Day3 Assignment

Pyspark RDD operations.

Actions/transformations performed on rdd values give us non rdd values..

The non rdd values specify that they are not stored in cluster

- **.collect()----------→Collects info about RDD**

```
print(rdd.collect())
rdd.collect()  #Actions give non rdd values
```

Output:

```
[('C', 85, 76, 87, 91), ('B', 85, 76, 87, 91), ('A', 85, 78, 96, 92), ('A', 92, 76, 8
```

- **.count()----------→Counts no. of elements in rdd**

```
#counting using a variable,count no of elements in rdd
Data_rdd=sc.parallelize([1,2,3,4,5,6,7,8,9])
print(Data_rdd.count())
```

Output:

```
...  9
```

- **.first()----------→Return the first element in RDD**

```
#return the first element
print(Data_rdd.first())
```

Output:

```
...  1
```

- **.take()----------→Print upto specific values,take(3) returns first 3 values**

```python
#printing only certain values..To print only 2 values we can pass 2 as argument
print(Data_rdd.take(2))
print(rdd.take(2))
```

Python

```
[1, 2]
[('C', 85, 76, 87, 91), ('B', 85, 76, 87, 91)]
```

- **.saveAsTextFile("File Name")---→Storing all the rdd data in local text file.can be viewed in catalog**

10:18 AM (3s) 6

```python
#This action is used to serve resultant rdd as a text file
Data_rdd.saveAsTextFile("save.txt")
```

(1) Spark Jobs

- **.reduce()----------→Tranformation to reduce rdd according to condition**

```python
#reduce function
print(Data_rdd.reduce(lambda x,y:x+y))
```

```
45
```

- **.map() -→returns new RDD-→transforms rdd values based on condition provided**

10:25 AM (<1s) 7

```python
#map action tranforms and return a rdd
print(Data_rdd.map(lambda x:x+10).collect())
```

(1) Spark Jobs

```
[11, 12, 13, 14, 15, 16, 17, 18, 19]
```

- **.filter()----→Filter data based on condition**

```
10:27 AM (1s)                    8

#filter elements from a PYspark RDD
print(Data_rdd.filter(lambda x:x%2==0).collect())

(1) Spark Jobs

[2, 4, 6, 8]
```

```
10:58 AM (1s)                    9

data2_rdd=sc.parallelize(['Rahul','Swati','Rohan','Shreya','Priya'])
print(data2_rdd.filter(lambda x: x.startswith('R')).collect())

(1) Spark Jobs

['Rahul', 'Rohan']
```

- **.FlatMap()**

```
10:51 AM (1s)                    10

data3_rdd=sc.parallelize(["hiii everyone","Welcome to pyspark session"])
print(data3_rdd.flatMap(lambda x:x.split(" ")).collect())

(1) Spark Jobs

['hiii', 'everyone', 'Welcome', 'to', 'pyspark', 'session']
```

- **.Union()----------→combine two rdds**

```
11:02 AM (1s)                    11

#Union function
data=sc.parallelize([2,4,5,6,7,8,9,10])
data1=data.filter(lambda x:x%2==0)
data2=data.filter(lambda x:x%3==0)
print(data1.union(data2).collect())

(1) Spark Jobs

[2, 4, 6, 8, 10, 6, 9]
```

Pyspark pair RDD operations

Key value pairs..similar to real world data

Pyspark Transformations in pair RDDS:

- reduceByKey()----------→*#reduce by key*
- *# It performs multiple parallel processes for each key in the data and combines the values for the same keys returns rdd as a result*

01:59 PM (1s) 14

```
#reduce by key
# It  performs multiple parallel processes for each
# key in the data and combines the values for the same keys.

# returns rdd as a result

marks_rdd = sc.parallelize([('Rahul', 25), ('Swati', 26), ('Shreya', 22),
('Abhay', 29), ('Rohan', 22),
('Rahul', 23), ('Swati', 19), ('Shreya', 28), ('Abhay', 26), ('Rohan',
22)])


print(marks_rdd.reduceByKey(lambda x, y: x + y).collect())
```

(1) Spark Jobs

```
[('Shreya', 50), ('Swati', 45), ('Rahul', 48), ('Abhay', 55), ('Rohan', 44)]
```

- sortByKey() ----

  The .sortByKey() transformation sorts the input data by keys from key-value pairs either in ascending or descending order. It returns a unique RDD as a result.

12:50 PM (2s) 15

```
#sort by key

# The .sortByKey() transformation sorts the input data by keys from
key-value pairs either in ascending or descending order. It returns a
unique RDD as a result.

marks_rdd = sc.parallelize([('Rahul', 25), ('Swati', 26), ('Shreya', 22),
('Abhay', 29), ('Rohan', 22),
('Rahul', 23), ('Swati', 19), ('Shreya', 28), ('Abhay', 26), ('Rohan',
22)])

print(marks_rdd.sortByKey('ascending').collect())
```

(3) Spark Jobs

```
[('Abhay', 29), ('Abhay', 26), ('Rahul', 25), ('Rahul', 23), ('Rohan', 22), ('R
ohan', 22), ('Shreya', 22), ('Shreya', 28), ('Swati', 26), ('Swati', 19)]
```

- groupBy() ---*The .groupByKey() transformation groups all the values in the given data with thesame key together. It returns a new RDD as a result.*

12:51 PM (1s) 16

```
#group by

# The .groupByKey() transformation groups all the values in the given data
with thesame key together. It returns a new RDD as a result.

marks_rdd = sc.parallelize([('Rahul', 25), ('Swati', 26), ('Shreya', 22),
('Abhay', 29), ('Rohan', 22),
('Rahul', 23), ('Swati', 19), ('Shreya', 28), ('Abhay', 26), ('Rohan',
22)])
dict_rdd = marks_rdd.groupByKey().collect()
for key, value in dict_rdd:
    print(key, list(value))
```

(1) Spark Jobs

```
Shreya [22, 28]
Swati [26, 19]
Rahul [25, 23]
Abhay [29, 26]
Rohan [22, 22]
```

Pyspark Actions in pair RDDS

- countByKey()

- *The .countByKey() option is used to count the number of values for each key in the given data. This action returns a dictionary and one can extract the keys and values by iterating over the extracted dictionary*
-

12:54 PM (1s) 17

```
#Count by
# The .countByKey() option is used to count the number of values for each
key in the given data. This action returns a dictionary and one can
extract the keys and values by iterating over the extracted dictionary
# using loops.
marks_rdd = sc.parallelize([('Rahul', 25), ('Swati', 26), ('Rohan', 22),
('Rahul', 23), ('Swati', 19),
('Shreya', 28), ('Abhay', 26), ('Rohan', 22)])
dict_rdd = marks_rdd.countByKey().items()
for key, value in dict_rdd:
    print(key, value)
```

(1) Spark Jobs

```
Rahul 2
Swati 2
Rohan 2
Shreya 1
Abhay 1
```

View and Temp View

```
04:27 PM (1s)                    23

from pyspark.sql import SparkSession
# Create spark session
spark = SparkSession \
.builder \
.appName("SparkByExamples.com") \
.enableHiveSupport() \
.getOrCreate()
data = [("James","Smith","USA","CA"),
("Michael","Rose","USA","NY"),
("Robert","Williams","USA","CA"),
("Maria","Jones","USA","FL")
]
columns = ["firstname","lastname","country","state"]
# Create dataframe
sampleDF = spark.sparkContext.parallelize(data).toDF(columns)
sampleDF.createOrReplaceTempView("Person")
sampleDF.createOrReplaceTempView("mydata")
sampleDF.show()
```

Output

```
sampleDF: pyspark.sql.dataframe.DataFrame = [firstname: string, lastname: string ... 2
more fields]

+---------+--------+-------+-----+
|firstname|lastname|country|state|
+---------+--------+-------+-----+
|    James|   Smith|    USA|   CA|
|  Michael|    Rose|    USA|   NY|
|   Robert|Williams|    USA|   CA|
|    Maria|   Jones|    USA|   FL|
+---------+--------+-------+-----+
```

Selecting renaming columns from rdd:

- **This PySpark script creates a Spark DataFrame with sample employee data, renames columns like "DOB" to "date of birth" and "Name" to "personname," and displays the updated DataFrame.**

```
# Importing necessary libraries
from pyspark.sql import SparkSession

# Create a spark session
spark = SparkSession.builder.appName('pyspark - example join').getOrCreate
()

# Create data in dataframe
data = [(('Ram'), '1991-04-01', 'M', 3000),
        (('Mike'), '2000-05-19', 'M', 4000),
        (('Rohini'), '1978-09-05', 'M', 4000),
        (('Maria'), '1967-12-01', 'F', 4000),
        (('Jenis'), '1980-02-17', 'F', 1200)]

# Column names in dataframe
columns = ["Name", "DOB", "Gender", "salary"]

# Create the spark dataframe
df = spark.createDataFrame(data=data,
                           schema=columns)
df.withColumnRenamed("DOB","date of birth").show()
df.withColumnRenamed("DOB","date of birth").withColumnRenamed("Name",
"personname").show()
```

Output

```
df: pyspark.sql.dataframe.DataFrame = [Name: string, DOB: string ... 2 more fields]
+------+-------------+------+------+
|  Name|date of birth|Gender|salary|
+------+-------------+------+------+
|   Ram|   1991-04-01|     M|  3000|
|  Mike|   2000-05-19|     M|  4000|
|Rohini|   1978-09-05|     M|  4000|
| Maria|   1967-12-01|     F|  4000|
| Jenis|   1980-02-17|     F|  1200|
+------+-------------+------+------+

+----------+-------------+------+------+
|personname|date of birth|Gender|salary|
+----------+-------------+------+------+
|       Ram|   1991-04-01|     M|  3000|
|      Mike|   2000-05-19|     M|  4000|
|    Rohini|   1978-09-05|     M|  4000|
|     Maria|   1967-12-01|     F|  4000|
|     Jenis|   1980-02-17|     F|  1200|
+----------+-------------+------+------+
```

t

- **This PySpark script creates a DataFrame with employee data, then uses selectExpr to rename the "Gender" column as "category," "Name" as "name," and retains other columns, displaying the final DataFrame.**

```python
# Importing necessary libraries using select exp
from pyspark.sql import SparkSession

# Create a spark session
spark = SparkSession.builder.appName('pyspark - example join').getOrCreate
()

# Create data in dataframe
data = [(('Ram'), '1991-04-01', 'M', 3000),
        (('Mike'), '2000-05-19', 'M', 4000),
        (('Rohini'), '1978-09-05', 'M', 4000),
        (('Maria'), '1967-12-01', 'F', 4000),
        (('Jenis'), '1980-02-17', 'F', 1200)]

# Column names in dataframe
columns = ["Name", "DOB", "Gender", "salary"]

# Create the spark dataframe
df = spark.createDataFrame(data=data,
                           schema=columns)


data = df.selectExpr("Gender as category","DOB","Name as name","salary")

data.show()
```

Output

```python
data = df.selectExpr("Gender as category","DOB","Name as name","salary")

data.show()
```

```
(3) Spark Jobs
df: pyspark.sql.dataframe.DataFrame = [Name: string, DOB: string ... 2 more fields]
data: pyspark.sql.dataframe.DataFrame = [category: string, DOB: string ... 2 more fields]
+--------+----------+------+------+
|category|       DOB|  name|salary|
+--------+----------+------+------+
|       M|1991-04-01|   Ram|  3000|
|       M|2000-05-19|  Mike|  4000|
|       M|1978-09-05|Rohini|  4000|
|       F|1967-12-01| Maria|  4000|
|       F|1980-02-17| Jenis|  1200|
+--------+----------+------+------+
```

- **This PySpark script uses the select function with column aliasing to rename the "salary" column to "Amount" while keeping other columns unchanged, and displays the updated DataFrame.**

04:26 PM (1s) 22

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

# Select the 'salary' as 'Amount' using aliasing
# Select remaining with their original name
data = df.select(col("Name"),col("DOB"),
                 col("Gender"),
                 col("salary").alias('Amount'))
data.show()
```

(3) Spark Jobs

data: pyspark.sql.dataframe.DataFrame = [Name: string, DOB: string ... 2 more fields]

```
+------+----------+------+------+
|  Name|       DOB|Gender|Amount|
+------+----------+------+------+
|   Ram|1991-04-01|     M|  3000|
|  Mike|2000-05-19|     M|  4000|
|Rohini|1978-09-05|     M|  4000|
| Maria|1967-12-01|     F|  4000|
| Jenis|1980-02-17|     F|  1200|
+------+----------+------+------+
```