

Data Engineering with Spark Databricks

Summary

1	Version	3
2	Introduction	4
3	Data Sourcing/Extraction	5
4	Data Transformation and Data Analysis.....	9
5	Data Loading.....	14
6	Conclusion	16
7	References.....	17

1 Version

This document was created by Cleber Zumba de Souza and can be distributed freely, as long as the source is mentioned.

Versão	Ação	Data
1.0	Document creation	21/06/2024

2 Introduction

San Francisco Fire Calls ETL and Analysis

SUMMARY

Fire Calls-For-Service includes all fire units' responses to 911 calls from the city's Computer-Aided Dispatch ("CAD") system. This includes responses to Medical Incidents requiring EMS staff. 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 involve multiple units, there are multiple records for each call number. Addresses are associated with an intersection or call box, not a specific address.

HOW TO USE THIS DATASET

This dataset is based on responses, and since most calls involve multiple units, there are multiple records for each call number. The most common call types are Medical Incidents, Alarms, Structure Fires, and Traffic Collisions.

ETL Process

```
graph LR;
    Source[Source Systems] -- Extract --> Transform[Transform];
    Transform -- Load --> Destination[Destination];
```

- This pipeline uses the San Francisco Fire Department's call event dataset and demonstrates:
 - End-to-end Data Engineering pipeline covers the extraction, transformation and loading (ETL) steps of large volumes of data, using PySpark for transformation and Spark SQL for queries. Caching techniques were implemented to optimize query performance, and data analysis was conducted to gain insights.
 - How to answer questions by analyzing data using Spark SQL
- Benefits of the Techniques Used:
 - Partitioning: Improves data reading and writing by dividing data into smaller, more manageable partitions.
 - Spark Settings: Tweaks like `spark.sql.shuffle.partitions` and `spark.sql.autoBroadcastJoinThreshold` help optimize shuffle and join operations.
 - Parquet Format: Parquet format storage improves reading and writing performance due to its columnar nature and support compression.
 - Cache: Caching frequently used DataFrames reduces subsequent data reading time.
 - Integrated Analysis: Analysis can be performed directly in Databricks, with integrated visualizations for easy interpretation of the results.
 - Using Databricks and Spark allows the pipeline to easily scale to large volumes of data.

3 Data Sourcing/Extraction

Dataset has been downloaded from [San Francisco Fire Department Calls](#)

The screenshot shows the DataSF website interface. The main header includes the DataSF logo and navigation links: OPEN DATA, SHOWCASE, PUBLISHING, ACADEMY, RESOURCES, and BLOG. Below the header, there are tabs for Explore, Browse Data, and Developers, along with a search bar and a Sign In button. The main content area displays the dataset 'Fire Department and Emergency Medical Services Dispatched Calls for Service' under the 'Public Safety' category. A summary section provides a brief description of the dataset, which includes fire units' responses to 911 calls and medical incidents requiring EMS staff. The 'About this Dataset' section contains metadata such as the update date (June 21, 2024), data last updated date (June 21, 2024), metadata last updated date (June 21, 2024), date created (December 17, 2015), views (94.3K), and downloads (35.6K). A 'Department Metrics' table shows the publishing department as 'Fire Department'. A 'Detailed Descriptive' table lists geographic unit and latitude/longitude. A 'Publishing Details' table shows publishing frequency as 'Daily', data change frequency as 'Daily', program link, is self-service as 'Yes', and geographic unit as 'Intersection or street segment'.

Department Metrics	
Publishing Department	Fire Department

Detailed Descriptive	
Geographic unit	Latitude/longitude

Publishing Details	
Publishing frequency	Daily
Data change frequency	Daily
Program link	
Is self-service	Yes
Geographic unit	Intersection or street segment

To load your data into DBFS, please refer to [Databricks Guide > Importing Data](#).

The screenshot shows the Databricks documentation page for 'Explore and create tables in DBFS'. The page is dated May 17, 2024. It features a sidebar with navigation links such as 'Databricks on AWS', 'Get started', 'What is Databricks?', 'DatabricksIQ', 'Release notes', 'Connect to data sources', 'Connect to compute', 'Discover data', 'Query data', 'Ingest data', 'Transform data', 'Monitor data and AI assets', 'Share data (Delta Sharing)', 'Databricks Marketplace', 'Data engineering', 'Generative AI & LLMs', 'Machine learning', 'Model serving', 'Data warehousing', 'Delta Lake', 'Developers', and 'Technology partners'. The main content area includes an 'Important!' notice stating that the documentation is retired and might not be updated. It also provides instructions on how to access the legacy DBFS file upload and table creation UI through the 'add data UI' or by clicking 'New > Data > DBFS'. A 'Note' section mentions that the availability of some elements varies based on workspace configurations. The page also includes a 'Neste artigo:' section with links to 'Import data', 'Create a table', 'View databases and tables', 'View table details', and 'Delete a table using the UI'.

The screenshot shows the 'Create New Table' interface in the Databricks workspace. The left sidebar contains navigation links: 'New', 'Workspace', 'Recents', 'Search', 'Catalog', 'Workflows', 'Compute', 'Machine Learning', and 'Experiments'. The main content area is titled 'Create New Table' and shows the 'Data source' as 'Upload File'. The 'DBFS Target Directory' is set to '/FileStore/tables/' with an optional field for a file name. A 'Select' button is available. Below this, a list of files is shown, including 'sf-fire-calls.csv' with a size of 1.1 GB and a 'Remove file' button. A green checkmark indicates that the file has been successfully uploaded to '/FileStore/tables/sf_fire_calls.csv'. At the bottom, there are two buttons: 'Create Table with UI' and 'Create Table in Notebook'.

The screenshot shows the Databricks interface for a notebook titled "San Francisco Fire Calls". The notebook is in Python mode. The first cell, labeled "1. Data Sourcing/Extraction", contains the following code:

```
dbutils.fs.mkdirs("/data/sf_fire")
dbutils.fs.mkdirs("/data/sf_fire/input")
dbutils.fs.mkdirs("/data/sf_fire/interim")
```

The output of this cell shows that the directories were successfully created. Below the code, there is a table view showing the contents of the file system:

path	name	size	modificationTime
dbfs:/FileStore/tables/sf_fire_calls.csv	sf_fire_calls.csv	1137925359	1718903006000

The table indicates that the file "sf_fire_calls.csv" has been successfully uploaded to the file system.

The screenshot shows the Databricks interface for the same notebook, now displaying the second cell. The code in this cell is as follows:

```
# Path to the newly uploaded file
uploaded_file_path = "/FileStore/tables/sf_fire_calls.csv"

# Path to input directory
input_dir = "/data/sf_fire/input"

# Move the file to the input directory
dbutils.fs.mv(uploaded_file_path, input_dir)

# List files in directories to check if the file has been moved/copied
dbutils.fs.ls(input_dir)
```

The output of this cell shows the result of moving the file to the input directory. Below the code, there is a table view showing the contents of the file system:

path	name	size	modificationTime
dbfs:/data/sf_fire/input/sf_fire_calls.csv	sf_fire_calls.csv	1137925359	1718903211000

The table indicates that the file "sf_fire_calls.csv" has been successfully moved to the input directory.

San Francisco Fire Calls Python

Inspect the data looks like before defining a schema

```
%fs head /data/sf_fire/input/sf_fire_calls.csv
```

[Truncated to first 65536 bytes]

Call Number,Unit ID,Incident Number,Call Type,Call Date,Watch Date,Call Final Disposition,Available DTM,Address,City,Zipcode of Incident,Battalion,Station Area,Box,OrigPriority,Priority,Final Priority,ALS Unit,Call Type Group,NumAlarms,UnitType,Unit sequence in Call dispatch,Fire Prevention District,Supervisor District,Neighborhood,Location,RowID,Delay

20110014,M29,2003234,Medical Incident,01/11/2002,01/10/2002,Other,01/11/2002 01:58:43 AM,10TH ST/HARKEY ST,SF,94103,802,36,2338,1,1,2,true,"",1,MEDIC,1,2,6,Tenderloin,"(37.7765408927181, -122.417501464907)",020110014-M29,5,2333333333333333

20110015,M08,2003233,Medical Incident,01/11/2002,01/10/2002,Other,01/11/2002 02:10:17 AM,300 Block of STH ST,SF,94107,803,08,2243,1,1,2,true,"",1,MEDIC,1,3,6,South of Market,"(37.7792841462441, -122.402061300134)",020110015-M08,3,0833333333333333

20110016,B02,2003235,Structure Fire,01/11/2002,01/10/2002,Other,01/11/2002 01:47:00 AM,2000 Block of CALIFORNIA ST,SF,94109,804,38,3362,3,3,3,false,"",1,CHIEF,6,4,5,Pacific Heights,"(37.7895840679362, -122.428071912459)",020110016-B02,3,05

20110016,B04,2003235,Structure Fire,01/11/2002,01/10/2002,Other,01/11/2002 01:51:54 AM,2000 Block of CALIFORNIA ST,SF,94109,804,38,3362,3,3,3,false,"",1,CHIEF,3,4,5,Pacific Heights,"(37.7895840679362, -122.428071912459)",020110016-B04,2,3166666666666667

20110016,D2,2003235,Structure Fire,01/11/2002,01/10/2002,Other,01/11/2002 01:47:00 AM,2000 Block of CALIFORNIA ST,SF,94109,804,38,3362,3,3,3,false,"",1,CHIEF,4,4,5,Pacific Heights,"(37.7895840679362, -122.428071912459)",020110016-D2,3,0166666666666667

20110016,E03,2003235,Structure Fire,01/11/2002,01/10/2002,Other,01/11/2002 01:47:00 AM,2000 Block of CALIFORNIA ST,SF,94109,804,38,3362,3,3,3,false,"",1,ENGINE,7,4,5,Pacific Heights,"(37.7895840679362, -122.428071912459)",020110016-E03,2,6833333333333333

20110016,E38,2003235,Structure Fire,01/11/2002,01/10/2002,Other,01/11/2002 01:51:17 AM,2000 Block of CALIFORNIA ST,SF,94109,804,38,3362,3,3,3,false,"",1,ENGINE,1,4,5,Pacific Heights,"(37.7895840679362, -122.428071912459)",020110016-E38,2,1

20110016,E41,2003235,Structure Fire,01/11/2002,01/10/2002,Other,01/11/2002 01:47:00 AM,2000 Block of CALIFORNIA ST,SF,94109,804,38,3362,3,3,3,false,"",1,ENGINE,8,4,5,Pacific Heights,"(37.7895840679362, -122.428071912459)",020110016-E41,2,7166666666666667

20110016,M03,2003235,Structure Fire,01/11/2002,01/10/2002,Other,01/11/2002 01:46:38 AM,2000 Block of CALIFORNIA ST,SF,94109,804,38,3362,3,3,3,true,"",1,M

San Francisco Fire Calls Python

```
# Importando pacotes necessários
from pyspark.sql.types import *
from pyspark.sql.functions import *

# Definindo o schema
fire_schema = StructType([StructField('CallNumber', IntegerType(), True),
                             StructField('UnitID', StringType(), True),
                             StructField('IncidentNumber', IntegerType(), True),
                             StructField('CallType', StringType(), True),
                             StructField('CallDate', StringType(), True),
                             StructField('WatchDate', StringType(), True),
                             StructField('CallFinalDisposition', StringType(), True),
                             StructField('AvailableDTM', StringType(), True),
                             StructField('Address', StringType(), True),
                             StructField('City', StringType(), True),
                             StructField('Zipcode', IntegerType(), True),
                             StructField('Battalion', StringType(), True),
                             StructField('StationArea', StringType(), True),
                             StructField('Box', StringType(), True),
                             StructField('OriginalPriority', StringType(), True),
                             StructField('Priority', StringType(), True),
                             StructField('FinalPriority', IntegerType(), True),
                             StructField('ALSUnit', BooleanType(), True),
                             StructField('CallTypeGroup', StringType(), True),
                             StructField('NumAlarms', IntegerType(), True),
                             StructField('UnitType', StringType(), True),
                             StructField('UnitSequenceInCallDispatch', IntegerType(), True),
                             StructField('FirePreventionDistrict', StringType(), True),
                             StructField('SupervisorDistrict', StringType(), True),
                             StructField('Neighborhood', StringType(), True),
                             StructField('Location', StringType(), True),
                             StructField('RowID', StringType(), True),
                             StructField('Delay', FloatType(), True)])
```

San Francisco Fire Calls Python

```
fire_df = spark.read.csv("/data/sf_fire/input/sf_fire_calls.csv",
                          schema=fire_schema,
                          header=True,
                          ignoreLeadingWhiteSpace=True,
                          ignoreTrailingWhiteSpace=True)
```

fire_df: pyspark.sql.dataframe.DataFrame = [CallNumber: integer, UnitID: string ... 26 more fields]

```
display(fire_df.limit(10))
```

(1) Spark Jobs

Table	CallNumber	UnitID	IncidentNumber	CallType	CallDate	WatchDate	CallFinalDisposition	AvailableDTM
1	20110014	M29	2003234	Medical Incident	01/11/2002	01/10/2002	Other	01/11/2002 01:58
2	20110015	M08	2003233	Medical Incident	01/11/2002	01/10/2002	Other	01/11/2002 02:10
3	20110016	B02	2003235	Structure Fire	01/11/2002	01/10/2002	Other	01/11/2002 01:47
4	20110016	B04	2003235	Structure Fire	01/11/2002	01/10/2002	Other	01/11/2002 01:51
5	20110016	D2	2003235	Structure Fire	01/11/2002	01/10/2002	Other	01/11/2002 01:47
6	20110016	E03	2003235	Structure Fire	01/11/2002	01/10/2002	Other	01/11/2002 01:47
7	20110016	E38	2003235	Structure Fire	01/11/2002	01/10/2002	Other	01/11/2002 01:51
8	20110016	E41	2003235	Structure Fire	01/11/2002	01/10/2002	Other	01/11/2002 01:47
9	20110016	M03	2003235	Structure Fire	01/11/2002	01/10/2002	Other	01/11/2002 01:46

10 rows | 4.74 seconds runtime

Refreshed 19 hours ago

4 Data Transformation and Data Analysis

San Francisco Fire Calls Python File Edit View Run Help Last edit was 15 hours ago Run all Terminated Share Publish

2. Data Transformation and Data Analysis

Cache the DataFrame since we will be performing some operations on it.

```
fire_df.cache()
```

Out[7]: DataFrame[callNumber: int, UnitID: string, IncidentNumber: int, CallType: string, CallDate: string, WatchDate: string, CallFinalDisposition: string, AvailableDTM: string, Address: string, City: string, Zipcode: int, Battalion: string, StationArea: string, Box: string, OriginalPriority: string, Priority: string, FinalPriority: int, ALSUnit: boolean, CallTypeGroup: string, NumAlarms: int, UnitType: string, UnitSequenceInCallDispatch: int, FirePreventionDistrict: string, SupervisorDistrict: string, Neighborhood: string, Location: string, RowID: string, Delay: float]

```
fire_df.count()
```

(2) Spark Jobs

Out[8]: 4388660

San Francisco Fire Calls Python File Edit View Run Help Last edit was 15 hours ago Run all Terminated Share Publish

```
fire_df.printSchema()
```

```
-- Address: string (nullable = true)
-- City: string (nullable = true)
-- Zipcode: integer (nullable = true)
-- Battalion: string (nullable = true)
-- StationArea: string (nullable = true)
-- Box: string (nullable = true)
-- OriginalPriority: string (nullable = true)
-- Priority: string (nullable = true)
-- FinalPriority: integer (nullable = true)
-- ALSUnit: boolean (nullable = true)
-- CallTypeGroup: string (nullable = true)
-- NumAlarms: integer (nullable = true)
-- UnitType: string (nullable = true)
-- UnitSequenceInCallDispatch: integer (nullable = true)
-- FirePreventionDistrict: string (nullable = true)
-- SupervisorDistrict: string (nullable = true)
-- Neighborhood: string (nullable = true)
-- Location: string (nullable = true)
-- RowID: string (nullable = true)
-- Delay: float (nullable = true)
```

San Francisco Fire Calls Python File Edit View Run Help Last edit was 15 hours ago Run all Terminated Share Publish

Filter out "Medical Incident" call types

`filter()` and `where()` methods on the DataFrame are similar.

```
few_fire_df = (fire_df.select("IncidentNumber", "AvailableDTM", "CallType")
               .where(col("CallType") == "Medical Incident"))

few_fire_df.show(5, truncate=False)
```

(1) Spark Jobs

```
few_fire_df: pyspark.sql.dataframe.DataFrame = [(IncidentNumber: integer, AvailableDTM: string ... 1 more field)]
```

IncidentNumber	AvailableDTM	CallType
2003234	01/11/2002 01:58:43 AM	Medical Incident
2003233	01/11/2002 02:10:17 AM	Medical Incident
2003236	01/11/2002 02:27:14 AM	Medical Incident
2003238	01/11/2002 02:28:30 AM	Medical Incident
2003240	01/11/2002 02:17:23 AM	Medical Incident

only showing top 5 rows

San Francisco Fire Calls Python File Edit View Run Help Last edit was 15 hours ago Run all Terminated Share Publish

1) How many distinct types of calls were made to the Fire Department?

To be sure, let's not count "null" strings in that column.

```
fire_df.select("CallType").where(col("CallType").isNotNull()).distinct().count()
```

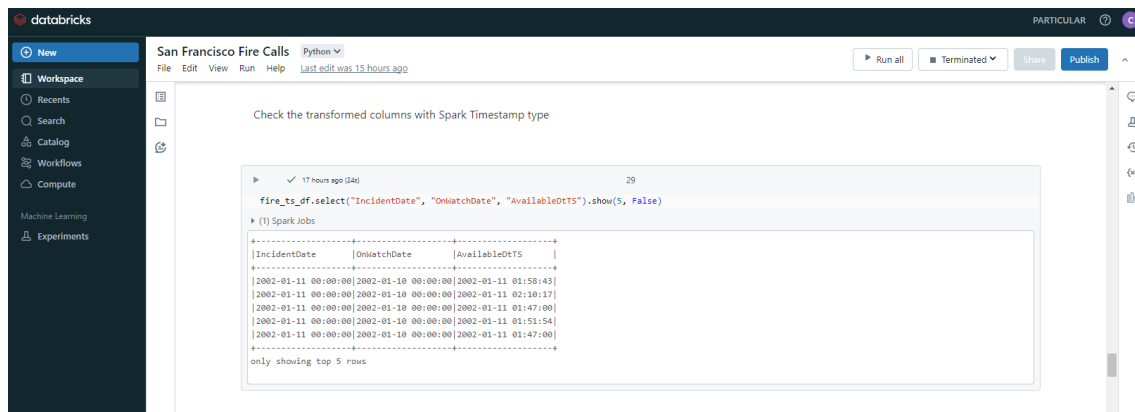
(3) Spark Jobs

Out[5]: 32

The screenshot shows a Databricks workspace with a notebook titled "San Francisco Fire Calls". The notebook is in Python mode and has a menu bar with File, Edit, View, Run, and Help. The "Run" button is highlighted. The notebook content includes a question: "2) What are distinct types of calls were made to the Fire Department?". Below the question, it says "These are all the distinct type of call to the SF Fire Department". A code cell shows a Spark SQL query: `fire_df.select("CallType").where(col("CallType").isNotNull()).distinct().show(10,False)`. The output shows a list of distinct call types: Alarms, Odor (Strange / Unknown), Citizen Assist / Service Call, Vehicle Fire, Other, Outside Fire, Electrical Hazard, Structure Fire, Medical Incident, and Fuel Spill. The output is truncated with "only showing top 10 rows".

The screenshot shows a Databricks workspace with a notebook titled "San Francisco Fire Calls". The notebook is in Python mode and has a menu bar with File, Edit, View, Run, and Help. The "Run" button is highlighted. The notebook content includes a question: "3) Which all response or delayed times greater than 5 mins?". Below the question, it lists three steps: 1. Rename the column Delay -> ReponseDelayedInMins, 2. Returns a new DataFrame, 3. Find out all calls where the response time to the fire site was delayed for more than 5 mins. A code cell shows a Spark SQL query: `new_fire_df = fire_df.withColumnRenamed("Delay", "ResponseDelayedInMins")` and `new_fire_df.select("ResponseDelayedInMins").where(col("ResponseDelayedInMins") > 5).show(5, False)`. The output shows a list of response times: 5.233333, 6.933333, 6.116667, 7.85, and 77.333336. The output is truncated with "only showing top 5 rows".

The screenshot shows a Databricks workspace with a notebook titled "San Francisco Fire Calls". The notebook is in Python mode and has a menu bar with File, Edit, View, Run, and Help. The "Run" button is highlighted. The notebook content includes three steps: 1. Transform the string dates to Spark Timestamp data type so we can make some time-based queries later, 2. Returns a transformed query, 3. Cache the new DataFrame. A code cell shows a Spark SQL query: `fire_ts_df = (new_fire_df.withColumn("IncidentDate", to_timestamp(col("CallDate"), "MM/dd/yyyy")).drop("CallDate").withColumn("OnWatchDate", to_timestamp(col("WatchDate"), "MM/dd/yyyy")).drop("WatchDate").withColumn("AvailableDtTs", to_timestamp(col("AvailableDtTs"), "MM/dd/yyyy hh:mm:ss a")).drop("AvailableDtTs"))`. The output shows a list of columns: CallNumber, UnitID, IncidentNumber, CallType, CallFinalDisposition, Address, City, Zipcode, Battalion, StationArea, Box, OriginalPriority, Priority, FinalPriority, ALSUnit, CallTypeGroup, and NumAlarms. The output is truncated with "only showing top 10 rows".



San Francisco Fire Calls Python

File Edit View Run Help Last edit was 15 hours ago

Run all Terminated Share Publish

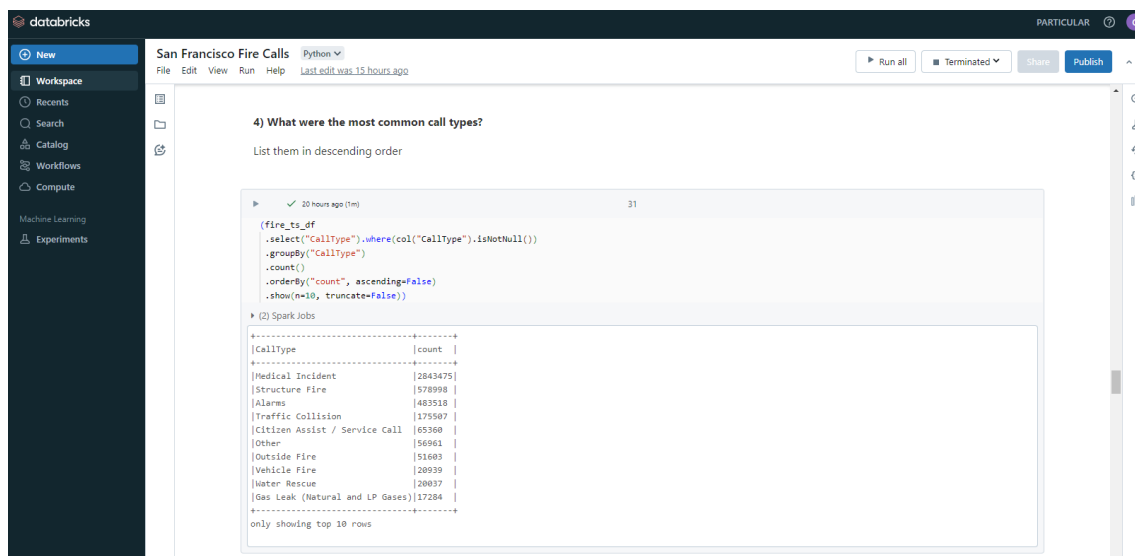
Check the transformed columns with Spark Timestamp type

```
fire_ts_df.select("IncidentDate", "OnWatchDate", "AvailableDTS").show(5, False)
```

(1) Spark Jobs

IncidentDate	OnWatchDate	AvailableDTS
2002-01-11 00:00:00	2002-01-10 00:00:00	2002-01-11 01:58:43
2002-01-11 00:00:00	2002-01-10 00:00:00	2002-01-11 02:10:17
2002-01-11 00:00:00	2002-01-10 00:00:00	2002-01-11 01:47:00
2002-01-11 00:00:00	2002-01-10 00:00:00	2002-01-11 01:51:54
2002-01-11 00:00:00	2002-01-10 00:00:00	2002-01-11 01:47:00

only showing top 5 rows



San Francisco Fire Calls Python

File Edit View Run Help Last edit was 15 hours ago

Run all Terminated Share Publish

4) What were the most common call types?

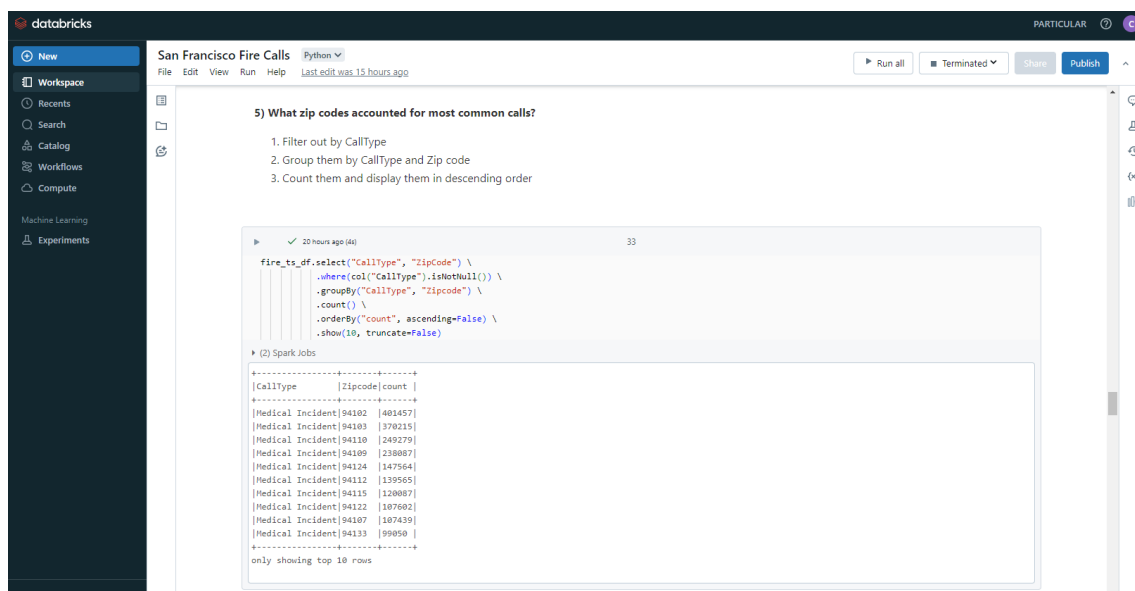
List them in descending order

```
(fire_ts_df.select("CallType").where(col("CallType").isNotNull()).groupBy("CallType").count().orderBy("count", ascending=False).show(n=10, truncate=False))
```

(2) Spark Jobs

CallType	count
Medical Incident	2843475
Structure Fire	578998
Alarms	483518
Traffic Collision	175507
Citizen Assist / Service Call	65360
Other	56961
Outside Fire	51603
Vehicle Fire	20939
Water Rescue	20037
Gas Leak (Natural and LP Gases)	17284

only showing top 10 rows



San Francisco Fire Calls Python

File Edit View Run Help Last edit was 15 hours ago

Run all Terminated Share Publish

5) What zip codes accounted for most common calls?

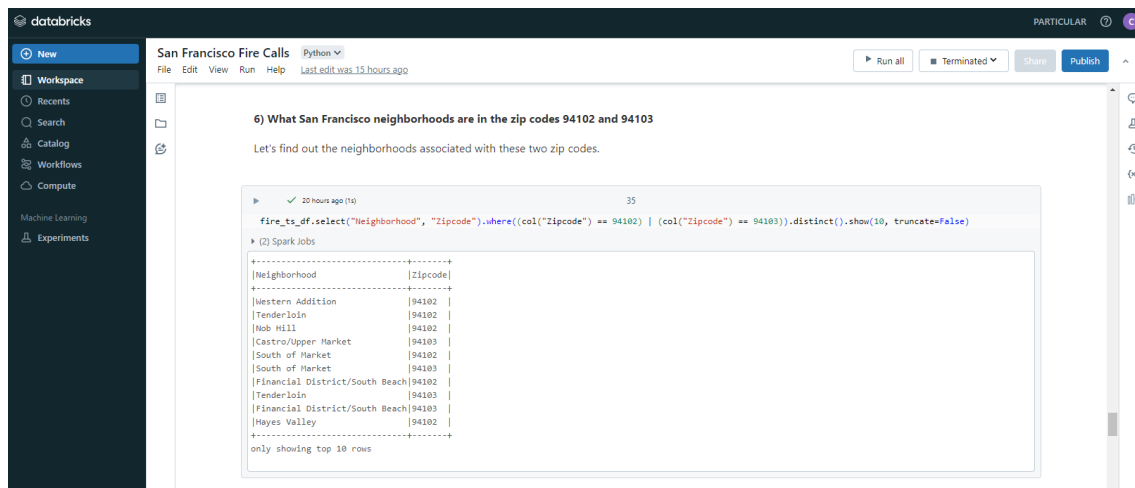
1. Filter out by CallType
2. Group them by CallType and Zip code
3. Count them and display them in descending order

```
fire_ts_df.select("CallType", "Zipcode").where(col("CallType").isNotNull()).groupBy("CallType", "Zipcode").count().orderBy("count", ascending=False).show(10, truncate=False)
```

(2) Spark Jobs

CallType	Zipcode	count
Medical Incident	94102	4801457
Medical Incident	94103	370215
Medical Incident	94110	249279
Medical Incident	94109	238087
Medical Incident	94124	147564
Medical Incident	94112	139565
Medical Incident	94115	120087
Medical Incident	94122	107602
Medical Incident	94107	107439
Medical Incident	94133	99090

only showing top 10 rows



San Francisco Fire Calls Python

File Edit View Run Help Last edit was 15 hours ago

Run all Terminated Share Publish

6) What San Francisco neighborhoods are in the zip codes 94102 and 94103

Let's find out the neighborhoods associated with these two zip codes.

```
fire_ts_df.select("Neighborhood", "Zipcode").where((col("Zipcode") == 94102) | (col("Zipcode") == 94103)).distinct().show(10, truncate=False)
```

(2) Spark Jobs

Neighborhood	Zipcode
Western Addition	94102
Tenderloin	94102
Nob Hill	94102
Castro/Upper Market	94103
South of Market	94102
South of Market	94103
Financial District/South Beach	94102
Tenderloin	94103
Financial District/South Beach	94103
Hayes Valley	94102

only showing top 10 rows



San Francisco Fire Calls Python

File Edit View Run Help Last edit was 15 hours ago

Run all Terminated Share Publish

7) What was the sum of all calls, average, min and max of the response times for calls?

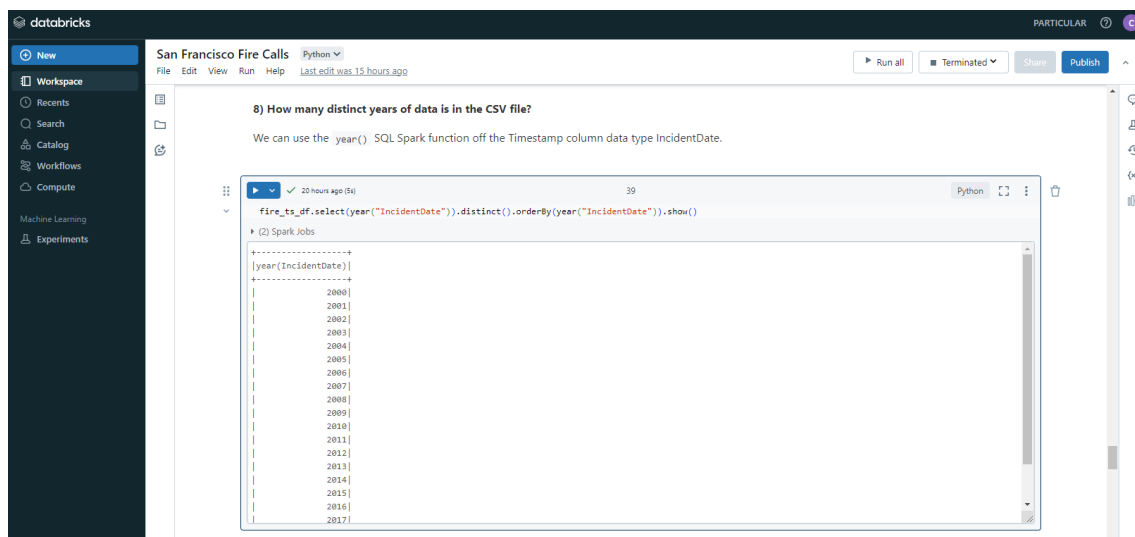
Let's use the built-in Spark SQL functions to compute the sum, avg, min, and max of few columns:

- Number of Total Alarms
- What were the min and max the delay in response time before the Fire Dept arrived at the scene of the call

```
fire_ts_df.select(sum("NumAlarms"), avg("ResponseDelayedInMins"), min("ResponseDelayedInMins"), max("ResponseDelayedInMins")).show()
```

(2) Spark Jobs

sum(NumAlarms)	avg(ResponseDelayedInMins)	min(ResponseDelayedInMins)	max(ResponseDelayedInMins)
4403441	3.902170335891614	0.016666668	1879.6167



San Francisco Fire Calls Python

File Edit View Run Help Last edit was 15 hours ago

Run all Terminated Share Publish

8) How many distinct years of data is in the CSV file?

We can use the `year()` SQL Spark function off the Timestamp column data type IncidentDate.

```
fire_ts_df.select(year("IncidentDate")).distinct().orderBy(year("IncidentDate")).show()
```

(2) Spark Jobs

year(IncidentDate)
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017

San Francisco Fire Calls Python

File Edit View Run Help Last edit was 15 hours ago

Run all Terminated Share Publish

9) What week of the year in 2018 had the most fire calls?

Note: Week 1 is the New Year's week and week 25 is the July 4th week. Loads of fireworks, so it makes sense the higher number of calls.

```
fire_ts_df.filter(year("IncidentDate") == 2018).groupBy(weekofyear("IncidentDate")).count().orderBy("count", ascending=False).show()
```

(2) Spark Jobs

weekofyear(IncidentDate)	count
1	6401
25	6163
13	6183
22	6060
44	6048
27	6042
16	6009
40	6000
43	5986
5	5946
2	5929
18	5917
9	5874
8	5843
6	5839
21	5821
38	5817
10	5806

San Francisco Fire Calls Python

File Edit View Run Help Last edit was 15 hours ago

Run all Terminated Share Publish

10) What neighborhoods in San Francisco had the worst response time in 2018?

```
fire_ts_df.select("Neighborhood", "ResponseDelayedInMins").filter(year("IncidentDate") == 2018).show(10, False)
```

(2) Spark Jobs

Neighborhood	ResponseDelayedInMins
Presidio Heights	2.4666667
Presidio Heights	2.8833334
Presidio Heights	2.1166666
Presidio Heights	4.25
Presidio Heights	2.8166666
Presidio Heights	3.4
Presidio Heights	2.7333333
Mission Bay	6.2666667
Mission Bay	6.7333333
Mission Bay	6.3333335

only showing top 10 rows

5 Data Loading

3. Data Loading

Load the transformed data to persistent storage, so that it's query-able across notebooks and clusters

```
fire_ts_df.write.format("parquet").mode("overwrite").save("/tmp/fireServiceParquet/")
```

(1) Spark Jobs

3. Data Loading

Load the transformed data to persistent storage, so that it's query-able across notebooks and clusters

```
%fs ls /tmp/fireServiceParquet/
```

Table	path	name
1	dbfs:/tmp/fireServiceParquet/_SUCCESS	_SUCCESS
2	dbfs:/tmp/fireServiceParquet/_committed_3026484276956406733	_committed_3026484276956406733
3	dbfs:/tmp/fireServiceParquet/_committed_8962660238422050142	_committed_8962660238422050142
4	dbfs:/tmp/fireServiceParquet/_committed_vacuum1553602930634289257	_committed_vacuum1553602930634289257
5	dbfs:/tmp/fireServiceParquet/_started_4084391089644339418	_started_4084391089644339418
6	dbfs:/tmp/fireServiceParquet/_started_8962660238422050142	_started_8962660238422050142
7	dbfs:/tmp/fireServiceParquet/part-00000-tid-8962660238422050142-1f363096-b320-4508-b0d0-cf532d746bc6-24-1-c000.snappy.parqu...	part-00000-tid-8962660238422050142-1f363096-b320
8	dbfs:/tmp/fireServiceParquet/part-00001-tid-8962660238422050142-1f363096-b320-4508-b0d0-cf532d746bc6-25-1-c000.snappy.parqu...	part-00001-tid-8962660238422050142-1f363096-b320
9	dbfs:/tmp/fireServiceParquet/part-00002-tid-8962660238422050142-1f363096-b320-4508-b0d0-cf532d746bc6-26-1-c000.snappy.parqu...	part-00002-tid-8962660238422050142-1f363096-b320
10	dbfs:/tmp/fireServiceParquet/part-00003-tid-8962660238422050142-1f363096-b320-4508-b0d0-cf532d746bc6-27-1-c000.snappy.parqu...	part-00003-tid-8962660238422050142-1f363096-b320
11	dbfs:/tmp/fireServiceParquet/part-00004-tid-8962660238422050142-1f363096-b320-4508-b0d0-cf532d746bc6-28-1-c000.snappy.parqu...	part-00004-tid-8962660238422050142-1f363096-b320
12	dbfs:/tmp/fireServiceParquet/part-00005-tid-8962660238422050142-1f363096-b320-4508-b0d0-cf532d746bc6-29-1-c000.snappy.parqu...	part-00005-tid-8962660238422050142-1f363096-b320
13	dbfs:/tmp/fireServiceParquet/part-00006-tid-8962660238422050142-1f363096-b320-4508-b0d0-cf532d746bc6-30-1-c000.snappy.parqu...	part-00006-tid-8962660238422050142-1f363096-b320
14	dbfs:/tmp/fireServiceParquet/part-00007-tid-8962660238422050142-1f363096-b320-4508-b0d0-cf532d746bc6-31-1-c000.snappy.parqu...	part-00007-tid-8962660238422050142-1f363096-b320

15 rows | 17.07 seconds runtime

12) How can we use Parquet SQL table to store data and read it back?

```
fire_ts_df.write.format("parquet").mode("overwrite").saveAsTable("FireServiceCalls")
```

(1) Spark Jobs

This command caches the specified table in Databricks cluster memory. Caching is a technique used to improve the performance of repeated queries by storing frequently accessed data in memory. When a table is cached, future queries to that table will be much faster because the data will not need to be read repeatedly from disk.

```
%sql
CACHE TABLE FireServiceCalls
```

(2) Spark Jobs

OK

Data Engineering with Spark Databricks

The screenshot shows the Databricks interface with a workspace titled "San Francisco Fire Calls". A SQL query is executed, displaying a table of fire call data. The table has columns: CallNumber, UnitID, IncidentNumber, CallType, CallFinalDisposition, Address, City, and Zipcode. The data is limited to 10 rows.

	CallNumber	UnitID	IncidentNumber	CallType	CallFinalDisposition	Address	City	Zipcode
1	111050354	E14	11034920	Medical Incident	Other	500 Block of 21ST AVE	SF	941
2	111050355	E03	11034921	Structure Fire	Other	HYDE ST/BUSH ST	SF	941
3	111050355	T03	11034921	Structure Fire	Other	HYDE ST/BUSH ST	SF	941
4	111050356	73	11034922	Structure Fire	Other	1000 Block of POTRERO AVE	SF	941
5	111050356	B06	11034922	Structure Fire	Other	1000 Block of POTRERO AVE	SF	941
6	111050356	B10	11034922	Structure Fire	Other	1000 Block of POTRERO AVE	SF	941
7	111050356	D3	11034922	Structure Fire	Other	1000 Block of POTRERO AVE	SF	941
8	111050356	E29	11034922	Structure Fire	Other	1000 Block of POTRERO AVE	SF	941
9	111050356	E37	11034922	Structure Fire	Other	1000 Block of POTRERO AVE	SF	941

The screenshot shows the Databricks interface with a workspace titled "San Francisco Fire Calls". A Python code cell is executed, reading data from a Parquet file. The code defines a variable `file_parquet_df` and displays the first 10 rows of the data.

```
file_parquet_df = spark.read.format("parquet").load("/tmp/fireServiceParquet/")  
display(file_parquet_df.limit(10))
```

The screenshot shows the Databricks interface with a workspace titled "San Francisco Fire Calls". A Python code cell is executed, displaying the first 10 rows of data from a Parquet file. The code defines a variable `file_parquet_df` and displays the first 10 rows of the data.

```
display(file_parquet_df.limit(10))
```

6 Conclusion

In this article, we explored the powerful capabilities of PySpark and SparkSQL within the Databricks platform, focusing on data analysis and transformation. Through our hands-on examples and code implementations, we demonstrated how to effectively leverage these tools to process and analyze large datasets efficiently.

We perform data extraction, transformation, and loading (ETL) processes, highlighting the seamless integration of PySpark for scalable data processing. Additionally, we utilized SparkSQL to perform queries, showcasing its ability to handle large-scale data analytics with ease.

We use techniques the performance optimization techniques, such as caching, which are essential for achieving high-performance data processing. By following best practices and utilizing the features of Databricks, we were able to enhance the efficiency and speed of our data analysis workflows.

I hope this article serves as a helpful guide for those looking to harness the power of PySpark and SparkSQL in their data engineering and analytics projects. The journey of data analysis is filled with continuous learning and innovation.

Thank you for taking the time to read this article. Your feedback and insights are always welcome as we continue to learn and grow in the ever-evolving field of data science.

Happy coding!

Cleber Zumba de Souza

7 References

PARSIAN, Mahmoud. **Data Algorithms with Spark**. Sebastopol, California, United States: O'Reilly Media, 2022.