CSE 4510/5310
Big Data

Instructor: Fitzroy Nembhard, Ph.D.

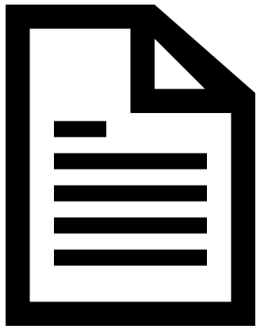
Processing Data in Spark

FLORIDA TECH
FLORIDA'S STEM UNIVERSITY®

Data Gathering

Download the sf-fire-calls.csv dataset from Canvas

The dataset is a large CSV file containing data on San Francisco Fire Department calls.



sf-fire-calls.csv

Properties of the Dataset

Rows	4.3M
Columns	28
Size	44MB

FLORIDA TECH
FLORIDA'S STEM UNIVERSITY

A snapshot of the Dataset

Here's a snapshot of the dataset.

sf-fire-calls

CallNumber	UnitID	IncidentNumber	CallType	CallDate	WatchDate	CallFinalDisposition	AvailableDtTm	Address	City	Zipcode	Battalion	StationArea	Box	OriginalPriority	Priority	FinalPriority	ALSUnit	CallTypeGroup	NumAlarms	UnitType	UnitSequenceInCallDispatch	FirePreventionDistrict	SupervisorDistrict	Neighborhood	Location	RowID	Delay
20110016	T13	2003235	Structure Fire	01/11/2002	01/10/2002	Other	01/11/2002 01:51:44 AM	2000 Block of CALIFORNIA ST	SF	94109	B04	38	3362	3	3	3	FALSE		1	TRUCK	2	4	5	Pacific Heights	(37.789584067932, -122.428071912459)	020110016-T13	2.95
20110022	M17	2003241	Medical Incident	01/11/2002	01/10/2002	Other	01/11/2002 03:01:18 AM	0 Block of SILVERVIEW DR	SF	94124	B10	42	6495	3	3	3	TRUE		1	MEDIC	1	10	10	Bayview Hunters Point	(37.7337633673897, -122.396113802632)	020110022-M17	4.7
20110023	M41	2003242	Medical Incident	01/11/2002	01/10/2002	Other	01/11/2002 02:39:50 AM	MARKET ST/MCALLISTER ST	SF	94102	B03	1	1455	3	3	3	TRUE		1	MEDIC	2	3	6	Tenderloin	(37.7811772186856, -122.411699931232)	020110023-M41	2.4333334
20110032	E11	2003250	Vehicle Fire	01/11/2002	01/10/2002	Other	01/11/2002 04:16:46 AM	APPLETON AV/MISSION ST	SF	94110	B06	32	5626	3	3	3	FALSE		1	ENGINE	1	6	9	Bernal Heights	(37.7388432849018, -122.423948785199)	020110032-E11	1.5
20110043	B04	2003259	Alarms	01/11/2002	01/10/2002	Other	01/11/2002 06:01:58 AM	1400 Block of SUTTER ST	SF	94109	B04	3	3223	3	3	3	FALSE		1	CHIEF	2	4	2	Western Addition	(37.7872893372638, -122.424236212664)	020110043-B04	3.4833333
20110072	T08	2003279	Structure Fire	01/11/2002	01/11/2002	Other	01/11/2002 08:03:26 AM	BEALE ST/FOLSOM ST	SF	94105	B03	35	2122	3	3	3	FALSE		1	TRUCK	2	3	6	Financial District/South Beach	(37.7886866619654, -122.392722833778)	020110072-T08	1.75
20110125	E33	2003301	Alarms	01/11/2002	01/11/2002	Other	01/11/2002 09:46:44 AM	0 Block of FARALLONES ST	SF	94112	B09	33	8324	3	3	3	FALSE		1	ENGINE	2	9	11	Oceanview/Merced/Ingleside	(37.7140353531157, -122.454117749916)	020110125-E33	2.7166667
20110130	E36	2003304	Alarms	01/11/2002	01/11/2002	Other	01/11/2002 09:58:53 AM	600 Block of POLK ST	SF	94102	B02	3	3114	3	3	3	FALSE		1	ENGINE	1	2	6	Tenderloin	(37.7826266328595, -122.41915582123)	020110130-E36	1.7833333
20110197	E05	2003343	Medical Incident	01/11/2002	01/11/2002	Other	01/11/2002 12:06:57 PM	1500 Block of WEBSTER ST	SF	94115	B04	5	3513	3	3	3	FALSE		1	ENGINE	1	4	5	Japantown	(37.784958590666, -122.43143527450	020110197-E05	1.5166667
20110215	E06	2003348	Medical Incident	01/11/2002	01/11/2002	Other	01/11/2002 01:08:40 PM	DIAMOND ST/MARKET ST	SF	94114	B05	6	5415	3	3	3	FALSE		1	ENGINE	1	5	8	Castro/Upper Market	(37.7618954753708, -122.437298717721)	020110215-E06	2.7666667
20110274	M07	2003381	Medical Incident	01/11/2002	01/11/2002	Other	01/11/2002 03:31:02 PM	2700 Block of MISSION ST	SF	94110	B06	11	5525	1	1	2	TRUE		1	MEDIC	1	6	9	Mission	(37.7530339738059, -122.41585898473)	020110274-M07	2.1833334
20110275	T15	2003382	Structure Fire	01/11/2002	01/11/2002	Other	01/11/2002 02:59:04 PM	BRUNSWICK ST/GUTTENBERG ST	SF	94112	B09	43	6218	3	3	3	FALSE		1	TRUCK	1	9	11	Excelsior	(37.7105545807996, -122.443335369545)	020110275-T15	2.5
20110304	E03	2003399	Medical Incident	01/11/2002	01/11/2002	Other	01/11/2002 04:22:49 PM	1000 Block of SUTTER ST	SF	94109	B04	3	1557	3	3	3	FALSE		1	ENGINE	1	4	3	Nob Hill	(37.7881263034393, -122.41765721404	020110304-E03	2.4166667
20110308	E14	2003403	Medical Incident	01/11/2002	01/11/2002	Other	01/11/2002 04:18:33 PM	100 Block of 21ST AVE	SF	94121	B07	14	7173	3	3	3	FALSE		1	ENGINE	1	7	1	Outer Richmond	(37.7850084431077, -122.480723607753)	020110308-E14	4.95
20110313	B10	2003408	Structure Fire	01/11/2002	01/11/2002	Other	01/11/2002 04:09:08 PM	700 Block of CAPP ST	SF	94110	B06	7	5472	3	3	3	FALSE		1	CHIEF	6	6	9	Mission	(37.7547064357942, -122.417513465479)	020110313-B10	1.4166666
20110313	D3	2003408	Structure Fire	01/11/2002	01/11/2002	Other	01/11/2002 04:09:08 PM	700 Block of CAPP ST	SF	94110	B06	7	5472	3	3	3	FALSE		1	CHIEF	4	6	9	Mission	(37.7547064357942, -122.417513465479)	020110313-D3	2.5333333
20110313	E32	2003408	Structure Fire	01/11/2002	01/11/2002	Other	01/11/2002 04:09:08 PM	700 Block of CAPP ST	SF	94110	B06	7	5472	3	3	3	TRUE		1	ENGINE	8	6	9	Mission	(37.7547064357942, -122.417513465479)	020110313-E32	1.8833333
20110315	RC2	2003409	Medical Incident	01/11/2002	01/11/2002	Other	01/11/2002 04:34:23 PM	200 Block of LAGUNA HONDA BLVD	SF	94116	B08	20	8635	3	3	3	TRUE		1	RESCUE CAPTAIN	2	8	7	West of Twin Peaks	(37.7501117393668, -122.460819155469)	020110315-RC2	5.35
20110330	E14	2003417	Medical Incident	01/11/2002	01/11/2002	Other	01/11/2002 04:51:31 PM	BALBOA ST/PARK PRESIDIO BL	SF	94118	B07	31	7145	3	3	3	FALSE		1	ENGINE	1	7	1	Inner Richmond	(37.7768682293368, -122.472039541478)	020110330-E14	2.0
20110330	M12	2003417	Medical Incident	01/11/2002	01/11/2002	Other	01/11/2002 04:51:12 PM	BALBOA ST/PARK PRESIDIO BL	SF	94118	B07	31	7145	3	3	3	TRUE		1	MEDIC	2	7	1	Inner Richmond	(37.7768682293368, -122.472039541478)	020110330-M12	1.8166667
20110344	T06	2003429	Odor (Strange / Unknown)	01/11/2002	01/11/2002	Other	01/11/2002 05:17:15 PM	2300 Block of MARKET ST	SF	94114	B05	6	5233	3	3	3	FALSE		1	TRUCK	2	5	8	Castro/Upper Market	(37.7635007029742, -122.434209429009)	020110344-T06	2.0833333
20110350	M41	2003435	Medical Incident	01/11/2002	01/11/2002	Other	01/11/2002 05:46:30 PM	500 Block of BROADWAY	SF	94133	B01	2	1311	2	2	2	TRUE		1	MEDIC	1	1	3	North Beach	(37.7980228452184, -122.405863216232)	020110350-M41	4.4666667
20110375	B05	2003453	Alarms	01/11/2002	01/11/2002	Other	01/11/2002 06:48:01 PM	100 Block of JOHNSTONE DR	SF	94131	B05	20	5278	3	3	3	FALSE		1	CHIEF	3	5	7	Inner Sunset	(37.7585821585787, -122.453613744703)	020110375-B05	1.8
20110425	B01	2003497	Structure Fire	01/11/2002	01/11/2002	Other	01/11/2002 09:03:17 PM	600 Block of OFARRELL ST	SF	94109	B04	3	1544	3	3	3	FALSE		1	CHIEF	4	4	6	Tenderloin	(37.7854670505017, -122.415977627827)	020110425-B01	2.5666666
20110428	M28	2003500	Medical Incident	01/11/2002	01/11/2002	Other	01/11/2002 10:08:48 PM	1000 Block of BATTERY ST	SF	94111	B01	13	1153	3	3	3	TRUE		1	MEDIC	2	1	3	Financial District/South Beach	(37.8006802693983, -122.401547948	020110428-M28	3.8833334
20110467	T19	2003529	Medical Incident	01/11/2002	01/11/2002	Other	01/11/2002 10:56:59 PM	3000 Block of 23RD AVE	SF	94132	B08	19	8704	3	3	3	FALSE		1	TRUCK	1	8	7	Sunset/Parkside	(37.7319470441445, -122.479112854493)	020110467-T19	3.3
20120016	E43	2003550	Medical Incident	01/12/2002	01/11/2002	Other	01/12/2002 02:04:06 AM	0 Block of BLYTHDALE AVE	SF	94134	B09	43	6244	3	3	3	TRUE		1	ENGINE	2	9	10	Visitacion Valley	(37.7106587183839, -122.417699843965)	020120016-E43	3.1
20120020	E36	2003554	Structure Fire	01/12/2002	01/11/2002	Other	01/12/2002 01:56:32 AM	9TH ST/HOWARD ST	SF	94103	B02	36	2335	3	3	3	FALSE		1	ENGINE	1	2	6	South of Market	(37.7749917496069, -122.413161109656	020120020-E36	3.3333333
20120044	M10	2003576	Medical Incident	01/12/2002	01/11/2002	Other	01/12/2002 04:17:22 AM	3800 Block of GEARY BLVD	SF	94118	B07	31	7113	3	3	3	TRUE		1	MEDIC	2	7	1	Inner Richmond	(37.781314527908, -122.460540716843)	020120044-M10	3.9666667
20120045	E21	2003577	Medical Incident	01/12/2002	01/11/2002	Other	01/12/2002 04:23:31 AM	300 Block of BAKER ST	SF	94117	B05	21	4252	3	3	3	FALSE		1	ENGINE	1	5	5	Lone Mountain/USF	(37.7740926787047, -122.4410842597)	020120045-E21	3.25
20120052	M36	2003584	Medical Incident	01/12/2002	01/11/2002	Other	01/12/2002 06:27:31 AM	400 Block of VALENCIA ST	SF	94103	B02	6	5226	3	3	3	TRUE		1	MEDIC	2	2	9	Mission	(37.7661343875141, -122.421935077177)	020120052-M36	4.2833333
20120061	M10	2003593	Medical Incident	01/12/2002	01/11/2002	Other		0 Block of TERRA VISTA AVE	SF	94115	B05	21	4256	3	3	3	TRUE		1	MEDIC	1	5	2	Lone Mountain/USF	(37.7814248717141, -122.441695636568	020120061-M10	2.3
20120111	E18	2003618	Odor (Strange / Unknown)	01/12/2002	01/12/2002	Other	01/12/2002 11:07:36 AM	2000 Block of 34TH AVE	SF	94116	B08	18	7556	3	3	3	FALSE		1	ENGINE	2	8	4	Sunset/Parkside	(37.7488650921071, -122.4922894922253)	020120111-E18	2.8
20120127	M38	2003630	Medical Incident	01/12/2002	01/12/2002	Other	01/12/2002 11:28:40 AM	LAGUNA ST/WASHINGTON ST	SF	94109	B04	38	3361	3	3	3	TRUE		1	MEDIC	1	4	2	Pacific Heights	(37.7922168970615, -122.42944719330	020120127-M38	2.3833334
20120142	E07	2003639	Medical Incident	01/12/2002	01/12/2002	Other	01/12/2002 12:15:25 PM	600 Block of SOUTH VAN NESS AVE	SF	94110	B02	7	5246	3	3	3	FALSE		1	ENGINE	1	2	9	Mission	(37.7627737268131, -122.417186682545	020120142-E07	2.45
20120147	M38	2003642	Medical Incident	01/12/2002	01/12/2002	Other	01/12/2002 01:23:04 PM	1300 Block of HYDE ST	SF	94109	B01	41	1564	1	1	2	TRUE		1	MEDIC	1	1	3	Russian Hill	(37.7932350074749, -122.417915793747)	020120147-M38	6.25
20120149	M01	2003643	Medical Incident	01/12/2002	01/12/2002	Other	01/12/2002 01:05:52 PM	16TH ST/MISSION ST	SF	94103	B02	7	5236	1	1	2	TRUE		1	MEDIC	1	2	9	Mission	(37.7650513381945, -122.419668973861)	020120149-M01	3.6166666
20120153	M41	2003647	Medical Incident	01/12/2002	01/12/2002	Other	01/12/2002 02:20:25 PM	500 Block of 30TH AVE	SF	94121	B07	14	722	3	3	3	TRUE		1	MEDIC	2	7	1	Outer Richmond	(37.778884735552 5, -122.4899547276 3)	020120153-M41	4.65
20120155	B08	2003649	Odor (Strange / Unknown)	01/12/2002	01/12/2002	Other	01/12/2002 01:03:10 PM	1500 Block of 5TH AVE	SF	94122	B08	12	7328	3	3	3	FALSE		1	CHIEF	3	8	7	Inner Sunset	(37.7598374406709, -122.4609429486 7)	020120155-B08	4.1833334
20120166	E01	2003656	Medical Incident	01/12/2002	01/12/2002	Other	01/12/2002 01:22:58 PM	4TH ST/MISSION ST	SF	94103	B03	1	2213	3	3	3	FALSE		1	ENGINE	1	3	6	South of Market	(37.7844787439707, -122.402666872501)	020120166-E01	1.95
20120210	E44	2003688	Medical Incident	01/12/2002	01/12/2002	Other	01/12/2002 04:25:35 PM	0 Block of AQUAVISTA WAY	SF	94131	B09	20	5362	3	3	3	TRUE		1	ENGINE	1	8	7	Twin Peaks	(37.732557468944, -122.449351606127)	020120210-E44	3.55
20120217	M03	2003695	Structure Fire	01/12/2002	01/12/2002	Other	01/12/2002 04:46:59 PM	400 Block of TURK ST	SF	94102	B02	3	1554	3	3	3	TRUE		1	MEDIC	3	2	6	Tenderloin	(37.7825569563078, -122.416349428183	020120217-M03	1.5666667
20120254	M17	2003724	Medical Incident	01/12/2002	01/12/2002	Other	01/12/2002 07:15:23 PM	0 Block of REARDON RD	SF	94124	B10	17	6625	3	3	3	TRUE		1	MEDIC	2	10	10	Bayview Hunters Point	(37.7294407135003, -122.377026608239	020120254-M17	2.7833333
20120268	M07	2003735	Medical Incident	01/12/2002	01/12/2002	Other	01/12/2002 07:26:52 PM	1800 Block of SILLIMAN ST	SF	94134	B09	43	6153	3	3	3	TRUE		1	MEDIC	2	9	9	Excelsior	(37.7264150130988, -122.423317090957)	020120268-M07	4.5333333

FLORIDA TECH
FLORIDA'S STEM UNIVERSITY

Import PySpark

First, Launch Jupyter notebook.

Name your notebook “processing_san_fran_fires”

Run the following code to import spark and initialize a spark session.

Note pyspark must be installed for this code to run.

Refresher: install pyspark using the links below:

https://sparkbyexamples.com/spark/apache-spark-installation-on-windows/

https://sparkbyexamples.com/spark/install-apache-spark-on-mac/

```
# This code creates a Spark Application


from pyspark.sql import SparkSession

# Create a Spark Application using SparkSession
spark = SparkSession.builder.appName("Francisco Fires").master ("local[*]").getOrCreate()
```

FLORIDA TECH
FLORIDA'S STEM UNIVERSITY

Create a Schema

Note that it is more efficient to define a schema than have Spark infer it.

Use the given schema to read the data.

StructField(name, dataType, nullable):
Represents a field in a StructType.

The name of a field is indicated by name.

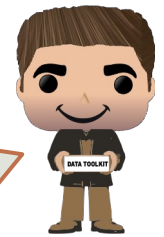
The data type of a field is indicated by datatype.

nullable is used to indicate if values of these fields can have null values.

```
from pyspark.sql.types import *

# Programmatic way to define a 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('AvailableDtTm', 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)])
```

We could also infer the schema from a sample of the data

```
sampleDF = spark \
.read \
.option("samplingRatio", 0.001)\
.option("header", "true")\
.csv("""data/sanfran_fire/Fire_Incidents.csv""")
```



FLORIDA TECH
FLORIDA'S STEM UNIVERSITY

Common Data Types

Here are some common data types in spark.

Read more here:
https://spark.apache.org/docs/latest/sql-ref-datatypes.html

Data type	Value type in Python	API to access or create a data type
ByteType	int or long **Note:** Numbers will be converted to 1-byte signed integer numbers at runtime. Please make sure that numbers are within the range of -128 to 127.	ByteType()
ShortType	int or long **Note:** Numbers will be converted to 2-byte signed integer numbers at runtime. Please make sure that numbers are within the range of -32768 to 32767.	ShortType()
IntegerType	int or long	IntegerType()
LongType	long **Note:** Numbers will be converted to 8-byte signed integer numbers at runtime. Please make sure that numbers are within the range of -9223372036854775808 to 9223372036854775807. Otherwise, please convert data to decimal.Decimal and use DecimalType.	LongType()
FloatType	float **Note:** Numbers will be converted to 4-byte single-precision floating point numbers at runtime.	FloatType()
DoubleType	float	DoubleType()
DecimalType	decimal.Decimal	DecimalType()
StringType	string	StringType()
BinaryType	bytearray	BinaryType()
BooleanType	bool	BooleanType()
TimestampType	datetime.datetime	TimestampType()
DateType	datetime.date	DateType()
DayTimeIntervalType	datetime.timedelta	DayTimeIntervalType()
ArrayType	list, tuple, or array	ArrayType(*elementType*, [*containsNull*]) **Note:**The default value of *containsNull* is True.
MapType	dict	MapType(*keyType*, *valueType*, [*valueContainsNull*]) **Note:**The default value of *valueContainsNull* is True.
StructType	list or tuple	StructType(*fields*) **Note:** *fields* is a Seq of StructFields. Also, two fields with the same name are not allowed.
StructField	The value type in Python of the data type of this field (For example, Int for a StructField with the data type IntegerType)	StructField(*name*, *dataType*, [*nullable*]) **Note:** The default value of *nullable* is True.

FLORIDA TECH
FLORIDA'S STEM UNIVERSITY

Read the Data

Use the following piece of code the read the data to create a DataFrame from the CSV file.

```
fire_df = spark.read.csv(sf_fire_file, header=True, schema=fire_schema)
```

FLORIDA TECH
FLORIDA'S STEM UNIVERSITY

Saving a DataFrame

You may use the following piece of code to save the data to a parquet file or table.

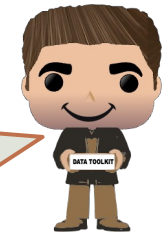
saveAsTable() creates a permanent, physical table stored in Hive Metastore using the Parquet format. The table metadata including the location of the file(s) is stored within the Hive metastore.

A Hive metastore (aka metastore_db) is a relational database to manage the metadata of the persistent relational entities, e.g. databases, tables, columns, partitions.

A Hive metastore warehouse (aka spark-warehouse) is the directory where Spark SQL persists tables.

```
# Save as a Parquet file
parquet_path = ""

fire_df.write.format("parquet").save(parquet_path)

df.write\
    .option("mode", "DROPMALFORMED")\
    .option("compression", "snappy")\
    .option("path","s3://....")\
    .mode("overwrite")\
    .format("parquet").save()

df.write.parquet("s3://bucket-name/folder/test.parquet",mode="overwrite")


# Save as a table
parquet_table = "" # name of the table
fire_df.write.format("parquet").saveAsTable(parquet_table)


######################
# Configuring Hive Support
spark = SparkSession \
    .builder \
    .appName("Python Spark SQL Hive integration example") \
    .config("spark.sql.warehouse.dir", warehouse_location) \
    .enableHiveSupport() \
    .getOrCreate()
```

Learn more about Spark and S3 Buckets here: https://sparkbyexamples.com/spark/write-read-csv-file-from-s3-into-dataframe/



Projections & Filters

A **projection** in relational parlance is a way to return only the rows matching a certain relational condition by using filters.

In Spark, projections are done with the select() method, while filters can be expressed using the filter() or where() method. We can use this technique to examine specific aspects of our SF Fire Department data set.

We may also import specific functions this way:

```
from pyspark.sql.functions import avg, sum
```

```
from pyspark.sql import functions as f

few_fire_df = (fire_df \
.select("IncidentNumber", "AvailableDtTm", "CallType") \
.where(f.col("CallType") != "Medical Incident"))

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

```
+--------------+----------------------+--------------+
|IncidentNumber|AvailableDtTm         |CallType      |
+--------------+----------------------+--------------+
|2003235       |01/11/2002 01:51:44 AM|Structure Fire|
|2003250       |01/11/2002 04:16:46 AM|Vehicle Fire  |
|2003259       |01/11/2002 06:01:58 AM|Alarms        |
|2003279       |01/11/2002 08:03:26 AM|Structure Fire|
|2003301       |01/11/2002 09:46:44 AM|Alarms        |
+--------------+----------------------+--------------+
```

FLORIDA TECH
FLORIDA'S STEM UNIVERSITY

Querying the Data

Let's answer a question from the data:

How many distinct CallTypes were recorded as the causes of the fire calls?

```
(fire_df
 .select("CallType")
 .where(f.col("CallType").isNotNull())
 .agg(f.countDistinct("CallType").alias("DistinctCallTypes"))
 .show())
```

```
+-----------------+
|DistinctCallTypes|
+-----------------+
|               30|
+-----------------+
```

FLORIDA TECH
FLORIDA'S STEM UNIVERSITY

Querying the Data

Let's answer a question from the data:

List the distinct call types in the data.

DataFrame.show(n=20, truncate=True, vertical=False)

Truncate: truncate strings longer than 20 chars by default. If set to a number greater than one, truncates long strings to length truncate and align cells right

Vertical: print output rows vertically (one line per column value)

```
(fire_df
 .select("CallType")
 .where(f.col("CallType").isNotNull())
 .distinct()
 .show(10, False))
```

```
+-----------------------------+
|CallType                     |
+-----------------------------+
|Elevator / Escalator Rescue  |
|Marine Fire                  |
|Aircraft Emergency           |
|Administrative               |
|Alarms                       |
|Odor (Strange / Unknown)     |
|Citizen Assist / Service Call|
|HazMat                       |
|Explosion                    |
|Oil Spill                    |
+-----------------------------+
only showing top 10 rows
```

FLORIDA TECH
FLORIDA'S STEM UNIVERSITY

Renaming columns

By specifying the desired column names in the schema with StructField, as we did, we effectively changed all names in the resulting DataFrame.

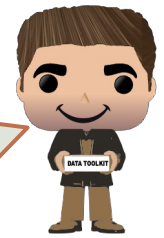
Alternatively, you could selectively rename columns with the withColumnRenamed() method.

Let's change the name of our Delay column to ResponseDelayedinMins and take a look at the response times that were longer than five minutes

```
new_fire_df = fire_df.withColumnRenamed("Delay", "ResponseDelayedinMins")

(new_fire_df
 .select("ResponseDelayedinMins")
 .where(f.col("ResponseDelayedinMins") > 5)
 .show(5, False))
```

```
+---------------------+
|ResponseDelayedinMins|
+---------------------+
|5.35                 |
|6.25                 |
|5.2                  |
|5.6                  |
|7.25                 |
+---------------------+
only showing top 5 rows
```

Because DataFrame transformations are immutable, when we rename a column using withColumnRenamed() we get a new DataFrame while retaining the original with the old column name.



FLORIDA TECH
FLORIDA'S STEM UNIVERSITY

Adding and dropping columns

In our SF Fire Department data set, the columns CallDate, WatchDate, and AlarmDtTm are strings rather than either Unix timestamps or SQL dates, both of which Spark supports and can easily manipulate during transformations or actions (e.g., during a date- or time- based analysis of the data).

spark.sql.functions has a set of to/from date/time- stamp functions such as to_timestamp() and to_date() that we can use for just this purpose:

```
from pyspark.sql import functions as f

fire_ts_df = (new_fire_df
        .withColumn("IncidentDate", f.to_timestamp(f.col("CallDate"), "MM/dd/yyyy"))
        .drop("CallDate")
        .withColumn("OnWatchDate", f.to_timestamp(f.col("WatchDate"), "MM/dd/yyyy"))
        .drop("WatchDate")
        .withColumn("AvailableDtTS", f.to_timestamp(f.col("AvailableDtTm"),
                                    "MM/dd/yyyy hh:mm:ss a"))
        .drop("AvailableDtTm"))

  # Select the converted columns
  (fire_ts_df
   .select("IncidentDate", "OnWatchDate", "AvailableDtTS")
   .show(5, False))
```

```
+-------------------+-------------------+-------------------+
|IncidentDate       |OnWatchDate        |AvailableDtTS      |
+-------------------+-------------------+-------------------+
|2002-01-11 00:00:00|2002-01-10 00:00:00|2002-01-11 01:51:44|
|2002-01-11 00:00:00|2002-01-10 00:00:00|2002-01-11 03:01:18|
|2002-01-11 00:00:00|2002-01-10 00:00:00|2002-01-11 02:39:50|
|2002-01-11 00:00:00|2002-01-10 00:00:00|2002-01-11 04:16:46|
|2002-01-11 00:00:00|2002-01-10 00:00:00|2002-01-11 06:01:58|
+-------------------+-------------------+-------------------+
only showing top 5 rows
```

FLORIDA TECH
FLORIDA'S STEM UNIVERSITY

Using the converted dates in a query

Now that we have modified the dates, we can query using functions from spark.sql.functions like month(), year(), and day() to explore our data further.

```
(fire_ts_df
 .select(f.year('IncidentDate'))
 .distinct()
 .orderBy(f.year('IncidentDate'))
 .show())
```

FLORIDA TECH
FLORIDA'S STEM UNIVERSITY

Aggregations

Transformations and actions on DataFrames, such as groupBy(), orderBy(), and count(), offer the ability to aggregate by column names and then aggregate counts across the dataframe.

Let's answer the following question using aggregation:

What were the top 10 most common types of fire calls?

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

```
+---------------------------------+------+
|CallType                         |count |
+---------------------------------+------+
|Medical Incident                 |113794|
|Structure Fire                   |23319 |
|Alarms                           |19406 |
|Traffic Collision                |7013  |
|Citizen Assist / Service Call    |2524  |
|Other                            |2166  |
|Outside Fire                     |2094  |
|Vehicle Fire                     |854   |
|Gas Leak (Natural and LP Gases)  |764   |
|Water Rescue                     |755   |
+---------------------------------+------+
only showing top 10 rows
```

FLORIDA TECH
FLORIDA'S STEM UNIVERSITY

Other common DataFrame operations

The DataFrame API also provides descriptive statistical methods like min(), max(), sum(), and avg()

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

```
+--------------+-------------------------+-------------------------+-------------------------+
|sum(NumAlarms)|avg(ResponseDelayedinMins)|min(ResponseDelayedinMins)|max(ResponseDelayedinMins)|
+--------------+-------------------------+-------------------------+-------------------------+
|        176170|         3.892364154521585|               0.016666668|                  1844.55|
+--------------+-------------------------+-------------------------+-------------------------+
```

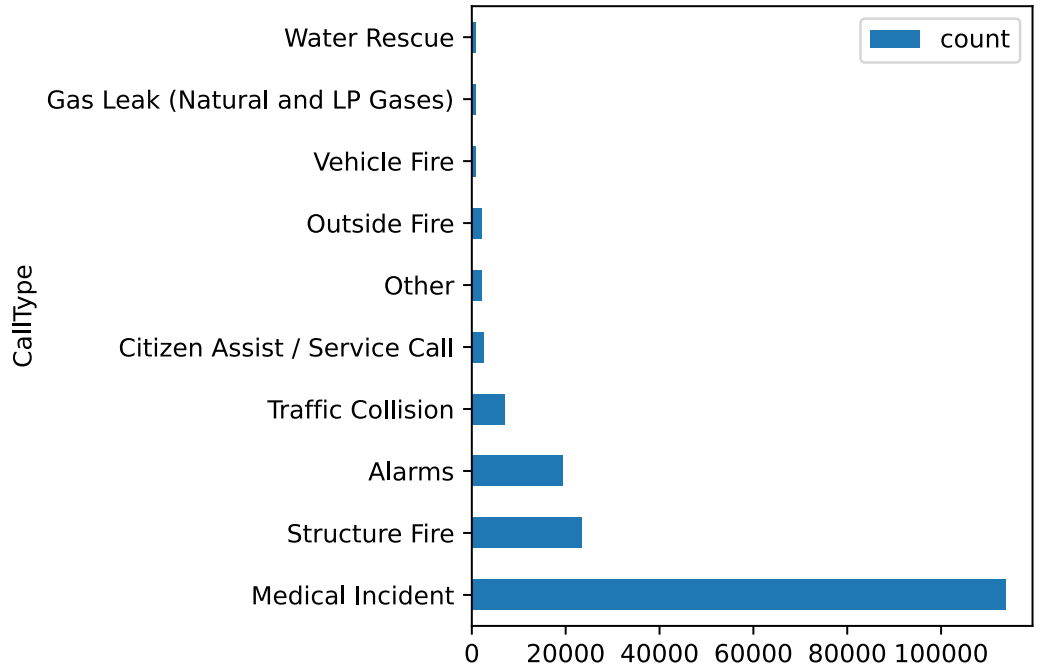
FLORIDA TECH
FLORIDA'S STEM UNIVERSITY

Plotting Spark Data using Pandas

Since Spark does not have plotting capabilities, we can convert the spark DataFrame to a Pandas dataframe to take advantage of the plotting capabilities of Pandas and Matplotlib.

```
pandas_df.plot(kind="barh", x="CallType", y="count")
```

```
pandas_df = (fire_ts_df
        .select("CallType")
        .where(f.col("CallType").isNotNull())
        .groupBy("CallType")
        .count()
        .orderBy("count", ascending=False)).toPandas().head(10)
```



FLORIDA TECH
FLORIDA'S STEM UNIVERSITY

Useful Resources

Feel free to use the following resources to quicky process your PySpark DataFrames and do interesting things.

Topic	Resource
Join DataFrames	https://sparkbyexamples.com/pyspark/pyspark-join-explained-with-examples/
Convert Pandas to Spark DataFrame	https://sparkbyexamples.com/pyspark/convert-pandas-to-pyspark-dataframe/
Concatenate PySpark Columns	https://sparkbyexamples.com/pyspark/pyspark-concatenate-columns/
User Defined Functions (UDF) : To apply a user function to one or more columns	https://sparkbyexamples.com/pyspark/pyspark-udf-user-defined-function/
Select Top N rows from a DataFrame	https://sparkbyexamples.com/spark/show-top-n-rows-in-spark-pyspark/
Display location data (e.g., lat/long) on a map	https://geopandas.org/en/stable/gallery/create_geopandas_from_pandas.html
Reverse Geocoding Data: Get a city using its lat/long	https://openweathermap.org/api/geocoding-api
PySpark Built-in Functions	https://spark.apache.org/docs/2.1.0/api/python/pyspark.sql.html#module-pyspark.sql.functions
Calculating Correlations in PySpark	https://www.projectpro.io/recipes/calculate-correlation-pyspark
Data Analysis with PySpark Example	https://www.nbshare.io/notebook/97969492/Data-Analysis-With-Pyspark-Dataframe/

FLORIDA TECH
FLORIDA'S STEM UNIVERSITY

References

The following reference(s) were used to create this tutorial.

Learning Spark Lightning-Fast Data Analytics, Jules S. Damji et al

FLORIDA TECH
FLORIDA'S STEM UNIVERSITY