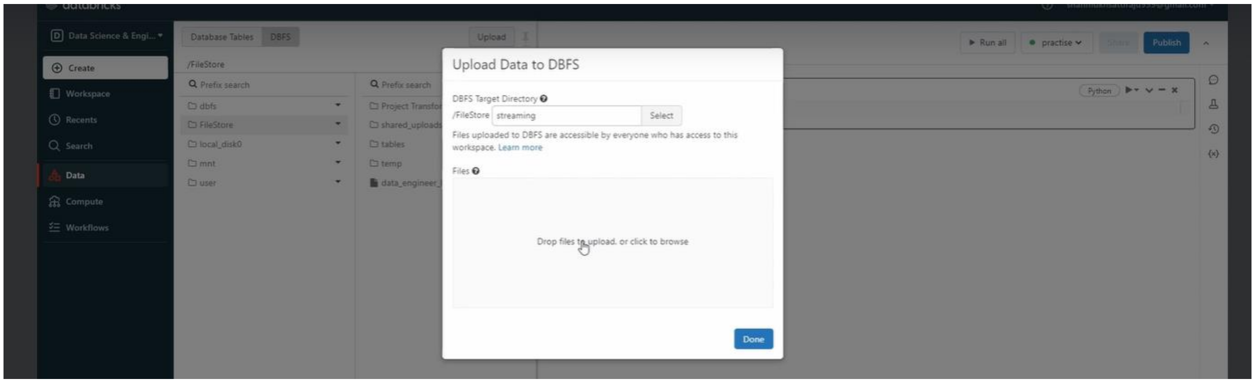
Streaming_practice



Upload Data to DBFS

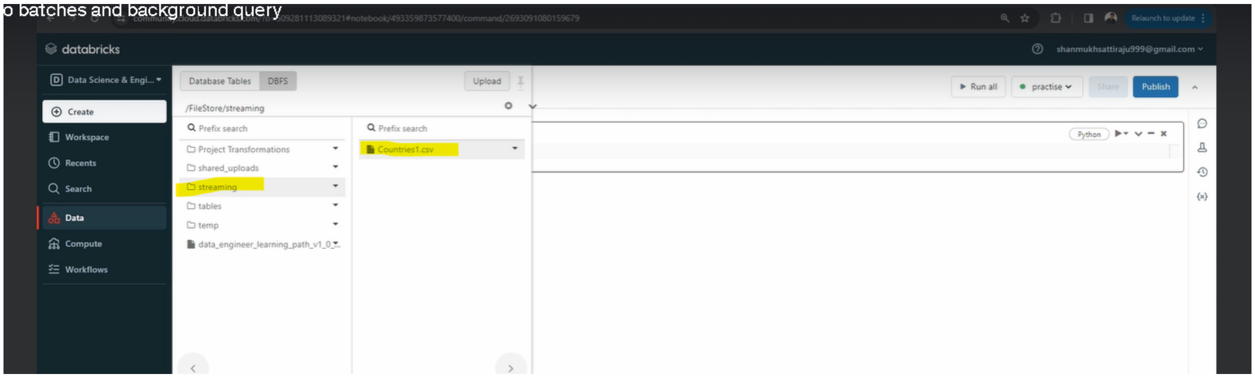
DBFS Target Directory: /FileStore streaming

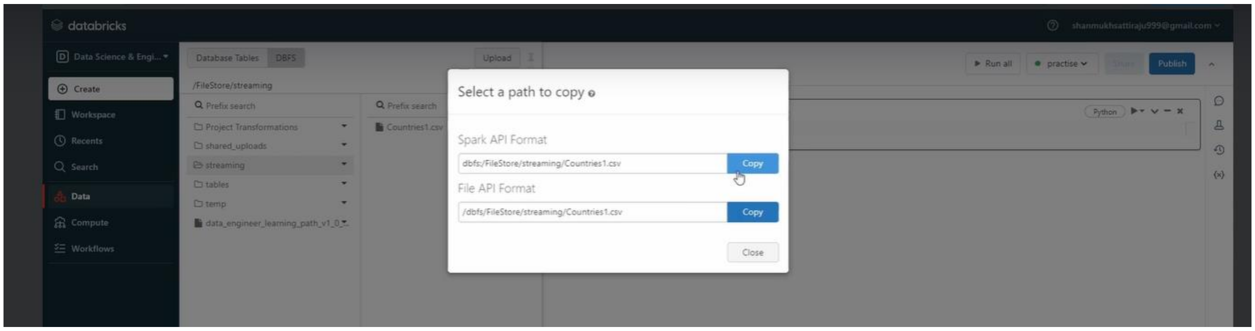
Files uploaded to DBFS are accessible by everyone who has access to this workspace. Learn more

Files: Drop files to upload, or click to browse

Upload a file to dbfs in the workspace creating a folder

o batches and background query



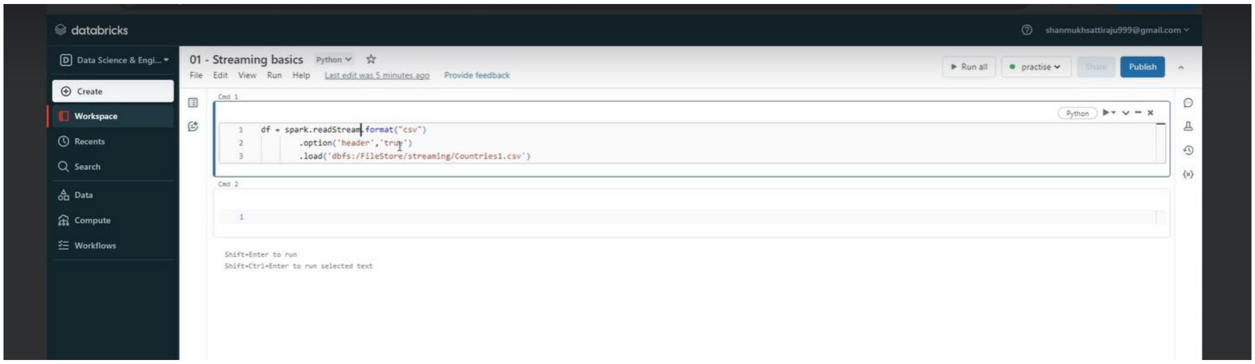


Select a path to copy

Spark API Format: dbfs:/FileStore/streaming/Countries1.csv

File API Format: /dbfs/FileStore/streaming/Countries1.csv

Reading the data

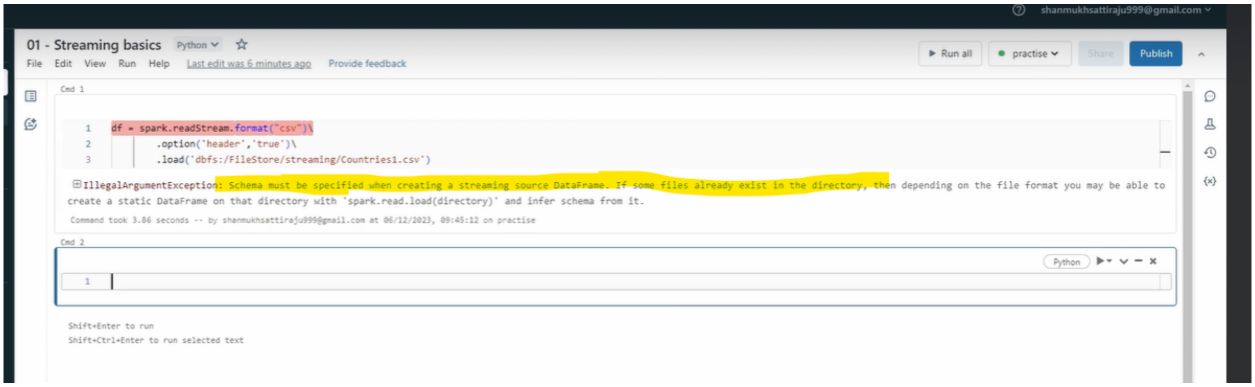


```
df = spark.readStream.format("csv")
        .option('header','true')
        .load('dbfs:/FileStore/streaming/Countries1.csv')
```

```
Shift+Enter to run
Shift+Ctrl+Enter to run selected text
```

Reading a streaming dataset

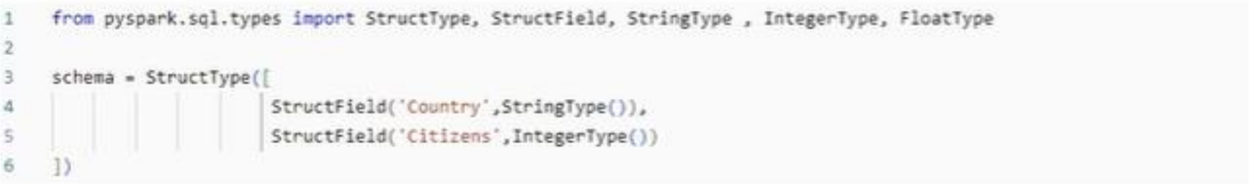
If we try to read the dataframe directly without specifying or inferring the schema, we will get the below error.



```
df = spark.readStream.format("csv")\
        .option('header','true')\
        .load('dbfs:/FileStore/streaming/Countries1.csv')
```

```
IllegalArgumentException: Schema must be specified when creating a streaming source DataFrame. If some files already exist in the directory, then depending on the file format you may be able to create a static DataFrame on that directory with 'spark.read.load(directory)' and infer schema from it.
Command took 3.86 seconds -- by shanmukhsattiraju999@gmail.com at 06/12/2023, 09:45:12 on practise
```

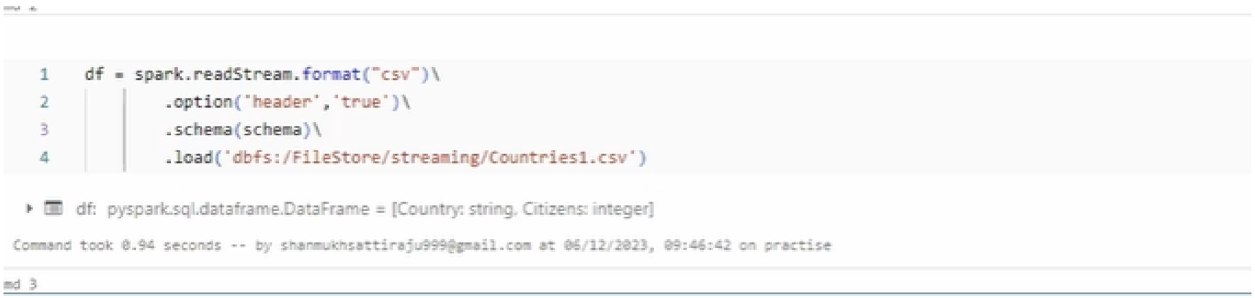
```
Shift+Enter to run
Shift+Ctrl+Enter to run selected text
```



```
from pyspark.sql.types import StructType, StructField, StringType , IntegerType, FloatType

schema = StructType([
                    StructField('Country',StringType()),
                    StructField('Citizens',IntegerType())
])
```

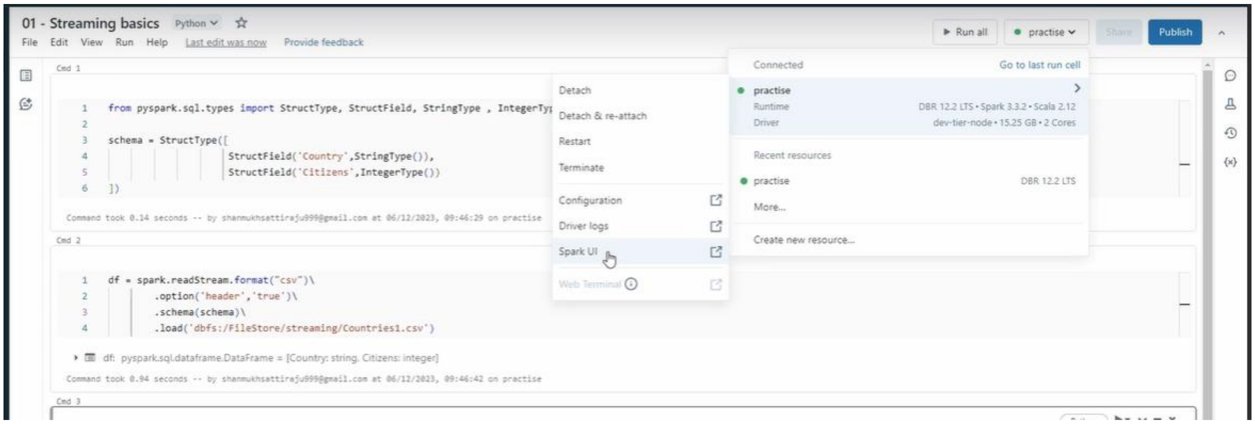
Define the schema and re-run the data frame



```
df = spark.readStream.format("csv")\
        .option('header','true')\
        .schema(schema)\
        .load('dbfs:/FileStore/streaming/Countries1.csv')
```

```
df: pyspark.sql.dataframe.DataFrame = [Country: string, Citizens: integer]
Command took 0.94 seconds -- by shanmukhsattiraju999@gmail.com at 06/12/2023, 09:46:42 on practise
```

Rerunning the data frame, specifying the schema will read the stream data



practise
Runtime DBR 12.2 LTS • Spark 3.3.2 • Scala 2.12
Driver dev-tier-node • 15.25 GB • 2 Cores

practise DBR 12.2 LTS

Cmd 1

```
from pyspark.sql.types import StructType, StructField, StringType , IntegerTyp

schema = StructType([
                    StructField('Country',StringType()),
                    StructField('Citizens',IntegerType())
])
```

```
Command took 0.14 seconds -- by shanmukhsattiraju999@gmail.com at 06/12/2023, 09:46:29 on practise
```

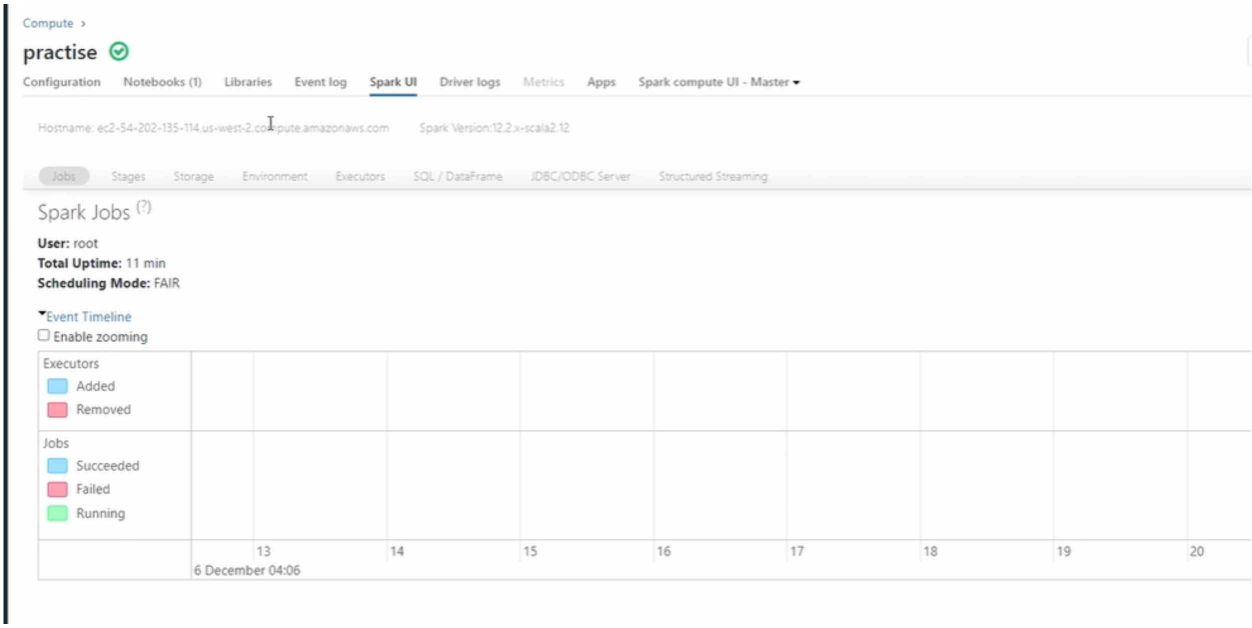
Cmd 2

```
df = spark.readStream.format("csv")\
        .option('header','true')\
        .schema(schema)\
        .load('dbfs:/FileStore/streaming/Countries1.csv')
```

```
df: pyspark.sql.dataframe.DataFrame = [Country: string, Citizens: integer]
Command took 0.94 seconds -- by shanmukhsattiraju999@gmail.com at 06/12/2023, 09:46:42 on practise
```

Cmd 3

Jobs initiated can be read through the spark UI interface


Compute ›
practise
Configuration Notebooks (1) Libraries Event log Spark UI Driver logs Metrics Apps Spark compute UI - Master
Jobs Stages Storage Environment Executors SQL / DataFrame JDBC/ODBC Server Structured Streaming
Event Timeline
Executors
Added
Removed
Jobs
Succeeded
Failed
Running
13
14
15
16
17
18
19
20
6 December 04:06

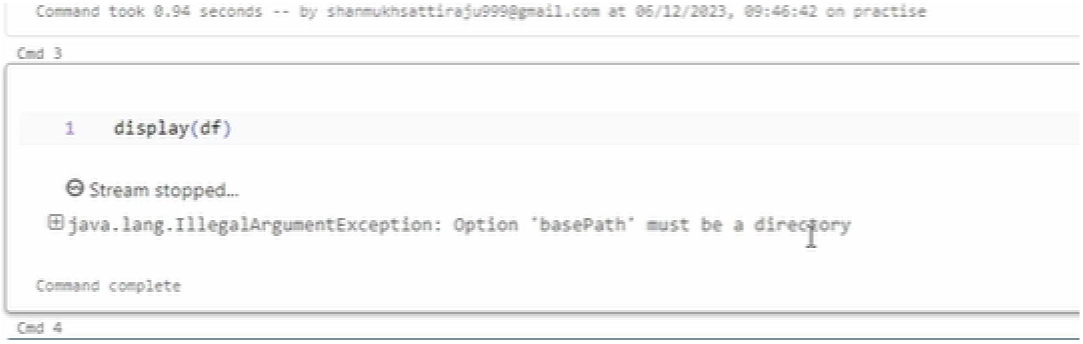
Hostname: ec2-54-202-135-114.us-west-2.compute.amazonaws.com Spark Version:12.2.x-scala2.12

Spark Jobs (?)

User: root
Total Uptime: 11 min
Scheduling Mode: FAIR

□ Enable zooming

Initially no jobs were initiated, jobs will be initiated only when trying to fetch some data.



```
Command took 0.94 seconds -- by shanmukhsattiraju999@gmail.com at 06/12/2023, 09:46:42 on practise
```

Cmd 3

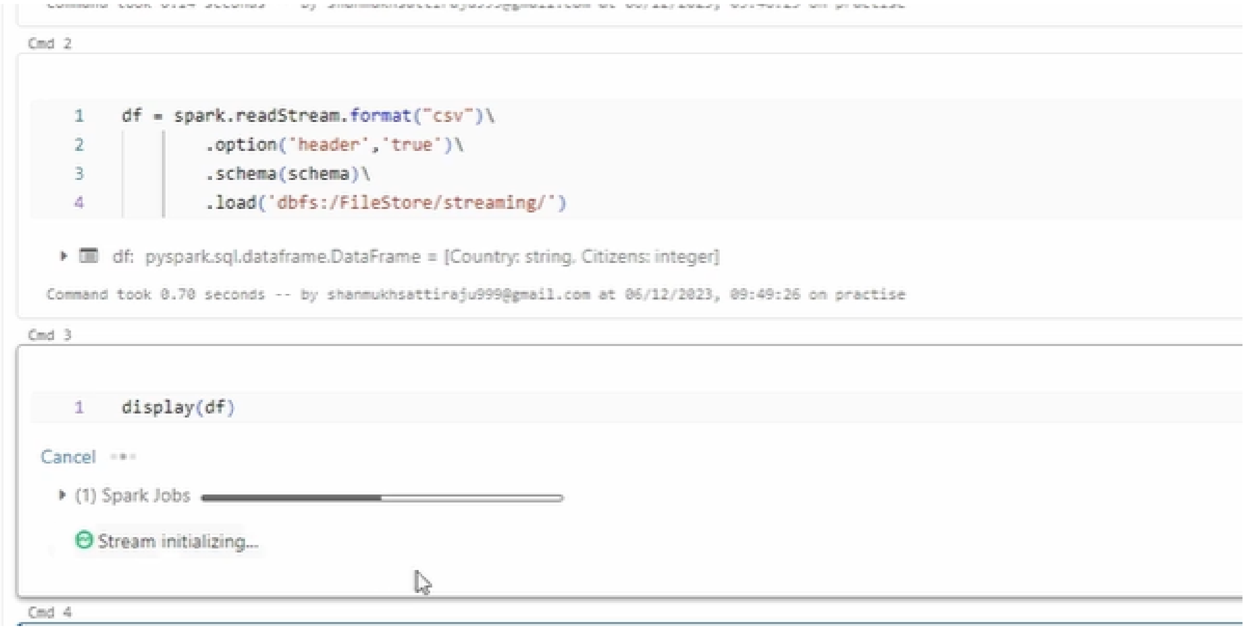
```
display(df)
```

```
Stream stopped...
java.lang.IllegalArgumentException: Option 'basePath' must be a directory

Command complete
```

Cmd 4

Streaming data can be read through display commad, however basepath must be a directory.



Cmd 2

```
df = spark.readStream.format("csv")\
        .option('header','true')\
        .schema(schema)\
        .load('dbfs:/FileStore/streaming/')
```

```
df: pyspark.sql.dataframe.DataFrame = [Country: string, Citizens: integer]
Command took 0.70 seconds -- by shanmukhsattiraju999@gmail.com at 06/12/2023, 09:49:26 on practise
```

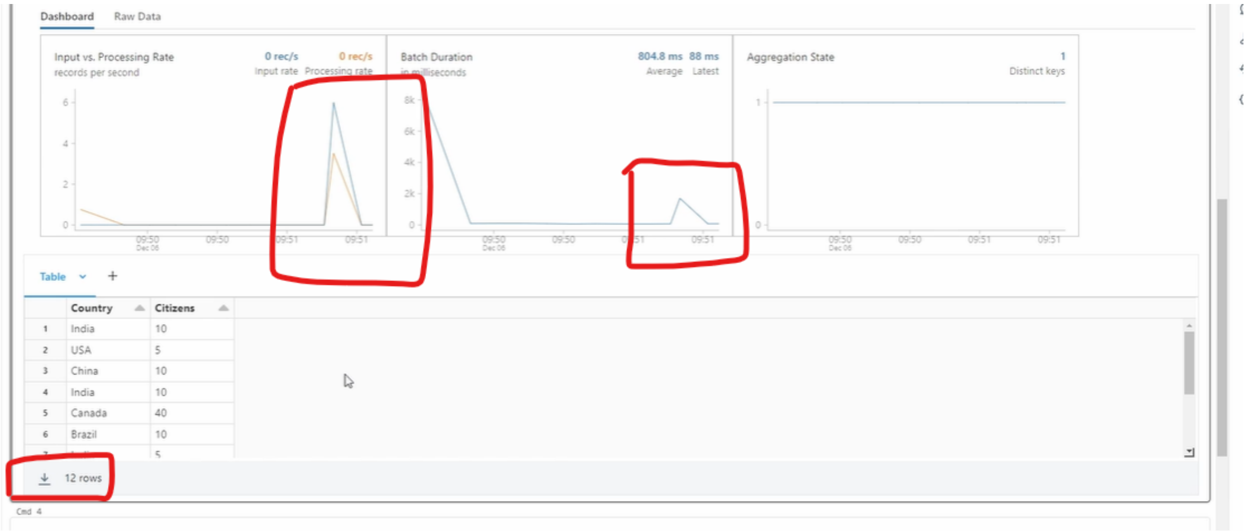
Cmd 3

```
display(df)
```

```
Cancel
(1) Spark Jobs
Stream initializing...
```

Cmd 4

To fix it change the load path to directly to directory level.

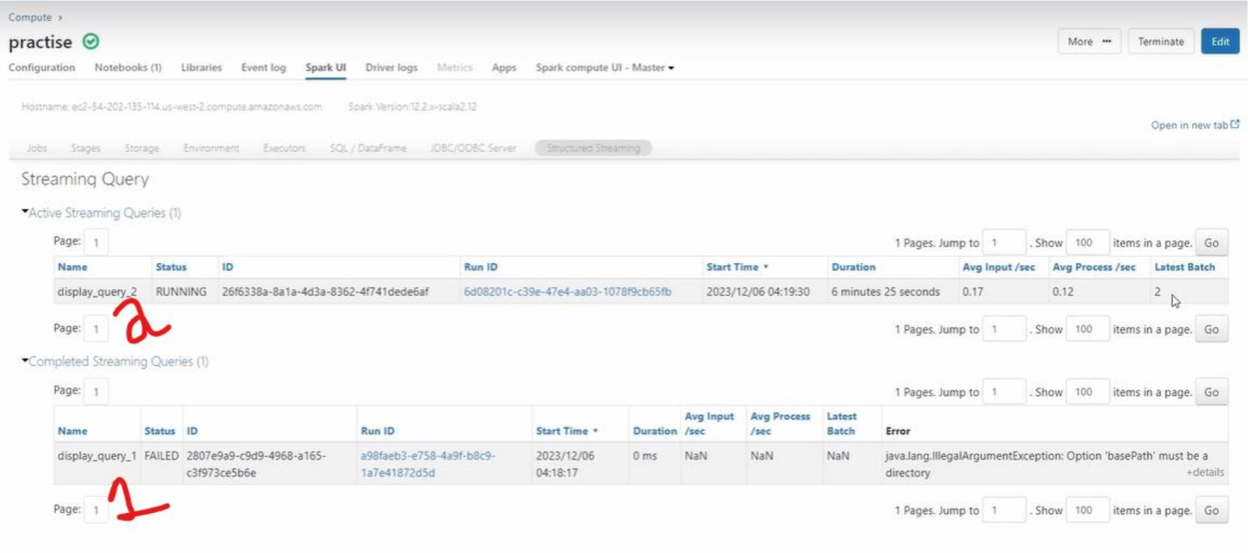


	Country	Citizens
1	India	10
2	USA	5
3	China	10
4	India	10
5	Canada	40
6	Brazil	10
7	[illegible]	5

12 rows

Cmd 4

To observe the change in the streaming input data, add a new file to the directory. With in few seconds will be noticed in the input and batch streaming graph, without re-executing the display command. Record count is changed to 12 from 6.



Streaming Query

Active Streaming Queries (1)

Page: 1 — 1 Pages. Jump to 1 . Show 100 items in a page. Go

Name	Status	ID	Run ID	Start Time	Duration	Avg Input /sec	Avg Process /sec	Latest Batch
display_query_2	RUNNING	26f6338a-8a1a-4d3a-8362-4f741dede6af	6d08201c-c39e-47e4-aa03-1078f9cb65fb	2023/12/06 04:19:30	6 minutes 25 seconds	0.17	0.12	2

Page: 1 — 1 Pages. Jump to 1 . Show 100 items in a page. Go

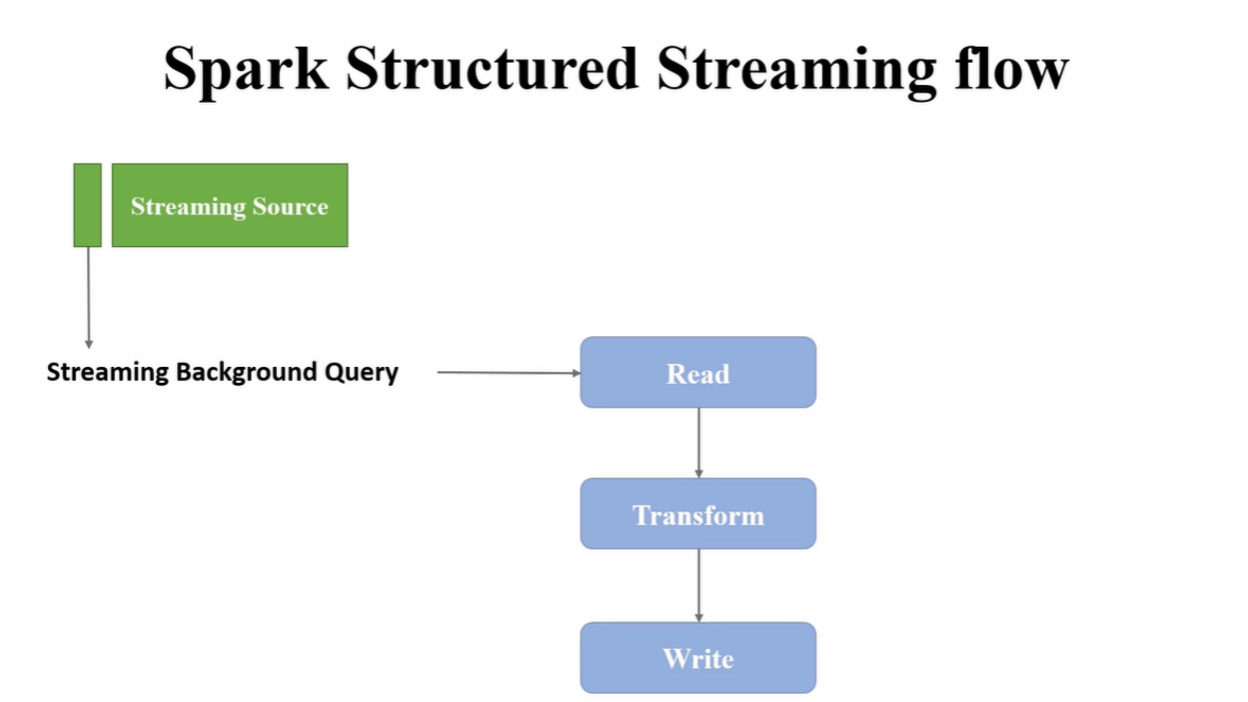
Completed Streaming Queries (1)

Page: 1 — 1 Pages. Jump to 1 . Show 100 items in a page. Go

Name	Status	ID	Run ID	Start Time	Duration	Avg Input /sec	Avg Process /sec	Latest Batch	Error
display_query_1	FAILED	2807e9a9-c9d9-4968-a165-c3f973ce5b6e	a98faeb3-e758-4a9f-b8c9-1a7e41872d5d	2023/12/06 04:18:17	0 ms	NaN	NaN	NaN	java.lang.IllegalArgumentException: Option 'basePath' must be a directory +details

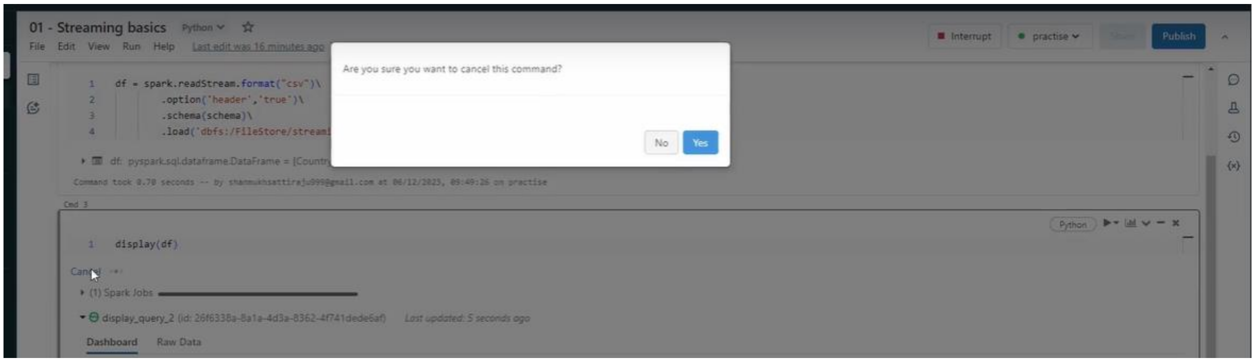
Page: 1 — 1 Pages. Jump to 1 . Show 100 items in a page. Go

Each file addition is read in micro batches.



Spark Structured Streaming flow

The background streaming query acts as a file watcher, whenever there is a new file added to the directory, a new job will be invoked, the file will be read, transformed and written. The query will be running continuously, until it is stopped explicitly.



Are you sure you want to cancel this command?

No   Yes

```
df = spark.readStream.format("csv")\
        .option('header','true')\
        .schema(schema)\
        .load('dbfs:/FileStore/streami
```

df: pyspark.sql.dataframe.DataFrame = [Country

Command took 0.70 seconds -- by shanmukhsattiraju999@gmail.com at 06/12/2023, 09:49:26 on practise

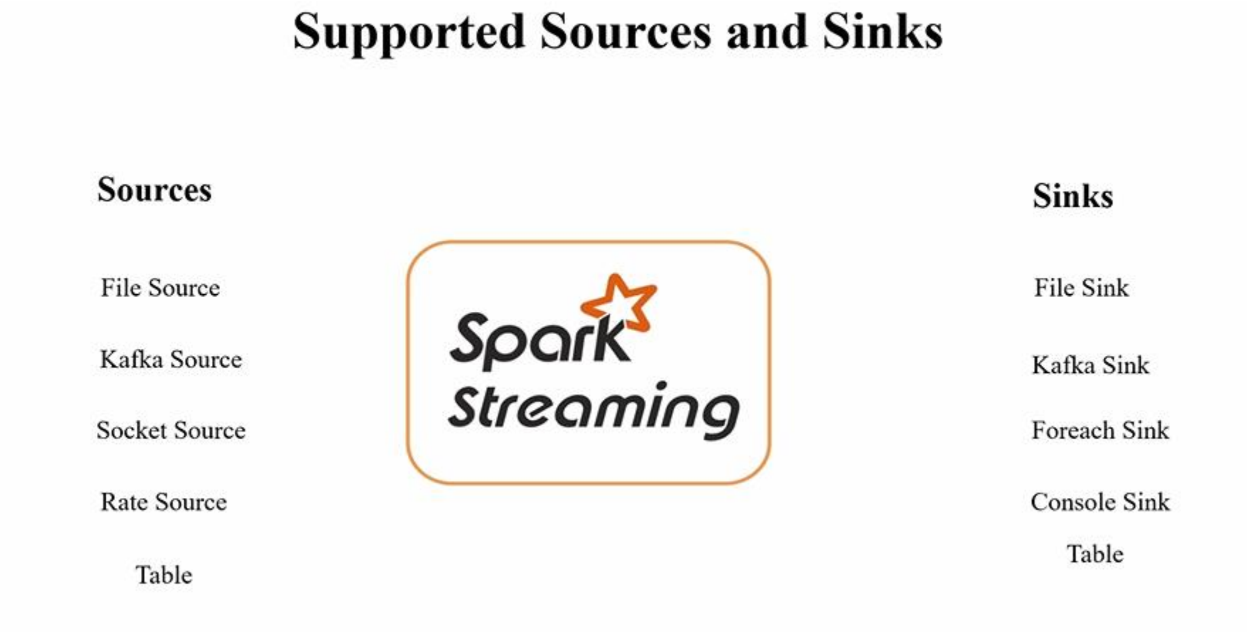
Cmd 3

```
display(df)
```

(1) Spark Jobs

display_query_2 (id: 26f6338a-8a1a-4d3a-8362-4f741dede6af) Last updated: 5 seconds ago

Streaming queries can be stopped explicitly by cancelling the query.



Supported Sources and Sinks

Sources

File Source

Kafka Source

Socket Source

Rate Source

Table

Sinks

File Sink

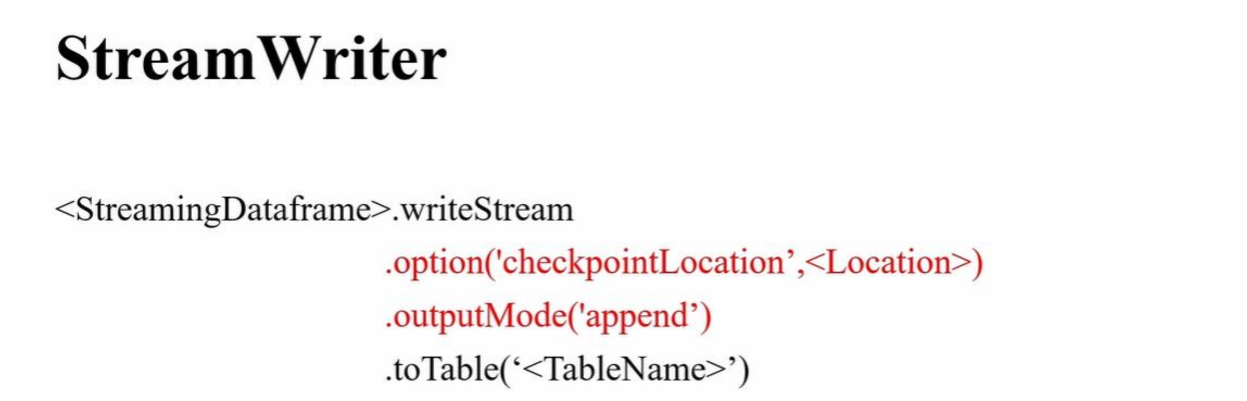
Kafka Sink

Foreach Sink

Console Sink

Table

Supported sources and sinks for the Spark Streaming.



StreamWriter

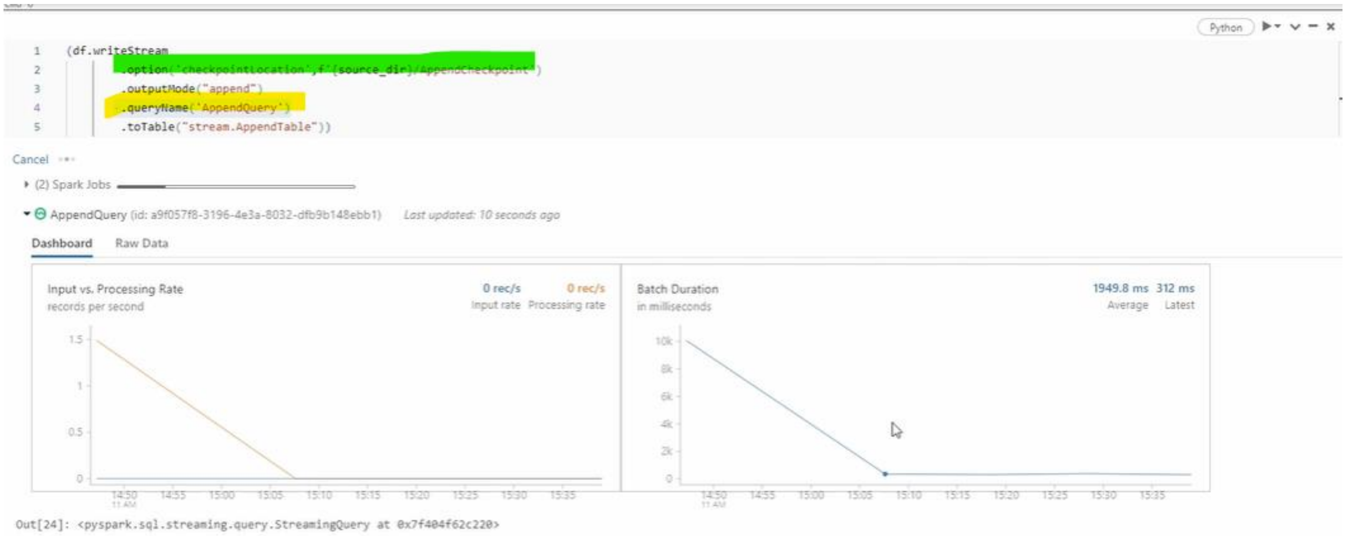
```
<StreamingDataframe>.writeStream
                          .option('checkpointLocation',<Location>)
                          .outputMode('append')
                          .toTable('<TableName>')
```

Checkpoint

To develop fault tolerant and resilient Spark APPS

It maintains intermediate state on fault-tolerant compatible files like DBFS, ADLS and S3 storage systems to recover from the failures.

It is unique to each stream.



```
(df.writeStream
    .option('checkpointLocation',f'{source_dir}/AppendCheckpoint')
    .outputMode("append")
    .queryName('AppendQuery')
    .toTable("stream.AppendTable"))
```

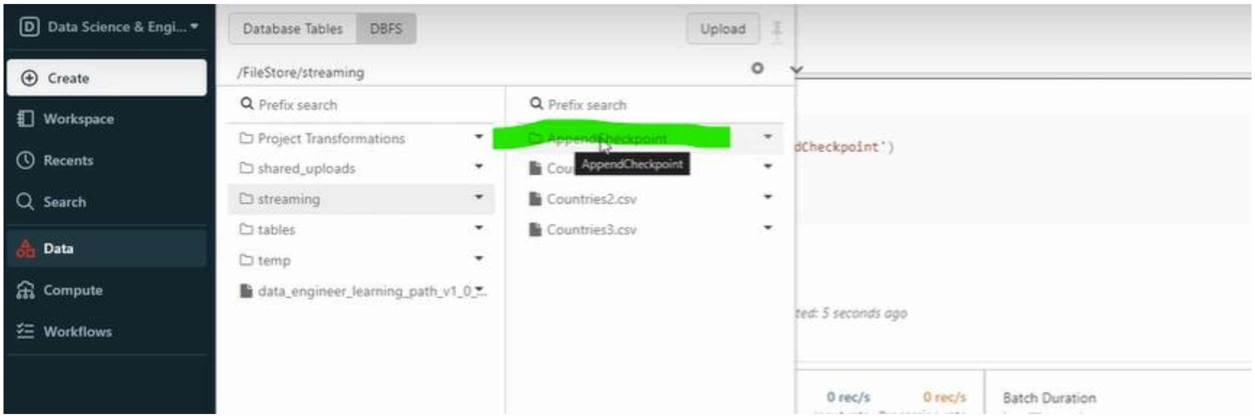
Cancel

(2) Spark Jobs

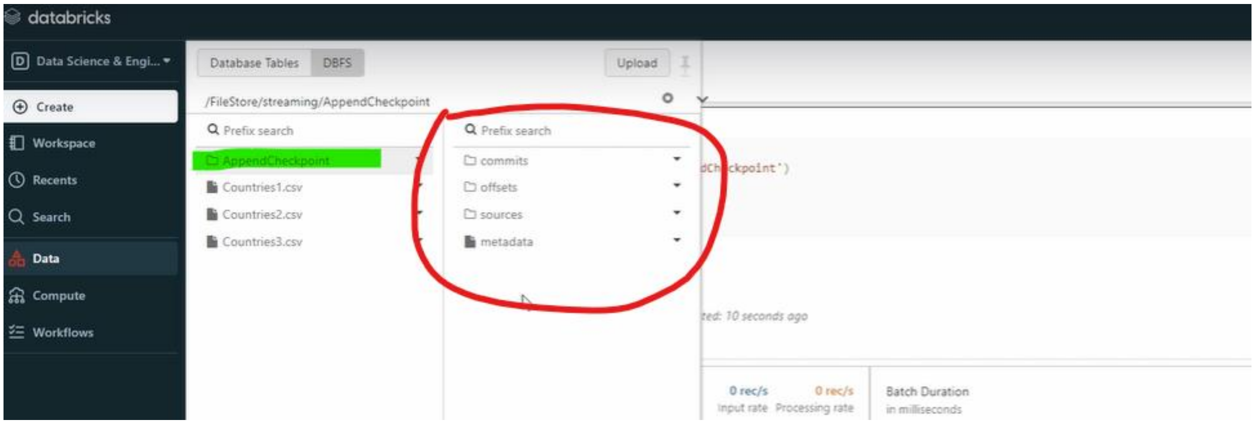
AppendQuery (id: a9f057f8-3196-4e3a-8032-dfb9b148ebb1) *Last updated: 10 seconds ago*

```
Out[24]: <pyspark.sql.streaming.query.StreamingQuery at 0x7f404f62c220>
```

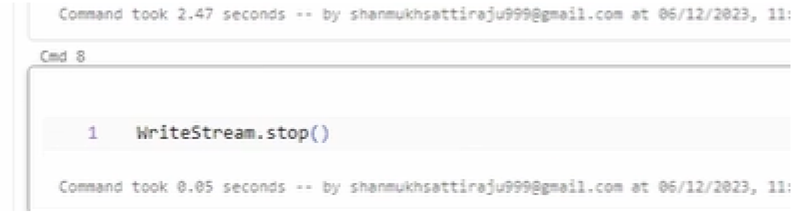
Streaming data frame, data is read, checkpoint location is specified and query name as well specified.



Now after executing the streaming query, a check point location folder is added in the location specified in the above query.



Contains the metadata of the streaming

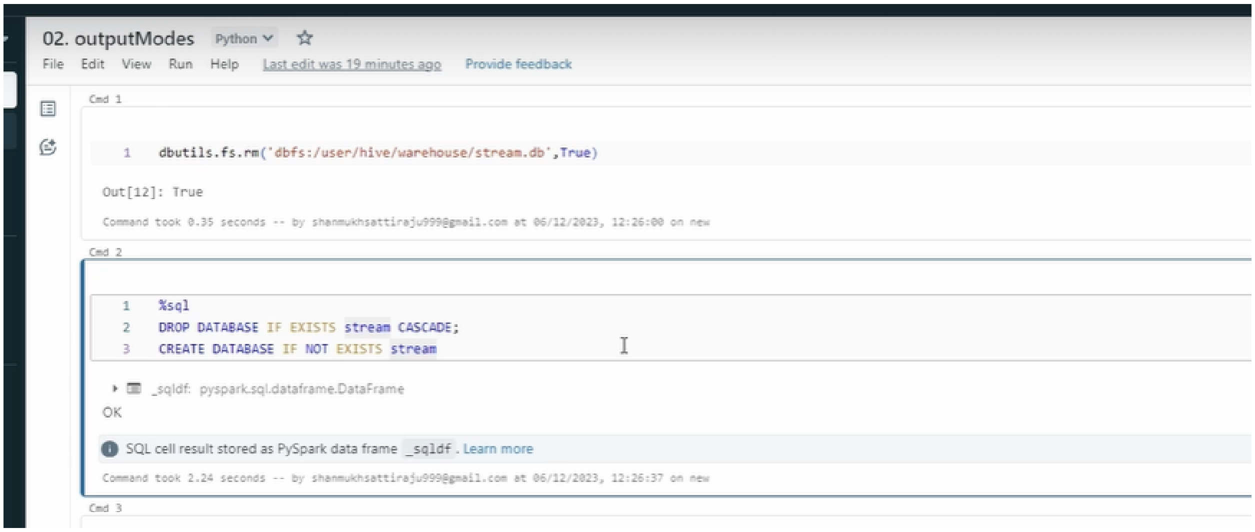


```
Command took 2.47 seconds -- by shanmukhsattiraju999@gmail.com at 06/12/2023, 11:
Cmd 8

WriteStream.stop()

Command took 0.05 seconds -- by shanmukhsattiraju999@gmail.com at 06/12/2023, 11:
```

Assign the streaming query to a variable and to stop the streaming use the command **<var>.stop()**


02. outputModes Python

File Edit View Run Help Last edit was 19 minutes ago Provide feedback

Cmd 1

```
dbutils.fs.rm('dbfs:/user/hive/warehouse/stream.db',True)
```

```
Out[12]: True

Command took 0.35 seconds -- by shanmukhsattiraju999@gmail.com at 06/12/2023, 12:26:00 on new
```

Cmd 2

```
%sql
DROP DATABASE IF EXISTS stream CASCADE;
CREATE DATABASE IF NOT EXISTS stream
```

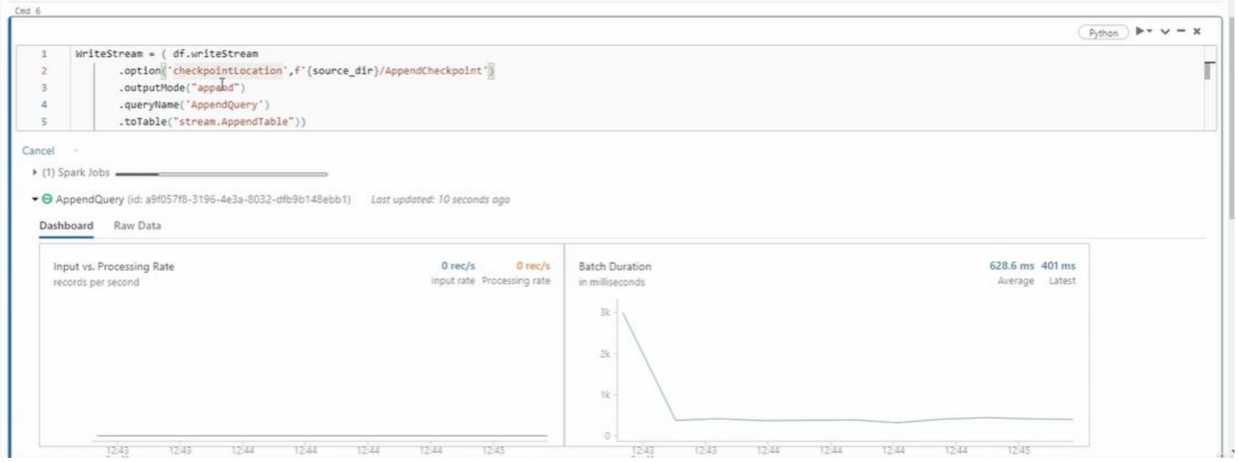
```
_sqldf: pyspark.sql.dataframe.DataFrame
OK
```

SQL cell result stored as PySpark data frame _sqldf. Learn more

```
Command took 2.24 seconds -- by shanmukhsattiraju999@gmail.com at 06/12/2023, 12:26:37 on new
```

Cmd 3

In the community edition when the cluster is terminated all the schema/database created in the working session will be terminated. Therefore, while using the community edition we should remove the files in the dbfs path, drop the database if exists and re-create it.


Cmd 6

```
WriteStream = ( df.writeStream
        .option('checkpointLocation',f'{source_dir}/AppendCheckpoint')
        .outputMode("append")
        .queryName('AppendQuery')
        .toTable("stream.AppendTable"))
```

Cancel

(1) Spark Jobs

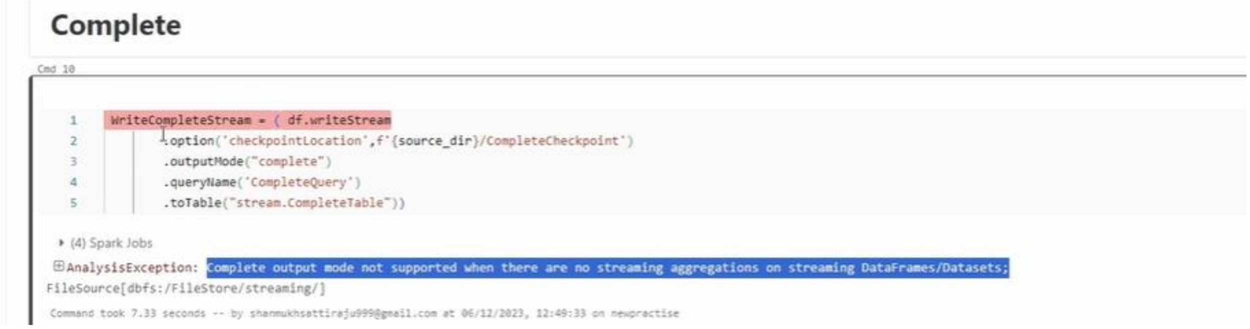
AppendQuery (id: a9f057f8-3196-4e3a-8032-dfb9b148ebb1) Last updated: 10 seconds ago

When we are trying to run the stream query initially it will verify the check point location, if there are any unprocessed files, only then it will show the updates in the graph

Output Modes

Deleting the checkpoint location will help to process the older data, without removing the data, just by removing the checkpoint, previous data will be processed.

Complete Mode



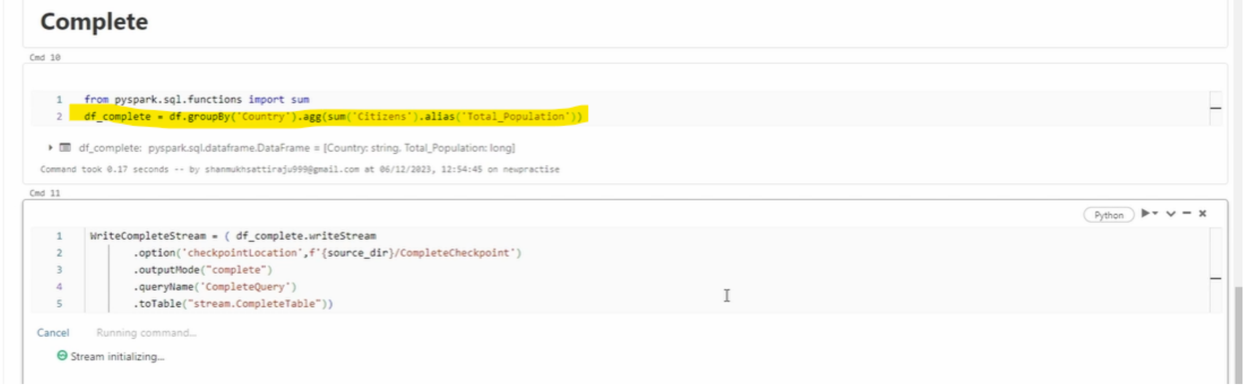
Complete

Cmd 10

```
WriteCompleteStream = ( df.writeStream
        .option('checkpointLocation',f'{source_dir}/CompleteCheckpoint')
        .outputMode("complete")
        .queryName('CompleteQuery')
        .toTable("stream.CompleteTable"))
```

```
(4) Spark Jobs
AnalysisException: Complete output mode not supported when there are no streaming aggregations on streaming DataFrames/Datasets;
FileSource[dbfs:/FileStore/streaming/]
Command took 7.33 seconds -- by shanmukhsattiraju999@gmail.com at 06/12/2023, 12:49:33 on newpractise
```

To use the complete output mode aggregation should be used on the data frame.



Complete

Cmd 10

```
from pyspark.sql.functions import sum
df_complete = df.groupBy('Country').agg(sum('Citizens').alias('Total_Population'))
```

```
df_complete: pyspark.sql.dataframe.DataFrame = [Country: string, Total_Population: long]
Command took 0.17 seconds -- by shanmukhsattiraju999@gmail.com at 06/12/2023, 12:54:45 on newpractise
```

Cmd 11

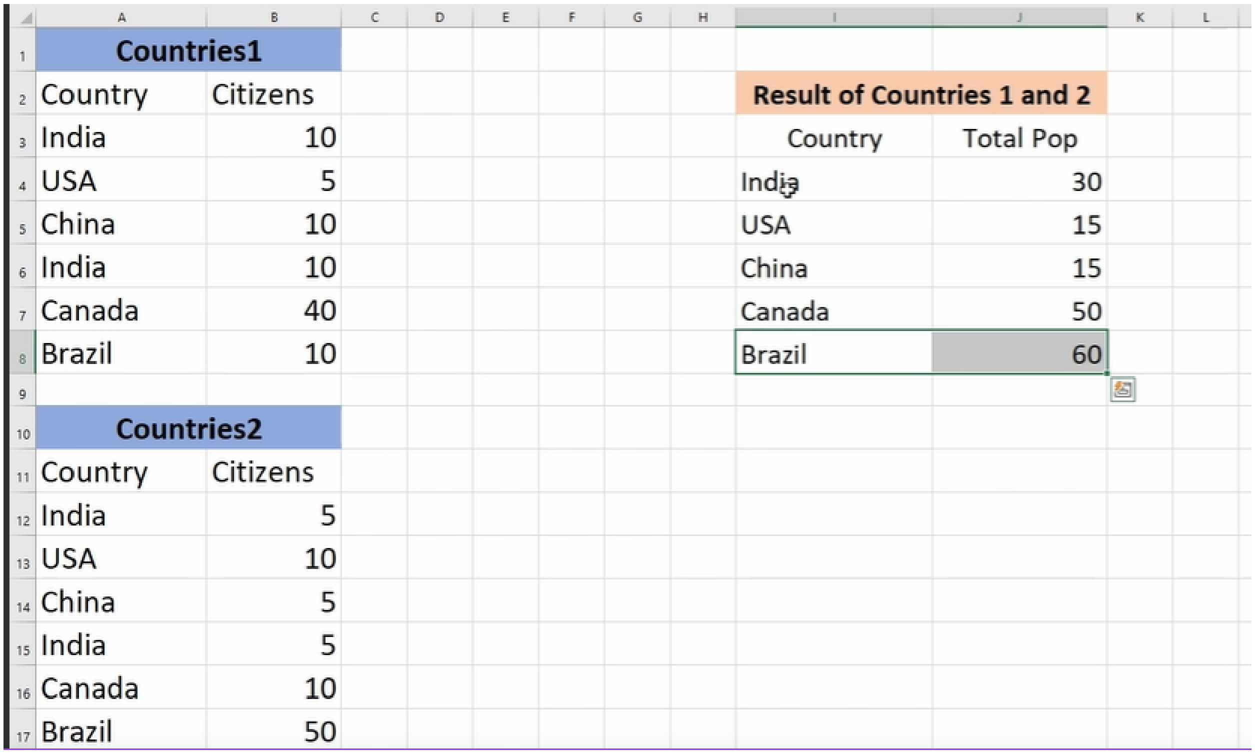
```
WriteCompleteStream = ( df_complete.writeStream
        .option('checkpointLocation',f'{source_dir}/CompleteCheckpoint')
        .outputMode("complete")
        .queryName('CompleteQuery')
        .toTable("stream.CompleteTable"))
```

```
Running command...
Stream initializing...
```

Applied aggregation to execute the ***complete output mode***

Use case of complete mode

When a new file or dataset is added, and we need to apply the aggregation on top it, to fetch the results of the query entire files should be read.

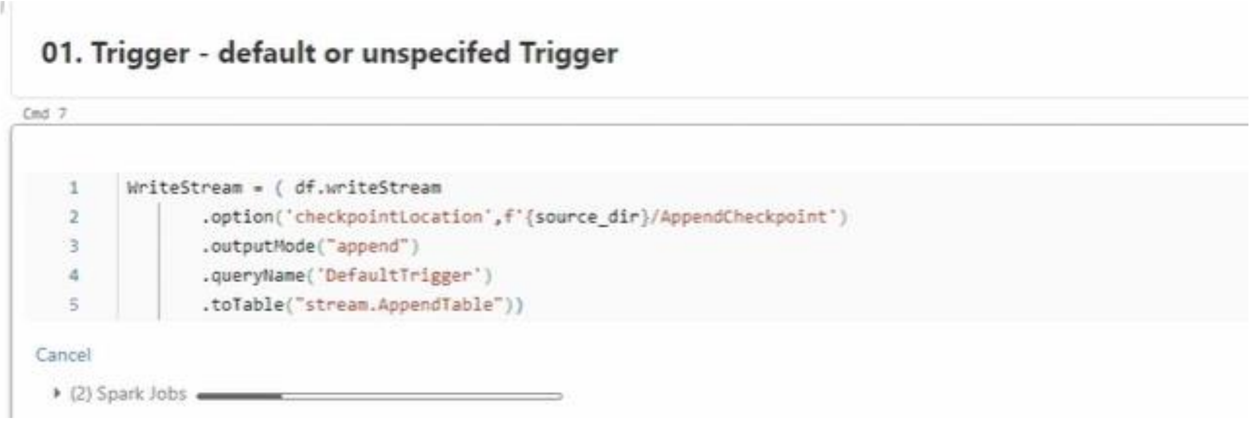


Countries1	
Country	Citizens
India	10
USA	5
China	10
India	10
Canada	40
Brazil	10

Countries2	
Country	Citizens
India	5
USA	10
China	5
India	5
Canada	10
Brazil	50

Result of Countries 1 and 2	
Country	Total Pop
India	30
USA	15
China	15
Canada	50
Brazil	60

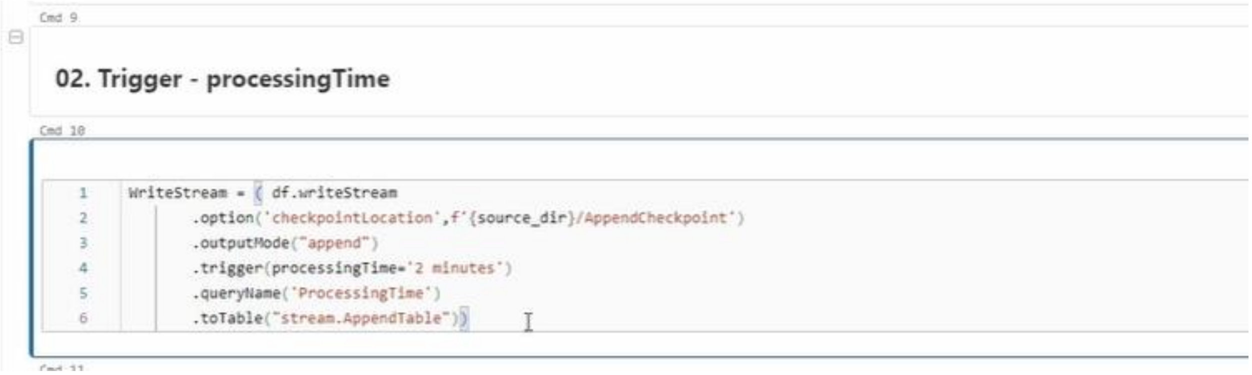
Trigger types



01. Trigger - default or unspecifed Trigger

```
WriteStream = ( df.writeStream
          .option('checkpointLocation',f'{source_dir}/AppendCheckpoint')
          .outputMode("append")
          .queryName('DefaultTrigger')
          .toTable("stream.AppendTable"))
```

If no trigger time is specified, by default for every 5sec, streaming query will look for new data. If the new file is added every 5 to 10 min, default trigger is not a good option.



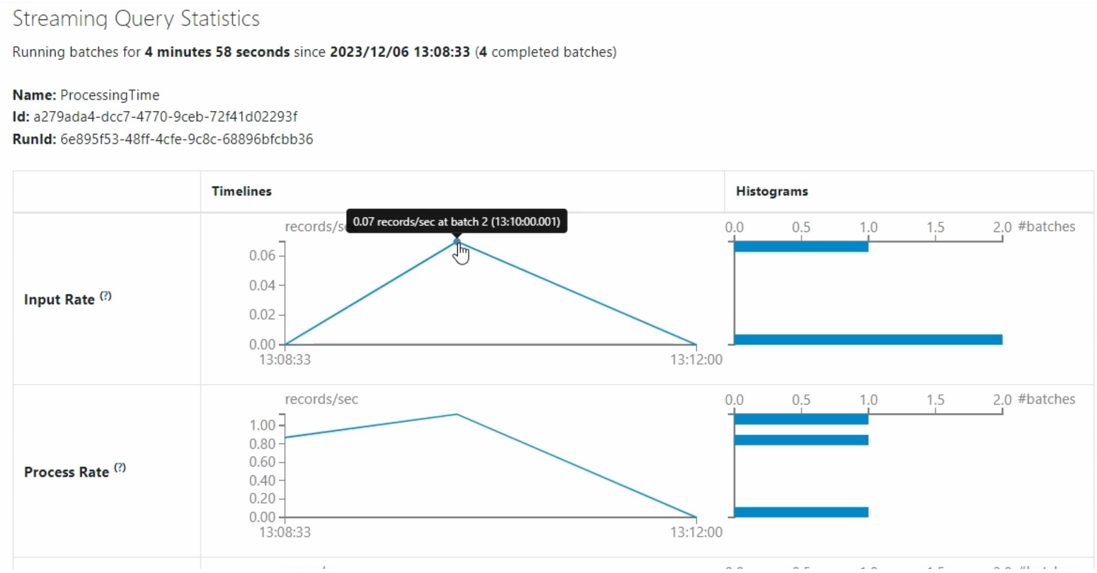
Cmd 9

02. Trigger - processingTime

Cmd 10

```
WriteStream = ( df.writeStream
        .option('checkpointLocation',f'{source_dir}/AppendCheckpoint')
        .outputMode("append")
        .trigger(processingTime='2 minutes')
        .queryName('ProcessingTime')
        .toTable("stream.AppendTable"))
```

Every 2 min, a micro batch will be processed.

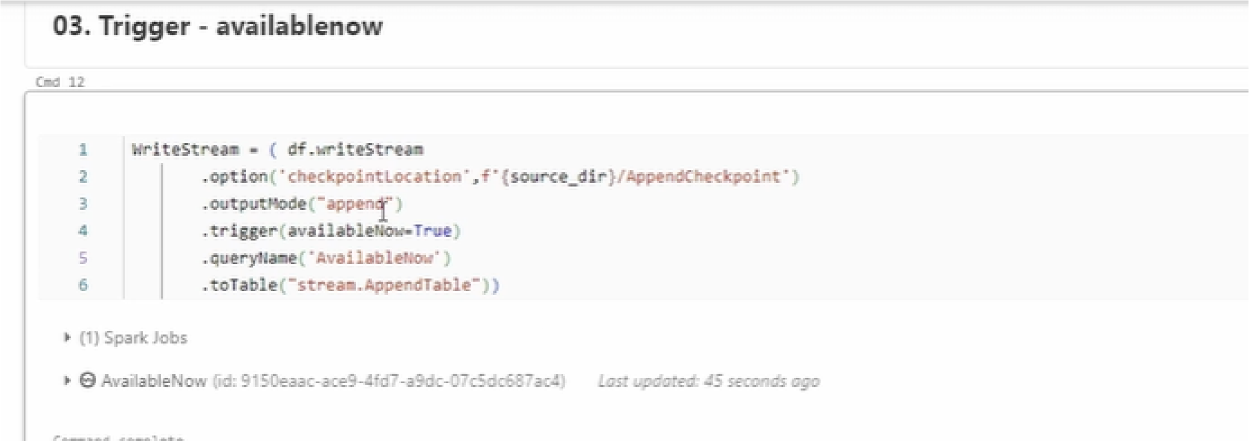


Streaming Query Statistics

Running batches for **4 minutes 58 seconds** since **2023/12/06 13:08:33** (**4** completed batches)

Name: ProcessingTime
Id: a279ada4-dcc7-4770-9ceb-72f41d02293f
RunId: 6e895f53-48ff-4cfe-9c8c-68896bfcbb36

At 13:08 initial processing occurred, even though data is available only after 2 min next micro batch will be processed.



03. Trigger - availablenow

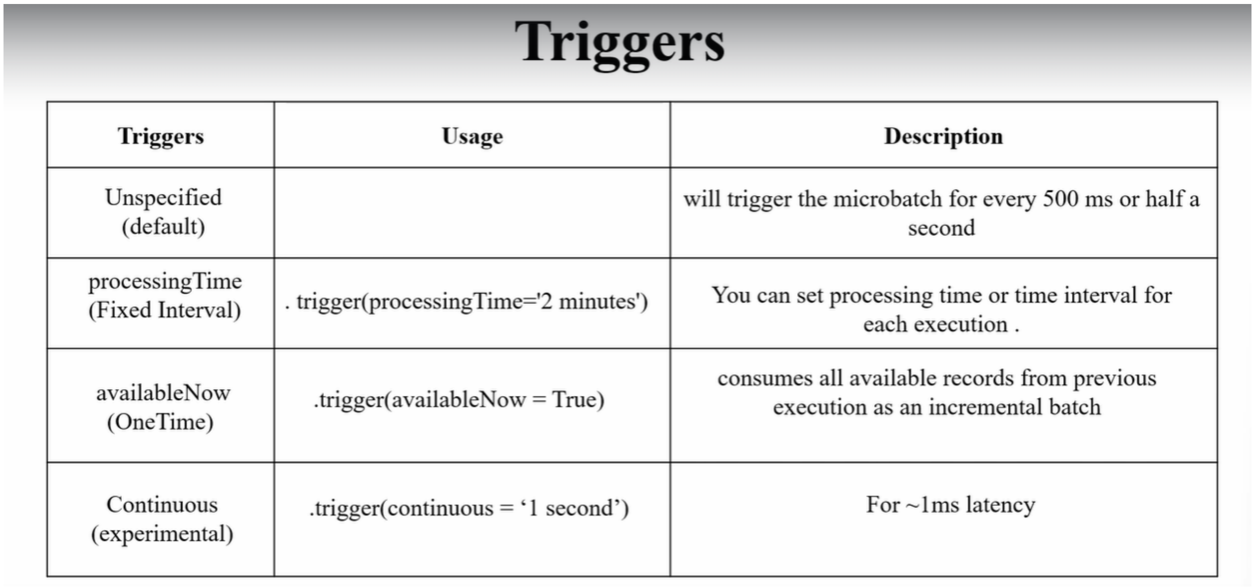
Cmd 12

```
WriteStream = ( df.writeStream
        .option('checkpointLocation',f'{source_dir}/AppendCheckpoint')
        .outputMode("append")
        .trigger(availableNow=True)
        .queryName('AvailableNow')
        .toTable("stream.AppendTable"))
```

▸ (1) Spark Jobs

▸ AvailableNow (id: 9150eaac-ace9-4fd7-a9dc-07c5dc687ac4)   *Last updated: 45 seconds ago*

Is equivalent to incremental batch processing, whenever there is a new data available query will process and stop the streaming

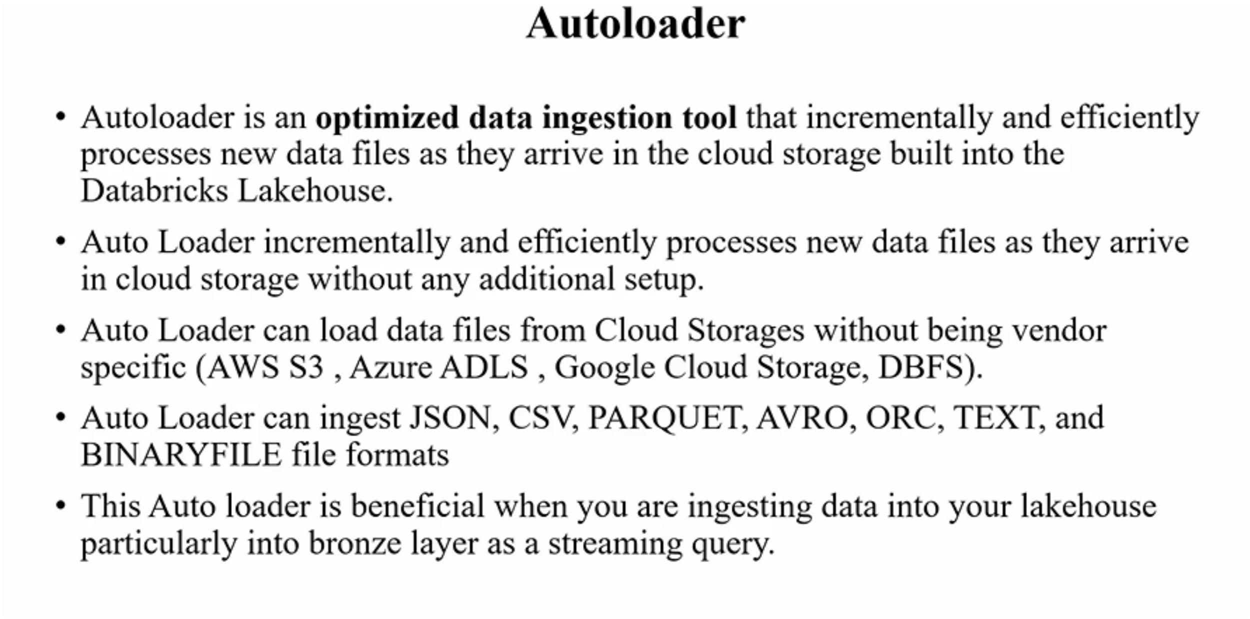


Triggers

Triggers	Usage	Description
Unspecified (default)		will trigger the microbatch for every 500 ms or half a second
processingTime (Fixed Interval)	. trigger(processingTime='2 minutes')	You can set processing time or time interval for each execution .
availableNow (OneTime)	.trigger(availableNow = True)	consumes all available records from previous execution as an incremental batch
Continuous (experimental)	.trigger(continuous = ‘1 second’)	For ~1ms latency

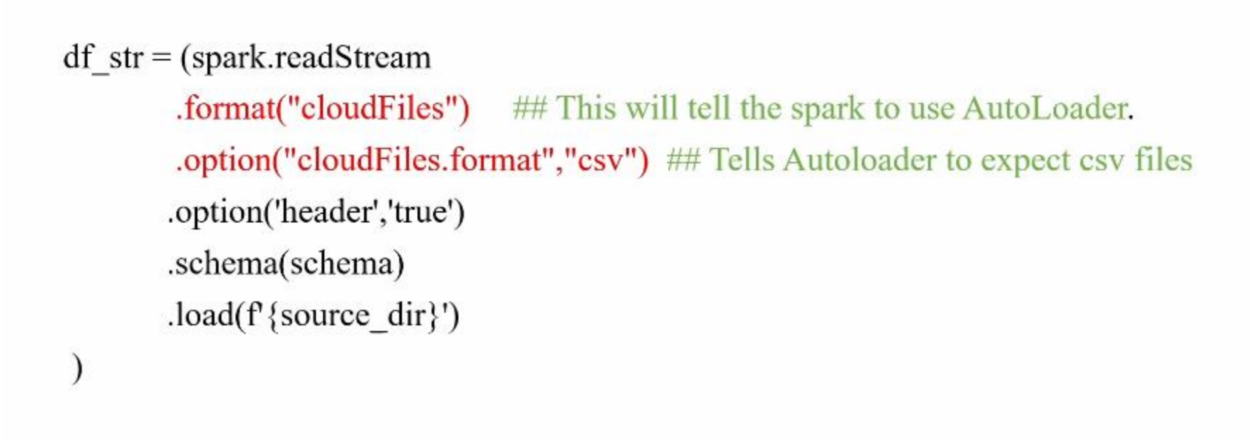
Types of streaming triggers

Autoloader



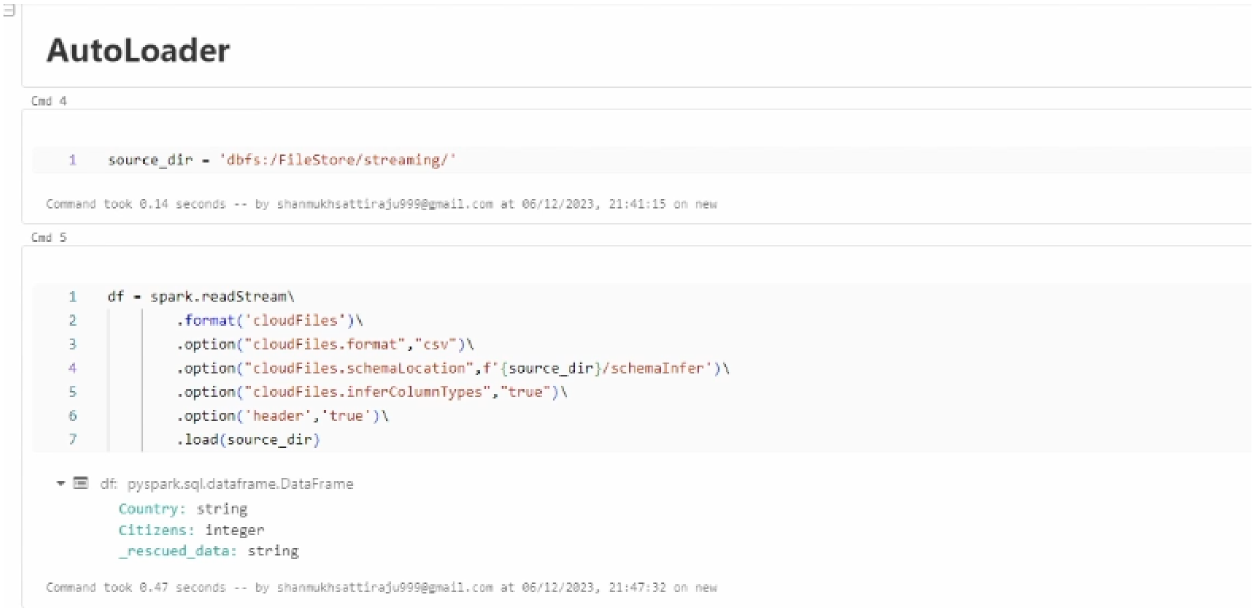
Autoloader

- Autoloader is an **optimized data ingestion tool** that incrementally and efficiently processes new data files as they arrive in the cloud storage built into the Databricks Lakehouse.
- Auto Loader incrementally and efficiently processes new data files as they arrive in cloud storage without any additional setup.
- Auto Loader can load data files from Cloud Storages without being vendor specific (AWS S3 , Azure ADLS , Google Cloud Storage, DBFS).
- Auto Loader can ingest JSON, CSV, PARQUET, AVRO, ORC, TEXT, and BINARYFILE file formats
- This Auto loader is beneficial when you are ingesting data into your lakehouse particularly into bronze layer as a streaming query.



```
df_str = (spark.readStream
          .format("cloudFiles")     ## This will tell the spark to use AutoLoader.
          .option("cloudFiles.format","csv")  ## Tells Autoloader to expect csv files
          .option('header','true')
          .schema(schema)
          .load(f'{source_dir}')
  )
```

To use the autoloader format("cloudFiles") needs to be specified



AutoLoader

Cmd 4

```
source_dir = 'dbfs:/FileStore/streaming/'
```

Command took 0.14 seconds -- by shanmukhsattiraju999@gmail.com at 06/12/2023, 21:41:15 on new

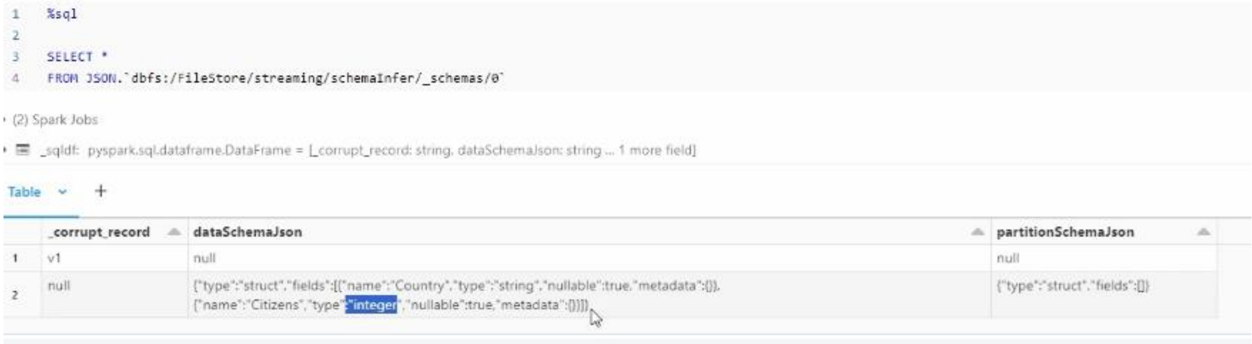
Cmd 5

```
df = spark.readStream\
        .format('cloudFiles')\
        .option("cloudFiles.format","csv")\
        .option("cloudFiles.schemaLocation",f'{source_dir}/schemaInfer')\
        .option("cloudFiles.inferColumnTypes","true")\
        .option('header','true')\
        .load(source_dir)
```

```
df: pyspark.sql.dataframe.DataFrame
    Country: string
    Citizens: integer
    _rescued_data: string
```

Command took 0.47 seconds -- by shanmukhsattiraju999@gmail.com at 06/12/2023, 21:47:32 on new

To get the correct data types, schema must be inferred.



```
%sql

SELECT *
FROM JSON.`dbfs:/FileStore/streaming/schemaInfer/_schemas/0`
```

(2) Spark Jobs

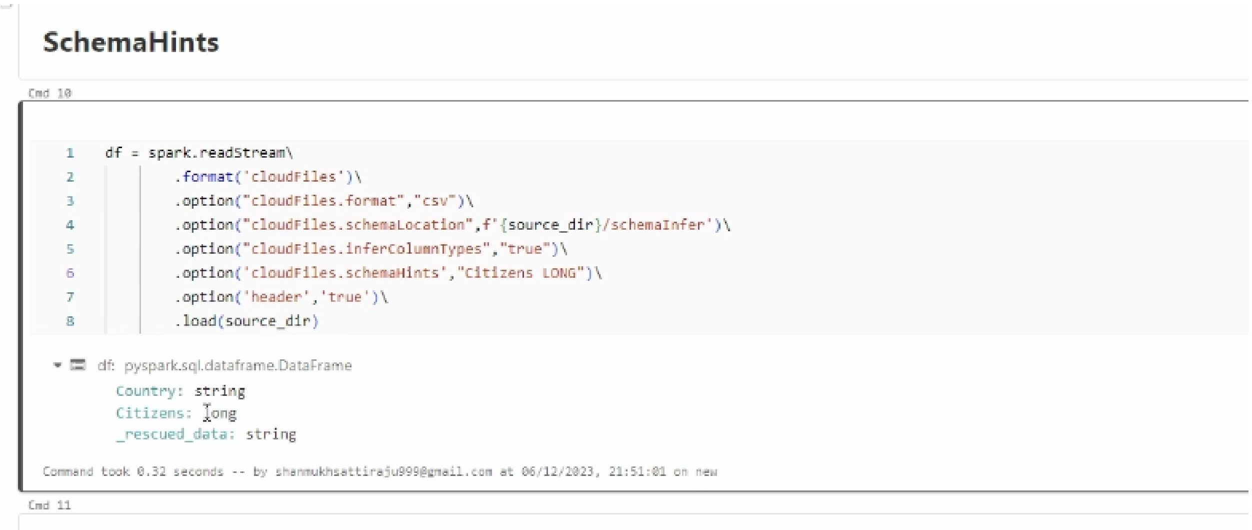
_sqldf: pyspark.sql.dataframe.DataFrame = [_corrupt_record: string, dataSchemaJson: string ... 1 more field]

	_corrupt_record	dataSchemaJson	partitionSchemaJson
1	v1	null	null
2	null	{"type":"struct","fields":[{"name":"Country","type":"string","nullable":true,"metadata":{}},{"name":"Citizens","type":"integer","nullable":true,"metadata":{}}]}	{"type":"struct","fields":[]}

Schema can be referred through the path, rescued data is the additional field created.

Schema Hints

For explicitly defining the data type of a column



SchemaHints

Cmd 10

```
df = spark.readStream\
        .format('cloudFiles')\
        .option("cloudFiles.format","csv")\
        .option("cloudFiles.schemaLocation",f'{source_dir}/schemaInfer')\
        .option("cloudFiles.inferColumnTypes","true")\
        .option('cloudFiles.schemaHints',"Citizens LONG")\
        .option('header','true')\
        .load(source_dir)
```

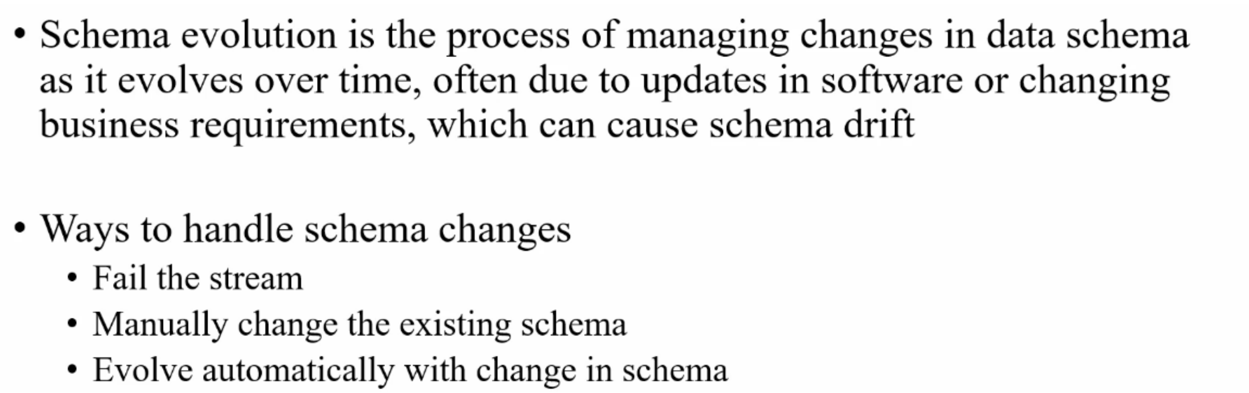
```
df: pyspark.sql.dataframe.DataFrame
    Country: string
    Citizens: long
    _rescued_data: string

Command took 0.32 seconds -- by shanmukhsattiraju999@gmail.com at 06/12/2023, 21:51:01 on new
```

Cmd 11

For explicitly defining the data type of a column

Schema Evolution



- Schema evolution is the process of managing changes in data schema as it evolves over time, often due to updates in software or changing business requirements, which can cause schema drift
- Ways to handle schema changes
  - Fail the stream
  - Manually change the existing schema
  - Evolve automatically with change in schema

There are 4 schema evolution types

1. AddNewColunms

   Stream will fail, if new column is added and existing columns do not evolve.

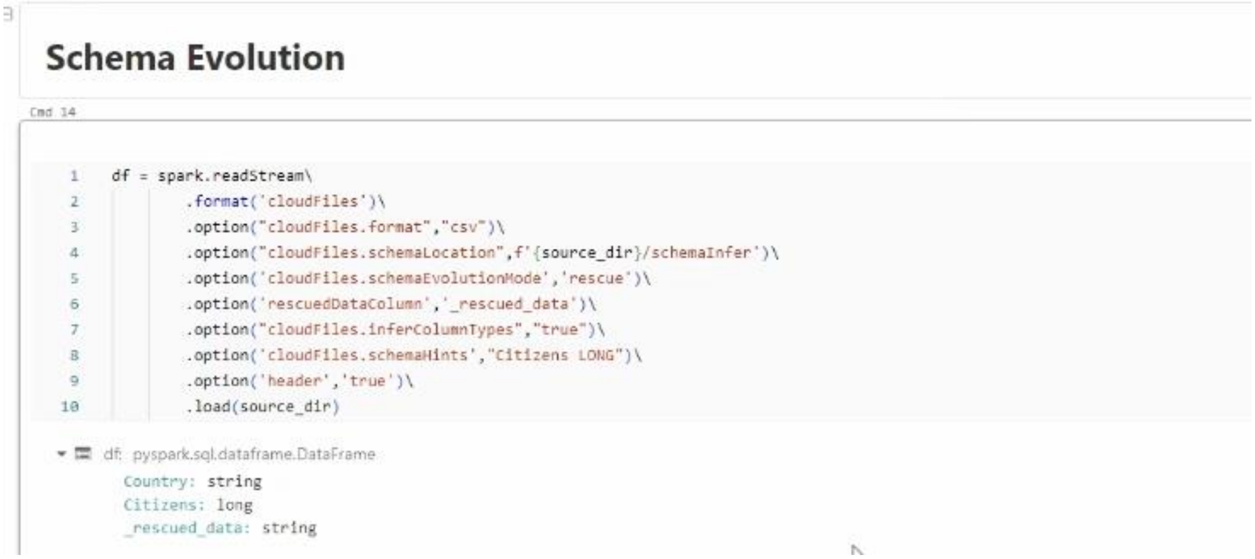
2. FailOnNewColumns

   Stream fails, stream does not restart until schema is updated or offending data is removed.

3. Rescue
   Schema never evolved; stream doesn't change due to schema changes. All new columns are recorded in the rescued data.
4. None

Ignore new columns, (doesn't evolve the schema, new columns are ignored, data is not rescued unless the rescuedDataColumn option is set. Stream doesn't fail due to schema changes)

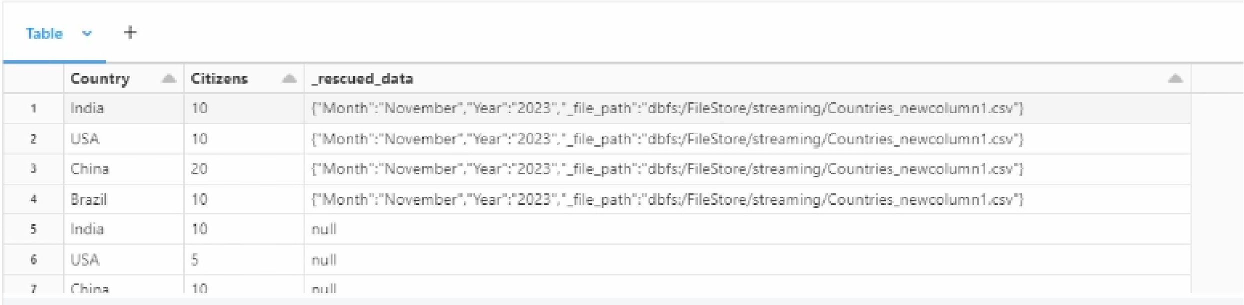
Using the **rescue type schema evolution**



Schema Evolution

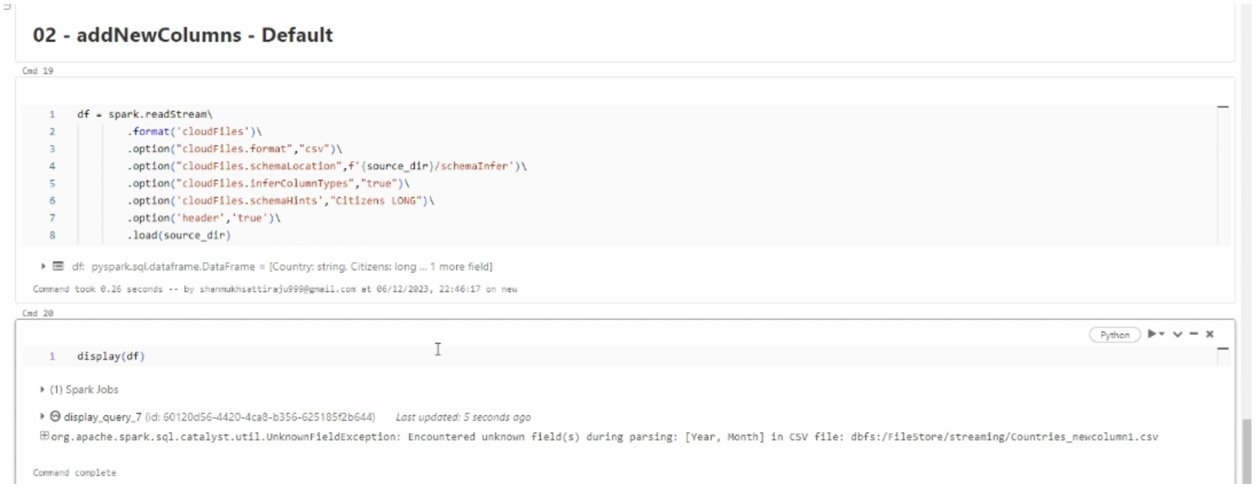
```
df = spark.readStream\
          .format('cloudFiles')\
          .option("cloudFiles.format","csv")\
          .option("cloudFiles.schemaLocation",f'{source_dir}/schemaInfer')\
          .option('cloudFiles.schemaEvolutionMode','rescue')\
          .option('rescuedDataColumn','_rescued_data')\
          .option("cloudFiles.inferColumnTypes","true")\
          .option('cloudFiles.schemaHints',"Citizens LONG")\
          .option('header','true')\
          .load(source_dir)
```

```
df:  pyspark.sql.dataframe.DataFrame
    Country: string
    Citizens: long
    _rescued_data: string
```



	Country	Citizens	_rescued_data
1	India	10	{"Month":"November","Year":"2023","_file_path":"dbfs:/FileStore/streaming/Countries_newcolumn1.csv"}
2	USA	10	{"Month":"November","Year":"2023","_file_path":"dbfs:/FileStore/streaming/Countries_newcolumn1.csv"}
3	China	20	{"Month":"November","Year":"2023","_file_path":"dbfs:/FileStore/streaming/Countries_newcolumn1.csv"}
4	Brazil	10	{"Month":"November","Year":"2023","_file_path":"dbfs:/FileStore/streaming/Countries_newcolumn1.csv"}
5	India	10	null
6	USA	5	null
7	China	10	null

Extra columns that are added handled by the rescued_data column



02 - addNewColumns - Default

Cmd 19

```
df = spark.readStream\
        .format('cloudFiles')\
        .option("cloudFiles.format","csv")\
        .option("cloudFiles.schemaLocation",f'{source_dir}/schemaInfer')\
        .option("cloudFiles.inferColumnTypes","true")\
        .option('cloudFiles.schemaHints',"Citizens LONG")\
        .option('header','true')\
        .load(source_dir)
```

df: pyspark.sql.dataframe.DataFrame = [Country: string, Citizens: long ... 1 more field]

Command took 0.26 seconds -- by shanmukhsattiraju999@gmail.com at 06/12/2023, 22:46:17 on new

Cmd 20

```
display(df)
```

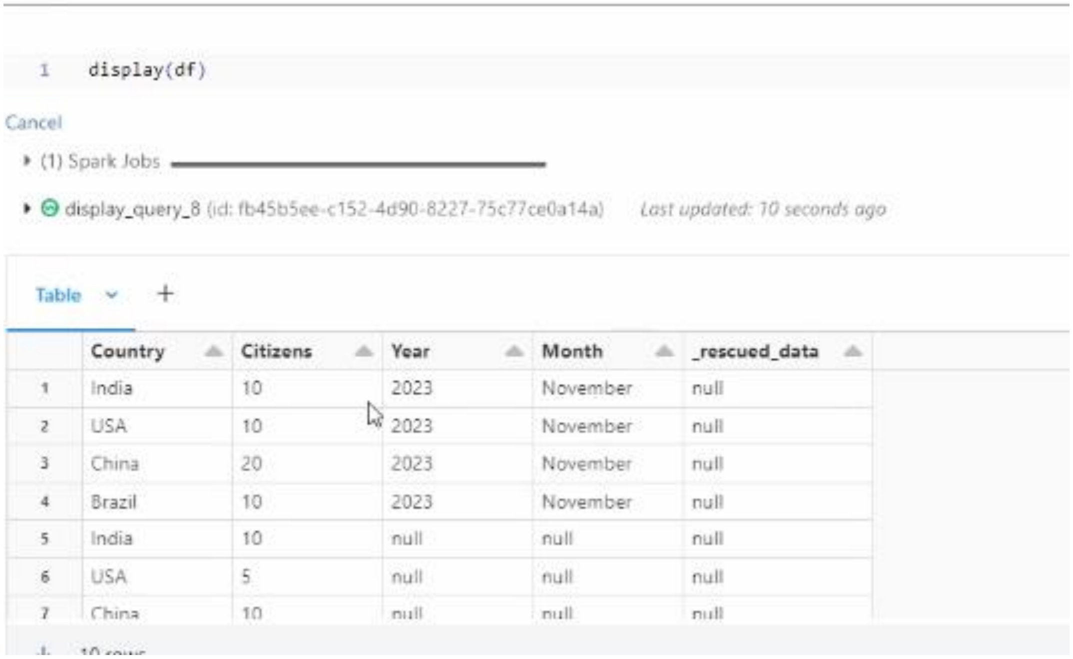
(1) Spark Jobs

display_query_7 (id: 60120d56-4420-4ca8-b356-625185f2b644) *Last updated: 5 seconds ago*

```
org.apache.spark.sql.catalyst.util.UnknownFieldException: Encountered unknown field(s) during parsing: [Year, Month] in CSV file: dbfs:/FileStore/streaming/Countries_newcolumn1.csv
```

Command complete

In the addNewColumns(default) type, after adding a file with extra columns, dataframe will be executed, however when we do a display it will fail.



```
display(df)
```

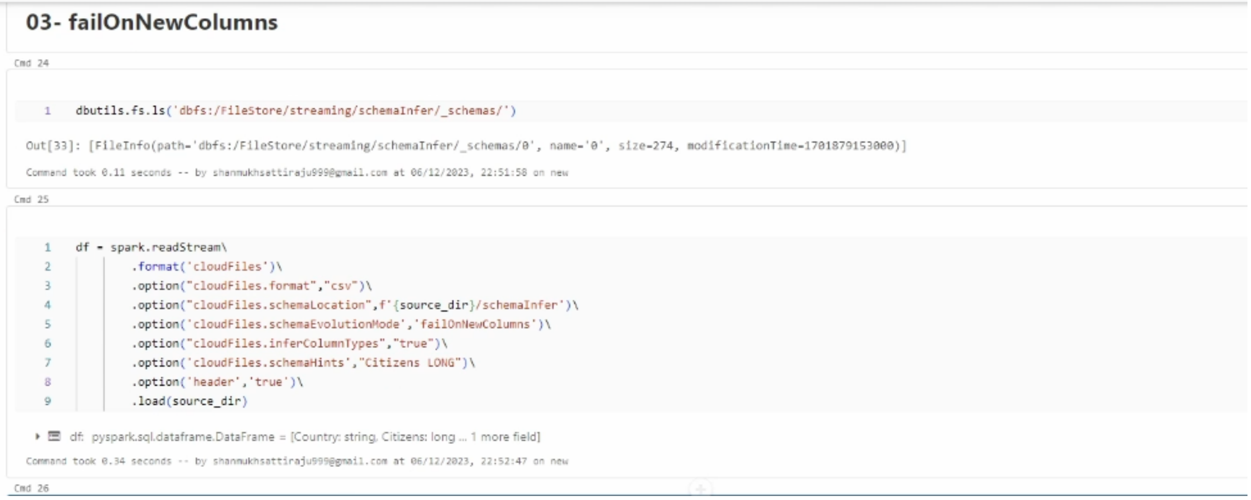
(1) Spark Jobs

display_query_8 (id: fb45b5ee-c152-4d90-8227-75c77ce0a14a) *Last updated: 10 seconds ago*

	Country	Citizens	Year	Month	_rescued_data
1	India	10	2023	November	null
2	USA	10	2023	November	null
3	China	20	2023	November	null
4	Brazil	10	2023	November	null
5	India	10	null	null	null
6	USA	5	null	null	null
7	China	10	null	null	null

10 rows

To fix this need to rerun the data frame and display to see the data with evolved schema.



03- failOnNewColumns

Cmd 24

```
dbutils.fs.ls('dbfs:/FileStore/streaming/schemaInfer/_schemas/')
```

```
Out[33]: [FileInfo(path='dbfs:/FileStore/streaming/schemaInfer/_schemas/0', name='0', size=274, modificationTime=1701879153000)]
```

Command took 0.11 seconds -- by shanmukhsattiraju999@gmail.com at 06/12/2023, 22:51:58 on new

Cmd 25

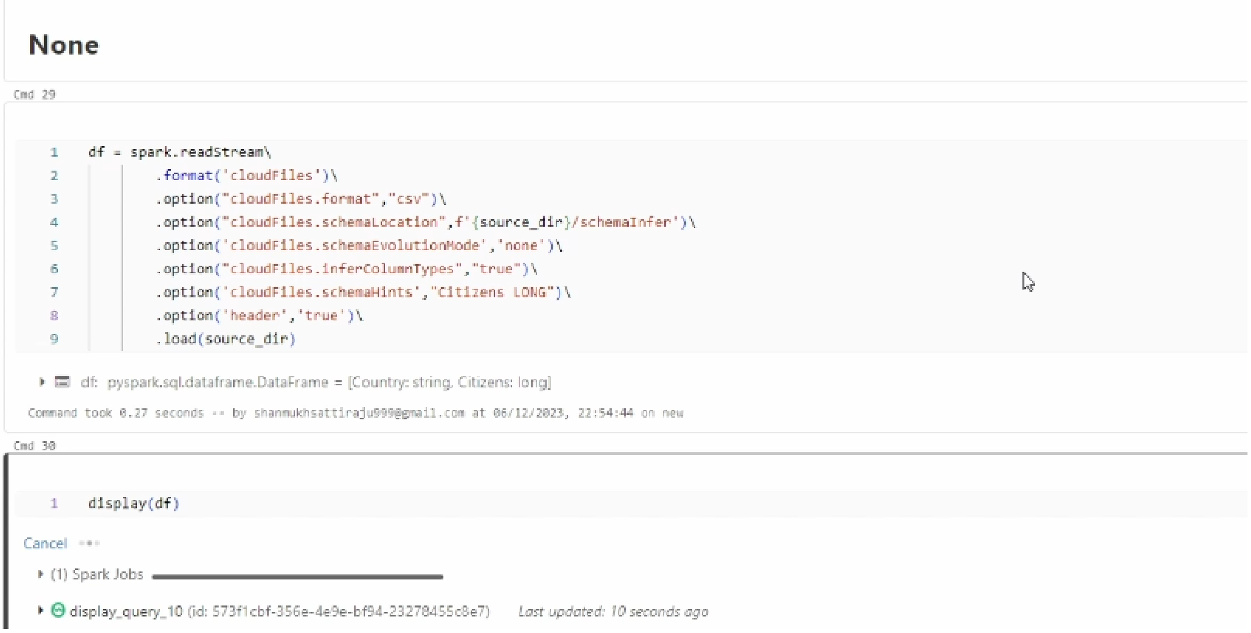
```
df = spark.readStream\
        .format('cloudFiles')\
        .option("cloudFiles.format","csv")\
        .option("cloudFiles.schemaLocation",f'{source_dir}/schemaInfer')\
        .option('cloudFiles.schemaEvolutionMode','failOnNewColumns')\
        .option("cloudFiles.inferColumnTypes","true")\
        .option('cloudFiles.schemaHints',"Citizens LONG")\
        .option('header','true')\
        .load(source_dir)
```

▸ df: pyspark.sql.dataframe.DataFrame = [Country: string, Citizens: long ... 1 more field]

Command took 0.34 seconds -- by shanmukhsattiraju999@gmail.com at 06/12/2023, 22:52:47 on new

Cmd 26

FailOnNewcolumns schema evolution mode, when a new file is added, display will fail, to fix either schema must be redefined or data should be changed.



None

Cmd 29

```
df = spark.readStream\
        .format('cloudFiles')\
        .option("cloudFiles.format","csv")\
        .option("cloudFiles.schemaLocation",f'{source_dir}/schemaInfer')\
        .option('cloudFiles.schemaEvolutionMode','none')\
        .option("cloudFiles.inferColumnTypes","true")\
        .option('cloudFiles.schemaHints',"Citizens LONG")\
        .option('header','true')\
        .load(source_dir)
```

▸ df: pyspark.sql.dataframe.DataFrame = [Country: string, Citizens: long]

Command took 0.27 seconds -- by shanmukhsattiraju999@gmail.com at 06/12/2023, 22:54:44 on new

Cmd 30

```
display(df)
```

▸ display_query_10 (id: 573f1cbf-356e-4e9e-bf94-23278455c8e7) *Last updated: 10 seconds ago*

None type of schema evolution ignores any changes in the schema, add the data to the existing schema.