**SparkCompy**

SparkCompy's SparkCompare class will join two dataframes on a list of join columns. It has the capability to map column names that may be different in each dataframe, including in the join columns. You are responsible for creating the dataframes from any source which Spark can handle and specifying a unique join key. There should be no duplicates in the two dataframes. Comparisons will be attempted even if dtypes don't match.

If your join keys have the same names in both the dataframes, then you may just pass the column name as a string item in the list. If the join keys have different names, then you will have to pass a tuple of string as an item in the list.

The SparkCompare class will compare columns that have the same name in both the dataframes. If the corresponding name is not found, that column is ignored. You can provide a column\_mapping for the ones that you want to compare when the names do not match.

Calling the report method will generate a report.xlsx in the same directory by default. You can override this by passing in the absolute/relative path with the file name. Note that the report.xlsx only contains differences for the keys that are common between the two dataframes. If you want to know which keys only exists in either of the dataframes then call the property method **get\_base\_only\_rows** & **get\_compare\_only\_rows.** Each will return a dataframe that you can use to do a collect or a write operation.

**Performance Implications**

Spark scales incredibly well, so you can use SparkCompare to compare billions of rows of data, provided you spin up a big enough cluster. Still, joining billions of rows of data is an inherently large task, so there are a couple of things you may want to take into consideration when getting into the cliched realm of "big data":

* SparkCompare will compare all columns in common in the dataframes. If there are columns in the data that you don't care to compare, pass a list of columns to the ignore\_columns argument while instantiating the SparkCompare class. You could also just select the columns you want to compare while creating your dataframes.
* For large datasets, adding cache\_intermediates=True to the SparkCompare call can help optimize performance by caching certain intermediate dataframes in memory, like an intermediate dataframe or the joined dataframe. Otherwise, Spark's lazy evaluation will recompute those each time it needs the data in a report or as you access instance attributes. This may be fine for smaller dataframes, but will be costly for larger ones. You do need to ensure that you have enough free cache memory before you do this, so this parameter is set to False by default.

**Basic Usage**

import datetime

import sparkcompy

from pyspark.sql import Row

# This example assumes you have a SparkSession named "spark" in your environment, as you do when running `pyspark` from the terminal for instance

data1 = [

Row(acct\_id=10000001234, dollar\_amt=123.45, name='George Maharis', float\_fld=14530.1555,

date\_fld=datetime.date(2017, 1, 1)),

Row(acct\_id=10000001235, dollar\_amt=0.45, name='Michael Bluth', float\_fld=1.0,

date\_fld=datetime.date(2017, 1, 1)),

Row(acct\_id=10000001236, dollar\_amt=1345.0, name='George Bluth', float\_fld=None,

date\_fld=datetime.date(2017, 1, 1)),

Row(acct\_id=10000001237, dollar\_amt=123456.0, name='Bob Loblaw', float\_fld=345.12,

date\_fld=datetime.date(2017, 1, 1)),

Row(acct\_id=10000001239, dollar\_amt=1.05, name='Lucille Bluth', float\_fld=None,

date\_fld=datetime.date(2017, 1, 1))

]

data2 = [

Row(ACC\_IDENTIFIER=10000001234, dollar\_amt=123.4, full\_name='George Michael Bluth', float\_fld=14530.155),

Row(ACC\_IDENTIFIER=10000001235, dollar\_amt=0.45, full\_name='Michael Bluth', float\_fld=None),

Row(ACC\_IDENTIFIER=10000001236, dollar\_amt=1345.0, full\_name='George Bluth', float\_fld=1.0),

Row(ACC\_IDENTIFIER=10000001237, dollar\_amt=123456.0, full\_name='Robert Loblaw', float\_fld=345.12),

Row(ACC\_IDENTIFIER=10000001238, dollar\_amt=1.05, full\_name='Loose Seal Bluth', float\_fld=111.0)

]

base\_df = spark.createDataFrame(data1)

compare\_df = spark.createDataFrame(data2)

comparison = sparkcompy.SparkCompare(spark, base\_df, compare\_df, join\_columns=[('acct\_id', 'ACC\_IDENTIFIER')], column\_mapping=[('name', 'full\_name')])

# If the join key was the same in both the dfs then you could just pass ['acct\_id']. Same goes for column\_mapping or ignore\_columns arguments

# This writes the differences to an excel file called report.xlsx in the current directory by default. You may override this behavior by passing a path string.

comparison.report()

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **acct\_id** | **base df column** | **base df value** | **compare df column** | **compare df value** |
| 10000001235 | float\_fld | 1 | float\_fld |  |
| 10000001234 | dollar\_amt | 123.45 | dollar\_amt | 123.4 |
| 10000001234 | float\_fld | 14530.1555 | float\_fld | 14530.155 |
| 10000001234 | name | George Maharis | full\_name | George Michael Bluth |
| 10000001237 | name | Bob Loblaw | full\_name | Robert Loblaw |
| 10000001236 | float\_fld |  | float\_fld | 1 |
|  |  |  |  |  |

**Tips for efficiency**Use parquet tables or files to create dataframes to take advantage of spark’s optimizations for parquet files.   
Incaseyou want to review some of the internal objects or speed up subsequent operations set the cacheIntermediates flag to True.  
If you are anticipating massive differences then you’d have to make sure the driver has enough memory for the collect operation. Also try increasing the maxResultSize.