7.9023971935295...|            2|
|[-8.4144221399337...|            2|
|[-2.7135451179107...|            2|
|[-9.1952445916975...|            2|
|[-7.5768466185928...|            2|
|[-8.1063340479438...|            2|
|[-7.5768466185928...|            2|
|[-8.1225492106801...|            2|
|[-7.5768466185928...|            2|
|[-8.1568505164685...|            2|
|[-9.9218086143052...|            2|
|[-8.5952835704541...|            2|
|[-7.2943285909179...|            2|
|[-6.4885597349444...|            2|
|[-7.7352562853244...|            2|
|[-7.9653868641590...|            2|
|[-7.1801987916585...|            2|
+--------------------+-------------+
only showing top 20 rows
```

```
splits = vfire_df.randomSplit([0.7, 0.3])
train_df = splits[0]
test_df = splits[1]
```

Training:Testing = 70:30
ImageImage

```
from pyspark.ml.classification import DecisionTreeClassifier
dt = DecisionTreeRegressor(featuresCol ='features', labelCol = 'Call_Type_num')
dt_model = dt.fit(train_df)
dt_predictions = dt_model.transform(test_df)

from pyspark.ml.evaluation import MulticlassClassificationEvaluator
dt_evaluator = MulticlassClassificationEvaluator(
    labelCol="Call_Type_num", predictionCol="prediction", metricName="accuracy")
accuracy = dt_evaluator.evaluate(dt_predictions)
```

```
print(" Decision Tree Accuracy = %g" % accuracy)
```

```
Decision Tree Accuracy = 0.789421
```

```
from pyspark.ml.classification import RandomForestClassifier
rf = RandomForestClassifier(featuresCol ='features', labelCol = 'Call_Type_num')
rf_model = rf.fit(train_df)
rf_predictions = rf_model.transform(test_df)

from pyspark.ml.evaluation import MulticlassClassificationEvaluator
rf_evaluator = MulticlassClassificationEvaluator(
    labelCol="Call_Type_num", predictionCol="prediction", metricName="accuracy")
accuracy = rf_evaluator.evaluate(rf_predictions)
```

```
print("Random Forest Classifier Accuracy = %g" % accuracy)
```

```
Random Forest Classifier Accuracy = 0.846172
```

```
from pyspark.ml.classification import GBTClassifier
gbt = GBTClassifier(featuresCol ='features', labelCol = 'Call_Type_num')
gbt_model = gbt.fit(train_df)
gbt_predictions = gbt_model.transform(test_df)

from pyspark.ml.evaluation import MulticlassClassificationEvaluator
gbt_evaluator = MulticlassClassificationEvaluator(
    labelCol="Call_Type_num", predictionCol="prediction", metricName="accuracy")
accuracy = gbt_evaluator.evaluate(gbt_predictions)
```

```
print("GBTClassifier Accuracy = %g" % accuracy)
```

```
GBTClassifier Accuracy = 0.708421
```

B. Decision Tree Classifier

Decision tree classifiers (DT) [8] are well-known for providing a more complete picture of performance results. In order to achieve high accuracy, all recognised data classifiers employ improved splitting parameters and increased tree pruning approaches (ID3, C4.5), which include CART, CHAID, and QUEST, among other things. Decision trees may be subject to a number of different considerations, including robustness, scalability adaptability, and height optimization. However, decision trees, in contrast to other approaches of data categorization, provide an efficient rule collection that is easy to comprehend.

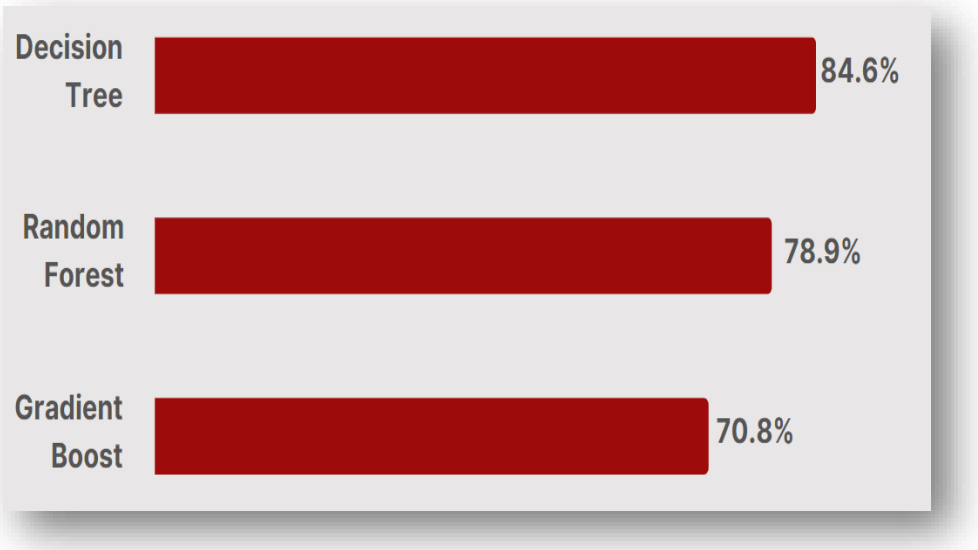
C. Random Forest Classifier

Random Forest (RF) [9] is a collection or ensemble of Classification and Regression Trees (CART) that have been trained using datasets from the training set. These datasets, known as bootstraps, are constructed by performing a random resampling on the training set. Afterwards, when the tree has been created, a collection of bootstraps which does not contain any specific record from the original dataset, is used as the test set. For the purpose of classifying incoming input data, each individual CART tree votes for one class, with the forest predicting the class that receives the most number of votes. RF follows a set of rules for tree growing, tree combination, self-testing, and post-processing; it is resistant to overfitting; and it is considered more stable in the presence of outliers and in very high-dimensional parameter spaces than other machine learning algorithms.

D. Gradient Boosted Classifier

GBDT (gradient boosting decision tree) is a machine learning technique that is frequently utilised owing to its efficiency, accuracy, and interpretability. GBDT delivers state-of-the-art performance in a wide range of machine learning problems, including multi-class classification [10], which is a common example. Recently, as a result of the growth of large amounts of data GBDT has faced new obstacles, particularly in the area of the tradeoff between accuracy and efficiency. Conventional GBDT implementations must scan all of the data instances for each feature in order to estimate the information gain of all of the potential split points, which is time-consuming. So their computational complexity will be proportional to both the number of features and the number of instances in respective datasets. When dealing with large amounts of data, this makes these methods very time-consuming. One basic approach to addressing this difficulty is to minimise the number of data instances and the number of features available for use. However, it turns out that this is a really difficult problem to solve.

VI. CONCLUSION



Using over 55 million fire department data in the last two decades, we presented spatial distribution of fire-related accidentst, variation of different types of fire accidents, identified vulnerable neighborhoods and created 3 ML models that can learn to predict a life threatening call based on location and time. Using the Decision Tree approach, we were able to classify types of fire department calls more accurately than the Random Forest and Gradient Boost Tree classifiers. Cross-featuring and other improvisation techniques might help to increase the accuracy of the model even further. When it comes to analysing this enormous dataset, Spark has significantly accelerated the process, with nearly all analysis taking just a few seconds to complete. The Spark MLlib framework for constructing machine learning pipelines makes it straightforward to perform feature extraction, selection, and transformations on the dataset. Using the Spark MLlib data pipeline, Custom pipelines can be easily added and removed from the system, which contains distributed implementations of clustering and classification algorithms

References

[1] Fire in the United States 2008-2017. U.S.Fire Administration. November 2019.

[2] Corcoran J, Higgs G, Brunsdon C, Ware A, Norman P. The use of spatial analytical techniques to explore patterns of fire incidence: A South Wales case study. Computers, Environment and Urban Systems. 2007; 31(6):623-47. doi:10.1016/j.compenvurbsys.2007.01.002.

[3] Plug C, Xia J, Caulfield C. Spatial and temporal visualisation techniques for crash analysis. Accident Analysis and Prevention. 2011; 43(6):1937-46. doi:10.1016/j.aap.2011.05.007.

[4] Soltani A, Askari S. Exploring spatial autocorrelation of traffic crashes based on severity. Injury.48(3):637-47

[5] Wuschke K, Clare J, Garis L. Temporal and geographic clustering of residential structure fires: A theoretical platform for targeted fire prevention. Fire Safety Journal. 2013;62(PART A):3-12. doi:10.1016/j.firesaf.2013.07.003.

[6] Asgary A, Ghaffari A, Levy J. Spatial and temporal analyses of structural fire incidents and their causes: A case of Toronto, Canada.

[7] https://data.sfgov.org/Public-Safety/Fire-Incidents/wr8u-ric/data.

[8] https://www.researchgate.net/publication/350386944_Classification_Based_on_Decision_Tree_Algorithm_for_Machine_Learning

[9] https://www.frontiersin.org/articles/10.3389/fnagi.2017.00329/full

[10] https://proceedings.neurips.cc/paper/2017/file/6449f44a102fde848669bdd9eb6b76fa-Paper.pdf