Transformations in RDD:

Using `map()

rdd = spark.sparkContext.parallelize([1, 2, 3, 4])

squared\_rdd = rdd.map(lambda x: x 2)

print(squared\_rdd.collect())

Using `flatMap()`

rdd = spark.sparkContext.parallelize(["Hello Spark", "Learning RDD"])

words\_rdd = rdd.flatMap(lambda line: line.split())

print(words\_rdd.collect())

Using `filter()`

rdd = spark.sparkContext.parallelize([1, 2, 3, 4, 5])

filtered\_rdd = rdd.filter(lambda x: x % 2 == 0)

print(filtered\_rdd.collect())

Using `distinct()`:

rdd = spark.sparkContext.parallelize([1, 2, 2, 3, 3, 4])

distinct\_rdd = rdd.distinct()

print(distinct\_rdd.collect())

Using `groupBy()`:

#Grouping by even and odd

rdd = spark.sparkContext.parallelize([1, 2, 3, 4, 5])

grouped\_rdd = rdd.groupBy(lambda x: x % 2)

print({k: list(v) for k, v in grouped\_rdd.collect()})

Using `union()`:

rdd1 = spark.sparkContext.parallelize([1, 2])

rdd2 = spark.sparkContext.parallelize([3, 4])

combined\_rdd = rdd1.union(rdd2)

print(combined\_rdd.collect()) (Combines two RDDs into one. )

Using intersection()

rdd1 = spark.sparkContext.parallelize([1, 2, 3])

rdd2 = spark.sparkContext.parallelize([3, 4, 5])

intersection\_rdd = rdd1.intersection(rdd2)

print(intersection\_rdd.collect())

Using subtract()

rdd1 = spark.sparkContext.parallelize([1, 2, 3])

rdd2 = spark.sparkContext.parallelize([2, 3, 4])

subtracted\_rdd = rdd1.subtract(rdd2)

print(subtracted\_rdd.collect())

Using cartesian()

rdd1 = spark.sparkContext.parallelize([1, 2])

rdd2 = spark.sparkContext.parallelize([3, 4])

cartesian\_rdd = rdd1.cartesian(rdd2)

print(cartesian\_rdd.collect())

Using groupBy()

rdd = spark.sparkContext.parallelize([1, 2, 3, 4, 5])

grouped\_rdd = rdd.groupBy(lambda x: x % 2)

print({k: list(v) for k, v in grouped\_rdd.collect()})

Using reduceByKey()

pair\_rdd = spark.sparkContext.parallelize([('A', 1), ('B', 1), ('A', 2)])

reduced\_rdd = pair\_rdd.reduceByKey(lambda a, b: a + b)

print(reduced\_rdd.collect())

Using groupByKey()

pair\_rdd = spark.sparkContext.parallelize([('A', 1), ('A', 2), ('B', 1)])

grouped\_rdd = pair\_rdd.groupByKey()

print({k: list(v) for k, v in grouped\_rdd.collect()})

Using sortBy()

rdd = spark.sparkContext.parallelize([1, 3, 2, 4])

sorted\_rdd = rdd.sortBy(lambda x: x, ascending=False)

print(sorted\_rdd.collect())

Using sortByKey()

pair\_rdd = spark.sparkContext.parallelize([('A', 1), ('B', 2), ('A', 3)])

sorted\_rdd = pair\_rdd.sortByKey()

print(sorted\_rdd.collect())

Using coalesce()

rdd = spark.sparkContext.parallelize([1, 2, 3, 4, 5], 5) Initial 5 partitions

coalesced\_rdd = rdd.coalesce(2) Reduce to 2 partitions

print(coalesced\_rdd.getNumPartitions())

Using repartition()

rdd = spark.sparkContext.parallelize([1, 2, 3, 4], 2)

repartitioned\_rdd = rdd.repartition(4)

print(repartitioned\_rdd.getNumPartitions())

Using sample()

rdd = spark.sparkContext.parallelize([1, 2, 3, 4, 5])

sampled\_rdd = rdd.sample(False, 0.5, seed=42) 50% sample without replacement

print(sampled\_rdd.collect())

Using join()

rdd1 = spark.sparkContext.parallelize([('A', 1), ('B', 2)])

rdd2 = spark.sparkContext.parallelize([('A', 3), ('B', 4)])

joined\_rdd = rdd1.join(rdd2)

print(joined\_rdd.collect())

Using cogroup()

rdd1 = spark.sparkContext.parallelize([('A', 1), ('B', 2)])

rdd2 = spark.sparkContext.parallelize([('A', 3), ('B', 4)])

cogrouped\_rdd = rdd1.cogroup(rdd2)

print({k: (list(v[0]), list(v[1])) for k, v in cogrouped\_rdd.collect()})

Using partitionBy()

dd = spark.sparkContext.parallelize([('A', 1), ('B', 2), ('A', 3)], 2)

partitioned\_rdd = rdd.partitionBy(3)

print(partitioned\_rdd.getNumPartitions())

Using keyBy()

rdd = spark.sparkContext.parallelize([1, 2, 3, 4])

keyed\_rdd = rdd.keyBy(lambda x: x % 2) Key by even or odd

print(keyed\_rdd.collect())

Actions in RDD:

reduce()

rdd = spark.sparkContext.parallelize([1, 2, 3, 4])

result = rdd.reduce(lambda a, b: a + b)

print(result)

countByKey()

pair\_rdd = spark.sparkContext.parallelize([('A', 1), ('B', 1), ('A', 1)])

result = pair\_rdd.countByKey()

print(result)

take()

rdd = spark.sparkContext.parallelize([5, 4, 3, 2, 1])

print(rdd.take(3))

collectAsMap()

pair\_rdd = spark.sparkContext.parallelize([('A', 1), ('B', 2), ('A', 3)])

result = pair\_rdd.collectAsMap()

print(result)

foreach()

rdd = spark.sparkContext.parallelize([1, 2, 3])

rdd.foreach(lambda x: print(x)) collect()

rdd = spark.sparkContext.parallelize([1, 2, 3, 4])

print(rdd.collect())

count()

rdd = spark.sparkContext.parallelize([1, 2, 3, 4])

print(rdd.count())

first()

rdd = spark.sparkContext.parallelize([1, 2, 3, 4])

print(rdd.first())

take(n)

rdd = spark.sparkContext.parallelize([1, 2, 3, 4])

print(rdd.take(2))

takeSample(withReplacement, num, [seed])

rdd = spark.sparkContext.parallelize([1, 2, 3, 4])

print(rdd.takeSample(False, 2))

takeOrdered(n, [key])

rdd = spark.sparkContext.parallelize([3, 1, 2, 4])

print(rdd.takeOrdered(2))

top(n, [key])

rdd = spark.sparkContext.parallelize([3, 1, 2, 4])

print(rdd.top(2))

reduce(func)

rdd = spark.sparkContext.parallelize([1, 2, 3, 4])

print(rdd.reduce(lambda x, y: x + y))

fold(zeroValue, func)

rdd = spark.sparkContext.parallelize([1, 2, 3, 4])

print(rdd.fold(0, lambda x, y: x + y))

aggregate(zeroValue, seqOp, combOp)

rdd = spark.sparkContext.parallelize([1, 2, 3, 4])

seqOp = (lambda x, y: (x[0] + y, x[1] + 1))

combOp = (lambda x, y: (x[0] + y[0], x[1] + y[1]))

result = rdd.aggregate((0, 0), seqOp, combOp)

average = result[0] / result[1]

print(average)

foreach(func)

rdd = spark.sparkContext.parallelize([1, 2, 3, 4])

rdd.foreach(lambda x: print(x))

countByKey()

rdd = spark.sparkContext.parallelize([('A', 1), ('B', 1), ('A', 2)])

print(rdd.countByKey())

saveAsTextFile(path)

rdd = spark.sparkContext.parallelize([1, 2, 3, 4])

rdd.saveAsTextFile("/path/to/output.txt")

saveAsSequenceFile(path)

pairRDD = spark.sparkContext.parallelize([("A", 1), ("B", 2)])

pairRDD.saveAsSequenceFile("/path/to/output")

saveAsObjectFile(path)

rdd = spark.sparkContext.parallelize([1, 2, 3, 4])

rdd.saveAsObjectFile("/path/to/output")

countByValue()

rdd = spark.sparkContext.parallelize([1, 2, 2, 3, 3, 3])

print(rdd.countByValue())

foreachPartition(func)

rdd = spark.sparkContext.parallelize([1, 2, 3, 4], numSlices=2)

rdd.foreachPartition(lambda partition: print(list(partition)))

takeOrdered()

rdd = spark.sparkContext.parallelize([3, 1, 2, 4])

print(rdd.takeOrdered(2))

Transformations in DataFrame:

Using select()

df = spark.createDataFrame([(1, "Alice"), (2, "Bob")], ["id", "name"])

df.select("name").show()

Using filter()/where()

df.filter(df.id > 1).show()

where()` is just an alias for `filter().

Using groupBy()

df.groupBy("name").count().show()

Using agg()

df.groupBy("name").agg({"id": "sum"}).show()

Using join()

df1 = spark.createDataFrame([(1, "Alice"), (2, "Bob")], ["id", "name"])

df2 = spark.createDataFrame([(1, "HR"), (2, "Engineering")], ["id", "dept"])

df1.join(df2, "id").show()

Using withColumn()

from pyspark.sql.functions import col

df.withColumn("id\_times\_two", df["id"] \* 2).show()

Using drop()

df.drop("id").show()

Using distinct()

df = spark.createDataFrame([(1, "Alice"), (1, "Alice"), (2, "Bob")], ["id", "name"])

df.distinct().show()

Using orderBy()/sort()

df.orderBy(df.id.desc()).show()

Using union()

df1 = spark.createDataFrame([(1, "Alice")], ["id", "name"])

df2 = spark.createDataFrame([(2, "Bob")], ["id", "name"])

df1.union(df2).show()

Using limit()

df.limit(1).show()

Using sample()

df.sample(False, 0.5, seed=123).show()

Using repartition()

df.repartition(5).rdd.getNumPartitions()

Using coalesce()

df.coalesce(1).rdd.getNumPartitions()

Using alias()

df.alias("alias\_df").select("alias\_df.name").show()

Using explode()

from pyspark.sql.functions import explode

df = spark.createDataFrame([(1, ["a", "b", "c"]), (2, ["d", "e"])], ["id", "letters"])

df.select(df.id, explode(df.letters)).show()

Using pivot()

df = spark.createDataFrame([("A", "2022", 10), ("A", "2023", 20), ("B", "2022", 30)], ["name", "year", "value"])

df.groupBy("name").pivot("year").sum("value").show()

Using cube()

df = spark.createDataFrame([("A", 10), ("A", 15), ("B", 30)], ["category", "value"])

df.cube("category").sum("value").show()

Using rollup()

df = spark.createDataFrame([("A", 10), ("A", 15), ("B", 30)], ["category", "value"])

df.rollup("category").sum("value").show()

Using crossJoin()

df1 = spark.createDataFrame([(1, "Alice")], ["id", "name"])

df2 = spark.createDataFrame([(2, "Bob")], ["id", "name"])

df1.crossJoin(df2).show()

**DataFrames actions**

collect()

df = sparkcreateDataFrame([(1, "Alice"), (2, "Bob")], ["id", "name"])

result = df.collect()

print(result)

count()

df = sparkcreateDataFrame([(1, "Alice"), (2, "Bob")], ["id", "name"])

print(df.count())

first()

df = sparkcreateDataFrame([(1, "Alice"), (2, "Bob")], ["id", "name"])

print(df.first())

head(n)

df = sparkcreateDataFrame([(1, "Alice"), (2, "Bob")], ["id", "name"])

print(dfhead(1))

take(n)

df = sparkcreateDataFrame([(1, "Alice"), (2, "Bob")], ["id", "name"])

print(dftake(1)) Output: [Row(id=1, name='Alice')]

show(n)

df = sparkcreateDataFrame([(1, "Alice"), (2, "Bob")], ["id", "name"])

dfshow(2) Displays 2 rows

describe()

df = sparkcreateDataFrame([(1, 100), (2, 200)], ["id", "salary"])

dfdescribe()show(

select()

df = sparkcreateDataFrame([(1, "Alice", 100), (2, "Bob", 200)], ["id", "name", "salary"])

dfselect("name", "salary")show()

agg()

from pysparksqlfunctions import sum, avg

df = sparkcreateDataFrame([(1, 100), (2, 200)], ["id", "salary"])

dfagg(sum("salary")alias("total\_salary"), avg("salary")alias("average\_salary"))show()

foreach(func)

df = sparkcreateDataFrame([(1, "Alice"), (2, "Bob")], ["id", "name"])

dfforeach(lambda row: print(row))

foreachPartition(func)

df = sparkcreateDataFrame([(1, "Alice"), (2, "Bob")], ["id", "name"])

dfforeachPartition(lambda partition: print(list(partition)))

toPandas()

df = sparkcreateDataFrame([(1, "Alice"), (2, "Bob")], ["id", "name"])

pandas\_df = dftoPandas()

print(pandas\_df)

write()

df = sparkcreateDataFrame([(1, "Alice"), (2, "Bob")], ["id", "name"])

dfwritecsv("/path/to/output")

writeStream()

df = sparkreadStreamformat("rate")load()

query = dfwriteStreamformat("console")start()

queryawaitTermination()

cache()

df = sparkcreateDataFrame([(1, "Alice"), (2, "Bob")], ["id", "name"])

dfcache()

dfshow()

persist()

from pyspark import StorageLevel

df = sparkcreateDataFrame([(1, "Alice"), (2, "Bob")], ["id", "name"])

dfpersist(StorageLevelDISK\_ONLY)

dfshow()

unpersist()

df = sparkcreateDataFrame([(1, "Alice"), (2, "Bob")], ["id", "name"])

dfcache()

dfunpersist()

dropDuplicates()

df = sparkcreateDataFrame([(1, "Alice"), (2, "Alice"), (3, "Bob")], ["id", "name"])

dfdropDuplicates(["name"])show()

dropna()

df = sparkcreateDataFrame([(1, "Alice"), (2, None), (3, "Bob")], ["id", "name"])

dfdropna()show()

limit(n)

df = sparkcreateDataFrame([(1, "Alice"), (2, "Bob")], ["id", "name"])

dflimit(1)show()