National University of Computer and Emerging Sciences

Laboratory Manual-12

for

Fundamentals of Big Data Lab

Course Instructor: Dr Iqra Safdar
Lab Instructors: Mr. Muhammad Mazarib; Mr Muhammad Aiss Shahid
Section: BDS-4B
Date: 03-May-2023
Semester: Spring 2023

Department of Computer Science

FAST-NU, Lahore, Pakistan

Actions Available on Pair RDDs

As with the transformations, all of the traditional actions available on the base RDD are also available on pair RDDs. Some additional actions are available on pair RDDs to take advantage of the key/value nature of the data; these are listed

Table Actions on pair RDDs (example ({(1, 2), (3, 4), (3, 6)}))

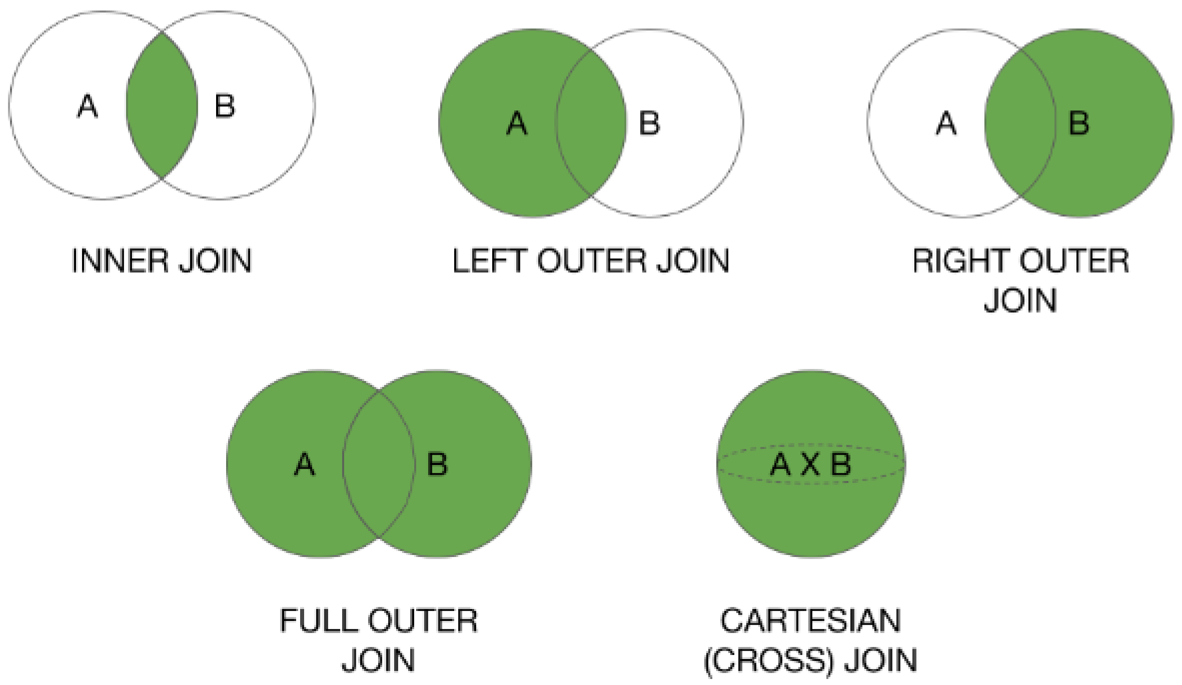
Function	Description	Example	Result
`countByKey()`	Count the number of elements for each key.	`rdd.countByKey()`	`{(1, 1), (3, 2)}`
`collectAsMap()`	Collect the result as a map to provide easy lookup.	`rdd.collectAsMap()`	`Map{(1, 2), (3, 4), (3, 6)}`
`lookup(key)`	Return all values associated with the provided key.	`rdd.lookup(3)`	`[4, 6]`

Transformations on two pair RDDs (rdd = {(1, 2), (3, 4), (3, 6)} other = {(3, 9)})

Function name	Purpose	Example	Result
`subtractByKey`	Remove elements with a key present in the other RDD.	`rdd.subtractByKey(other)`	`{(1, 2)}`
`join`	Perform an inner join between two RDDs.	`rdd.join(other)`	`{(3, (4, 9)), (3, (6, 9))}`
`rightOuterJoin`	Perform a join between two RDDs where the key must be present in the first RDD.	`rdd.rightOuterJoin(other)`	`{(3,(Some(4),9)), (3,(Some(6),9))}`
`leftOuterJoin`	Perform a join between two RDDs where the key must be present in the other RDD.	`rdd.leftOuterJoin(other)`	`{(1,(2,None)), (3, (4,Some(9))), (3, (6,Some(9)))}`
`cogroup`	Group data from both RDDs sharing the same key.	`rdd.cogroup(other)`	`{(1,([2],[])), (3, ([4, 6],[9]))}`

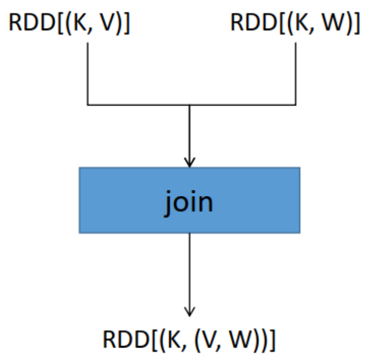
Join Operations in Pyspark

The given below illustration explains the various types of joins.



1. PySpark RDD join/ inner join

   Return an RDD containing all pairs of elements with matching keys in *self* and *other*. Each pair of elements will be returned as a (k, (v1, v2)) tuple, where (k, v1) is in *self* and (k, v2) is in *other*.



```
Example:

rdd1 = sc.parallelize([("a", 1), ("b", 4)])

rdd2 = sc.parallelize([("a", 2), ("a", 3)])

sorted(rdd1.join(rdd2).collect())

Output = [('a', (1, 2)), ('a', (1, 3))]
```

2. PySpark RDD leftOuterJoin

   Perform a left outer join of self and other. For each element (k, v) in self, the resulting RDD will either contain all pairs (k, (v, w)) for w in other, or the pair (k, (v, None)) if no elements in other have key k.

```
rdd1 = sc.parallelize([("a", 1), ("b", 4)])

rdd2 = sc.parallelize([("a", 2)])

sorted(rdd1.leftOuterJoin(rdd2).collect())

output = [('a', (1, 2)), ('b', (4, None))]
```

3. PySpark RDD rightOuterJoin

   Perform a right outer join of self and other. For each element (k, w) in other, the resulting RDD will either contain all pairs (k, (v, w)) for v in this, or the pair (k, (None, w)) if no elements in self have key k.

```
rdd1 = sc.parallelize([("a", 1), ("b", 4)])

rdd2 = sc.parallelize([("a", 2)])
```

```
sorted(rdd2.rightOuterJoin(rdd1).collect())

Output = [('a', (2, 1)), ('b', (None, 4))]
```

4. PySpark RDD fullOuterJoin

   Perform a right outer join of self and other. For each element (k, v) in self, the resulting RDD will either contain all pairs (k, (v, w)) for w in other, or the pair (k, (v, None)) if no elements in other have key k.

   Similarly, for each element (k, w) in other, the resulting RDD will either contain all pairs (k, (v, w)) for v in self, or the pair (k, (None, w)) if no elements in self have key k.

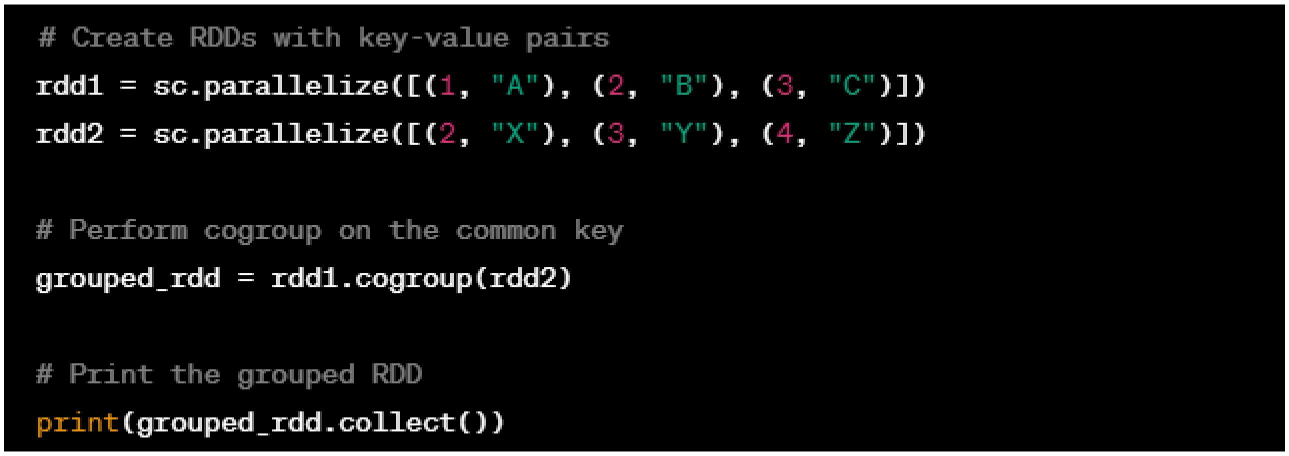
```
rdd1 = sc.parallelize([("a", 1), ("b", 4)])

rdd2 = sc.parallelize([("a", 2), ("c", 8)])

sorted(rdd1.fullOuterJoin(rdd2).collect())

output=[('a', (1, 2)), ('b', (4, None)), ('c', (None, 8))]
```

5. RDD cogroup()

   cogroup() is another transformation in PySpark that allows you to group the values of multiple RDDs by a common key. It returns an RDD of key-value pairs, where the key is the common key, and the value is a tuple containing all the values from the input RDDs that share the same key.

   For each key k in self or other, return a resulting RDD that contains a tuple with the list of values for that key in self as well as other.

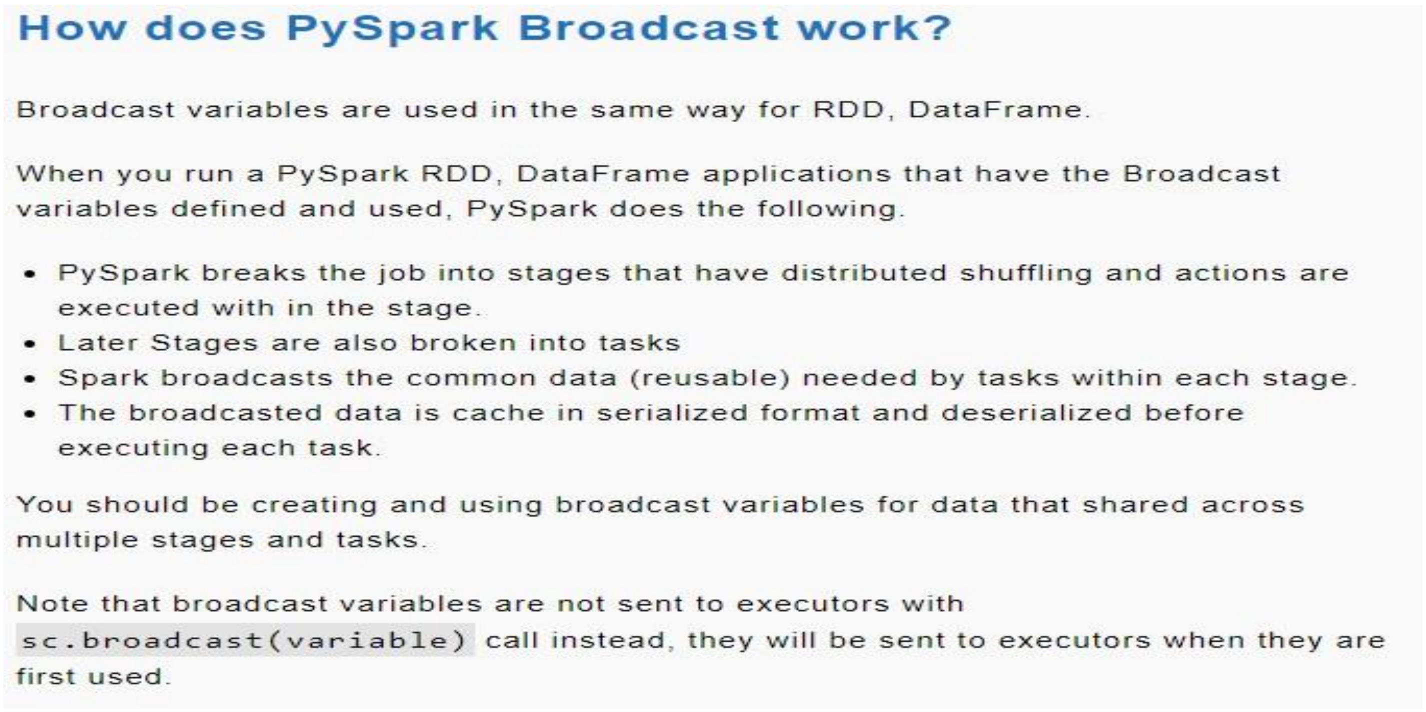
```
rdd1 = sc.parallelize([("a", 1), ("b", 4)])

rdd2 = sc.parallelize([("a", 2)])

[(x, tuple(map(list, y))) for x, y in sorted(list(rdd1.cogroup(rdd2).collect()))]

Output= [('a', ([1], [2])), ('b', ([4], []))]
```



```
# Create RDDs with key-value pairs
rdd1 = sc.parallelize([(1, "A"), (2, "B"), (3, "C")])
rdd2 = sc.parallelize([(2, "X"), (3, "Y"), (4, "Z")])

# Perform cogroup on the common key
grouped_rdd = rdd1.cogroup(rdd2)

# Print the grouped RDD
print(grouped_rdd.collect())
```

Output of the above code: [(1, (['A'], [])), (2, (['B'], ['X'])), (3, (['C'], ['Y'])), (4, ([], ['Z']))]

Broadcast: Broadcasting is a technique used to efficiently share a read-only variable across all the worker nodes in a distributed computing environment. In PySpark, when you perform operations like join or filter on large datasets, Spark may shuffle and transfer data across different nodes, which can be expensive in terms of time and network bandwidth. Broadcasting allows you to send a small amount of data, such as lookup tables or configuration parameters, to all the worker nodes so that they can access it locally, avoiding unnecessary data shuffling. This can significantly improve the performance of Spark applications.



How does PySpark Broadcast work?

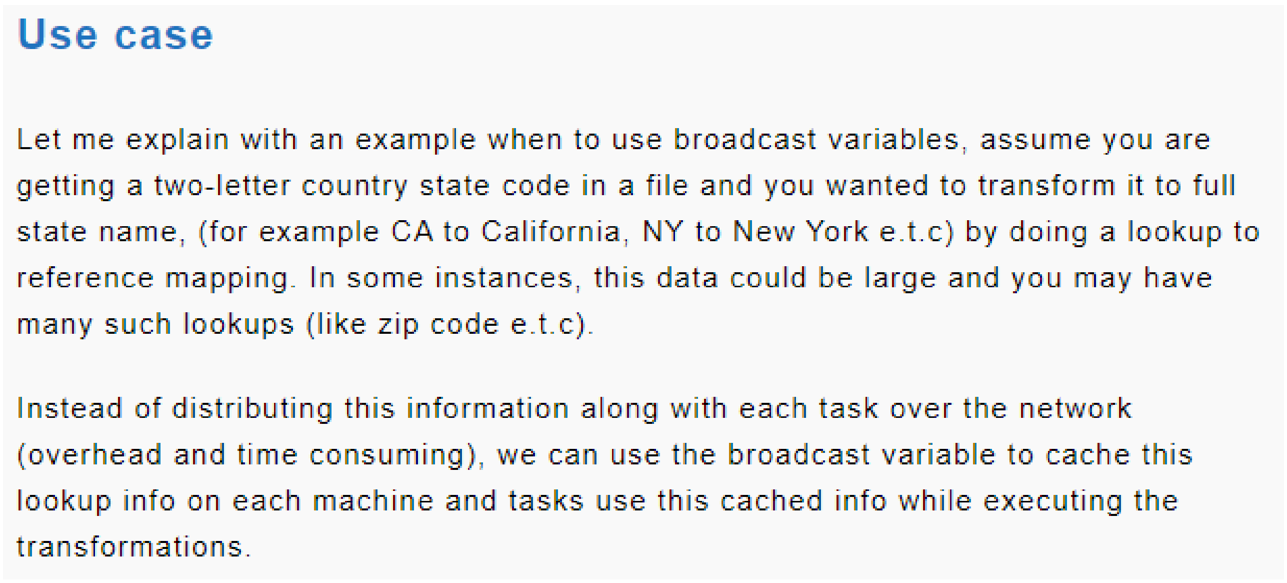
Broadcast variables are used in the same way for RDD, DataFrame.

When you run a PySpark RDD, DataFrame applications that have the Broadcast variables defined and used, PySpark does the following.

- PySpark breaks the job into stages that have distributed shuffling and actions are executed with in the stage.
- Later Stages are also broken into tasks
- Spark broadcasts the common data (reusable) needed by tasks within each stage.
- The broadcasted data is cache in serialized format and deserialized before executing each task.

You should be creating and using broadcast variables for data that shared across multiple stages and tasks.

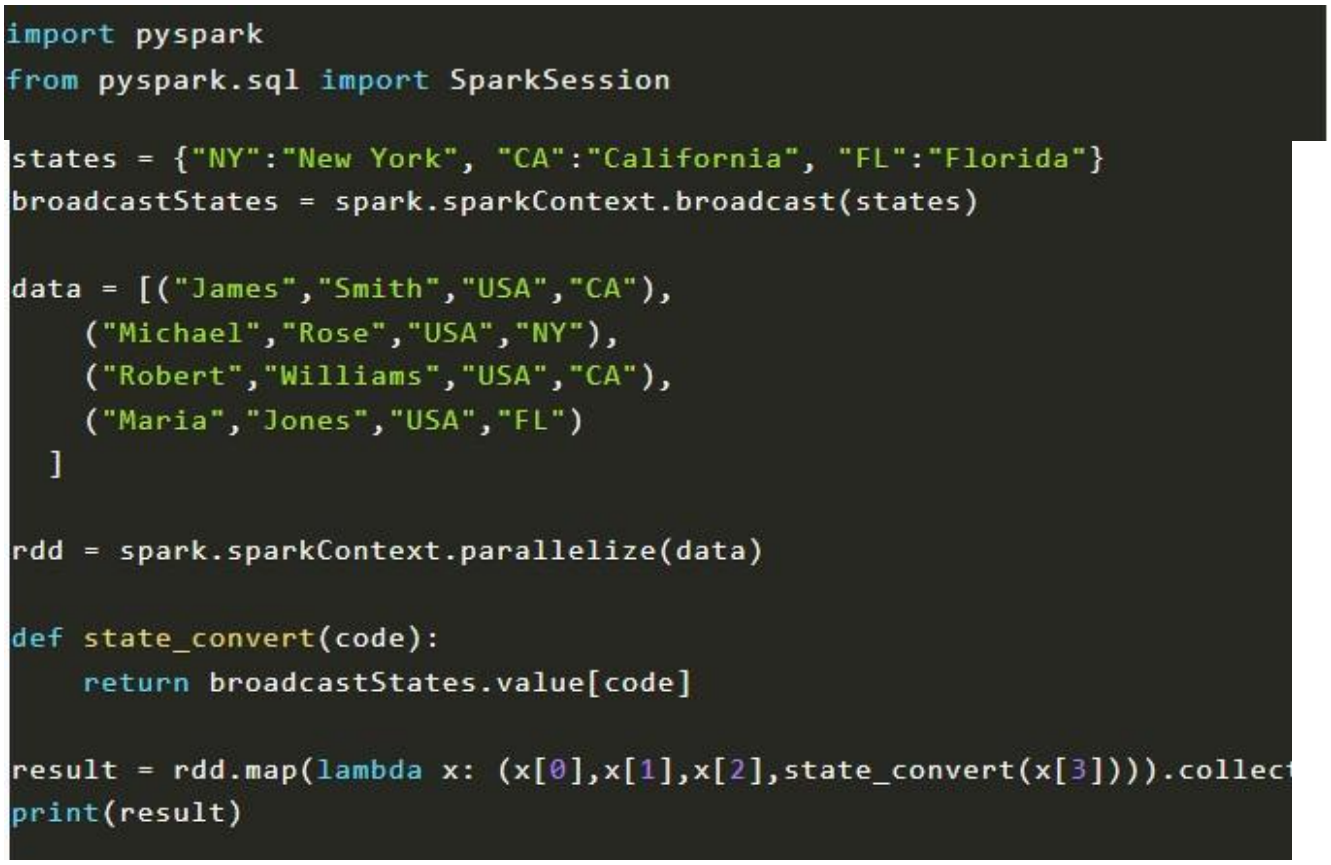
Note that broadcast variables are not sent to executors with `sc.broadcast(variable)` call instead, they will be sent to executors when they are first used.



Use case

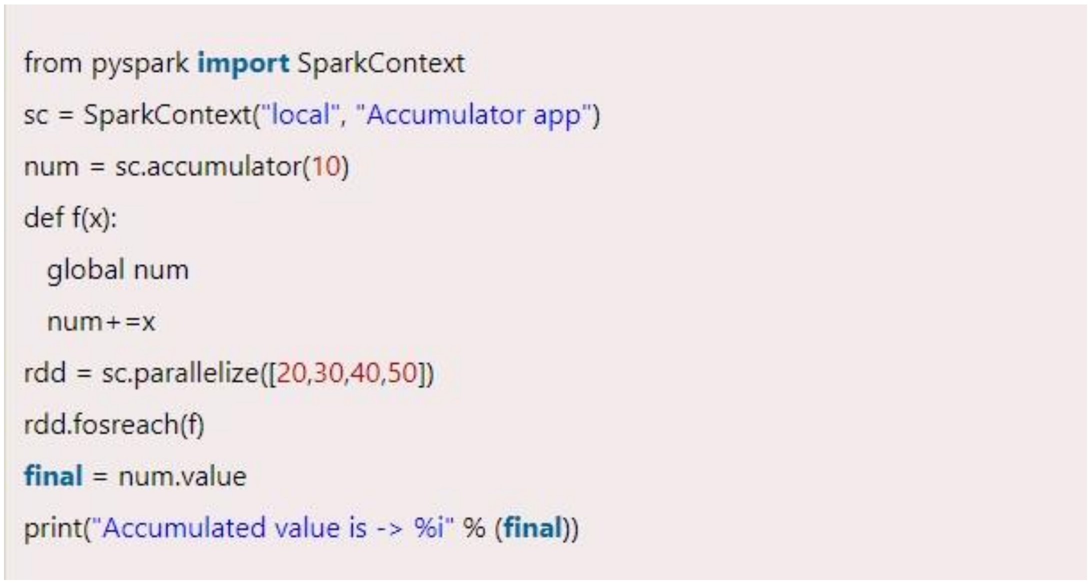
Let me explain with an example when to use broadcast variables, assume you are getting a two-letter country state code in a file and you wanted to transform it to full state name, (for example CA to California, NY to New York e.t.c) by doing a lookup to reference mapping. In some instances, this data could be large and you may have many such lookups (like zip code e.t.c).

Instead of distributing this information along with each task over the network (overhead and time consuming), we can use the broadcast variable to cache this lookup info on each machine and tasks use this cached info while executing the transformations.



```
import pyspark
from pyspark.sql import SparkSession

states = {"NY":"New York", "CA":"California", "FL":"Florida"}
broadcastStates = spark.sparkContext.broadcast(states)

data = [("James","Smith","USA","CA"),
    ("Michael","Rose","USA","NY"),
    ("Robert","Williams","USA","CA"),
    ("Maria","Jones","USA","FL")
  ]

rdd = spark.sparkContext.parallelize(data)

def state_convert(code):
    return broadcastStates.value[code]

result = rdd.map(lambda x: (x[0],x[1],x[2],state_convert(x[3]))).collec
print(result)
```

Accumulator: An accumulator is a distributed variable that is used to accumulate values across different worker nodes in a distributed computing environment. It is a write-only variable that can be updated in parallel by multiple tasks running on different worker nodes. Accumulators are used for aggregating information or collecting statistics during the execution of a Spark job.



```
from pyspark import SparkContext
sc = SparkContext("local", "Accumulator app")
num = sc.accumulator(10)
def f(x):
  global num
  num+=x
rdd = sc.parallelize([20,30,40,50])
rdd.fosreach(f)
final = num.value
print("Accumulated value is -> %i" % (final))
```

LAB TASKS

Your task is to perform the following RDD join operations using PySpark:

1. Inner Join: Perform an inner join on emp_id between rdd1 and rdd2 to get an RDD containing the employee ID, employee name, employee department, and employee salary for employees who exist in both RDDs.
2. Left Outer Join: Perform a left outer join on emp_id between rdd1 and rdd2 to get an RDD containing the employee ID, employee name, employee department, and employee salary for all employees in rdd1 and matching employees in rdd2. For employees in rdd1 that do not have a matching employee ID in rdd2, the employee salary should be set to None.
3. Calculate the total salary of employees in RDD1 by joining it with RDD2 on "emp_id" and multiplying "emp_salary" with "emp_experience".
4. Calculate the maximum salary of employees
5. Perform an inner join on RDD1 and RDD2 on "emp_id" and "dept_id", and calculate the total salary of employees in each department.
6. Perform a left outer join on RDD1 and RDD2 on "emp_id" and "dept_id", and find the employees who do not belong to any department.

7. Write a PySpark code to perform a broadcast join between two pair RDDs. The first RDD, contains key value pair: "id" and "name". The second RDD contains pair of "id" and "age". Perform a broadcast join on the "id" column and return a new RDD, result, containing "id", "name", and "age".

8. Write a PySpark code to calculate the sum of all values in an RDD using an accumulator. Create an RDD of integers from 1 to 10. Define an accumulator with an initial value of 0. Use the foreach() action on the RDD to update the accumulator with each element. Finally, print the final value of the accumulator after processing all elements in the RDD.