PySpark

1. What is PySpark ?

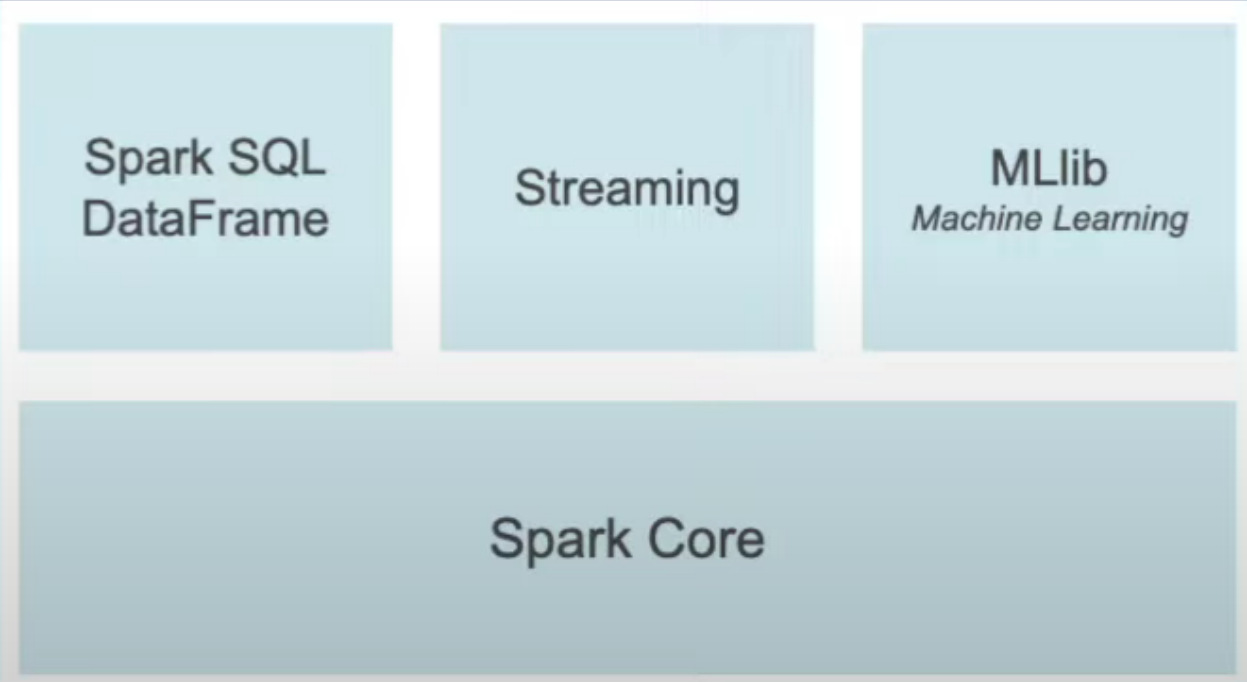
PySpark

PySpark is an interface for Apache Spark in python. It not only allows you to write Spark application using Python APIs, but also provides the PySpark shell for interactively analyzing your data in a distributed manner environment. Pyspark supports most of Spark's features such as Spark SQL, Dataframe , Streaming , MLlib (Machine Learning) and Spark core.

PySpark is nothing but a kind of python libraries.

Using R , SQL or C# you can also intercat with PySpark.

Using pyspark libraries you can modify the dataframe table.



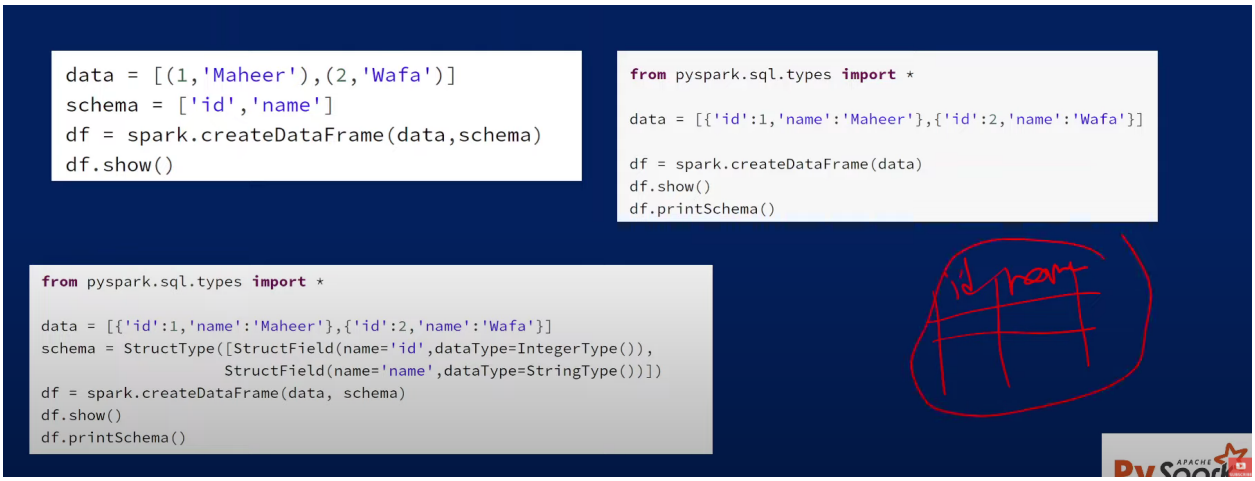
- Apache Spark is used for big data.
- It is built up in Scala
- Dataframe is reading the data and storing into one kind of table.

2. Creating DataFrame manually with hard coded values in PySpark

- Create a Dataframe manually with hard coded values in PySpark.
- We will be using createDataframe() method from Spark session object.

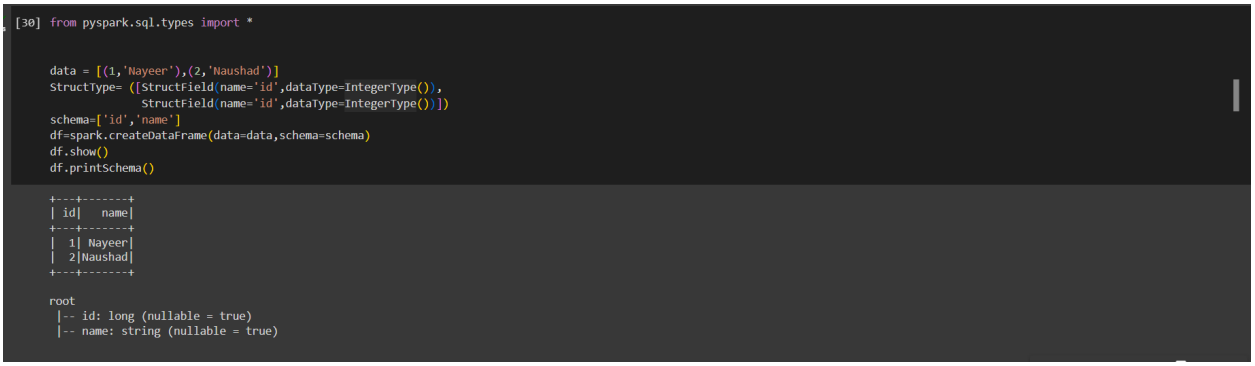
createDataframe()

DataFrame is a distributed collection of data organized into named columns. It is conceptually equivalent to a table in a relational database.



Below type is of long type

But here we converting it using schema type is of integer types



Using dictionaries

```
[31] data = [{'id':1,'name':'Nayeer'},
             {'id':2,'name':'Naushad'}]

     df= spark.createDataFrame(data=data)
     df.show()

     +---+-------+
     | id|   name|
     +---+-------+
     |  1| Nayeer|
     |  2|Naushad|
     +---+-------+
```

```
     data = [{'id':1,'name':'Nayeer'},
             {'id':2,'name':'Naushad'}]

     df= spark.createDataFrame(data=data,schema=schema)
     df.show()
     df.printSchema()

     +---+-------+
     | id|   name|
     +---+-------+
     |  1| Nayeer|
     |  2|Naushad|
     +---+-------+

     root
      |-- id: long (nullable = true)
      |-- name: string (nullable = true)
```

3. Read CSV file in to Dataframe using PySpark

We will see How to read single csv file or multiple csv files or all csv files in to Dataframe using Pyspark.

Using csv("Path") or format("csv").load("path") of DataFrameReader, you can read a csv file into pyspark dataFrame.

→ Reading Single CSV file

```
[18] ##Read CSV files
     df = spark.read.csv("dataset_Loan.csv")
     display(df)
     df.printSchema()

     DataFrame[_c0: string, _c1: string, _c2: string, _c3: string, _c4: string, _c5: string, _c6: string, _c7: string, _c8: string, _c9: string, _c10: string, _c11: string, _c12: string]
     root
      |-- _c0: string (nullable = true)
      |-- _c1: string (nullable = true)
      |-- _c2: string (nullable = true)
      |-- _c3: string (nullable = true)
      |-- _c4: string (nullable = true)
      |-- _c5: string (nullable = true)
      |-- _c6: string (nullable = true)
      |-- _c7: string (nullable = true)
      |-- _c8: string (nullable = true)
      |-- _c9: string (nullable = true)
      |-- _c10: string (nullable = true)
      |-- _c11: string (nullable = true)
      |-- _c12: string (nullable = true)
```

→ Reading Multiple CSV files

```
[27] ##multiple csv file
     df = spark.read.csv(['dataset_Loan.csv','spam.csv'],header=True)
     display(df)
     df.printSchema()

     DataFrame[v1: string, v2: string, _c2: string, _c3: string, _c4: string]
     root
      |-- v1: string (nullable = true)
      |-- v2: string (nullable = true)
      |-- _c2: string (nullable = true)
      |-- _c3: string (nullable = true)
      |-- _c4: string (nullable = true)
```

→ Reading all the CSV Files

```
[43] ##reading All the csv file at once
     df = spark.read.csv(path='/content/CSV',header=True)
     display(df)
```

Schema →

```
1  from pyspark.sql.types import *
2
3  schema = StructType().add(field='id',data_type=IntegerType())\
4                       .add(field='name',data_type=StringType())\
5                       .add(field='gender',data_type=StringType())\
6                       .add(field='salary',data_type=IntegerType())
7
8  df = spark.read.csv(path='dbfs:/FileStore/data/', schema=schema, header=True)
9  display(df)
10 df.printSchema()
```

```
Showing all 4 rows. | 0.41 seconds runtime

root
 |-- id: integer (nullable = true)
 |-- name: string (nullable = true)
 |-- gender: string (nullable = true)
 |-- salary: integer (nullable = true)
```

4. Write DataFrame into CSV file using PySpark

- How to read write dataframe into CSV file using PySpark
- Options available while saving to CSV file
- Savings modes

Write DataFrame into CSV

Use the object() method of the PySpark DataFrameWriter object to write PySpark DataFrame to a CSV file.

```
[24] df.write.option("header", True).csv("/content/gth")
```

While writing a CSV file you can use several options. for example, header to output the DataFrame column names as header record and delimiter to specify the delimiter on the CSV output File.

```
[26] df.write.options(header="True",delimiter=',').csv("/content/gas")
```

For reading

```
df2=spark.read.csv(path="/content/gth",header=True)
df2.show()
```

```
+---+-------+
| id|   name|
+---+-------+
|  2|Naushad|
|  1| Nayeer|
+---+-------+
```

- there are various mode you can select one of that is ignore mode

```
[37] df.write.csv(path='/content/gg',header=True,mode='ignore')
```

- another is error ie if the directory is not found then there will an error will occur

```
df.write.csv(path='/content/gg',header=True,mode='error')
```

```
---------------------------------------------------------------------------
AnalysisException                         Traceback (most recent call last)
<ipython-input-38-37c49ae6f98d> in <cell line: 1>()
----> 1 df.write.csv(path='/content/gg',header=True,mode='error')

                                    2 frames
/usr/local/lib/python3.10/dist-packages/pyspark/errors/exceptions/captured.py in deco(*a, **kw)
    183                 # Hide where the exception came from that shows a non-Pythonic
    184                 # JVM exception message.
--> 185                 raise converted from None
    186             else:
    187                 raise

AnalysisException: [PATH_ALREADY_EXISTS] Path file:/content/gg already exists. Set mode as "overwrite" to overwrite the existing path.
```

Next steps: Explain error

- overwrite

```
[42] #overwrite
     df.write.csv(path="/content/gth",header=True,mode='overwrite')
```

```
[43] df2=spark.read.csv(path="/content/gth",header=True)
     df2.show()
```

```
+---+-------+
| id|   name|
+---+-------+
|  2|Naushad|
|  1| Nayeer|
+---+-------+
```

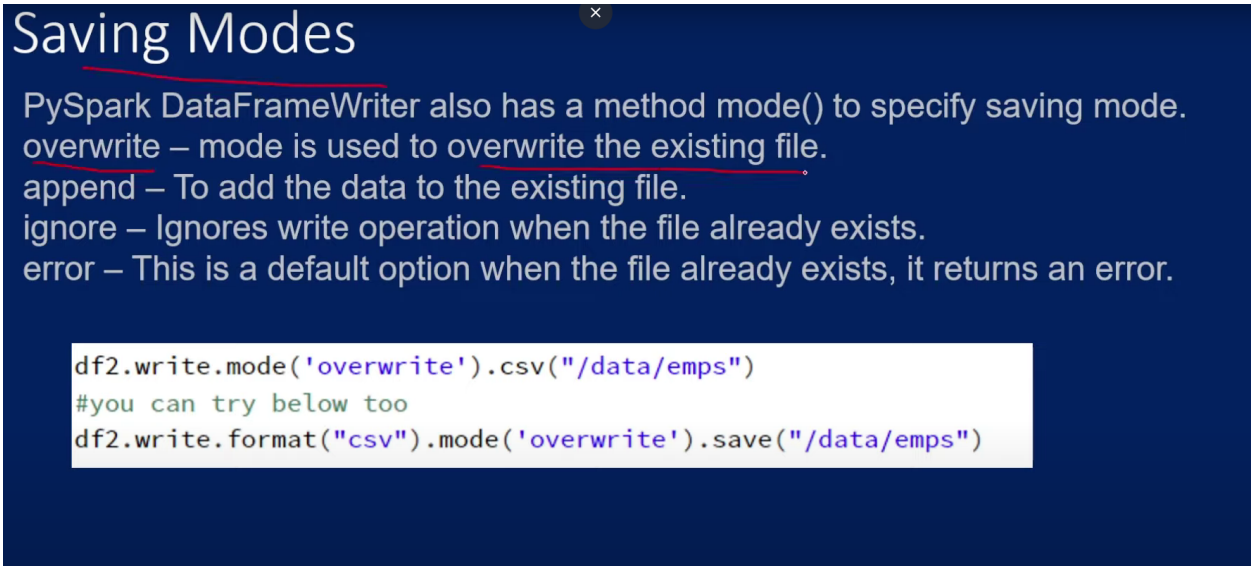
- Append

```
#append
df.write.csv(path="/content/gth",header=True,mode='append')
```

```
[45] df=spark.read.csv(path="/content/gth",header=True)
     df.show()
```

```
+---+-------+
| id|   name|
+---+-------+
|  2|Naushad|
|  2|Naushad|
|  1| Nayeer|
|  1| Nayeer|
+---+-------+
```

Savings Mode



5. Read json file in to Dataframe using PySpark

Agenda

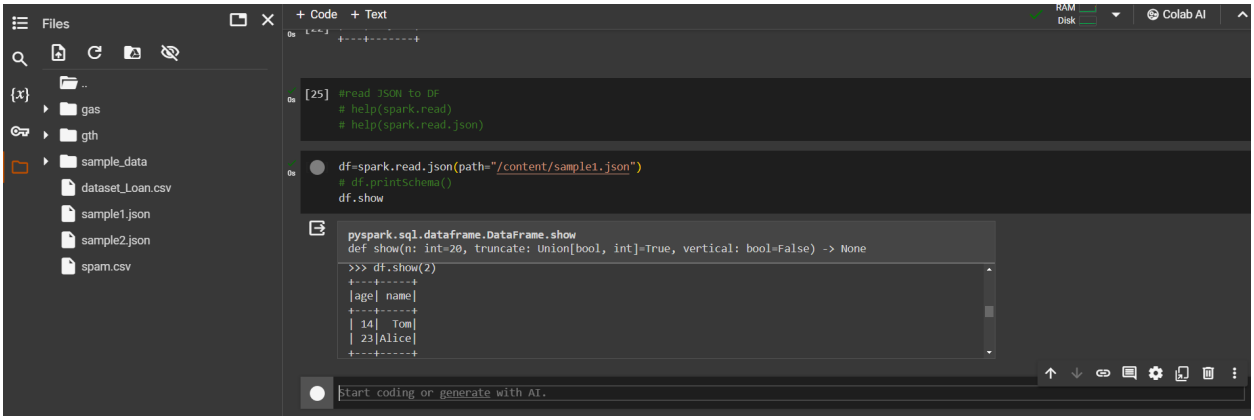
- How to read single json file or multiple json files or all json files in to DataFrame using Pyspark
- How to read single line and multi line json files into DataFrame using PySpark

Read JSON data into DataFrame

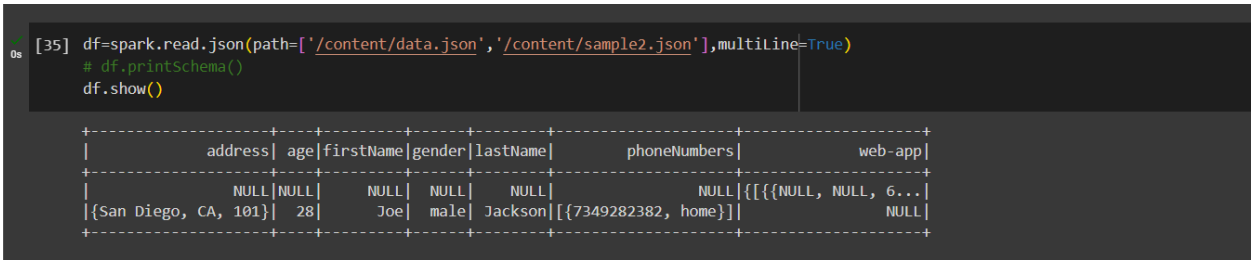
Using read.json("path") or read.format("json").load("path") you can read a JSON file into a PySpark DataFrame

Use MultiLine option to read JSON files scattered across multiple lines. By default multiline option , is set to false.

- Single Line



- MultiLine



```
[32] df=spark.read.json(path="/content/data.json",multiLine=True)
     df.printSchema()
     df.show()

     root
      |-- web-app: struct (nullable = true)
      |    |-- servlet: array (nullable = true)
      |    |    |-- element: struct (containsNull = true)
      |    |    |    |-- init-param: struct (nullable = true)
      |    |    |    |    |-- adminGroupID: long (nullable = true)
      |    |    |    |    |-- betaServer: boolean (nullable = true)
      |    |    |    |    |-- cachePackageTagsRefresh: long (nullable = true)
      |    |    |    |    |-- cachePackageTagsStore: long (nullable = true)
      |    |    |    |    |-- cachePackageTagsTrack: long (nullable = true)
      |    |    |    |    |-- cachePagesDirtyRead: long (nullable = true)
      |    |    |    |    |-- cachePagesRefresh: long (nullable = true)
      |    |    |    |    |-- cachePagesStore: long (nullable = true)
      |    |    |    |    |-- cachePagesTrack: long (nullable = true)
      |    |    |    |    |-- cacheTemplatesRefresh: long (nullable = true)
      |    |    |    |    |-- cacheTemplatesStore: long (nullable = true)
      |    |    |    |    |-- cacheTemplatesTrack: long (nullable = true)
      |    |    |    |    |-- configGlossary:adminEmail: string (nullable = true)
      |    |    |    |    |-- configGlossary:installationAt: string (nullable = true)
      |    |    |    |    |-- configGlossary:poweredBy: string (nullable = true)
      |    |    |    |    |-- configGlossary:poweredByIcon: string (nullable = true)
      |    |    |    |    |-- configGlossary:staticPath: string (nullable = true)
      |    |    |    |    |-- dataLog: long (nullable = true)
      |    |    |    |    |-- dataLogLocation: string (nullable = true)
```

- Read all the JSON files

```
[37] df=spark.read.json(path='/content/*.json')
     df.show()

+--------------------+------+----+
|     _corrupt_record|number|type|
+--------------------+------+----+
|                   {|  NULL|NULL|
|        "web-app": {|  NULL|NULL|
|        "servlet": [|  NULL|NULL|
|                   {|  NULL|NULL|
|        "servlet-...|  NULL|NULL|
|        "servlet-...|  NULL|NULL|
|        "init-par...|  NULL|NULL|
|          "config...|  NULL|NULL|
|          "config...|  NULL|NULL|
|          "config...|  NULL|NULL|
|          "config...|  NULL|NULL|
|          "config...|  NULL|NULL|
|          "templa...|  NULL|NULL|
|          "templa...|  NULL|NULL|
|          "templa...|  NULL|NULL|
|          "templa...|  NULL|NULL|
|          "defaul...|  NULL|NULL|
|          "defaul...|  NULL|NULL|
|          "useJSP...|  NULL|NULL|
|          "jspLis...|  NULL|NULL|
+--------------------+------+----+
```

- Changing all the data type to integer types

6. Write Dataframe into json file using PySpark

Agenda

- How to read write DataFrame into JSON file using PySpark
- Saving Modes

Write DataFrama into json

Use the DataFrameWriter object to write PySpark Dataframe to a JSON file.

```
[47] df=spark.read.json('/content/neww.json')
     df.show

pyspark.sql.dataframe.DataFrame.show
def show(n: int=20, truncate: Union[bool, int]=True, vertical: bool=False) -> None
+---+----+
| 14| Tom|
| 23| Ali|
| 16| Bob|
+---+----+

Show :class:`DataFrame` verticallv.
```

```
df.write.json('/content/neww.json',mode='ignore')

df.write.json('/content/neww.json',mode='append')
df.write.json('/content/neww.json',mode='overwrite')
```

7. Read Parquet file into DataFrame using Pyspark

→ Agenda

- How to read single parquet file or all parquet files in to DataFrame files in to DataFrame using PySpark

```
[61] #Read Parquet file into DataFrame using Pyspark
     from pyspark.sql.types import StructType, StructField, StringType, IntegerType

[63] pip install pyarrow

     Requirement already satisfied: pyarrow in /usr/local/lib/python3.10/dist-packages (14.0.2)
     Requirement already satisfied: numpy>=1.16.6 in /usr/local/lib/python3.10/dist-packages (from pyarrow) (1.25.2)

     df= spark.read.parquet('/content/MT cars.parquet')
     df.show()
     print(df.count())

     +------------------+----+---+-----+---+----+-----+----+---+----+----+
     |             model| mpg|cyl| disp| hp|drat|   wt| qsec| vs| am|gear|carb|
     +------------------+----+---+-----+---+----+-----+----+---+----+----+
     |         Mazda RX4|21.0|  6|160.0|110| 3.9| 2.62|16.46|  0|  1|   4|   4|
     |     Mazda RX4 Wag|21.0|  6|160.0|110| 3.9|2.875|17.02|  0|  1|   4|   4|
     |        Datsun 710|22.8|  4|108.0| 93|3.85| 2.32|18.61|  1|  1|   4|   1|
     |    Hornet 4 Drive|21.4|  6|258.0|110|3.08|3.215|19.44|  1|  0|   3|   1|
     | Hornet Sportabout|18.7|  8|360.0|175|3.15| 3.44|17.02|  0|  0|   3|   2|
     |           Valiant|18.1|  6|225.0|105|2.76| 3.46|20.22|  1|  0|   3|   1|
     |         Duster 360|14.3|  8|360.0|245|3.21| 3.57|15.84|  0|  0|   3|   4|
     |          Merc 240D|24.4|  4|146.7| 62|3.69| 3.19| 20.0|  1|  0|   4|   2|
     |           Merc 230|22.8|  4|140.8| 95|3.92| 3.15| 22.9|  1|  0|   4|   2|
     |           Merc 280|19.2|  6|167.6|123|3.92| 3.44| 18.3|  1|  0|   4|   4|
     |          Merc 280C|17.8|  6|167.6|123|3.92| 3.44| 18.9|  1|  0|   4|   4|
     |         Merc 450SE|16.4|  8|275.8|180|3.07| 4.07| 17.4|  0|  0|   3|   3|
     |         Merc 450SL|17.3|  8|275.8|180|3.07| 3.73| 17.6|  0|  0|   3|   3|
     |        Merc 450SLC|15.2|  8|275.8|180|3.07| 3.78| 18.0|  0|  0|   3|   3|
     | Cadillac Fleetwood|10.4|  8|472.0|205|2.93| 5.25|17.98|  0|  0|   3|   4|
     |Lincoln Continental|10.4|  8|460.0|215| 3.0|5.424|17.82|  0|  0|   3|   4|
     |  Chrysler Imperial|14.7|  8|440.0|230|3.23|5.345|17.42|  0|  0|   3|   4|
     |           Fiat 128|32.4|  4| 78.7| 66|4.08|  2.2|19.47|  1|  1|   4|   1|
     |        Honda Civic|30.4|  4| 75.7| 52|4.93|1.615|18.52|  1|  1|   4|   2|
     |     Toyota Corolla|33.9|  4| 71.1| 65|4.22|1.835| 19.9|  1|  1|   4|   1|
     +------------------+----+---+-----+---+----+-----+----+---+----+----+
     only showing top 20 rows
```

→ read single parquet file

```
#read single parquet file
df= spark.read.parquet('/content/MT cars.parquet')
df.show()
print(df.count())
```

```
+------------------+----+---+-----+---+----+-----+-----+---+---+----+----+
|             model| mpg|cyl| disp| hp|drat|   wt| qsec| vs| am|gear|carb|
+------------------+----+---+-----+---+----+-----+-----+---+---+----+----+
|         Mazda RX4|21.0|  6|160.0|110| 3.9| 2.62|16.46|  0|  1|   4|   4|
|     Mazda RX4 Wag|21.0|  6|160.0|110| 3.9|2.875|17.02|  0|  1|   4|   4|
|        Datsun 710|22.8|  4|108.0| 93|3.85| 2.32|18.61|  1|  1|   4|   1|
|    Hornet 4 Drive|21.4|  6|258.0|110|3.08|3.215|19.44|  1|  0|   3|   1|
| Hornet Sportabout|18.7|  8|360.0|175|3.15| 3.44|17.02|  0|  0|   3|   2|
|           Valiant|18.1|  6|225.0|105|2.76| 3.46|20.22|  1|  0|   3|   1|
|        Duster 360|14.3|  8|360.0|245|3.21| 3.57|15.84|  0|  0|   3|   4|
|         Merc 240D|24.4|  4|146.7| 62|3.69| 3.19| 20.0|  1|  0|   4|   2|
|          Merc 230|22.8|  4|140.8| 95|3.92| 3.15| 22.9|  1|  0|   4|   2|
|          Merc 280|19.2|  6|167.6|123|3.92| 3.44| 18.3|  1|  0|   4|   4|
|         Merc 280C|17.8|  6|167.6|123|3.92| 3.44| 18.9|  1|  0|   4|   4|
|        Merc 450SE|16.4|  8|275.8|180|3.07| 4.07| 17.4|  0|  0|   3|   3|
|        Merc 450SL|17.3|  8|275.8|180|3.07| 3.73| 17.6|  0|  0|   3|   3|
|       Merc 450SLC|15.2|  8|275.8|180|3.07| 3.78| 18.0|  0|  0|   3|   3|
| Cadillac Fleetwood|10.4|  8|472.0|205|2.93| 5.25|17.98|  0|  0|   3|   4|
|Lincoln Continental|10.4|  8|460.0|215| 3.0|5.424|17.82|  0|  0|   3|   4|
|  Chrysler Imperial|14.7|  8|440.0|230|3.23|5.345|17.42|  0|  0|   3|   4|
|          Fiat 128|32.4|  4| 78.7| 66|4.08|  2.2|19.47|  1|  1|   4|   1|
|       Honda Civic|30.4|  4| 75.7| 52|4.93|1.615|18.52|  1|  1|   4|   2|
|    Toyota Corolla|33.9|  4| 71.1| 65|4.22|1.835| 19.9|  1|  1|   4|   1|
+------------------+----+---+-----+---+----+-----+-----+---+---+----+----+
only showing top 20 rows

32
```

→ read all parquet files

```
#read all the parquet file
df= spark.read.parquet('/content/*.parquet')
df.show()
print(df.count())
```

```
+------------+-----------+------------+-----------+-------+
|sepal.length|sepal.width|petal.length|petal.width|variety|
+------------+-----------+------------+-----------+-------+
|        NULL|       NULL|        NULL|       NULL|   NULL|
|        NULL|       NULL|        NULL|       NULL|   NULL|
|        NULL|       NULL|        NULL|       NULL|   NULL|
|        NULL|       NULL|        NULL|       NULL|   NULL|
|        NULL|       NULL|        NULL|       NULL|   NULL|
|        NULL|       NULL|        NULL|       NULL|   NULL|
|        NULL|       NULL|        NULL|       NULL|   NULL|
|        NULL|       NULL|        NULL|       NULL|   NULL|
|        NULL|       NULL|        NULL|       NULL|   NULL|
|        NULL|       NULL|        NULL|       NULL|   NULL|
|        NULL|       NULL|        NULL|       NULL|   NULL|
|        NULL|       NULL|        NULL|       NULL|   NULL|
|        NULL|       NULL|        NULL|       NULL|   NULL|
|        NULL|       NULL|        NULL|       NULL|   NULL|
|        NULL|       NULL|        NULL|       NULL|   NULL|
|        NULL|       NULL|        NULL|       NULL|   NULL|
|        NULL|       NULL|        NULL|       NULL|   NULL|
|        NULL|       NULL|        NULL|       NULL|   NULL|
|        NULL|       NULL|        NULL|       NULL|   NULL|
|        NULL|       NULL|        NULL|       NULL|   NULL|
+------------+-----------+------------+-----------+-------+
only showing top 20 rows

182
```

✓ 0s completed at 12:22 PM

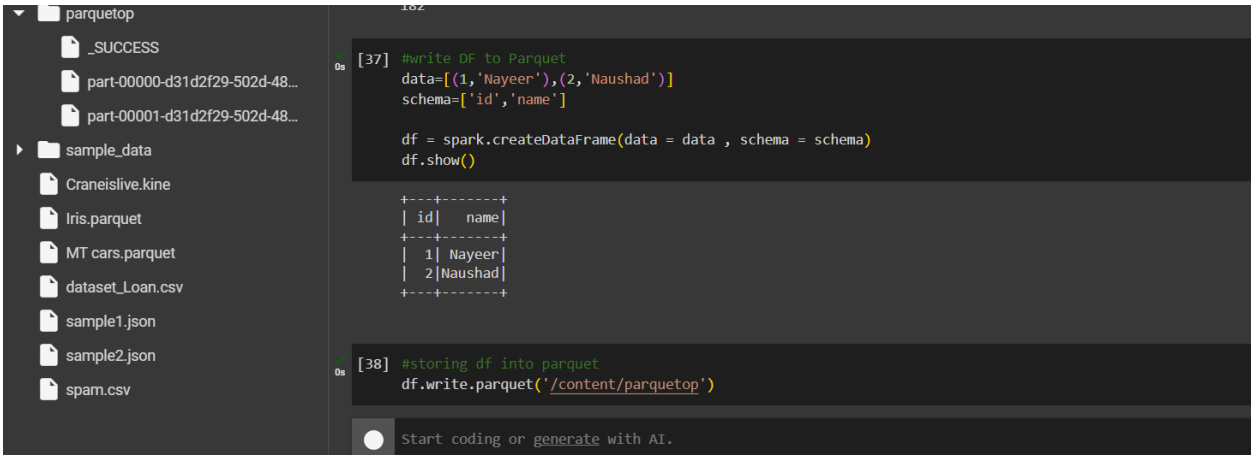
8. Write Dataframe into parquent file using PySpark

Agenda

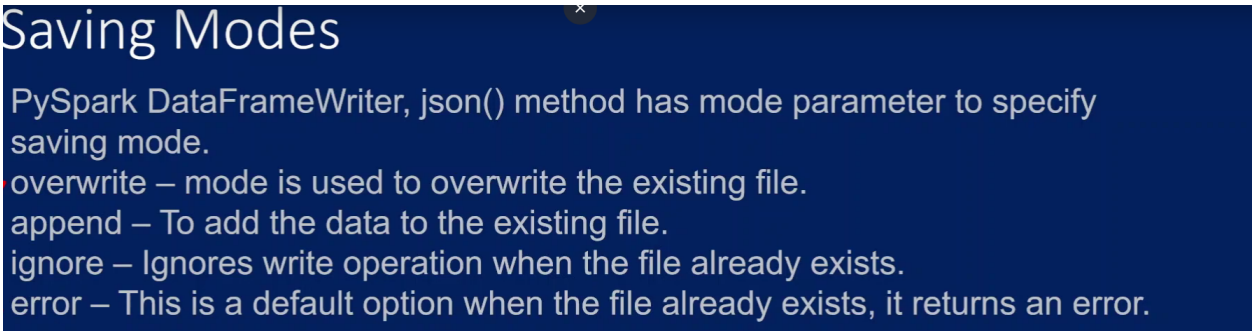
- How to read write dataframe into PARQUET File using PySpark
- Saving modes

Write Dataframe into parquet

Use the DataFrameWrite object to write PySpark DataFrame to parquet file.



Saving Modes



- Append

```
[46] #storing df into parquet
     df.write.parquet('/content/parquetop1',mode='append')
```

```
[48] df1= spark.read.parquet('/content/parquetop1')
     df1.show()
```

```
+---+--------+
| id|    name|
+---+--------+
|  2|Naushad|
|  2|Naushad|
|  1| Nayeer|
|  1| Nayeer|
+---+--------+
```

- OverWrite

```
[49] df.write.parquet('/content/parquetop1',mode='overwrite')
```

```
[50] df1= spark.read.parquet('/content/parquetop1')
     df1.show()
```

```
+---+-------+
| id|   name|
+---+-------+
|  2|Naushad|
|  1| Nayeer|
+---+-------+
```

- Error

```
[52] df.write.parquet('/content/parquetop1',mode='error')

---------------------------------------------------------------------------
AnalysisException                         Traceback (most recent call last)
<ipython-input-52-3e87397ab671> in <cell line: 1>()
----> 1 df.write.parquet('/content/parquetop1',mode='error')

                                    2 frames
/usr/local/lib/python3.10/dist-packages/pyspark/errors/exceptions/captured.py in deco(*a, **kw)
    183                 # Hide where the exception came from that shows a non-Pythonic
    184                 # JVM exception message.
--> 185                 raise converted from None
    186             else:
    187                 raise

AnalysisException: [PATH_ALREADY_EXISTS] Path file:/content/parquetop1 already exists. Set mode as "overwrite" to overwrit

Next steps:  Explain error

[53] df1= spark.read.parquet('/content/parquetop1')
     df1.show()

+---+-------+
| id|   name|
+---+-------+
|  2|Naushad|
|  1| Nayeer|
+---+-------+
```

- Ignore

```
[54] df.write.parquet('/content/parquetop1',mode='ignore')

[55] df1= spark.read.parquet('/content/parquetop1')
     df1.show()

+---+-------+
| id|   name|
+---+-------+
|  2|Naushad|
|  1| Nayeer|
+---+-------+

Start coding or generate with AI.
```

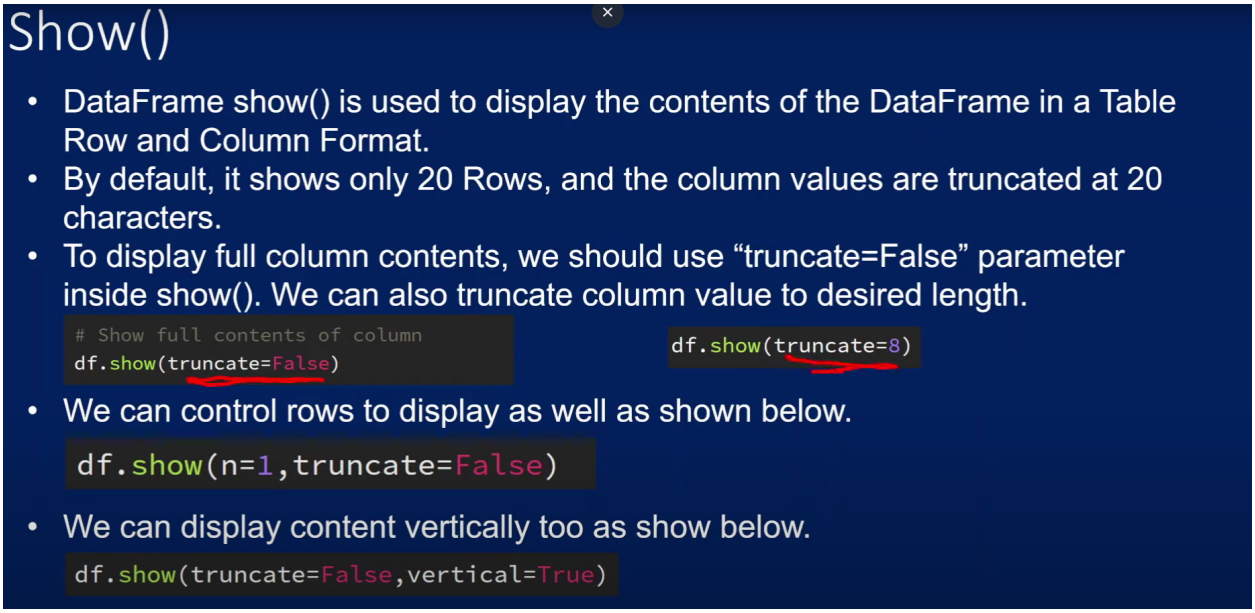
9. Show() - Display Dataframe contents in the table

Agenda

- DataFrame show() = to display contents in the table.

```
[64] #showNoteBook
data=[(1,'Kingnvksnvnsvnvnskvnksnvksnvksvnknvkvnksnvk'),
      (2,'Princsggssvvsvsvsvse'),
      (3,'Hitmansvsvsrvsrv'),
      (4,'Jaddusrgsrggsrjgsrjogrgosrisrjg')]

schema=['id','name']

df = spark.createDataFrame(data=data,schema=schema)

df.show()

df.show(truncate=False) # for showing full data value
df.show(truncate=2)
df.show(truncate=False,vertical=True)
df.show(n=2,truncate=False)
```

```
+---+--------------------+
| id|                name|
+---+--------------------+
|  1|Kingnvksnvnsvnvns...|
|  2|Princsggssvvsvsvsvse|
|  3|    Hitmansvsvsrvsrv|
|  4|Jaddusrgsrggsrjgs...|
+---+--------------------+

+---+-----------------------------------------+
|id |name                                     |
+---+-----------------------------------------+
|1  |Kingnvksnvnsvnvnskvnksnvksnvksvnknvkvnksnvk|
|2  |Princsggssvvsvsvsvse                     |
|3  |Hitmansvsvsrvsrv                         |
|4  |Jaddusrgsrggsrjgsrjogrgosrisrjg          |
+---+-----------------------------------------+

+---+----+
| id|name|
+---+----+
|  1|  Ki|
|  2|  Pr|
|  3|  Hi|
|  4|  Ja|
+---+----+

-RECORD 0-----------------------------------------
 id   | 1
 name | Kingnvksnvnsvnvnskvnksnvksnvksvnknvkvnksnvk
-RECORD 1-----------------------------------------
 id   | 2
 name | Princsggssvvsvsvsvse
-RECORD 2-----------------------------------------
 id   | 3
 name | Hitmansvsvsrvsrv
-RECORD 3-----------------------------------------
 id   | 4
 name | Jaddusrgsrggsrjgsrjogrgosrisrjg

+---+-----------------------------------------+
|id |name                                     |
+---+-----------------------------------------+
|1  |Kingnvksnvnsvnvnskvnksnvksnvksvnknvkvnksnvk|
|2  |Princsggssvvsvsvsvse                     |
+---+-----------------------------------------+
only showing top 2 rows
```



10. withColumn() usage in PySpark

Add new column or change value in column or change column DataTypes in Dataframe.

Agenda

- Dataframe withcolumn() - usage

  * Change column datatypes

  * Update value of existing column

  * Create a new column

withColumn()

withColumn()

- PySpark withColumn() is a transformation function of DataFrame which is used to change the value, convert the datatype of an existing column, create a new column, and many more

```python
#showNoteBook
from pyspark.sql.functions import col
data=[(1,'786'),
      (2,'Princsggssvvsvsvsvse'),
      (3,'Hitmansvsvsrvsrv'),
      (4,'Jaddusrgsrggsrjgsrjogrgosrisrjg')]

columns=['id','name']

df = spark.createDataFrame(data=data,schema=columns)

df.show()
df1=df.withColumn(colName='name',col=col('name').cast('Integer'))

df1.show()
df1.printSchema()


df2=df1.withColumn('name',col('name')*2)
df2.show()
df2.printSchema()
```

```
+---+--------------------+
| id|                name|
+---+--------------------+
|  1|                 786|
|  2|Princsggssvvsvsvsvse|
|  3|    Hitmansvsvsrvsrv|
|  4|Jaddusrgsrggsrjgs...|
+---+--------------------+

+---+----+
| id|name|
+---+----+
|  1| 786|
|  2|NULL|
|  3|NULL|
|  4|NULL|
+---+----+

root
 |-- id: long (nullable = true)
 |-- name: integer (nullable = true)

+---+----+
| id|name|
+---+----+
|  1|1572|
|  2|NULL|
|  3|NULL|
|  4|NULL|
+---+----+

root
 |-- id: long (nullable = true)
 |-- name: integer (nullable = true)
```

```
from pyspark.sql.functions import lit
df3 = df2.withColumn('salary',lit('2000'))  ## lit will add new column
df3.show()
```

```
+---+----+------+
| id|name|salary|
+---+----+------+
|  1|1572|  2000|
|  2|NULL|  2000|
|  3|NULL|  2000|
|  4|NULL|  2000|
+---+----+------+
```

```
] Start coding or generate with AI.
```

```
df4 = df3.withColumn('copiedsalary',col('salary'))  ## to make duplicate
df4.show()
```

```
+---+----+------+------------+
| id|name|salary|copiedsalary|
+---+----+------+------------+
|  1|1572|  2000|        2000|
|  2|NULL|  2000|        2000|
|  3|NULL|  2000|        2000|
|  4|NULL|  2000|        2000|
+---+----+------+------------+
```

11. withColumnRename() usage in Pyspark

Rename column in Dataframe

Agenda

- Dataframe withColumnReturn() - usage
  - Rename column in Dataframe

```
#renaming using withColumnRename()
data=[(1,'Kingnvksnvnsvnvnskvnksnvksnvksvnknvkvnksnvk'),
      (2,'Princsggssvvsvsvsvse'),
      (3,'Hitmansvsvsrvsrv'),
      (4,'Jaddusrgsrggsrjgsrjogrgosrisrjg')]

schema=['id','name']

df = spark.createDataFrame(data=data,schema=columns)

df1=df.withColumnRenamed('name','names')
df1.show()
```

```
+---+--------------------+
| id|               names|
+---+--------------------+
|  1|Kingnvksnvnsvnvns...|
|  2|Princsggssvvsvsvsvse|
|  3|    Hitmansvsvsrvsrv|
|  4|Jaddusrgsrggsrjgs...|
+---+--------------------+
```

12. StructType() & StructField()

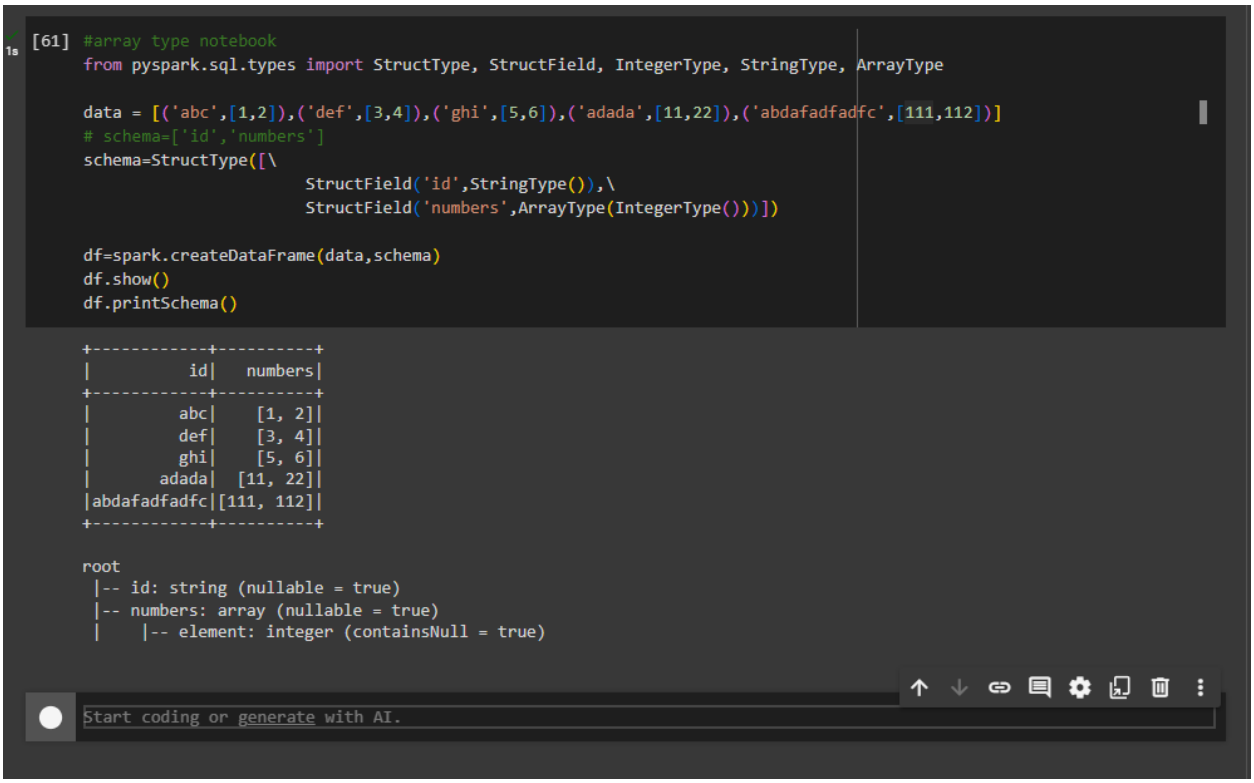
StructType() & StructField()

- PySpark StructType & StructField classes are used to programmatically specify the schema to the DataFrame and create complex columns like nested struct, array, and map columns
- StructType is a collection of StructField's

```
[71] #structType structField
from pyspark.sql import SparkSession
from pyspark.sql.types import StructType, StructField, IntegerType, StringType

data= [(1,('Nay','King'),4500),(2,('Eer','Prince'),4000)]

structName=StructType([\
                       StructField('firstname',StringType()),\
                       StructField('lastname',StringType())])

schema=StructType([\
                   StructField(name='id',dataType=IntegerType()),\
                   StructField(name='name',dataType=structName),\
                   StructField(name='Salary',dataType=IntegerType())\
                   ])

df=spark.createDataFrame(data,schema)
df.show()
df.printSchema()
```

```
+---+-------------+------+
| id|         name|Salary|
+---+-------------+------+
|  1|  {Nay, King}|  4500|
|  2|{Eer, Prince}|  4000|
+---+-------------+------+

root
 |-- id: integer (nullable = true)
 |-- name: struct (nullable = true)
 |    |-- firstname: string (nullable = true)
 |    |-- lastname: string (nullable = true)
 |-- Salary: integer (nullable = true)
```

13. ArrayType Columns in PySpark

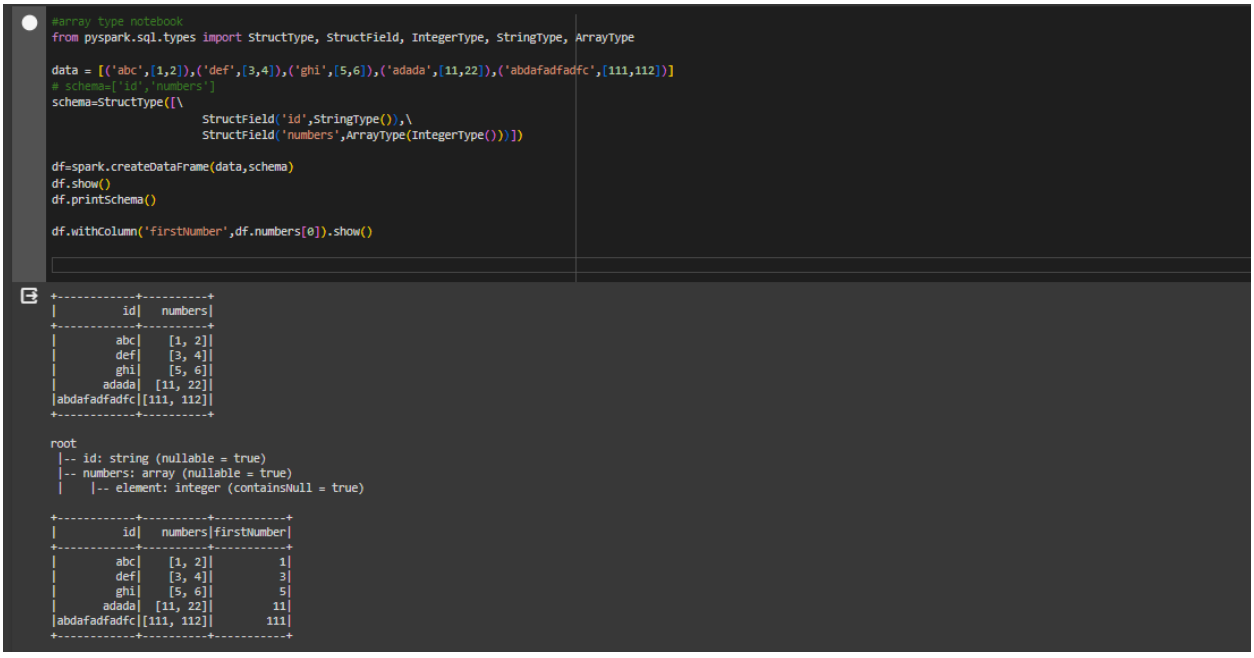


```
[61] #array type notebook
     from pyspark.sql.types import StructType, StructField, IntegerType, StringType, ArrayType

     data = [('abc',[1,2]),('def',[3,4]),('ghi',[5,6]),('adada',[11,22]),('abdafadfadfc',[111,112])]
     # schema=['id','numbers']
     schema=StructType([\
                         StructField('id',StringType()),\
                         StructField('numbers',ArrayType(IntegerType()))])

     df=spark.createDataFrame(data,schema)
     df.show()
     df.printSchema()
```

```
+------------+----------+
|          id|   numbers|
+------------+----------+
|         abc|    [1, 2]|
|         def|    [3, 4]|
|         ghi|    [5, 6]|
|       adada|  [11, 22]|
|abdafadfadfc|[111, 112]|
+------------+----------+

root
 |-- id: string (nullable = true)
 |-- numbers: array (nullable = true)
 |    |-- element: integer (containsNull = true)
```

```
Start coding or generate with AI.
```

Fetch Value from array as new column



```
#array type notebook
from pyspark.sql.types import StructType, StructField, IntegerType, StringType, ArrayType

data = [('abc',[1,2]),('def',[3,4]),('ghi',[5,6]),('adada',[11,22]),('abdafadfadfc',[111,112])]
# schema=['id','numbers']
schema=StructType([\
                    StructField('id',StringType()),\
                    StructField('numbers',ArrayType(IntegerType()))])

df=spark.createDataFrame(data,schema)
df.show()
df.printSchema()

df.withColumn('firstNumber',df.numbers[0]).show()
```

```
+------------+----------+
|          id|   numbers|
+------------+----------+
|         abc|    [1, 2]|
|         def|    [3, 4]|
|         ghi|    [5, 6]|
|       adada|  [11, 22]|
|abdafadfadfc|[111, 112]|
+------------+----------+

root
 |-- id: string (nullable = true)
 |-- numbers: array (nullable = true)
 |    |-- element: integer (containsNull = true)

+------------+----------+-----------+
|          id|   numbers|firstNumber|
+------------+----------+-----------+
|         abc|    [1, 2]|          1|
|         def|    [3, 4]|          3|
|         ghi|    [5, 6]|          5|
|       adada|  [11, 22]|         11|
|abdafadfadfc|[111, 112]|        111|
+------------+----------+-----------+
```

Combine columns to array

```
[72] from pyspark.sql.functions import col,array
     df=spark.createDataFrame(
         [(33,44),(55,66)],["num1","num2"]
     )
     df.show()

     df.withColumn("numbers",array(df.num1,df.num2)).show()
```

```
+----+----+
|num1|num2|
+----+----+
|  33|  44|
|  55|  66|
+----+----+

+----+----+--------+
|num1|num2| numbers|
+----+----+--------+
|  33|  44|[33, 44]|
|  55|  66|[55, 66]|
+----+----+--------+
```

14. exploded(), split(), array() & array_contains() functions

explode()

Use explode() function to create a new row for each element in the given array column.

```
[76] ##Use explode() function to create a new row for each element in the given array column.
     from pyspark.sql.types import StructType , StructField , StringType , IntegerType , ArrayType
     from pyspark.sql.functions import col,explode

     data = [('abc',[1,2]),('def',[3,4]),('ghi',[5,6]),('adada',[11,22]),('abdafadfadfc',[111,112])]
     # schema=['id','numbers']
     schema=StructType([\
                         StructField('id',StringType()),\
                         StructField('numbers',ArrayType(IntegerType()))])

     df=spark.createDataFrame(data,schema)
     df.show()
     df1=df.withColumn('explodedCol',explode(col('numbers'))).select('id','explodedCol')
     df1.show()
```

```
+------------+----------+
|          id|   numbers|
+------------+----------+
|         abc|    [1, 2]|
|         def|    [3, 4]|
|         ghi|    [5, 6]|
|       adada|  [11, 22]|
|abdafadfadfc|[111, 112]|
+------------+----------+

+------------+-----------+
|          id|explodedCol|
+------------+-----------+
|         abc|          1|
|         abc|          2|
|         def|          3|
|         def|          4|
|         ghi|          5|
|         ghi|          6|
|       adada|         11|
|       adada|         22|
|abdafadfadfc|        111|
|abdafadfadfc|        112|
+------------+-----------+
```

Use split() function to create a new row for each element in the given array column.

```
##Use split() function to create a new row for each element in the given array column.
# from pyspark.sql.types import StructType , StructField , StringType , IntegerType , ArrayType
from pyspark.sql.functions import split
data=[(1,'Nayeer','dyna,cae,wew'),(2,'Naushad','cloud,py,aws')]
schema=['id','name','skills']

df = spark.createDataFrame(data  , schema)
df=df.withColumn('skills',split('skills',','))

df.show()
```

```
+---+-------+---------------+
| id|   name|         skills|
+---+-------+---------------+
|  1| Nayeer|[dyna, cae, wew]|
|  2|Naushad|[cloud, py, aws]|
+---+-------+---------------+
```

Use array() function to create a new row for each element in the given array column.

```
[82] ##Use array() function to create a new row for each element in the given array column.
# from pyspark.sql.types import StructType , StructField , StringType , IntegerType , ArrayType
from pyspark.sql.functions import split , array , col
data=[(1,'Nayeer','dyna','cae'),(2,'Naushad','cloud','py')]
schema=['id','name','skillsf','skillss']

df = spark.createDataFrame(data  , schema)
df.show()
df1=df.withColumn('skills',array(col('skillsf'),col('skillss')))

df1.show()
df.printSchema()
```

```
+---+-------+-------+-------+
| id|   name|skillsf|skillss|
+---+-------+-------+-------+
|  1| Nayeer|   dyna|    cae|
|  2|Naushad|  cloud|     py|
+---+-------+-------+-------+

+---+-------+-------+-------+-----------+
| id|   name|skillsf|skillss|     skills|
+---+-------+-------+-------+-----------+
|  1| Nayeer|   dyna|    cae|[dyna, cae]|
|  2|Naushad|  cloud|     py|[cloud, py]|
+---+-------+-------+-------+-----------+

root
 |-- id: long (nullable = true)
 |-- name: string (nullable = true)
 |-- skillsf: string (nullable = true)
 |-- skillss: string (nullable = true)
```

Use arrayContains() function to create a new row for each element in the given array column.

```
[84] ##Use arraycontains() function to create a new row for each element in the given array column.
     # from pyspark.sql.types import StructType , StructField , StringType , IntegerType , ArrayType
     from pyspark.sql.functions import split , array , col , array_contains
     data=[(1,'Nayeer',['dyna','cae']),(2,'Naushad',['cloud','py'])]
     schema=['id','name','skills']

     df = spark.createDataFrame(data  , schema)
     df.show()
     df1=df.withColumn('hasJavaSkill',array_contains(col('skills'),'java'))

     df1.show()
     df.printSchema()
```

```
+---+-------+-----------+
| id|   name|     skills|
+---+-------+-----------+
|  1| Nayeer|[dyna, cae]|
|  2|Naushad|[cloud, py]|
+---+-------+-----------+

+---+-------+-----------+------------+
| id|   name|     skills|hasJavaSkill|
+---+-------+-----------+------------+
|  1| Nayeer|[dyna, cae]|       false|
|  2|Naushad|[cloud, py]|       false|
+---+-------+-----------+------------+

root
 |-- id: long (nullable = true)
 |-- name: string (nullable = true)
 |-- skills: array (nullable = true)
 |    |-- element: string (containsNull = true)
```

15. MapType Column in PySpark

- PySpark MapType is used to represent map key-value pair similar to python Dictionary (Dict).

```
[59] #- PySpark MapType is used to represent map key-value pair similar to python Dictionary (Dict).
     data=[('nayeer',{'hair':'black','eye':'brown'}),('naushad',{'hair':'brown','eye':'brown'})]
     schema=['name','properties']
     df=spark.createDataFrame(data,schema)
     df.show()
     df.printSchema()
```

```
+-------+--------------------+
|   name|          properties|
+-------+--------------------+
| nayeer|{eye -> brown, ha...|
|naushad|{eye -> brown, ha...|
+-------+--------------------+

root
 |-- name: string (nullable = true)
 |-- properties: map (nullable = true)
 |    |-- key: string
 |    |-- value: string (valueContainsNull = true)
```

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

data=[('nayeer',{'hair':'black','eye':'brown'}),('naushad',{'hair':'brown','eye':'brown'})]
schema=StructType([\
                   StructField('name',StringType()),\
                   StructField('properties',MapType(StringType(),StringType()))\
                   ])
df=spark.createDataFrame(data,schema)
df.show(truncate=False)
display(df)
df.printSchema()
```

```
+-------+---------------------------+
|name   |properties                 |
+-------+---------------------------+
|nayeer |{eye -> brown, hair -> black}|
|naushad|{eye -> brown, hair -> brown}|
+-------+---------------------------+

DataFrame[name: string, properties: map<string,string>]
root
 |-- name: string (nullable = true)
 |-- properties: map (nullable = true)
 |    |-- key: string
 |    |-- value: string (valueContainsNull = true)
```

```
[73] #Access maptype elements
     df=df.withColumn('hair',df.properties['hair'])
     df.show()
```

```
+-------+--------------------+-----+
|   name|          properties| hair|
+-------+--------------------+-----+
| nayeer|{eye -> brown, ha...|black|
|naushad|{eye -> brown, ha...|brown|
+-------+--------------------+-----+
```

16. map_keys(), map_values() & explode() functions to work with MapType Columns in PySpark

- explode()
- map_keys()
- map_values()

explode()

```
[74] #explode
     from pyspark.sql.functions import explode
     df.select('name','properties',explode(df.properties)).show(truncate=False)
```

```
+-------+-----------------------------+----+-----+
|name   |properties                   |key |value|
+-------+-----------------------------+----+-----+
|nayeer |{eye -> brown, hair -> black}|eye |brown|
|nayeer |{eye -> brown, hair -> black}|hair|black|
|naushad|{eye -> brown, hair -> brown}|eye |brown|
|naushad|{eye -> brown, hair -> brown}|hair|brown|
+-------+-----------------------------+----+-----+
```

- map_keys()

```
[78] #map_keys()
     #explode
     from pyspark.sql.functions import map_keys
     df1=df.withColumn('keys',map_keys(df.properties))
     df1.show(truncate=False)
```

```
+-------+-----------------------------+-----+----------+
|name   |properties                   |hair |keys      |
+-------+-----------------------------+-----+----------+
|nayeer |{eye -> brown, hair -> black}|black|[eye, hair]|
|naushad|{eye -> brown, hair -> brown}|brown|[eye, hair]|
+-------+-----------------------------+-----+----------+
```

- map_values()

```
##map_keys()
#explode
from pyspark.sql.functions import map_values
df1=df.withColumn('values',map_values(df.properties))
df1.show(truncate=False)
```

```
+-------+-----------------------------+-----+--------------+
|name   |properties                   |hair |values        |
+-------+-----------------------------+-----+--------------+
|nayeer |{eye -> brown, hair -> black}|black|[brown, black]|
|naushad|{eye -> brown, hair -> brown}|brown|[brown, brown]|
+-------+-----------------------------+-----+--------------+
```

17. Row() class in PySpark

- Row()

pyspark.sql.row which is represented as record row in fdataframe , one can create a row object by using named argumnets or create a custom row like class,

```
[82] #pyspark.sql.row which is represented as record row in fdataframe , one
     # can create a row object by using named argumnets or create a custom row like class,

     from pyspark.sql import Row
     row=Row('nayeer',2000)
     print(row[0]+' '+str(row[1]))
```

```
nayeer 2000
```

Using named argument

```python
#Using named argument
from pyspark.sql import Row
row=Row(name='Nayeer',salary=2000)
print(row.name+' '+str(row.salary))
```

```
Nayeer 2000
```

```python
from pyspark.sql import Row

row1 = Row(name = 'maheer',salary = 2000)

row2 = Row(name = 'wafa', salary = 3000)

data = [row1,row2]

df = spark.createDataFrame(data)
df.show()
```

▸ (3) Spark Jobs

▸ df: pyspark.sql.dataframe.DataFrame = [name: string, salary: long]

```
+------+------+
|  name|salary|
+------+------+
|maheer|  2000|
|  wafa|  3000|
+------+------+
```

```python
#we can also create row like this
person=Row('name','salary')
p1=person('nayeer',20000)
p2=person('naushad',20002)

print(p1.name+' '+p2.name)
print(str(p1.salary)+' '+str(p2.salary))
```

```
nayeer naushad
20000 20002
```

```python
#we can also create row like this
person=Row('name','salary')
p1=person('nayeer',20000)
p2=person('naushad',20002)

print(p1.name+' '+p2.name)
print(str(p1.salary)+' '+str(p2.salary))
```

```
nayeer naushad
20000 20002
```

Row()

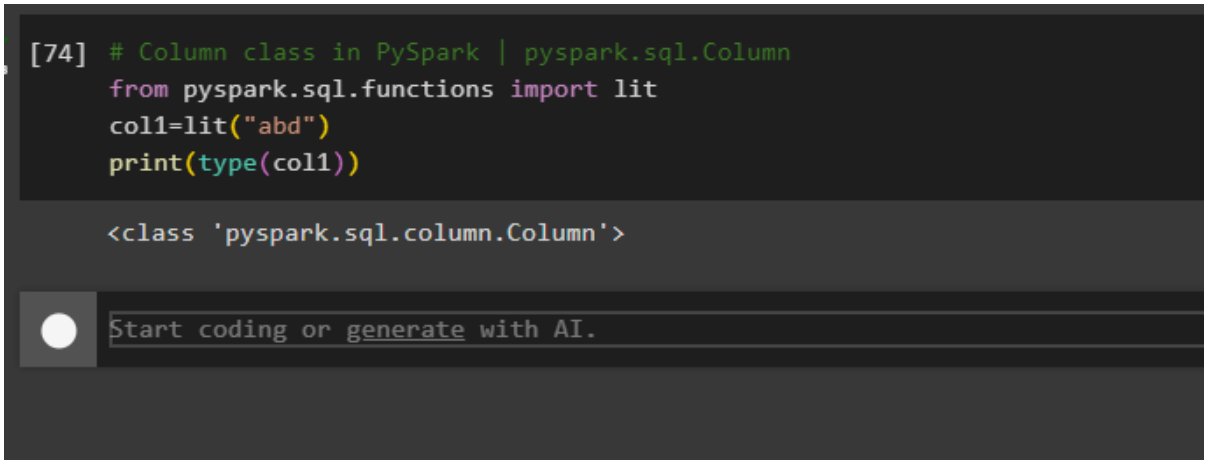
- We can create nested struct type also using Row().

```python
from pyspark.sql import Row
data=[Row(name="maheer",prop=Row(hair="black",eye="blue")),
      Row(name="wafa",prop=Row(hair="grey",eye="black"))]
df=spark.createDataFrame(data)
df.printSchema()
```

18. Column class in PySpark | pyspark.sql.Column

Column

- PySpark Column class represents a single Column in a DataFrame.
- pyspark.sql.Column class provides several functions to work with DataFrame to manipulate the Column values, evaluate the boolean expression to filter rows, retrieve a value or part of a value from a DataFrame column
- One of the simplest ways to create a Column class object is by using PySpark lit() SQL function



Access column from Dataframe

- You can access column multiple ways from dataframe.

```
[79] from pyspark.sql.functions import col
     df.select(df.gender).show()
     df.select(df['gender']).show()
     df.select(col('gender')).show()
```

```
+------+
|gender|
+------+
|  male|
|  male|
+------+

+------+
|gender|
+------+
|  male|
|  male|
+------+

+------+
|gender|
+------+
|  male|
|  male|
+------+
```

```
from pyspark.sql.functions import col
df.select(df.properties.hair).show()
df.select(df['properties.hair']).show()
df.select(col('properties.hair')).show()
```

```
+----------------+
|properties[hair]|
+----------------+
|           black|
|           brown|
+----------------+

+-----+
| hair|
+-----+
|black|
|brown|
+-----+

+-----+
| hair|
+-----+
|black|
|brown|
+-----+
```

you can use struct also

19. when() & otherwise() functions in PySpark

When() and otherwise()

```
from pyspark.sql.functions import when
data=[(1,'nayeer','male'),(2,'naca','feMale'),(3,'nadcadca','unknown')]
schema=['id','name','gender']
df=spark.createDataFrame(data,schema)

df.show()
df.printSchema()

df1=df.select(df.id,df.name,when(condition=df.gender=='male',value='male')\
              .when (condition=df.gender=='feMale',value='female')\
              .otherwise('unknown').alias('gender'))
df1.show()
```

```
+---+--------+-------+
| id|    name| gender|
+---+--------+-------+
|  1|  nayeer|   male|
|  2|    naca| feMale|
|  3|nadcadca|unknown|
+---+--------+-------+

root
 |-- id: long (nullable = true)
 |-- name: string (nullable = true)
 |-- gender: string (nullable = true)

+---+--------+-------+
| id|    name| gender|
+---+--------+-------+
|  1|  nayeer|   male|
|  2|    naca| female|
|  3|nadcadca|unknown|
+---+--------+-------+
```

20. alias(), asc(), desc(), cast() & like() functions on Columns of dataframe in PySpark

alias

- Provides alias to the column

```
data = [(1,'maheer','M',2000),(2,'wafa','M',4000),(3,'asi','F',3000)]

schema = ['id','name','gender','salary']

df = spark.createDataFrame(data,schema)

df.select(df.id.alias('emp_id'),df.name.alias('emp_name')).show()
```

asc() & desc()

- Sort columns ascending or descending order.

```
data = [(1,'maheer','M',2000),(2,'wafa','M',4000),(3,'asi','F',3000)]

schema = ['id','name','gender','salary']

df = spark.createDataFrame(data,schema)

df.sort(df.name.desc()).show()
```

```
#alias
from pyspark.sql.functions import when
data=[(1,'nayeer','male'),(2,'naca','feMale'),(3,'nadcadca','unknown')]
schema=['id','name','gender']
df=spark.createDataFrame(data,schema)
df.select(df.id.alias('emp_id'),df.name.alias('emp_name')).show()
```

```
+------+--------+
|emp_id|emp_name|
+------+--------+
|     1|  nayeer|
|     2|    naca|
|     3|nadcadca|
+------+--------+
```

```
[79] #asc() desc()
from pyspark.sql.functions import when
data=[(1,'nayeer','male'),(2,'naca','feMale'),(3,'nadcadca','unknown')]
schema=['id','name','gender']
df=spark.createDataFrame(data,schema)

#df1=df.sort(df.name.asc())

df1=df.sort(df.name.desc())
df1.show()
```

```
+---+--------+-------+
| id|    name| gender|
+---+--------+-------+
|  1|  nayeer|   male|
|  3|nadcadca|unknown|
|  2|    naca| feMale|
+---+--------+-------+
```

cast()

- convert the datatype

```
data = [(1,'maheer','M',2000),(2,'wafa','M',4000),(3,'asi','F',3000)]

schema = ['id','name','gender','salary']

df = spark.createDataFrame(data,schema)
df1 = df.select(df.salary.cast('int'))
df1.printSchema()
```

like()

- Similar to SQL LIKE expression

```
data = [(1,'maheer','M',2000),(2,'wafa','M',4000),(3,'asi','F',3000)]

schema = ['id','name','gender','salary']

df = spark.createDataFrame(data,schema)
df.filter(df.name.like('m%')).show()
```

```
[83] #cast()
from pyspark.sql.functions import when
data=[(1,'nayeer','male'),(2,'naca','feMale'),(3,'nadcadca','unknown')]
schema=['id','name','gender']
df=spark.createDataFrame(data,schema)
df1=df.select(df.name.cast('int'))
df1.printSchema()

root
 |-- name: integer (nullable = true)
```

```
[86]  #like()
from pyspark.sql.functions import when
data=[(1,'mayeer','male'),(2,'naca','feMale'),(3,'nadcadca','unknown')]
schema=['id','name','gender']
df=spark.createDataFrame(data,schema)
df2=df.filter(df.name.like('m%')).show()

+---+------+------+
| id|  name|gender|
+---+------+------+
|  1|mayeer|  male|
+---+------+------+
```

21. filter() & where() in PySpark

where()

```
#where  functions
 #like()
from pyspark.sql.functions import when
data=[(1,'mayeer','male'),(2,'naca','feMale'),(3,'nadcadca','unknown')]
schema=['id','name','gender']
df=spark.createDataFrame(data,schema)
df.where(df.gender=='male').show()
```

```
+---+------+------+
| id|  name|gender|
+---+------+------+
|  1|mayeer|  male|
+---+------+------+
```

Filter()

```
#  Filter() functions

from pyspark.sql.functions import when
data=[(1,'mayeer','male'),(2,'naca','feMale'),(3,'nadcadca','unknown')]
schema=['id','name','gender']
df=spark.createDataFrame(data,schema)
df.filter(df.gender=='male').show()
```

```
+---+------+------+
| id|  name|gender|
+---+------+------+
|  1|mayeer|  male|
+---+------+------+
```

In case of multiple where/filter

```
1  df.where((df.gender == 'male') & (df.salary == 2000)).show()
```

```
▸ (3) Spark Jobs
+---+------+------+------+
| id|  name|gender|salary|
+---+------+------+------+
|  1|maheer|  male|  2000|
+---+------+------+------+
```

22. distinct() & dropDuplicates() in PySpark

distinct() & dropDuplicates()

- PySpark distinct() function is used to remove the duplicate rows (all columns)
- dropDuplicates() is used to drop rows based on selected (one or multiple) columns.
- So basically, using these functions we can get distinct rows

```
#\22. distinct() & dropDuplicates() in PySpark
from pyspark.sql.functions import when
data=[(1,'mayeer','male'),(2,'naca','feMale'),(3,'nadcadca','unknown'),(4,'nadcadca','unknown'),(3,'nadcadca','unknown')]
schema=['id','name','gender']
df=spark.createDataFrame(data,schema)
df.distinct().show()
df.dropDuplicates().show()
df.dropDuplicates(['gender']).show()
```

```
+---+--------+-------+
| id|    name| gender|
+---+--------+-------+
|  1|  mayeer|   male|
|  2|    naca| feMale|
|  3|nadcadca|unknown|
|  4|nadcadca|unknown|
+---+--------+-------+

+---+--------+-------+
| id|    name| gender|
+---+--------+-------+
|  1|  mayeer|   male|
|  2|    naca| feMale|
|  3|nadcadca|unknown|
|  4|nadcadca|unknown|
+---+--------+-------+

+---+--------+-------+
| id|    name| gender|
+---+--------+-------+
|  2|    naca| feMale|
|  1|  mayeer|   male|
|  3|nadcadca|unknown|
+---+--------+-------+
```

23. orderBy() & sort() in PySpark

orderBy() & sort()

- You can use either sort() or orderBy() function of PySpark DataFrame to sort DataFrame by ascending or descending order based on single or multiple columns.
- By default, sorting will happen in ascending order. We can explicitly mention ascending or descending using asc(), desc() functions.

order() and sort()

```
[87] #23. orderBy() & sort() in PySpark
     from pyspark.sql.functions import when
     data=[(1013100,50000),(223231,50000),(379236973,989)]
     schema=['id','name',]
     df=spark.createDataFrame(data,schema)
     df.sort('name','id').show()
     df.orderBy(df.name.desc(),df.id.asc()).show()
```

```
+---------+-----+
|       id| name|
+---------+-----+
|379236973|  989|
|   223231|50000|
|  1013100|50000|
+---------+-----+

+---------+-----+
|       id| name|
+---------+-----+
|   223231|50000|
|  1013100|50000|
|379236973|  989|
+---------+-----+
```

```
[88] #23. orderBy() & sort() in PySpark
     from pyspark.sql.functions import when
     data=[(1013100,50000),(223231,50000),(379236973,989)]
     schema=['id','name',]
     df=spark.createDataFrame(data,schema)
     df.orderBy('name','id').show()
     df.sort(df.name.desc(),df.id.asc()).show()
```

```
+---------+-----+
|       id| name|
+---------+-----+
|379236973|  989|
|   223231|50000|
|  1013100|50000|
+---------+-----+

+---------+-----+
|       id| name|
+---------+-----+
|   223231|50000|
|  1013100|50000|
|379236973|  989|
+---------+-----+
```

24. union() & unionAll() in PySpark

union() & unionAll()

- union() and unionAll() transformations are used to merge two or more DataFrame's of the same schema or structure.
- union() & unionAll() method merges two DataFrames and returns the new DataFrame with all rows from two Dataframes regardless of duplicate data.
- To remove duplicates use distinct() function

```
[89] #24. union() & unionAll() in PySpark
     from pyspark.sql.functions import when
     data=[(1013100,50000),(223231,50000),(379236973,989)]
     schema=['id','name',]
     df1=spark.createDataFrame(data,schema)
     df2=spark.createDataFrame(data,schema)

     df1.show()
     df2.show()
     df1.unionAll(df2).show()
     df1.union(df2).show()
```

```
+---------+-----+
|       id| name|
+---------+-----+
|  1013100|50000|
|   223231|50000|
|379236973|  989|
+---------+-----+

+---------+-----+
|       id| name|
+---------+-----+
|  1013100|50000|
|   223231|50000|
|379236973|  989|
+---------+-----+

+---------+-----+
|       id| name|
+---------+-----+
|  1013100|50000|
|   223231|50000|
|379236973|  989|
|  1013100|50000|
|   223231|50000|
|379236973|  989|
+---------+-----+

+---------+-----+
|       id| name|
+---------+-----+
|  1013100|50000|
|   223231|50000|
|379236973|  989|
|  1013100|50000|
|   223231|50000|
|379236973|  989|
+---------+-----+
```

25. groupBy() in PySpark

groupBy()

- Similar to SQL GROUP BY clause, PySpark groupBy() function is used to collect the identical data into groups on DataFrame and perform count, sum, avg, min, max functions on the grouped data

```
92] #25. groupBy() in PySpark
    data=[(1,'naheey','M',5000,'IT'),\
          (2,'baheey','F',50100,'OIT'),\
          (3,'gaheey','M',52000,'RIT'),\
          (4,'taheey','F',53000,'GIT'),\
          (5,'raheey','M',25000,'NIT'),\
          (6,'eaheey','F',15000,'KIT'),\
          (7,'waheey','M',95000,'CIT')]

    schema=['id','name','gender','salary','dep']
    df=spark.createDataFrame(data,schema)
    df.groupBy('dep').count().show()
    df.groupBy('dep','gender').count().show()
    df.groupBy('dep').min('salary').show()
    df.groupBy('dep').max('salary').show()
    df.groupBy('dep').avg('salary').show()
```

```
+---+-----+
|dep|count|
+---+-----+
|OIT|    1|
| IT|    1|
|RIT|    1|
|CIT|    1|
|KIT|    1|
|NIT|    1|
|GIT|    1|
+---+-----+
```

```
+---+------+-----+
|dep|gender|count|
+---+------+-----+
|OIT|     F|    1|
|RIT|     M|    1|
| IT|     M|    1|
|KIT|     F|    1|
|NIT|     M|    1|
|GIT|     F|    1|
|CIT|     M|    1|
+---+------+-----+

+---+-----------+
|dep|min(salary)|
+---+-----------+
|OIT|      50100|
| IT|       5000|
|RIT|      52000|
|CIT|      95000|
|KIT|      15000|
|NIT|      25000|
|GIT|      53000|
+---+-----------+
```

```
+---+-----------+
|dep|max(salary)|
+---+-----------+
|OIT|      50100|
| IT|       5000|
|RIT|      52000|
|CIT|      95000|
|KIT|      15000|
|NIT|      25000|
|GIT|      53000|
+---+-----------+

+---+-----------+
|dep|avg(salary)|
+---+-----------+
|OIT|    50100.0|
| IT|     5000.0|
|RIT|    52000.0|
|CIT|    95000.0|
|KIT|    15000.0|
|NIT|    25000.0|
|GIT|    53000.0|
+---+-----------+
```

OR

```
4s
+---+-----------+
|dep|min(salary)|
+---+-----------+
| IT|       5000|
|RIT|      52000|
|CIT|      95000|
|KIT|      15000|
|NIT|      25000|
|GIT|      53000|
+---+-----------+

+---+-----------+
|dep|max(salary)|
+---+-----------+
| IT|      50100|
|RIT|      52000|
|CIT|      95000|
|KIT|      15000|
|NIT|      25000|
|GIT|      53000|
+---+-----------+

+---+-----------+
|dep|avg(salary)|
+---+-----------+
| IT|    27550.0|
|RIT|    52000.0|
|CIT|    95000.0|
|KIT|    15000.0|
|NIT|    25000.0|
|GIT|    53000.0|
+---+-----------+
```

26. GroupBy agg() function in PySpark

```
[85] #agg()
     #25. groupBy() in PySpark
     data=[(1,'naheey','M',5000,'IT'),\
           (2,'baheey','F',50100,'IT'),\
           (3,'gaheey','M',52000,'RIT'),\
           (4,'taheey','F',53000,'GIT'),\
           (5,'raheey','M',25000,'NIT'),\
           (6,'eaheey','F',15000,'KIT'),\
           (7,'waheey','M',95000,'CIT')]

     schema=['id','name','gender','salary','dep']
     df=spark.createDataFrame(data,schema)
     df.groupBy('dep').count().show()

     from pyspark.sql.functions import count,min,max
     df.groupBy('dep').agg(count('*').alias('countOfEmps'),\
                           min('salary').alias('minSal'),\
                           max('salary').alias('maxSal')).show()
```

```
+---+-----+
|dep|count|
+---+-----+
| IT|    2|
|RIT|    1|
|CIT|    1|
|KIT|    1|
|NIT|    1|
|GIT|    1|
+---+-----+

+---+-----------+------+------+
|dep|countOfEmps|minSal|maxSal|
+---+-----------+------+------+
| IT|          2|  5000| 50100|
|RIT|          1| 52000| 52000|
|CIT|          1| 95000| 95000|
|KIT|          1| 15000| 15000|
|NIT|          1| 25000| 25000|
|GIT|          1| 53000| 53000|
+---+-----------+------+------+
```

27. unionByName() function in PySpark

unionByName()

- unionByName() lets you to merge/union two DataFrames with a different number of columns (different schema) by passing allowMissingColumns with the value true

```python
#27. unionByName() function in PySpark
from pyspark.sql.functions import when
data1=[(10100,'king'),(2231,'prince'),(3726973,'ada')]
schema1=['id','name',]
data2=[(10100,'king'),(379236973,'adada'),(3726973,'dghi')]
schema2=['id','gtae',]

df1=spark.createDataFrame(data1,schema1)
df2=spark.createDataFrame(data2,schema2)

df1.union(df2).show()

df1.unionByName(allowMissingColumns=True,other=df2).show()
```

```
+---------+------+
|       id|  name|
+---------+------+
|    10100|  king|
|     2231|prince|
|  3726973|   ada|
|    10100|  king|
|379236973| adada|
|  3726973|  dghi|
+---------+------+

+---------+------+-----+
|       id|  name| gtae|
+---------+------+-----+
|    10100|  king| NULL|
|     2231|prince| NULL|
|  3726973|   ada| NULL|
|    10100|  NULL| king|
|379236973|  NULL|adada|
|  3726973|  NULL| dghi|
+---------+------+-----+
```

28. select() function in PySpark

select()

- select() function is used to select single, multiple, column by index, all columns from the list and the nested columns from a DataFrame

```
[93] #28. select() function in PySpark
     from pyspark.sql.functions import lit
     data=[('nayeer','male',20000),('naushad','male',300000)]
     schema=['name','gender','salary']
     df=spark.createDataFrame(data,schema)

     #select single or multi columns
     df.select('salary','name').show()
     df.select(df.name,df.gender).show()
     df.select(df['name'],df['gender']).show()

     #using col() function
     from pyspark.sql.functions import col
     df.select(col('salary'),col('name')).show()
     df.select(['gender','name']).show()
```

```
+------+-------+
|salary|   name|
+------+-------+
| 20000| nayeer|
|300000|naushad|
+------+-------+

+-------+------+
|   name|gender|
+-------+------+
| nayeer|  male|
|naushad|  male|
+-------+------+

+-------+------+
|   name|gender|
+-------+------+
| nayeer|  male|
|naushad|  male|
+-------+------+

+------+-------+
|salary|   name|
+------+-------+
| 20000| nayeer|
|300000|naushad|
+------+-------+

+------+-------+
|gender|   name|
+------+-------+
|  male| nayeer|
|  male|naushad|
+------+-------+
```

select()

- select() function to select all columns

```
#select() for all column
df.select([col for col in df.columns]).show()
df.select('*').show()
```

```
+-------+------+------+
|   name|gender|salary|
+-------+------+------+
| nayeer|  male| 20000|
|naushad|  male|300000|
+-------+------+------+

+-------+------+------+
|   name|gender|salary|
+-------+------+------+
| nayeer|  male| 20000|
|naushad|  male|300000|
+-------+------+------+
```

29. join() function in PySpark

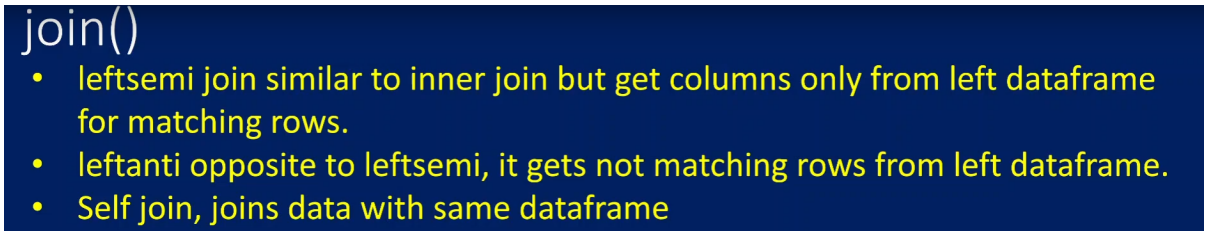
join()

- join() is like SQL JOIN. We can combine columns from different DataFrames based on condition. It supports all basic join types such as INNER, LEFT OUTER, RIGHT OUTER, LEFT ANTI, LEFT SEMI, CROSS, SELF

```
#29. join() function in PySpark
from pyspark.sql.functions import when
data1=[(1,'nwafa',2000,2),(2,'fsfs',5555,1),(3,'abcd',1111,1)]
schema1=['id','name','salary','dep']
data2=[(1,'nwafa'),(379236973,'adada'),(3726973,'dghi')]
schema2=['id','gtae',]

empDf=spark.createDataFrame(data1,schema1)
depDf=spark.createDataFrame(data2,schema2)
empDf.show()
depDf.show()

empDf.join(depDf,empDf.dep==depDf.id,'inner').show()
empDf.join(depDf,empDf.dep==depDf.id,'left').show()
empDf.join(depDf,empDf.dep==depDf.id,'right').show()
empDf.join(depDf,empDf.dep==depDf.id,'full').show()
```

```
+---+-----+------+---+
| id| name|salary|dep|
+---+-----+------+---+
|  1|nwafa|  2000|  2|
|  2| fsfs|  5555|  1|
|  3| abcd|  1111|  1|
+---+-----+------+---+

+---------+-----+
|       id| gtae|
+---------+-----+
|        1|nwafa|
|379236973|adada|
|  3726973| dghi|
+---------+-----+

+---+----+------+---+---+-----+
| id|name|salary|dep| id| gtae|
+---+----+------+---+---+-----+
|  2|fsfs|  5555|  1|  1|nwafa|
|  3|abcd|  1111|  1|  1|nwafa|
+---+----+------+---+---+-----+

+---+-----+------+---+----+-----+
| id| name|salary|dep|  id| gtae|
+---+-----+------+---+----+-----+
|  1|nwafa|  2000|  2|NULL| NULL|
|  2| fsfs|  5555|  1|   1|nwafa|
|  3| abcd|  1111|  1|   1|nwafa|
+---+-----+------+---+----+-----+

+----+----+------+----+---------+-----+
| id|name|salary| dep|       id| gtae|
+----+----+------+----+---------+-----+
|   3|abcd|  1111|   1|        1|nwafa|
|   2|fsfs|  5555|   1|        1|nwafa|
|NULL|NULL|  NULL|NULL|379236973|adada|
|NULL|NULL|  NULL|NULL|  3726973| dghi|
+----+----+------+----+---------+-----+

+----+-----+------+----+---------+-----+
|  id| name|salary| dep|       id| gtae|
+----+-----+------+----+---------+-----+
|   2| fsfs|  5555|   1|        1|nwafa|
|   3| abcd|  1111|   1|        1|nwafa|
|   1|nwafa|  2000|   2|     NULL| NULL|
|NULL| NULL|  NULL|NULL|  3726973| dghi|
|NULL| NULL|  NULL|NULL|379236973|adada|
+----+-----+------+----+---------+-----+
```

30. join() function in PySpark Continuation




```
[101] #30. join() function in PySpark Continuation
      #29. join() function in PySpark
      from pyspark.sql.functions import when
      data1=[(1,'nwafa',2000,2),(2,'fsfs',5555,1),(3,'abcd',1111,1)]
      schema1=['id','name','salary','dep']
      data2=[(1,'nwafa'),(379236973,'adada'),(3726973,'dghi')]
      schema2=['id','gtae',]

      empDf=spark.createDataFrame(data1,schema1)
      depDf=spark.createDataFrame(data2,schema2)
      # empDf.show()
      # depDf.show()

      empDf.join(depDf,empDf.dep==depDf.id,'leftsemi').show()
      empDf.join(depDf,empDf.dep==depDf.id,'leftanti').show()
```

```
+---+----+------+---+
| id|name|salary|dep|
+---+----+------+---+
|  2|fsfs|  5555|  1|
|  3|abcd|  1111|  1|
+---+----+------+---+

+---+-----+------+---+
| id| name|salary|dep|
+---+-----+------+---+
|  1|nwafa|  2000|  2|
+---+-----+------+---+
```

SelfJoin

```
[104] data1=[(1,'nwafa',0),(2,'fsfs',1),(3,'abcd',2)]
      schema1=['id','name','manager_id']
      df=spark.createDataFrame(data1,schema1)
      from pyspark.sql.functions import col
      df.alias('emp').join(df.alias('manager'),col('emp.manager_id')==col('manager.id'),'left')\
          .select(col('emp.name').alias('empName'),col('manager.name').alias('managerName')).show()
```

```
+-------+-----------+
|empName|managerName|
+-------+-----------+
|  nwafa|       NULL|
|   fsfs|      nwafa|
|   abcd|       fsfs|
+-------+-----------+
```

31. pivot() function in PySpark

pivot()

- It's used to rotate data in one column into multiple columns.
- It is an aggregation where one of the grouping column values will be converted in individual columns.

```
[91] #25. groupBy() in PySpark
     data=[(1,'naheey','M',5000,'IT'),\
           (2,'baheey','F',50100,'IT'),\
           (3,'gaheey','M',52000,'RIT'),\
           (4,'taheey','F',53000,'GIT'),\
           (5,'raheey','M',25000,'NIT'),\
           (6,'eaheey','F',15000,'KIT'),\
           (7,'waheey','M',95000,'CIT')]

     schema=['id','name','gender','salary','dep']
     df=spark.createDataFrame(data,schema)
     df.show()

     df.groupBy('dep').pivot('gender').count().show()
```

```
+---+------+------+------+---+
| id|  name|gender|salary|dep|
+---+------+------+------+---+
|  1|naheey|     M|  5000| IT|
|  2|baheey|     F| 50100| IT|
|  3|gaheey|     M| 52000|RIT|
|  4|taheey|     F| 53000|GIT|
|  5|raheey|     M| 25000|NIT|
|  6|eaheey|     F| 15000|KIT|
|  7|waheey|     M| 95000|CIT|
+---+------+------+------+---+

+---+----+----+
|dep|   F|   M|
+---+----+----+
|CIT|NULL|   1|
|KIT|   1|NULL|
|NIT|NULL|   1|
| IT|   1|   1|
|RIT|NULL|   1|
|GIT|   1|NULL|
+---+----+----+
```

31. pivot() function in PySpark

Unpivot Dataframe

- Unpivot is rotating columns into rows. PySpark SQL doesn't have unpivot function hence will use the stack() function.

```
[94] #32. unpivot Dataframe in PySpark
     data=[('IT',8,5),\
           ('Payroll',3,2),\
           ('HR',2,4)\
           ]

     schema=['dep','male','female']

     df=spark.createDataFrame(data,schema)
     df.show()

     from pyspark.sql.functions import expr
     df.select('dep',expr("stack(2,'male',male,'female',female) as (gender, count)")).show()
```

```
+-------+----+------+
|    dep|male|female|
+-------+----+------+
|     IT|   8|     5|
|Payroll|   3|     2|
|     HR|   2|     4|
+-------+----+------+

+-------+------+-----+
|    dep|gender|count|
+-------+------+-----+
|     IT|  male|    8|
|     IT|female|    5|
|Payroll|  male|    3|
|Payroll|female|    2|
|     HR|  male|    2|
|     HR|female|    4|
+-------+------+-----+
```

33. fill() & fillna() functions in PySpark

fill() & fillna()

- fillna() or DataFrameNaFunctions.fill() is used to replace NULL/None values on all or selected multiple DataFrame columns with either zero(0), empty string, space, or any constant literal values.

```
[99] #33. fill() & fillna() functions in PySpark
     from pyspark.sql.functions import lit
     data=[('nayeer','male',None),('naushad',None,300000)]
     schema=['name','gender','salary']
     df=spark.createDataFrame(data,schema)
     df.show()

     df.na.fill('unknown',['gender']).show()

     df.fillna('unknown',['salary']).show()
```

```
+-------+------+------+
|   name|gender|salary|
+-------+------+------+
| nayeer|  male|  NULL|
|naushad|  NULL|300000|
+-------+------+------+

+-------+-------+------+
|   name| gender|salary|
+-------+-------+------+
| nayeer|   male|  NULL|
|naushad|unknown|300000|
+-------+-------+------+

+-------+------+------+
|   name|gender|salary|
+-------+------+------+
| nayeer|  male|  NULL|
|naushad|  NULL|300000|
+-------+------+------+
```

34. sample() function in PySpark

sample()

- To get the random sampling subset from the large dataset
- Use fraction to indicate what percentage of data to return and seed value to make sure every time to get same random sample.

```
#34. sample() function in PySpark
df=spark.range(start=1, end=101)
df1=df.sample(fraction=0.1,seed=123)
df2=df.sample(fraction=0.1,seed=123)  # we put seed tjust to get same random number
df1.show()
df2.show()
```

```
+---+
| id|
+---+
| 37|
| 38|
| 42|
| 44|
| 70|
| 76|
| 85|
| 98|
+---+

+---+
| id|
+---+
| 37|
| 38|
| 42|
| 44|
| 70|
| 76|
| 85|
| 98|
+---+
```

35. collect() function in PySpark

collect()

- collect() retrieves all elements in a DataFrame as an Array of Row type to the driver node.
- collect() is an action hence it does not return a DataFrame instead, it returns data in an Array to the driver. Once the data is in an array, you can use python for loop to process it further.
- collect() use it with small DataFrames. With big DataFrames it may result in out of memory error as its return entier data to single node(driver)

```
#35. collect() function in PySpark
from pyspark.sql.functions import lit
data=[('nayeer','male',None),('naushad',None,300000)]
schema=['name','gender','salary']
df=spark.createDataFrame(data,schema)
df.show()

dataRows=df.collect() #it is similar to select() but here the op is collected in array
print(dataRows)
print(dataRows[0])
print(dataRows[0][0])
```

```
+-------+------+------+
|   name|gender|salary|
+-------+------+------+
| nayeer|  male|  NULL|
|naushad|  NULL|300000|
+-------+------+------+

[Row(name='nayeer', gender='male', salary=None), Row(name='naushad', gender=None, salary=300000)]
Row(name='nayeer', gender='male', salary=None)
nayeer
```

36. DataFrame.transform() function in PySpark

Dataframe.transform()

- It’s is used to chain the custom transformations and this function returns the new DataFrame after applying the specified transformations.

```
#36. DataFrame.transform() function in PySpark

data=[('nayeer','male',5678),('naushad',None,300000)]
schema=['name','gender','salary']
df=spark.createDataFrame(data,schema)
df.show()

from pyspark.sql.functions import upper

def convertToUpper(df):
  return df.withColumn('name',upper(df.name))

def doubleTheSalary(df):
  return df.withColumn('salary',df.salary*2)

df1=df.transform(convertToUpper)\
      .transform(doubleTheSalary)
df1.show()
```

```
+-------+------+------+
|   name|gender|salary|
+-------+------+------+
| nayeer|  male|  5678|
|naushad|  NULL|300000|
+-------+------+------+

+-------+------+------+
|   name|gender|salary|
+-------+------+------+
| NAYEER|  male| 11356|
|NAUSHAD|  NULL|600000|
+-------+------+------+
```

37. pyspark.sql.functions.transform()

pyspark.sql.functions.transform()

- It’s is used to apply the transformation on a column of type Array. This function applies the specified transformation on every element of the array and returns an object of ArrayType.

```
[10] #37. pyspark.sql.functions.transform()
data=[(1,'nayeer',['azure','dot']),(2,'naaefef',['aafae','tod'])]
schema=['id','name','skills']
df=spark.createDataFrame(data,schema)
df.show()
df.printSchema()

from pyspark.sql.functions import transform
df.select('id','name',transform('skills',lambda x: upper(x)).alias('skills')).show()

def convertToUpper(x):
  return upper(x)

from pyspark.sql.functions import transform, upper
df.select(transform('skills',convertToUpper).alias('skills')).show()
```

```
+---+-------+------------+
| id|   name|      skills|
+---+-------+------------+
|  1| nayeer|[azure, dot]|
|  2|naaefef|[aafae, tod]|
+---+-------+------------+

root
 |-- id: long (nullable = true)
 |-- name: string (nullable = true)
 |-- skills: array (nullable = true)
 |    |-- element: string (containsNull = true)

+---+-------+------------+
| id|   name|      skills|
+---+-------+------------+
|  1| nayeer|[AZURE, DOT]|
|  2|naaefef|[AAFAE, TOD]|
+---+-------+------------+

+------------+
|      skills|
+------------+
|[AZURE, DOT]|
|[AAFAE, TOD]|
+------------+
```

38. createOrReplaceTempView() function in PySpark

createOrReplaceTempView()

- Advantage of Spark, you can work with SQL along with DataFrames. That means, if you are comfortable with SQL, you can create temporary view on Dataframe by using createOrReplaceTempView() and use SQL to select and manipulate data.
- Temp Views are session scoped and cannot be shared between the sessions.

```
[121] #38. createOrReplaceTempView() function in PySpark
      data=[(1,'nayeer','male',5678),(2,'naushad',None,300000)]
      schema=['id','name','gender','salary']
      df=spark.createDataFrame(data,schema)

      df.createOrReplaceTempView('employees')
      df1=spark.sql('SELECT * FROM employees')
      df1.show()
```

```
+---+-------+------+------+
| id|   name|gender|salary|
+---+-------+------+------+
|  1| nayeer|  male|  5678|
|  2|naushad|  NULL|300000|
+---+-------+------+------+
```

```
1  %sql
2  SELECT id,UPPER(name) FROM employees
```

▸ (3) Spark Jobs

▸ _sqldf: pyspark.sql.dataframe.DataFrame = [id: long, upper(name): string]

Table +

	id	upper(name)
1	1	MAHEER
2	2	WAFA

39. createOrReplaceGlobalTempView()

createOrReplaceGlobalTempView()

- It's used to create temp views or tables globally, when can be accessed across the sessions with in Spark Application.
- To query these tables, we need append global_temp.<tablename>

```
%scala
spark
```

```
res1: org.apache.spark.sql.SparkSession = org.apache.spark.sql.SparkSession@b609dbc
```

```
data = [(1,'maheer'),(2,'wafa')]
schema = ['id','name']

df = spark.createDataFrame(data,schema)
df.createOrReplaceGlobalTempView('empGlobal')
```

```
df: pyspark.sql.dataframe.DataFrame = [id: long, name: string]
Command took 0.10 seconds -- by maheer3222@gmail.com at 12/22/2022, 12:20:12 PM on SHAIK MAHEER BASHA's Cluster
```

```
%sql
SELECT * FROM global_temp.empGlobal
```

(2) Spark Jobs

_sqldf: pyspark.sql.dataframe.DataFrame = [id: long, name: string]

Table +

	id	name
1	1	maheer
2	2	wafa

Showing all 2 rows. | 0.21 seconds runtime

- GlobalTempView are accessible accross the session

```
# list tables from current sessions
spark.catalog.listTables(spark.catalog.currentDatabase())

#To view global temp tables
spark.catalog.listTables('global_temp')
```

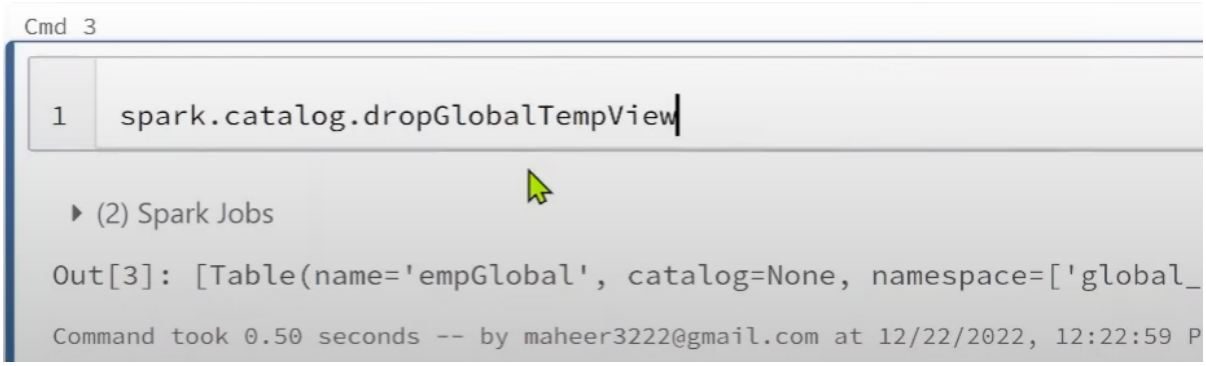
```
spark.catalog.currentDatabase()
```

```
ıt[7]: 'default'
mmand took 0.10 seconds -- by maheer3222@gmail.com at 12/22/2022, 12:22:13 PM on SHAIK MAHEER BASHA's Cluster
```

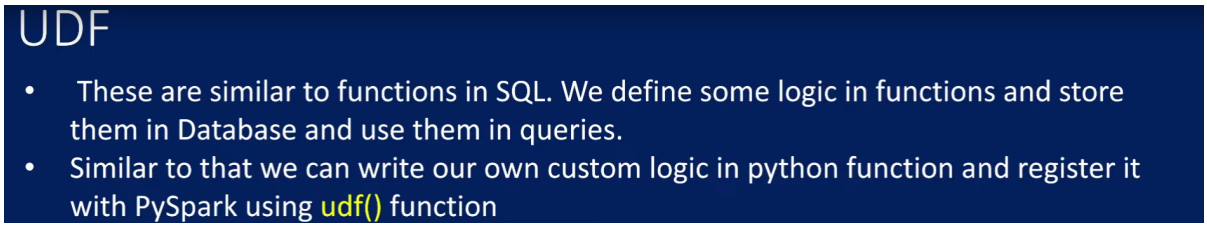
```
spark.catalog.listTables('default')
```

```
(2) Spark Jobs
ıt[8]: [Table(name='empT', catalog=None, namespace=[], description=None, tableType='TEMPORARY', isTemporary=True)]
```

For dropping temp view



40. UDF(user defined function)



```
[131] #40. UDF(user defined function)
      data =[(1,'nayeer',2000,500),(2,'naushad',4000,1000)]

      schema=['id','name','salary','bonus']

      df=spark.createDataFrame(data,schema)
      df.show()
```

```
+---+-------+------+-----+
| id|   name|salary|bonus|
+---+-------+------+-----+
|  1| nayeer|  2000|  500|
|  2|naushad|  4000| 1000|
+---+-------+------+-----+
```

```
def totalPay(s,b):
  return s+b

from pyspark.sql.functions import udf,col
from pyspark.sql.types import IntegerType

TotalPay=udf(lambda x,y: totalPay(x,y),IntegerType())

df.select('*',TotalPay(col('salary'),col('bonus')).alias('totalPay')).show()
df.withColumn('tolPay',TotalPay(df.salary,df.bonus)).show()
```

```
+---+-------+------+-----+--------+
| id|   name|salary|bonus|totalPay|
+---+-------+------+-----+--------+
|  1| nayeer|  2000|  500|    2500|
|  2|naushad|  4000| 1000|    5000|
+---+-------+------+-----+--------+

+---+-------+------+-----+------+
| id|   name|salary|bonus|tolPay|
+---+-------+------+-----+------+
|  1| nayeer|  2000|  500|  2500|
|  2|naushad|  4000| 1000|  5000|
+---+-------+------+-----+------+
```

41. Convert RDD to Dataframe in PySpark

RDD(Resilient Distributed Dataset)

- Its collection of objects similar to list in Python. Its Immutable and In memory processing.
- By using parallelize() function of SparkContext you can create an RDD.

```
[101] #41. Convert RDD to Dataframe in PySpark
      data=[(1,'nayeer'),(2,'wasim')]
      rdd=spark.sparkContext.parallelize(data)
      print(rdd.collect())

      [(1, 'nayeer'), (2, 'wasim')]
```

Convert RDD to Dataframe

```
[102] df=rdd.toDF(['id','name'])
      df.show()

      +---+------+
      | id|  name|
      +---+------+
      |  1|nayeer|
      |  2| wasim|
      +---+------+
```

```
      df=spark.createDataFrame(rdd,['id','name'])
      df.show()

      +---+------+
      | id|  name|
      +---+------+
      |  1|nayeer|
      |  2| wasim|
      +---+------+
```

42. map() transformation in PySpark

map()

- Its RDD transformation used to apply function(lambda) on every element of RDD and returns new RDD.
- Dataframe doesn't have map() transformation to use with Dataframe you need to generate RDD first.

```
data =[('maheer','shaik'),('wafa','shaik')]
```

```
[107] #42. map() transformation in PySpark
      data=[('king','nayeer'),('kohli','wasim')]
      rdd=spark.sparkContext.parallelize(data)
      rdd1=rdd.map(lambda x: x + (x[0]+x[1],) )
      print(rdd1.collect())

      [('king', 'nayeer', 'kingnayeer'), ('kohli', 'wasim', 'kohliwasim')]
```

```
      def FullName(x):
        x=x+(x[0]+' '+x[1],)
        return x

      data=[('king','nayeer'),('kohli','wasim')]
      rdd1=spark.createDataFrame(data,['fn','ln'])
      rdd1=rdd.map(lambda x: FullName(x))
      df1=rdd1.toDF(['fn','ln','FullName'])
      df1.show()

      +-----+------+-----------+
      |   fn|    ln|   FullName|
      +-----+------+-----------+
      | king|nayeer|king nayeer|
      |kohli| wasim|kohli wasim|
      +-----+------+-----------+
```

```
[116] data=[('king','nayeer'),('kohli','wasim')]
      rdd1=spark.createDataFrame(data,['fn','ln'])
      rdd1=rdd.map(lambda x: x + (x[0]+' '+x[1],) )
      df2=rdd1.toDF(['fn','ln','FullName'])
      df2.show()

      +-----+------+-----------+
      |   fn|    ln|   FullName|
      +-----+------+-----------+
      | king|nayeer|king nayeer|
      |kohli| wasim|kohli wasim|
      +-----+------+-----------+
```

43. flatMap() transformation in PySpark

flatMap()

- flatMap() is a transformation operation that flattens the RDD (array/map DataFrame columns) after applying the function on every element and returns a new PySpark RDD
- Its not available in dataframes. Explode() functions can be used in dataframes to flatten arrays.

```
[120] #43. flatMap() transformation in PySpark
      data=[('king nayeer'),('kohli wasim')]
      rdd=spark.sparkContext.parallelize(data)

      for item in rdd.collect():
        print(item)


      rdd1=rdd.flatMap(lambda x: x.split(' '))
      for item in rdd1.collect():
        print(item)
```

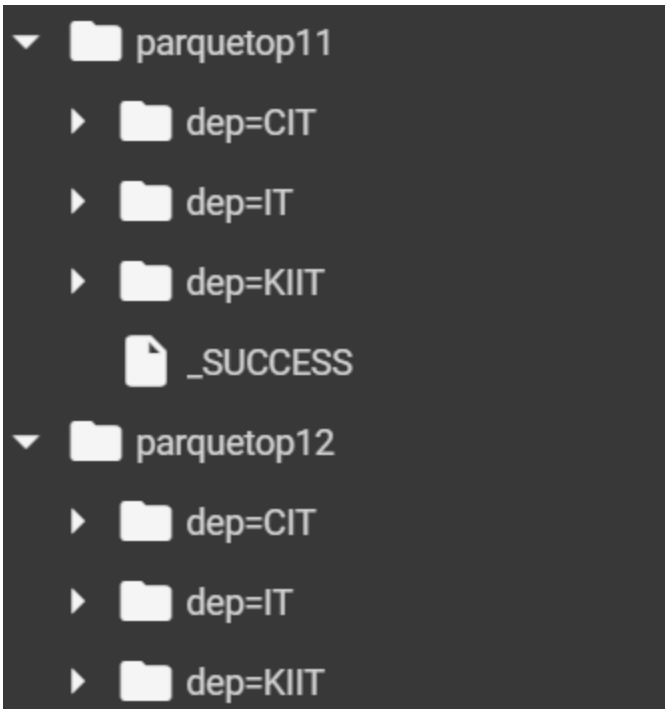
```
king nayeer
kohli wasim
king
nayeer
kohli
wasim
```

44. partitionBy function in PySpark

partitionBy()

- Its used to partition large Dataset into smaller files based on one or multiple columns

```
[122] #44. partitionBy function in PySpark
      data=[(1,'nayeer','male','IT'),(2,'yayeer','female','CIT'),(7,'nareer','male','KIIT')]
      schema=['id','name','gender','dep']

      df=spark.createDataFrame(data,schema)
      df.write.parquet(path='/content/parquetop12',mode='overwrite',partitionBy='dep')
      df.write.parquet(path='/content/parquetop11',mode='overwrite',partitionBy=['dep','gender'])
```



45. from_json() function to convert json string in to MapType in Pyspark

```python
#45. from_json() function to convert json string in to MapType in Pyspark
data=[('nayeer','{"hair":"black","eye":,"brown"}')]
schema=['name','props']
df=spark.createDataFrame(data,schema)
df.show(truncate=False)
df.printSchema()
```

```
+------+------------------------------+
|name  |props                         |
+------+------------------------------+
|nayeer|{"hair":"black","eye":,"brown"}|
+------+------------------------------+

root
 |-- name: string (nullable = true)
 |-- props: string (nullable = true)
```

```python
[136] from pyspark.sql.functions import from_json
      from pyspark.sql.types import MapType, StringType
      MapTypeSchema = MapType(StringType(), StringType())
      #propsMap column with MapType gets generates fropm json string
      df1= df.withColumn('propsMap',from_json(df.props,MapTypeSchema))
      df1.show(truncate=False)
      df1.printSchema()
```

```
+------+------------------------------+------------------------------+
|name  |props                         |propsMap                      |
+------+------------------------------+------------------------------+
|nayeer|{"hair":"black","eye":"brown"}|{hair -> black, eye -> brown}|
+------+------------------------------+------------------------------+

root
 |-- name: string (nullable = true)
 |-- props: string (nullable = true)
 |-- propsMap: map (nullable = true)
 |    |-- key: string
 |    |-- value: string (valueContainsNull = true)
```

```python
#accessing 'eye' key from MapType column 'propsMap'
df2=df1.withColumn('eye',df1.propsMap.eye)
df2.show(truncate=False)
```

```
+------+------------------------------+------------------------------+-----+
|name  |props                         |propsMap                      |eye  |
+------+------------------------------+------------------------------+-----+
|nayeer|{"hair":"black","eye":"brown"}|{hair -> black, eye -> brown}|brown|
+------+------------------------------+------------------------------+-----+
```

46. from_json() function to convert json string into StructType in Pyspark

from_json()

- It's used to convert json string in to MapType or StructType. In this video we discuss about converting it to MapType.

```
#46. from_json() function to convert json string into StructType in Pyspark
data=[('nayeer','{"hair":"black","eye":"brown"}')]
schema=['name','props']
df=spark.createDataFrame(data,schema)
# df.show(truncate=False)
# df.printSchema()


from pyspark.sql.functions import from_json
from pyspark.sql.types import MapType, StringType
structSchema = StructType([\
                           StructField('hair',StringType()),\
                           StructField('eye',StringType()),\
                           ])

MapTypeSchema = MapType(StringType(), StringType())
#propsMap column with MapType gets generates fropm json string
df1= df.withColumn('propsStruct',from_json(df.props,structSchema))
df1.show(truncate=False)
df1.printSchema()

#accessing 'eye' key from MapType column 'propsMap'
df2=df1.withColumn('eye',df1.propsStruct.eye)
df2.show(truncate=False)
```

```
+------+-----------------------------+--------------+
|name  |props                        |propsStruct   |
+------+-----------------------------+--------------+
|nayeer|{"hair":"black","eye":"brown"}|{black, brown}|
+------+-----------------------------+--------------+

root
 |-- name: string (nullable = true)
 |-- props: string (nullable = true)
 |-- propsStruct: struct (nullable = true)
 |    |-- hair: string (nullable = true)
 |    |-- eye: string (nullable = true)

+------+-----------------------------+--------------+-----+
|name  |props                        |propsStruct   |eye  |
+------+-----------------------------+--------------+-----+
|nayeer|{"hair":"black","eye":"brown"}|{black, brown}|brown|
```

47. to_json() function in PySpark

to_json()

- to_json() is used to convert DataFrame column MapType or Struct Type to JSON string.

```
[152] #47. to_json() function in PySpark
      from pyspark.sql.functions import to_json
      from pyspark.sql.types import StructType, StructField , StringType

      data=[('nayeer',{"hair":"black","eye":"brown"})]
      schema=['name','properties']
      df=spark.createDataFrame(data,schema)
      df.show()
      df.printSchema()
      #here 'prop' column will get generated as json string
      df1=df.withColumn('props',to_json(df.properties))
      df1.show(truncate=False)
      df1.printSchema()
```

```
+------+--------------------+
|  name|          properties|
+------+--------------------+
|nayeer|{eye -> brown, ha...|
+------+--------------------+

root
 |-- name: string (nullable = true)
 |-- properties: map (nullable = true)
 |    |-- key: string
 |    |-- value: string (valueContainsNull = true)

+------+----------------------------+----------------------------+
|name  |properties                  |props                       |
+------+----------------------------+----------------------------+
|nayeer|{eye -> brown, hair -> black}|{"eye":"brown","hair":"black"}|
+------+----------------------------+----------------------------+

root
 |-- name: string (nullable = true)
 |-- properties: map (nullable = true)
 |    |-- key: string
 |    |-- value: string (valueContainsNull = true)
 |-- props: string (nullable = true)
```

```
from pyspark.sql.types import StructType, StructField , StringType , StructType
data=[('maheer',{'black','brown'})]
schema=StructType([\
                   StructField('name',StringType()),\
                   StructField('properties',StructType([StructField('hair',StringType()),StructField('eye',StringType())]))\
                   ])

df=spark.createDataFrame(data,schema)
df.show()
df.printSchema()

from pyspark.sql.types import json
#here 'prop' column will get generated as json string
df1=df.withColumn('props',to_json(df.properties))
df1.show(truncate=False)
df1.printSchema()
```

```
+------+--------------+
|  name|    properties|
+------+--------------+
|maheer|{black, brown}|
+------+--------------+

root
 |-- name: string (nullable = true)
 |-- properties: struct (nullable = true)
 |    |-- hair: string (nullable = true)
 |    |-- eye: string (nullable = true)

+------+--------------+-----------------------------+
|name  |properties    |props                        |
+------+--------------+-----------------------------+
|maheer|{black, brown}|{"hair":"black","eye":"brown"}|
+------+--------------+-----------------------------+

root
 |-- name: string (nullable = true)
 |-- properties: struct (nullable = true)
 |    |-- hair: string (nullable = true)
 |    |-- eye: string (nullable = true)
 |-- props: string (nullable = true)
```

48. json_tuple() function in PySpark

json_tuple()

- json_tuple() function is used to query or extract elements from json string column and create as new columns.

```
[160] data=[('nayeer','{"hair":"black","eye":"brown" , "skin":"brown"}'),
           ('golden','{"hair":"golden","eye":"blue" , "skin":"white"}')]
      schema=['name','props']
      df=spark.createDataFrame(data,schema)
      df.show()
      df.printSchema()

      from pyspark.sql.functions import json_tuple
      df2=df.select(df.name, json_tuple(df.props,'hair','skin').alias('hair','skin'))
      df2.show()
```

```
+------+--------------------+
|  name|               props|
+------+--------------------+
|nayeer|{"hair":"black","...|
|golden|{"hair":"golden",...|
+------+--------------------+

root
 |-- name: string (nullable = true)
 |-- props: string (nullable = true)

+------+------+-----+
|  name|  hair| skin|
+------+------+-----+
|nayeer| black|brown|
|golden|golden|white|
+------+------+-----+
```

49. get_ json_object() function in PySpark

get_json_object()

- It's used to extract the JSON string based on path from the JSON column.

```
#49. get_ json_object() function in PySpark
data=[('nayeer','{"address":{"city":"hyd","state":"telegana"},"gender":"male"}'),
      ('kingi','{"address":{"city":"dyh","state":"gana"},"gender":"female"}')]

schema=['name','props']

df=spark.createDataFrame(data,schema)
df.show(truncate=False)
df.printSchema()
from pyspark.sql.functions import get_json_object
df1=df.select('name',get_json_object('props','$.address.city').alias('city'))
df1.show(truncate=False)
df1.printSchema()
```

```
+------+-----------------------------------------------------------------+
|name  |props                                                            |
+------+-----------------------------------------------------------------+
|nayeer|{"address":{"city":"hyd","state":"telegana"},"gender":"male"}|
|kingi |{"address":{"city":"dyh","state":"gana"},"gender":"female"}  |
+------+-----------------------------------------------------------------+

root
 |-- name: string (nullable = true)
 |-- props: string (nullable = true)

+------+----+
|name  |city|
+------+----+
|nayeer|hyd |
|kingi |dyh |
+------+----+

root
 |-- name: string (nullable = true)
 |-- city: string (nullable = true)
```

50. Date functions in PySpark | current_date(), to_date(), date_format() functions

Date Functions

- DateType default fomart is yyyy-MM-dd
- current_date() get the current system date. By default, the data will be returned in yyyy-dd-mm format.
- date_format() to parses the date and converts from yyyy-MM-dd to specified format.
- to_date() converts date string in to datetype. We need to specify format of date in the string in the function.

```
#50. Date functions in PySpark | current_date(), to_date(), date_format() functions
from pyspark.sql.functions import current_date,date_format,lit,to_date

df=spark.range(1)

#datatype default format is yyyy-MM-dd
#gives current date in yyyy-MM-dd format
df.withColumn('todaysDate',current_date()).show()

#converts yyyy-MM-dd datatypes to specified format
df.withColumn('newFormat',date_format(lit('2024-05-14'),'MM.dd.yyyy')).show()

#converts string values of date to DateType
df.withColumn ('newDateCol', to_date(lit('14-05-2024'),'dd-MM-yyyy')).show()
```

```
+---+----------+
| id|todaysDate|
+---+----------+
|  0|2024-05-13|
+---+----------+

+---+----------+
| id| newFormat|
+---+----------+
|  0|05.14.2024|
+---+----------+

+---+----------+
| id|newDateCol|
+---+----------+
|  0|2024-05-14|
+---+----------+
```

51. datediff(), months_between(), add_months(), date_add(), month(), year() functions in PySpark

```
[121] #51. datediff(), months_between(), add_months(), date_add(), month(), year() functions in PySpark
      from pyspark.sql.functions import datediff , months_between, add_months, date_add, year , month
      df=spark.createDataFrame([('2015-04-08','2015-05-08')],['d1','d2'])
      df.withColumn('diff',datediff(df.d2,df.d1)).show()
      df.withColumn('months_between',months_between(df.d2,df.d1)).show()
      df.withColumn('add_months',add_months(df.d2,4)).show()
      df.withColumn('submonth',add_months(df.d2,-4)).show()
```

```
+----------+----------+----+
|        d1|        d2|diff|
+----------+----------+----+
|2015-04-08|2015-05-08|  30|
+----------+----------+----+

+----------+----------+--------------+
|        d1|        d2|months_between|
+----------+----------+--------------+
|2015-04-08|2015-05-08|           1.0|
+----------+----------+--------------+

+----------+----------+----------+
|        d1|        d2|add_months|
+----------+----------+----------+
|2015-04-08|2015-05-08|2015-09-08|
+----------+----------+----------+

+----------+----------+----------+
|        d1|        d2|  submonth|
+----------+----------+----------+
|2015-04-08|2015-05-08|2015-01-08|
+----------+----------+----------+
```

```
df.withColumn('addDate',date_add(df.d2,4)).show()
df.withColumn('subDate',date_add(df.d2,-4)).show()
df.withColumn('year',year(df.d2)).show()
df.withColumn('month',month(df.d2)).show()
```

```
+----------+----------+----------+
|        d1|        d2|   addDate|
+----------+----------+----------+
|2015-04-08|2015-05-08|2015-05-12|
+----------+----------+----------+

+----------+----------+----------+
|        d1|        d2|   subDate|
+----------+----------+----------+
|2015-04-08|2015-05-08|2015-05-04|
+----------+----------+----------+

+----------+----------+----+
|        d1|        d2|year|
+----------+----------+----+
|2015-04-08|2015-05-08|2015|
+----------+----------+----+

+----------+----------+-----+
|        d1|        d2|month|
+----------+----------+-----+
|2015-04-08|2015-05-08|    5|
+----------+----------+-----+
```

52. Timestamp Functions in PySpark

Timestamp Functions

- TimestampType default fomart is yyyy-MM-dd HH:mm:ss.SS
- current_timestamp() get the current timestamp. By default, the data will be returned in default format.
- to_timestamp() converts timestamp string in to TimestampType. We need to specify format of timestamp in the string in the function.
- hour(), minute(), second() functions

```
[124] #52. Timestamp Functions in PySpark
      from pyspark.sql.functions import current_timestamp , to_timestamp , lit , hour , minute , second
      df=spark.range(2)
      df.show()
      df1=df.withColumn('timestamp',current_timestamp())
      df1.show(truncate=False)
      df1.printSchema()
      df2=df1.withColumn('toTimestamp',to_timestamp(lit('25.12.2022 06.10.13'), 'dd.MM.yyyy HH.mm.ss'))
      df2.show(truncate=False)
      df2.printSchema()
      df2.select('id',hour(current_timestamp()).alias('hour'),\
                 minute(current_timestamp()).alias('minute'),\
                 second(current_timestamp()).alias('second')).show()
```

```
+---+
| id|
+---+
|  0|
|  1|
+---+

+---+--------------------------+
|id |timestamp                 |
+---+--------------------------+
|0  |2024-05-14 08:47:58.716124|
|1  |2024-05-14 08:47:58.716124|
+---+--------------------------+

root
 |-- id: long (nullable = false)
 |-- timestamp: timestamp (nullable = false)

+---+--------------------------+-------------------+
|id |timestamp                 |toTimestamp        |
+---+--------------------------+-------------------+
|0  |2024-05-14 08:47:58.82591|2022-12-25 06:10:13|
|1  |2024-05-14 08:47:58.82591|2022-12-25 06:10:13|
+---+--------------------------+-------------------+

root
 |-- id: long (nullable = false)
 |-- timestamp: timestamp (nullable = false)
 |-- toTimestamp: timestamp (nullable = true)

+---+----+------+------+
| id|hour|minute|second|
+---+----+------+------+
|  0|   8|    47|    59|
|  1|   8|    47|    59|
+---+----+------+------+
```

53. approx_count_distinct(), avg(), collect_list(), collect_set(), countDistinct(), count()

Aggregate Functions

- Aggregate functions operate on a group of rows and calculate a single return value for every group.

approx_count_distinct() – returns the count of distinct items in a group of rows

avg() – returns average of values in a group of rows

collect_list() – returns all values from input column as list with duplicates.

collect_set() – returns all values from input column as list without duplicates.

countDistinct() – returns number of distinct elements in input column.

count() – returns number of elements in a column.

```
[130] #53. approx_count_distinct(), avg(), collect_list(), collect_set(), countDistinct(), count()
      from pyspark.sql.functions import approx_count_distinct, avg, collect_list , collect_set , countDistinct, count

      simpleData = [('Payeer','HR',4500),\
                    ('Fayeer','SDE',1300),\
                    ('Yayeer','Manager',4500)]

      schema= ["employee_name",'dep','salary']
      df=spark.createDataFrame(data=simpleData,schema=schema)
      df.printSchema()
      df.show(truncate = False)
      df.select(approx_count_distinct('salary')).show()
      df.select(avg('salary')).show()
      df.select(collect_list('salary')).show()
      df.select(collect_set('salary')).show()
      df.select(countDistinct('salary')).show()
      df.select(count('salary')).show()
```

```
root
 |-- employee_name: string (nullable = true)
 |-- dep: string (nullable = true)
 |-- salary: long (nullable = true)

+-------------+-------+------+
|employee_name|dep    |salary|
+-------------+-------+------+
|Payeer       |HR     |4500  |
|Fayeer       |SDE    |1300  |
|Yayeer       |Manager|4500  |
+-------------+-------+------+

+----------------------------+
|approx_count_distinct(salary)|
+----------------------------+
|                           2|
+----------------------------+

+------------------+
|       avg(salary)|
+------------------+
|3433.3333333333335|
+------------------+

+--------------------+
|collect_list(salary)|
+--------------------+
|  [4500, 1300, 4500]|
+--------------------+

+------------------+
|collect_set(salary)|
+------------------+
|       [4500, 1300]|
+------------------+

+----------------------+
|count(DISTINCT salary)|
+----------------------+
|                     2|
+----------------------+

+-------------+
|count(salary)|
+-------------+
|            3|
+-------------+
```

54. row_number(), rank(), dense_rank() functions in PySpark

Ranking Functions

- we need to partition the data using Window.partitionBy() , and for row number and rank function we need to additionally order by on partition data using orderBy clause.
- row_number() window function is used to give the sequential row number starting from 1 to the result of each window partition
- rank() window function is used to provide a rank to the result within a window partition. This function leaves gaps in rank when there are ties.
- dense_rank() window function is used to get the result with rank of rows within a window partition without any gaps. This is similar to rank() function difference being rank function leaves gaps in rank when there are ties.

Row _number()

```
[142] #54. row_number(), rank(), dense_rank() functions in PySpark
      from pyspark.sql.functions import row_number , rank , dense_rank
      from pyspark.sql.window import Window
      data=[(1,'naheey','M',5000,'IT'),\
            (2,'baheey','F',50100,'IT'),\
            (3,'gaheey','M',52000,'IT'),\
            (4,'taheey','F',53000,'GIT'),\
            (5,'raheey','M',25000,'GIT'),\
            (6,'eaheey','F',15000,'KIT'),\
            (7,'waheey','M',95000,'KIT')]

      schema=['id','name','gender','salary','dep']
      df=spark.createDataFrame(data,schema)

      # df.sort('dep').show()

      window=Window.partitionBy('dep').orderBy('salary')
      df.withColumn('rowNumber',row_number().over(window)).show()
```

```
+---+------+------+------+---+---------+
| id|  name|gender|salary|dep|rowNumber|
+---+------+------+------+---+---------+
|  5|raheey|     M| 25000|GIT|        1|
|  4|taheey|     F| 53000|GIT|        2|
|  1|naheey|     M|  5000| IT|        1|
|  2|baheey|     F| 50100| IT|        2|
|  3|gaheey|     M| 52000| IT|        3|
|  6|eaheey|     F| 15000|KIT|        1|
|  7|waheey|     M| 95000|KIT|        2|
+---+------+------+------+---+---------+
```

Rank

```
df.withColumn('rank',rank().over(window)).show()
```

```
+---+------+------+------+---+----+
| id|  name|gender|salary|dep|rank|
+---+------+------+------+---+----+
|  5|raheey|     M| 25000|GIT|   1|
|  4|taheey|     F| 53000|GIT|   2|
|  1|naheey|     M|  5000| IT|   1|
|  2|baheey|     F| 50100| IT|   2|
|  3|gaheey|     M| 52000| IT|   3|
|  6|eaheey|     F| 15000|KIT|   1|
|  7|waheey|     M| 95000|KIT|   2|
+---+------+------+------+---+----+
```

denserRank

```
[141] df.withColumn('denserrank',dense_rank().over(window)).show()
```

```
+---+------+------+------+---+----------+
| id|  name|gender|salary|dep|denserrank|
+---+------+------+------+---+----------+
|  5|raheey|     M| 25000|GIT|         1|
|  4|taheey|     F| 53000|GIT|         2|
|  1|naheey|     M|  5000| IT|         1|
|  2|baheey|     F| 50100| IT|         2|
|  3|gaheey|     M| 52000| IT|         3|
|  6|eaheey|     F| 15000|KIT|         1|
|  7|waheey|     M| 95000|KIT|         2|
+---+------+------+------+---+----------+
```