pySparkPrac

```
schema = ['name','age','gender','occupation']
data = [("John", 25, "Male", "Engineer"),
        ("Jane", 30, "Female", "Doctor"),
        ("Bob", 45, "Male", "Lawyer"),
        ("sam", 30, "Male", "Teacher"),
        ("alexa", 20, "Female", "Lawyer")]

df1 = spark.createDataFrame(data, schema=schema)
df1.show()
```

```
+-----+---+------+----------+
| name|age|gender|occupation|
+-----+---+------+----------+
| John| 25|  Male|  Engineer|
| Jane| 30|Female|    Doctor|
|  Bob| 45|  Male|    Lawyer|
|  sam| 30|  Male|   Teacher|
|alexa| 20|Female|    Lawyer|
+-----+---+------+----------+
```

```
col = ["name", "age","gender","city","country"]


data=[
('John', 25,    'Male', 'New York', 'USA'),
('Emily', 30,   'Female', 'London', 'UK'),
('Michael', 40, 'Male', 'Sydney', 'Australia'),
('Anna', 28,    'Female', 'Paris', 'France'),
('David', 35,   'Male', 'Toronto', 'Canada')]
df2=spark.createDataFrame(data=data,schema=col)
df2.show()
```

```
+-------+---+------+--------+---------+
|   name|age|gender|    city|  country|
+-------+---+------+--------+---------+
|   John| 25|  Male|New York|      USA|
|  Emily| 30|Female|  London|       UK|
|Michael| 40|  Male|  Sydney|Australia|
|   Anna| 28|Female|   Paris|   France|
|  David| 35|  Male| Toronto|   Canada|
+-------+---+------+--------+---------+
```

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

```
from pyspark.sql.types import StructType, StructField, IntegerType, StringType
from pyspark.sql import SparkSession
import random
from pyspark.sql.functions import *
from pyspark.sql import functions as f
```

```
def insertrecords(x):
    x=random.randint(1,20)
    return x
udf1=udf(lambda x: insertrecords(x),IntegerType())
```

```
df1idc=df1.withColumn("id",udf1(col("occupation")))
dfid.show()
```

```
+-------+---+------+--------+---------+---+
|   name|age|gender|    city|  country| id|
+-------+---+------+--------+---------+---+
|   John| 25|  Male|New York|      USA|  4|
|  Emily| 30|Female|  London|       UK|  6|
|Michael| 40|  Male|  Sydney|Australia|  7|
|   Anna| 28|Female|   Paris|   France|  6|
|  David| 35|  Male| Toronto|   Canada|  2|
+-------+---+------+--------+---------+---+
```

```
dfid.withColumnRenamed("id","customer_id").show()
```

```
+-------+---+------+--------+---------+-----------+
|   name|age|gender|    city|  country|customer_id|
+-------+---+------+--------+---------+-----------+
|   John| 25|  Male|New York|      USA|          2|
|  Emily| 30|Female|  London|       UK|          1|
|Michael| 40|  Male|  Sydney|Australia|          9|
|   Anna| 28|Female|   Paris|   France|          1|
|  David| 35|  Male| Toronto|   Canada|          2|
+-------+---+------+--------+---------+-----------+
```

```
dfid.withColumnRenamed("id","customer_id").drop("city").show()
```

```
+-------+---+------+---------+-----------+
|   name|age|gender|  country|customer_id|
+-------+---+------+---------+-----------+
|   John| 25|  Male|      USA|          6|
|  Emily| 30|Female|       UK|          9|
|Michael| 40|  Male|Australia|          6|
|   Anna| 28|Female|   France|          5|
|  David| 35|  Male|   Canada|          3|
+-------+---+------+---------+-----------+
```

```
df1id=df1idc
df1id.show()
```

```
+-----+---+------+----------+---+
| name|age|gender|occupation| id|
+-----+---+------+----------+---+
| John| 25|  Male|  Engineer|  8|
| Jane| 30|Female|    Doctor|  3|
|  Bob| 45|  Male|    Lawyer|  2|
|  sam| 30|  Male|   Teacher|  4|
|alexa| 20|Female|    Lawyer|  8|
+-----+---+------+----------+---+
```

```
df2id=df2.withColumn("id",udf1(col("country")))
dfid.show()
```

```
+-------+---+------+--------+---------+---+
|   name|age|gender|    city|  country| id|
+-------+---+------+--------+---------+---+
|   John| 25|  Male|New York|      USA|  3|
|  Emily| 30|Female|  London|       UK|  3|
|Michael| 40|  Male|  Sydney|Australia|  9|
|   Anna| 28|Female|   Paris|   France|  3|
|  David| 35|  Male| Toronto|   Canada|  3|
+-------+---+------+--------+---------+---+
```

```
dfleftjoin=df1id.join(df2id, df1id.id==df2id.id,'left')
dfleftjoin.show()
dfleftjoin1=df1id.join(df2id,["id"],'left') # no repeated key column
dfleftjoin1.show()
```

```
+-----+---+------+----------+---+-------+----+------+--------+---------+----+
| name|age|gender|occupation| id|   name| age|gender|    city|  country|  id|
+-----+---+------+----------+---+-------+----+------+--------+---------+----+
|  Bob| 45|  Male|    Lawyer|  6|   null|null|  null|    null|     null|null|
|alexa| 20|Female|    Lawyer|  6|   null|null|  null|    null|     null|null|
| John| 25|  Male|  Engineer|  9|   null|null|  null|    null|     null|null|
| Jane| 30|Female|    Doctor|  9|   null|null|  null|    null|     null|null|
|  sam| 30|  Male|   Teacher|  2|  David|  35|  Male| Toronto|   Canada|   2|
|  sam| 30|  Male|   Teacher|  2|   John|  25|  Male|New York|      USA|   2|
|  sam| 30|  Male|   Teacher|  2|Michael|  40|  Male|  Sydney|Australia|   2|
+-----+---+------+----------+---+-------+----+------+--------+---------+----+
+---+-----+---+------+----------+-------+----+------+--------+---------+
| id| name|age|gender|occupation|   name| age|gender|    city|  country|
+---+-----+---+------+----------+-------+----+------+--------+---------+
|  3|alexa| 20|Female|    Lawyer|   null|null|  null|    null|     null|
|  5|  Bob| 45|  Male|    Lawyer|   null|null|  null|    null|     null|
|  5|  sam| 30|  Male|   Teacher|   null|null|  null|    null|     null|
|  9| John| 25|  Male|  Engineer|  David|  35|  Male| Toronto|   Canada|
```

```
dfrightjoin=df1id.join(df2id,df1id.id==df2id.id,'right')
dfrightjoin.show()
```

```
+----+----+------+----------+----+-------+---+------+--------+---------+---+
|name| age|gender|occupation|  id|   name|age|gender|    city|  country| id|
+----+----+------+----------+----+-------+---+------+--------+---------+---+
|null|null|  null|      null|null|   John| 25|  Male|New York|      USA|  1|
|null|null|  null|      null|null|Michael| 40|  Male|  Sydney|Australia|  1|
| Bob|  45|  Male|    Lawyer|   9|  David| 35|  Male| Toronto|   Canada|  9|
| sam|  30|  Male|   Teacher|   9|  David| 35|  Male| Toronto|   Canada|  9|
|John|  25|  Male|  Engineer|   2|  Emily| 30|Female|  London|       UK|  2|
|Jane|  30|Female|    Doctor|   2|  Emily| 30|Female|  London|       UK|  2|
|John|  25|  Male|  Engineer|   2|   Anna| 28|Female|   Paris|   France|  2|
|Jane|  30|Female|    Doctor|   2|   Anna| 28|Female|   Paris|   France|  2|
+----+----+------+----------+----+-------+---+------+--------+---------+---+
```

```
dfinnerjoin=df1id.join(df2id,df1id.id==df2id.id,'inner')
dfinnerjoin.show()
```

```
+----+---+------+----------+---+-------+---+------+--------+---------+---+
|name|age|gender|occupation| id|   name|age|gender|    city|  country| id|
+----+---+------+----------+---+-------+---+------+--------+---------+---+
|John| 25|  Male|  Engineer|  2|   John| 25|  Male|New York|      USA|  2|
|John| 25|  Male|  Engineer|  2|Michael| 40|  Male|  Sydney|Australia|  2|
|John| 25|  Male|  Engineer|  2|  David| 35|  Male| Toronto|   Canada|  2|
+----+---+------+----------+---+-------+---+------+--------+---------+---+
```

```
dfouterjoin=df1id.join(df2id,df1id.id==df2id.id,"outer")
dfouterjoin.show()
```

```
+-----+----+------+----------+----+-------+----+------+--------+---------+----+
| name| age|gender|occupation|  id|   name| age|gender|    city|  country|  id|
+-----+----+------+----------+----+-------+----+------+--------+---------+----+
| null|null|  null|      null|null|  Emily|  30|Female|  London|       UK|   1|
| null|null|  null|      null|null|   Anna|  28|Female|   Paris|   France|   1|
|  sam|  30|  Male|   Teacher|   6|   null|null|  null|    null|     null|null|
| John|  25|  Male|  Engineer|   9|   null|null|  null|    null|     null|null|
|  Bob|  45|  Male|    Lawyer|   9|   null|null|  null|    null|     null|null|
| Jane|  30|Female|    Doctor|   2|   John|  25|  Male|New York|      USA|   2|
| Jane|  30|Female|    Doctor|   2|Michael|  40|  Male|  Sydney|Australia|   2|
| Jane|  30|Female|    Doctor|   2|  David|  35|  Male| Toronto|   Canada|   2|
|alexa|  20|Female|    Lawyer|   2|   John|  25|  Male|New York|      USA|   2|
|alexa|  20|Female|    Lawyer|   2|Michael|  40|  Male|  Sydney|Australia|   2|
|alexa|  20|Female|    Lawyer|   2|  David|  35|  Male| Toronto|   Canada|   2|
+-----+----+------+----------+----+-------+----+------+--------+---------+----+
```

```
df1id.show()
```

```
+-----+---+------+----------+---+
| name|age|gender|occupation| id|
+-----+---+------+----------+---+
| John| 25|  Male|  Engineer|  1|
| Jane| 30|Female|    Doctor|  2|
|  Bob| 45|  Male|    Lawyer|  2|
|  sam| 30|  Male|   Teacher|  2|
|alexa| 20|Female|    Lawyer|  9|
+-----+---+------+----------+---+
```

narrow transformation

```
# this simply means dependencies on another dataframe if its narrow.

##Any transformation for which a single output partition can be calculated from only one input partition is a narrow transformation.

# For example filter() and contains() operations can produce output partition from a single input partition without needing any data exchange across the executors. Therefore, they are called narrow transformations.

### comp----However, transformations like groupBy() and orderBy() need data from multiple partitions and force data shuffle from each of the executors across the cluster before producing the output partition.

#Narrow transformations are the result of map() and filter() functions and these compute data that live on a single partition meaning there will not be any data movement between partitions to execute narrow transformations.


# Functions such as map(), mapPartition(), flatMap(), filter(), union() are some examples of narrow transformation
```

Wider Transformation

```
# it needs data from varios partitions, shuffle and store them in a new transformation

#Wider transformations are the result of groupByKey() and reduceByKey() functions and these compute data that live on many partitions meaning there will be data movements between partitions to execute wider transformations. Since these shuffles the data, they also called shuffle transformations.

# Functions such as groupByKey(), aggregateByKey(), aggregate(), join(), repartition()
```

pySparkPrac

difference between reducebykey and reduce

```
#reduceByKey and reduce are both transformations in Apache Spark that operate on RDDs (Resilient Distributed Datasets) and perform aggregation operations. However, they have some important differences.

##reduce takes an RDD and applies a binary operator to the elements in it, resulting in a single aggregated value. This is similar to the reduce operation in functional programming, and can be used for operations such as finding the sum or maximum of an RDD.

##reduceByKey operates on RDDs of key-value pairs, where the values are aggregated based on their corresponding keys. It applies a binary operator to values that have the same key, resulting in a new RDD where each key is associated with a single, aggregated value. This transformation is commonly used in word count applications, where you want to count the occurrences of each word in a document.

###In summary, reduce operates on the entire RDD, while reduceByKey operates on key-value pairs and groups values by their keys.
```

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

```
"""
REDUCE                                                                      REDUCEBYKEY
Action                                                                      Transformation
reduce must pull the entire dataset down into                          reduceByKey reduces one value for each key
a single location because it is reducing to one
final value

reduce() is a function that operates on an RDD of objects.        reduceByKey() is a function that operates on an RDD of key-value pairs

reduce() function is a member of RDD[T] class                     reduceByKey() is a member of the PairRDDFunctions[K, V] class

Output is a collection not an RDD and is is not added to DAG .    Output is an RDD and is added to DAG
                                                                  The reduce cannot result in an RDD simply because it is a single value as output.

def reduce(f: (T, T) => T): T                                     def reduceByKey(func: (V, V) ⇒ V): RDD[(K, V)]

reduce is an action which Aggregate the elements of the dataset   reduceByKey When called on a dataset of (K, V) pairs, returns a dataset of (K, V) pairs where the values
                                                  for each key are aggregated using the given reduce function func, which must be of type (V,V) => V.
using a function func (which takes two arguments and returns one)
"""
```

```
#--https://sparkbyexamples.com/spark/spark-rdd-reduce-function-example/

data = [('Project', 1),
('Gutenbergs', 1),
('Alices', 1),
('Adventures', 1),
('in', 1),
('Wonderland', 1),
('Project', 1),
('Gutenbergs', 1),
('Adventures', 1),
('in', 1),
('Wonderland', 1),
('Project', 1),
('Gutenbergs', 1)]

rdd=spark.sparkContext.parallelize(data)
```

```
reduceByKey(func, numPartitions=None, partitionFunc=)
```

```
print(rdd.collect())

rdd2=rdd.reduceByKey(lambda a,b: a+b)
for element in rdd2.collect():
    print(element)
```

```
[('Project', 1), ('Gutenbergs', 1), ('Alices', 1), ('Adventures', 1), ('in', 1), ('Wonderland', 1), ('Project', 1), ('Gutenbergs', 1), ('Adventures', 1), ('in', 1), ('Wonderland', 1), ('Proje
ct', 1), ('Gutenbergs', 1)]
('Project', 3)
('Gutenbergs', 3)
('Wonderland', 2)
('Adventures', 2)
('Alices', 1)
('in', 2)
```

Spark RDD repartition() vs coalesce()

```
'''repartion() is for increase or decrease the RDD

and coalesce() is used for deceasing the partitions in efficient way

https://sparkbyexamples.com/spark/spark-repartition-vs-coalesce/'''
```