How to read and write to same Parquet file?

-Lipsa

https://www.linkedin.com/in/lipsa-biswas/

Overwriting Parquet files

⚠ Please note that the approach discussed here to overwrite the parquet file is a workaround, and not an efficient or recommended way to use in production environment or while working with billions of records.

For transactional support and efficient handling of updates, you might consider technologies like *Delta lake* which extends Apache Spark to provide ACID transactions on top of your data lakes. You may want to use the workaround for testing on small files in development environment.

Task – Issue - Workaround

Task - You have a parquet file, you do transformations on it, and now want to write it back to same parquet file. How will you do it?

Issue - You may ask, what is the issue? Use mode-overwrite and overwrite the file. Following code will raise an exception

df_parquet.write.format("parquet").mode("overwrite").save("data/parque_data")

FileNotFoundException: File does not exist: XXXX

As we are selecting mode override , Spark in it's execution plan adding to delete the path first, then trying to read that path which is already vacant. Hence, we get FileNotFoundException

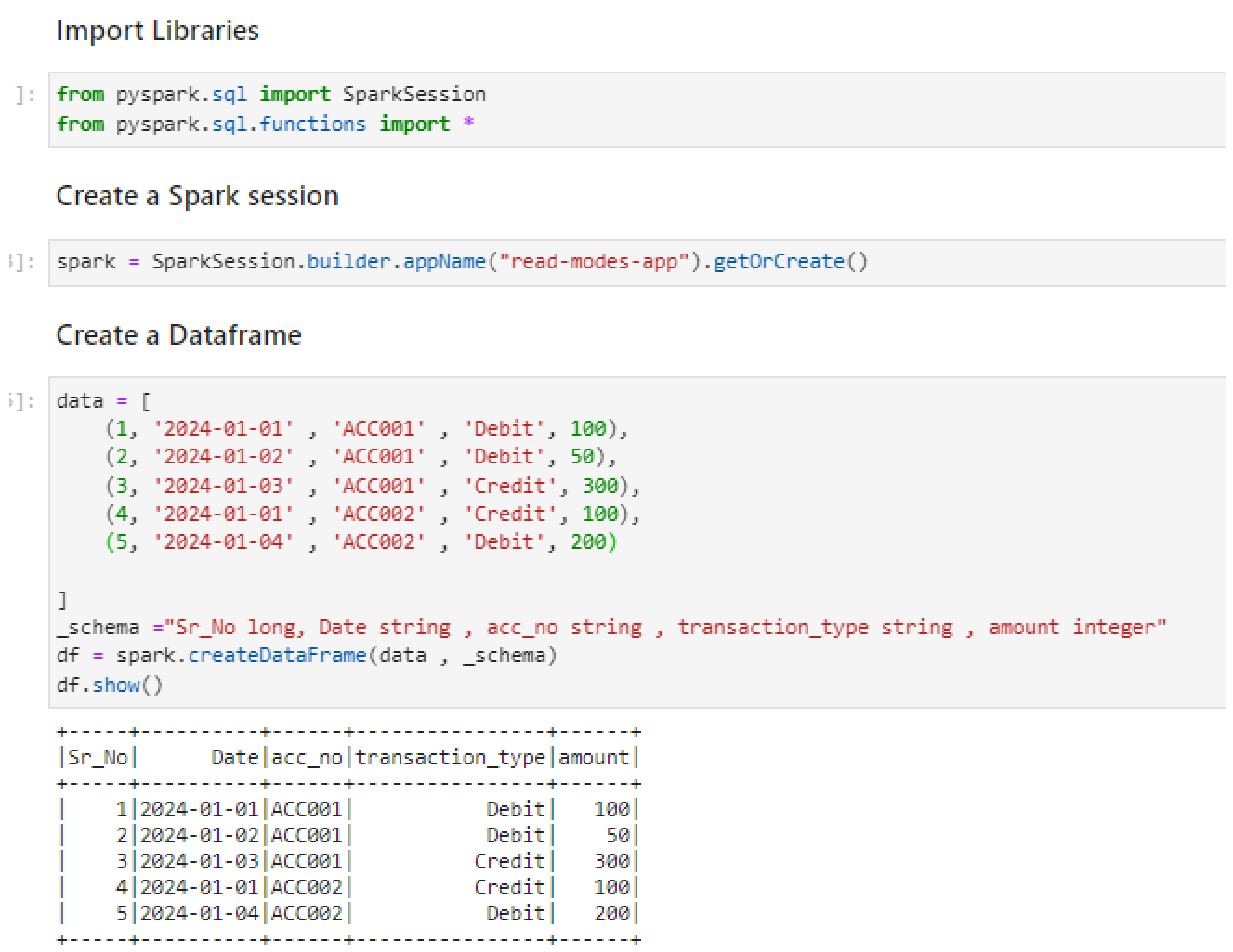
Workaround – We will cache the transformed dataframe before writing it back to same Parquet file

Let’s code

Approach:

1. Create a Spark dataframe , say **df**
2. Save it as a Parquet file , say to a path **data/parquet_data**
3. Read from the parquet file (created in step#2) and save into a dataframe , say **df_parquet**
4. Do required transformation on the dataframe df_parquet
5. Cache the dataframe df_parquet (This is our “fix” code step)
6. Save the cached dataframe to the same parquet file as you did in step#2 , **data/parquet_data**
7. Read from the updated parquet file and verify the records

Code -1



Import Libraries

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import *
```

Create a Spark session

```
spark = SparkSession.builder.appName("read-modes-app").getOrCreate()
```

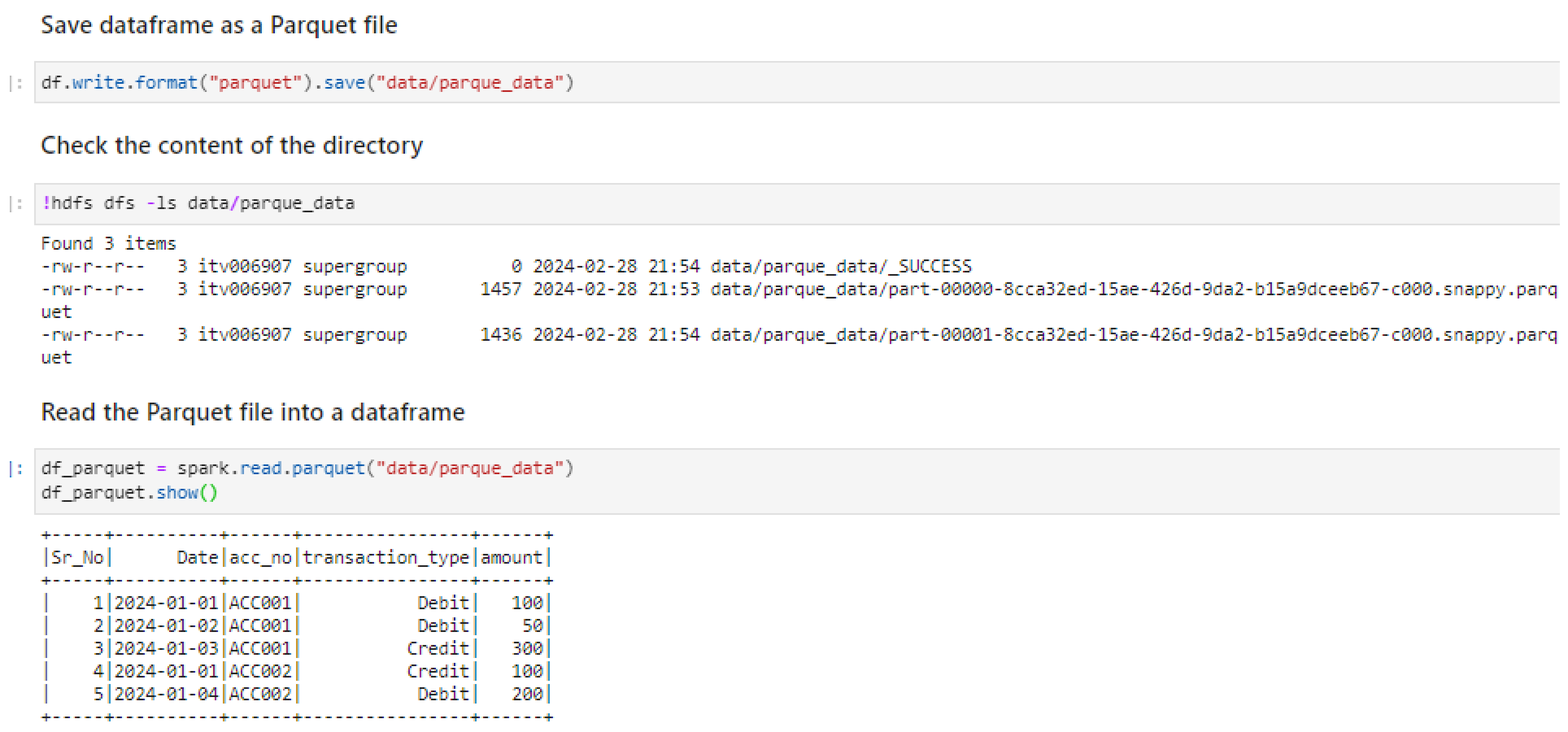
Create a Dataframe

```
data = [
    (1, '2024-01-01' , 'ACC001' , 'Debit', 100),
    (2, '2024-01-02' , 'ACC001' , 'Debit', 50),
    (3, '2024-01-03' , 'ACC001' , 'Credit', 300),
    (4, '2024-01-01' , 'ACC002' , 'Credit', 100),
    (5, '2024-01-04' , 'ACC002' , 'Debit', 200)

]
_schema ="Sr_No long, Date string , acc_no string , transaction_type string , amount integer"
df = spark.createDataFrame(data , _schema)
df.show()
```

```
+-----+----------+------+----------------+------+
|Sr_No|      Date|acc_no|transaction_type|amount|
+-----+----------+------+----------------+------+
|    1|2024-01-01|ACC001|           Debit|   100|
|    2|2024-01-02|ACC001|           Debit|    50|
|    3|2024-01-03|ACC001|          Credit|   300|
|    4|2024-01-01|ACC002|          Credit|   100|
|    5|2024-01-04|ACC002|           Debit|   200|
+-----+----------+------+----------------+------+
```

Code -2



Save dataframe as a Parquet file

```
df.write.format("parquet").save("data/parque_data")
```

Check the content of the directory

```
!hdfs dfs -ls data/parque_data
```

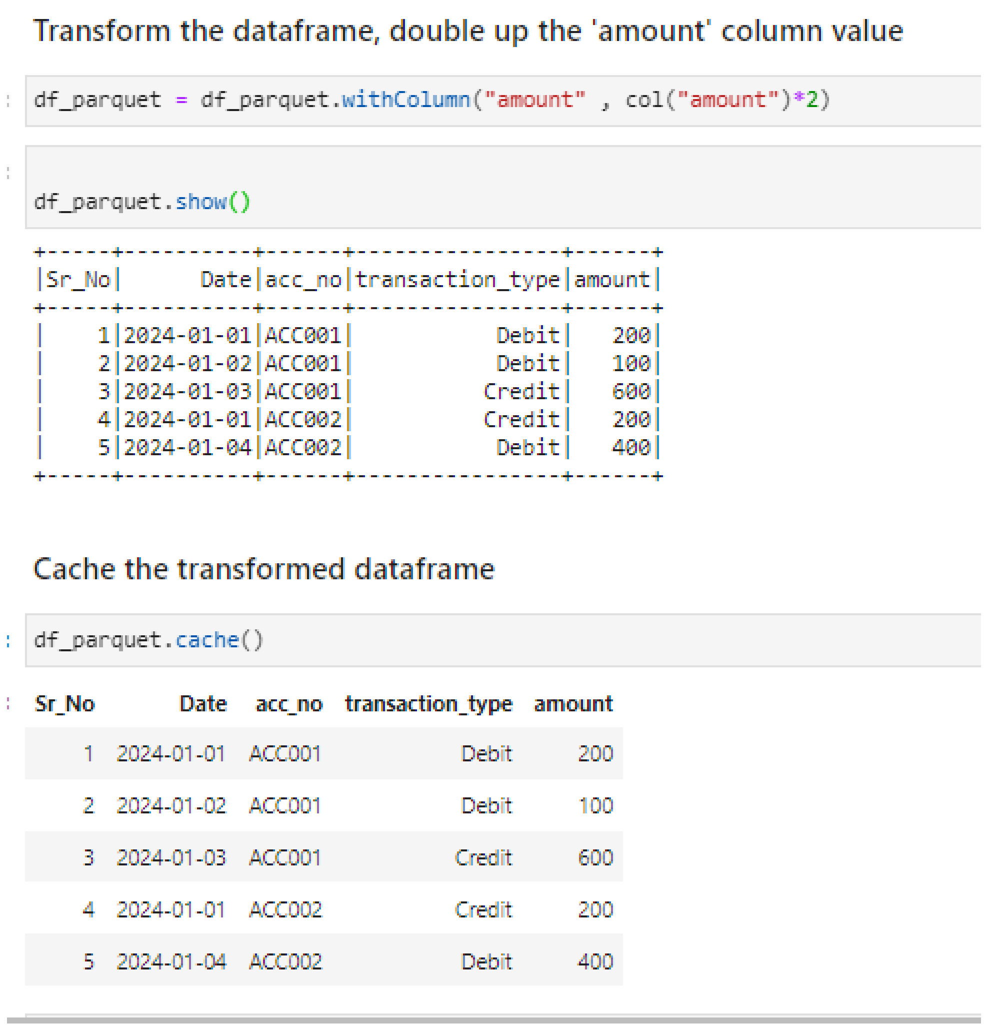
```
Found 3 items
-rw-r--r--   3 itv006907 supergroup          0 2024-02-28 21:54 data/parque_data/_SUCCESS
-rw-r--r--   3 itv006907 supergroup       1457 2024-02-28 21:53 data/parque_data/part-00000-8cca32ed-15ae-426d-9da2-b15a9dceeb67-c000.snappy.parq
uet
-rw-r--r--   3 itv006907 supergroup       1436 2024-02-28 21:54 data/parque_data/part-00001-8cca32ed-15ae-426d-9da2-b15a9dceeb67-c000.snappy.parq
uet
```

Read the Parquet file into a dataframe

```
df_parquet = spark.read.parquet("data/parque_data")
df_parquet.show()
```

```
+-----+----------+------+----------------+------+
|Sr_No|      Date|acc_no|transaction_type|amount|
+-----+----------+------+----------------+------+
|    1|2024-01-01|ACC001|           Debit|   100|
|    2|2024-01-02|ACC001|           Debit|    50|
|    3|2024-01-03|ACC001|          Credit|   300|
|    4|2024-01-01|ACC002|          Credit|   100|
|    5|2024-01-04|ACC002|           Debit|   200|
+-----+----------+------+----------------+------+
```

Code -3



Transform the dataframe, double up the 'amount' column value

```
df_parquet = df_parquet.withColumn("amount" , col("amount")*2)
```

```
df_parquet.show()
```

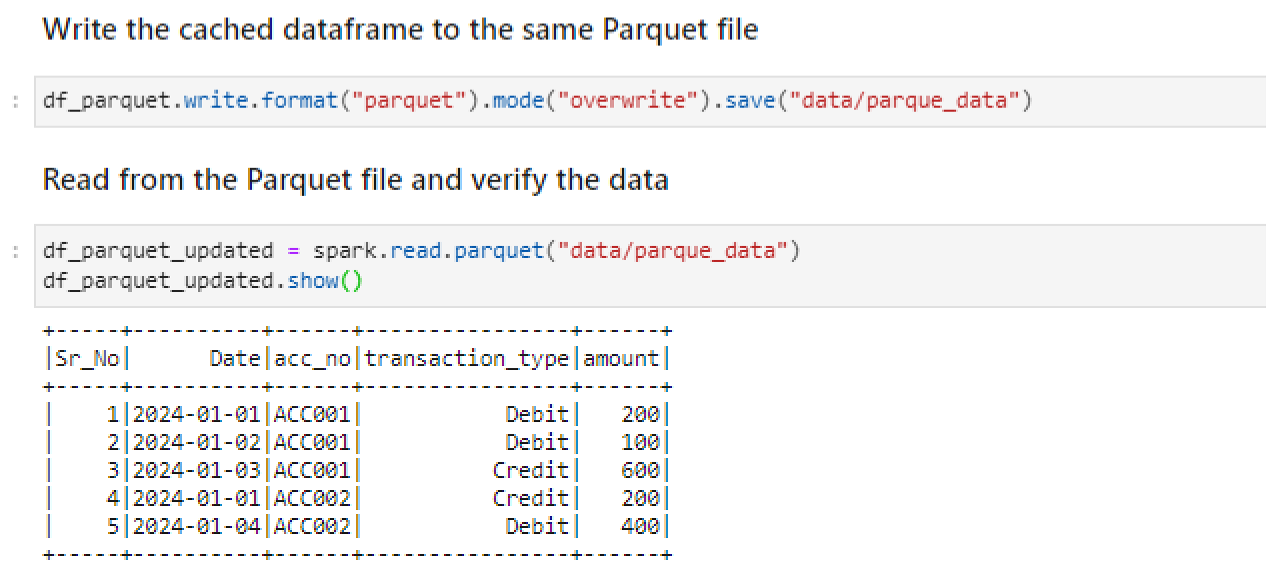
```
+-----+----------+------+----------------+------+
|Sr_No|      Date|acc_no|transaction_type|amount|
+-----+----------+------+----------------+------+
|    1|2024-01-01|ACC001|           Debit|   200|
|    2|2024-01-02|ACC001|           Debit|   100|
|    3|2024-01-03|ACC001|          Credit|   600|
|    4|2024-01-01|ACC002|          Credit|   200|
|    5|2024-01-04|ACC002|           Debit|   400|
+-----+----------+------+----------------+------+
```

Cache the transformed dataframe

```
df_parquet.cache()
```

Sr_No	Date	acc_no	transaction_type	amount
1	2024-01-01	ACC001	Debit	200
2	2024-01-02	ACC001	Debit	100
3	2024-01-03	ACC001	Credit	600
4	2024-01-01	ACC002	Credit	200
5	2024-01-04	ACC002	Debit	400

Code -4



Write the cached dataframe to the same Parquet file

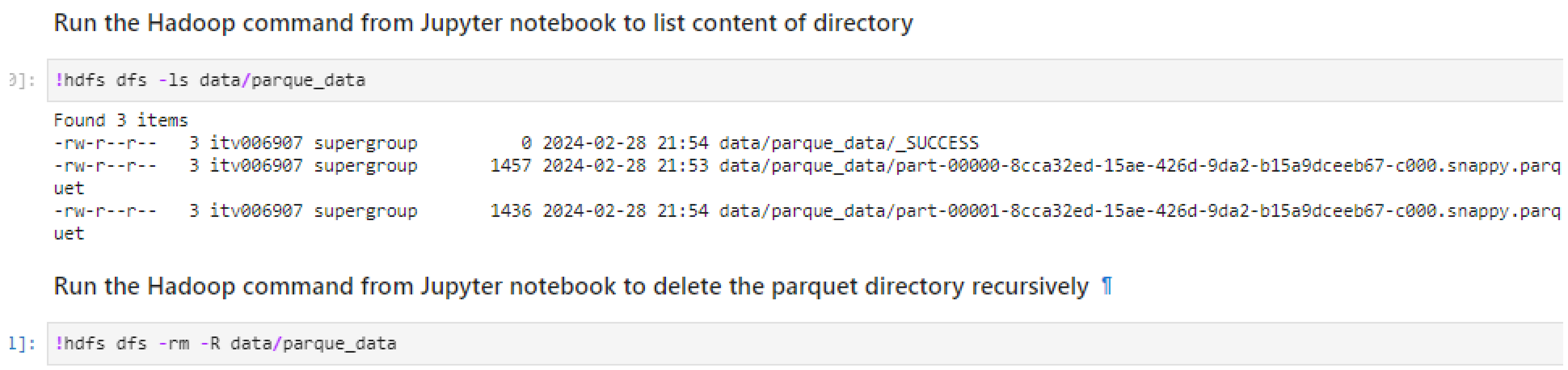
```
df_parquet.write.format("parquet").mode("overwrite").save("data/parque_data")
```

Read from the Parquet file and verify the data

```
df_parquet_updated = spark.read.parquet("data/parque_data")
df_parquet_updated.show()
```

```
+-----+----------+------+----------------+------+
|Sr_No|      Date|acc_no|transaction_type|amount|
+-----+----------+------+----------------+------+
|    1|2024-01-01|ACC001|           Debit|   200|
|    2|2024-01-02|ACC001|           Debit|   100|
|    3|2024-01-03|ACC001|          Credit|   600|
|    4|2024-01-01|ACC002|          Credit|   200|
|    5|2024-01-04|ACC002|           Debit|   400|
+-----+----------+------+----------------+------+
```

Code -5



Run the Hadoop command from Jupyter notebook to list content of directory

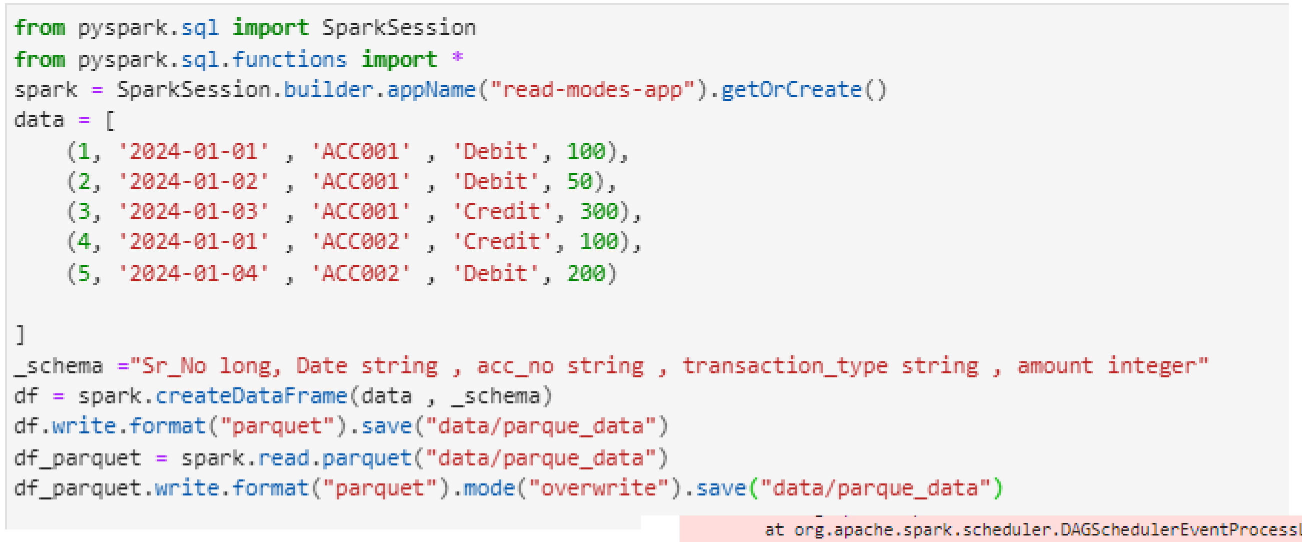
```
0]: !hdfs dfs -ls data/parque_data
```

```
Found 3 items
-rw-r--r--   3 itv006907 supergroup          0 2024-02-28 21:54 data/parque_data/_SUCCESS
-rw-r--r--   3 itv006907 supergroup       1457 2024-02-28 21:53 data/parque_data/part-00000-8cca32ed-15ae-426d-9da2-b15a9dceeb67-c000.snappy.parq
uet
-rw-r--r--   3 itv006907 supergroup       1436 2024-02-28 21:54 data/parque_data/part-00001-8cca32ed-15ae-426d-9da2-b15a9dceeb67-c000.snappy.parq
uet
```

Run the Hadoop command from Jupyter notebook to delete the parquet directory recursively ¶

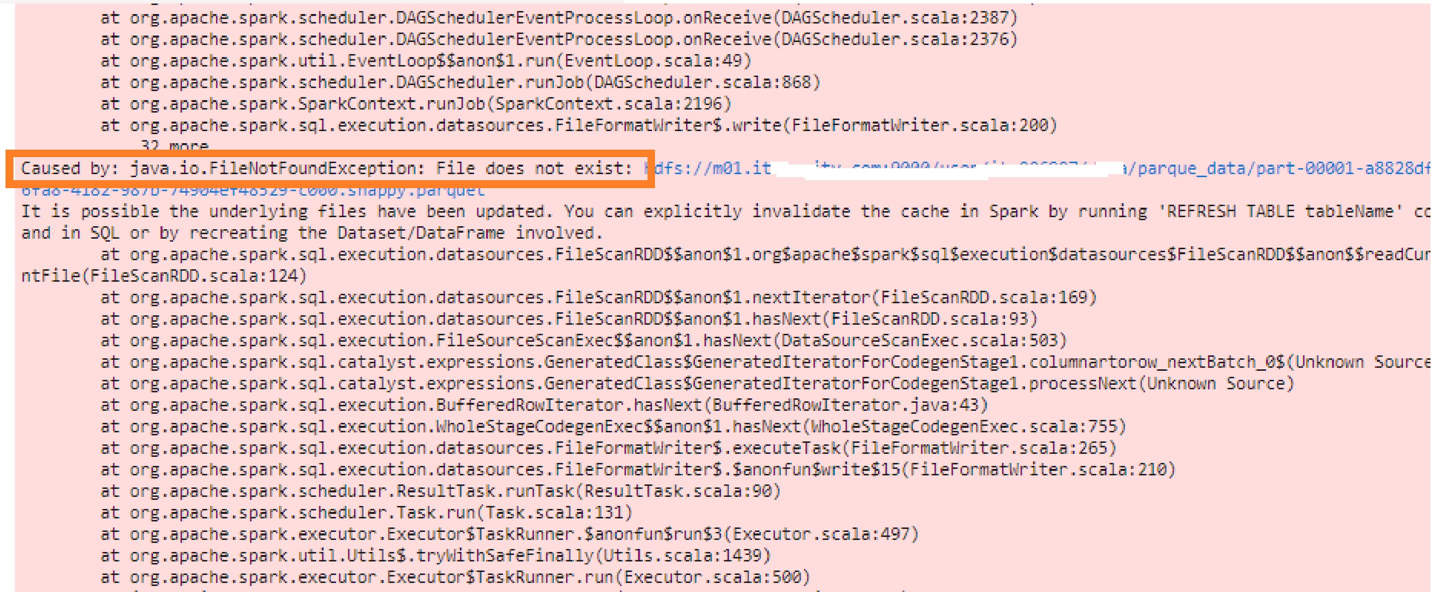
```
1]: !hdfs dfs -rm -R data/parque_data
```

Issue w/o caching the dataframe



```
from pyspark.sql import SparkSession
from pyspark.sql.functions import *
spark = SparkSession.builder.appName("read-modes-app").getOrCreate()
data = [
    (1, '2024-01-01' , 'ACC001' , 'Debit', 100),
    (2, '2024-01-02' , 'ACC001' , 'Debit', 50),
    (3, '2024-01-03' , 'ACC001' , 'Credit', 300),
    (4, '2024-01-01' , 'ACC002' , 'Credit', 100),
    (5, '2024-01-04' , 'ACC002' , 'Debit', 200)

]
_schema ="Sr_No long, Date string , acc_no string , transaction_type string , amount integer"
df = spark.createDataFrame(data , _schema)
df.write.format("parquet").save("data/parque_data")
df_parquet = spark.read.parquet("data/parque_data")
df_parquet.write.format("parquet").mode("overwrite").save("data/parque_data")
```



```
        at org.apache.spark.scheduler.DAGSchedulerEventProcessLoop.onReceive(DAGScheduler.scala:2387)
        at org.apache.spark.scheduler.DAGSchedulerEventProcessLoop.onReceive(DAGScheduler.scala:2376)
        at org.apache.spark.util.EventLoop$$anon$1.run(EventLoop.scala:49)
        at org.apache.spark.scheduler.DAGScheduler.runJob(DAGScheduler.scala:868)
        at org.apache.spark.SparkContext.runJob(SparkContext.scala:2196)
        at org.apache.spark.sql.execution.datasources.FileFormatWriter$.write(FileFormatWriter.scala:200)
            ... 32 more
Caused by: java.io.FileNotFoundException: File does not exist: hdfs://m01.it[redacted]a/parque_data/part-00001-a8828df
6fa8-4182-9870-74904ef48529-c000.snappy.parquet
It is possible the underlying files have been updated. You can explicitly invalidate the cache in Spark by running 'REFRESH TABLE tableName' co
and in SQL or by recreating the Dataset/DataFrame involved.
        at org.apache.spark.sql.execution.datasources.FileScanRDD$$anon$1.org$apache$spark$sql$execution$datasources$FileScanRDD$$anon$$readCur
ntFile(FileScanRDD.scala:124)
        at org.apache.spark.sql.execution.datasources.FileScanRDD$$anon$1.nextIterator(FileScanRDD.scala:169)
        at org.apache.spark.sql.execution.datasources.FileScanRDD$$anon$1.hasNext(FileScanRDD.scala:93)
        at org.apache.spark.sql.execution.FileSourceScanExec$$anon$1.hasNext(DataSourceScanExec.scala:503)
        at org.apache.spark.sql.catalyst.expressions.GeneratedClass$GeneratedIteratorForCodegenStage1.columnartorow_nextBatch_0$(Unknown Source
        at org.apache.spark.sql.catalyst.expressions.GeneratedClass$GeneratedIteratorForCodegenStage1.processNext(Unknown Source)
        at org.apache.spark.sql.execution.BufferedRowIterator.hasNext(BufferedRowIterator.java:43)
        at org.apache.spark.sql.execution.WholeStageCodegenExec$$anon$1.hasNext(WholeStageCodegenExec.scala:755)
        at org.apache.spark.sql.execution.datasources.FileFormatWriter$.executeTask(FileFormatWriter.scala:265)
        at org.apache.spark.sql.execution.datasources.FileFormatWriter$.$anonfun$write$15(FileFormatWriter.scala:210)
        at org.apache.spark.scheduler.ResultTask.runTask(ResultTask.scala:90)
        at org.apache.spark.scheduler.Task.run(Task.scala:131)
        at org.apache.spark.executor.Executor$TaskRunner.$anonfun$run$3(Executor.scala:497)
        at org.apache.spark.util.Utils$.tryWithSafeFinally(Utils.scala:1439)
        at org.apache.spark.executor.Executor$TaskRunner.run(Executor.scala:500)
```