RDD and DataFrame Operations

**1) How do we create RDD's in spark ? Define the number of Partitions while defining it ?**

from pyspark import SparkContext, SparkConf

conf = SparkConf()

sc = SparkContext(conf=conf)

rdd = sc.textfile(“/path/of/file”, 4) #4 means no.of partitions

rdd1 = sc.parallelize(range(1,10),4)

rdd2 = sc.parallelize([(1,"sun"),(2,"moon"),(3,"mercury")],4)

rdd.getNumPartitions() #will return 4 as we mentioned above

rdd1.collect()

**2) How do we create a dataframe in Spark?**

sqlContext = sql.SQLContext(sc)

2.1)

df1 = sc.parallelize([(1,"sun"),(2,"moon"),(3,"mercury")]).toDF(["no", "Planet"])

df1.show()

2.2)

depRDD = sc.textFile("dept.txt", 4)

dep\_col = Row('DEPTNO', 'DNAME', 'DLOC')

depRow = depRDD.map(lambda x: x.split(',')).map(lambda r: dep\_col(\*r))

depDF = sqlContext.createDataFrame(depRow)

2.3) After v.2.0

from pyspark.sql import SparkSession

spark = SparkSession.builder.master("local").config("spark.executor.memory", "4g").getOrCreate()

dept = spark.read.csv("dept.txt", header=True, inferSchema=True)

dept.show()

**Dataframe Operations:**

**1. Creating New column, Renaming New column, Changing DataType of column, selecting subset of column**

**1.1) Creating New column**

df.withColumn(“colName”, “expr”)

**1.2) Renaming new column**

<https://stackoverflow.com/questions/34077353/how-to-change-dataframe-column-names-in-pyspark>

emp.withColumnRenamed("EMPNO", "ENO").show()

**1.3) Changing DataType of the column**

emp.withColumn(“EMPNO”, to\_date('HIREDATE')).withColumn("SAL", empDF.SAL.cast(“double”))

**Functional way:**

def toDouble(d):

return d.cast("double")

emp.withColumn("SAL", toDouble(empDF.SAL))

**1.4)** **selecting subset of column**

emp.select(emp.columns[:2])

emp.show(5)

**2. GroupBy, Conditional GroupBy, with various aggregrate fnc ( Max, min, sum, avergae, median, count,)**

emp.groupBy("DEPTNO").agg(sum(when(col("COMM").isNull(),col("SAL")).otherwise(col("SAL")+col("COMM")))).show()

**3. Counting null values, handling and filling null values**

**3.1 Counting null values:**

emp.filter(col("COMM").isNull()).count()

**3.2 Filling null values:**

emp.na.fill({'COMM': 0}).show()

**3.3 Droping null values:**

emp.na.drop().show() #clean the rows if null persist at any column

emp.na.drop(subset = "COMM").show() # clean the row whereever COMM is null

4. **Filter operation ( and , or) on single/multiple columns**

emp.filter((col("SAL") == allenSAL) | (col("JOB") == allenJOB)).show()

emp.filter((col("SAL") == allenSAL) & (col("JOB") == allenJOB)).show()

5. **Ranking on dataframe columns ( Window Functions)**

# 103. Find out least 5 earners of the company

emp.withColumn("rank", dense\_rank().over(Window.orderBy(asc("SAL")))).filter("rank < 6").show()

6. **Writing UDF's with/ without lambda**

Tutorial: <https://changhsinlee.com/pyspark-udf/>

# 159) Find the name and Job of the emps who earn Max salary and Commission.

def netpay(sal, comm):

return when(comm.isNull(), sal).otherwise(comm+sal)

netpay\_udf = netpay

emp.withColumn("NETPAY", netpay\_udf(emp.SAL, emp.COMM)).groupby("ENAME","JOB").agg(max("SAL")).show()

7. **Other Fuctionality like : lit(), head(), collectAsMap(), collectAsList()**

**lit():**

lit will help us to add some hardcoded value to the dataframe columns.

# 127. List the Ename and Sal is increased by 15% and expressed as no.of Dollars.

emp.select("ENAME", concat(lit("$"), emp.SAL + emp.SAL\*15/100).alias("inc.15%")).show()

**head(): # below will return single integer value of empno of name blake**

blakeID = emp.filter('ENAME = "BLAKE"').select("EMPNO").head()[0]

**collectAsMap():**

**collectAsMap()**

This method will help us to convert the any paired rdd to dictionary type. The following rdd has converted to Dict and returns 2nd position value as moon

a = sc.parallelize([(1,"sun"),(2,"moon"),(3,"mercury")])

a.collectAsMap()[2]

output: ‘moon’

8. **How to print Schema, show, truncate, save**

printSchema():

emp.printSchema() # will explain our table schema details and types

show():

emp.show() # will return the values from table. usually it will return 20 rows

Save():

Spark 1.4+

df.write.format('com.databricks.spark.csv').save('mycsv.csv')

In Spark 2.0+ you can use csv data source directly:

df.write.csv('mycsv.csv')

9. **Joins and where to use them**

**Three Major types of join we are using, inner, left, right, self**

The below DF joins three tables with condition value also by primary key value using inner join.

emp.join(sal, emp.SAL.between(sal.LOSAL, sal.HISAL), 'inner')\

.join(dept, "DEPTNO", 'inner').filter('DNAME = "SALES"').show()

if we want to join by left, right replace inner with left or right.

emp.alias("e").join(emp.alias("m"), [col("e.MGR") == col("m.EMPNO"), col("e.SAL") > col("m.SAL")], "inner").show()

10. **Dataframe reparation(), paritionby(), coalesce() functions**

**Repartition**(): Repartition will help us to increase the number of partition with full shuffle.

**Coalesce**(): coalesce with help us to decrease the number of partition with partial shuffle. It is always good to go with coalesce for reduce the no.of partitions

**partitionBy**(): it will take all the number 4 to starting indexes

pairs = sc.parallelize([1, 2, 3, 4, 2, 4, 1]).map(lambda x: (x, x))

sets = pairs.partitionBy(4).collect()

set(sets[0])

**11. distinct(),union(),intersection() and substract() operations**

print("Testing: **union**(otherDataset)")

u01\_rdd = sc.parallelize(range(1,9))

u02\_rdd = sc.parallelize(range(5,15))

union\_rdd = u01\_rdd.union(u02\_rdd).collect()

union\_rdd

Testing: union(otherDataset)

[1, 2, 3, 4, 5, 6, 7, 8, 5, 6, 7, 8]

print("Testing: **intersection**(otherDataset)")

i01\_rdd = sc.parallelize(range(1,9))

i02\_rdd = sc.parallelize(range(1,15))

intersection\_rdd = i01\_rdd.intersection(i02\_rdd).collect()

intersection\_rdd

Testing: intersection(otherDataset)

[1, 2, 3, 4, 5, 6, 7, 8]

print("Testing: **intersection**(otherDataset)")

s01\_rdd = sc.parallelize(['A', 'B'])

s02\_rdd = sc.parallelize(['B', 'C'])

subtract\_rdd = s01\_rdd.subtract(s02\_rdd).collect()

subtract\_rdd

Testing: intersection(otherDataset)

['A']

**# Good to understand the below scenerio**

**# Say I have the data set one A,B and the dataset B,C here I want the result A,C Because I want to subtract b from both the sets.**

set1 = sc.parallelize(['A', 'B'])

set2 = sc.parallelize(['B', 'C'])

u\_set = set1.union(set2) #result A,B,B,C

i\_set = set1.intersection(set2) #result B

u\_set.subtract(i\_set).collect()

Output:

['A', 'C']

**12. collect() ( Spark 1.6 and spark 2). What is disadvantage of collect())**

Collect (Action) - Return all the elements of the dataset as an array at the driver program. This is usually useful after a filter or other operation that returns a sufficiently small subset of the data.

Collect will return all the values from the RDD so, incase our rdd is too big we will give the overload to the network. So we can use collect after filtering and our dataset is low in size.

So always prefer, take() and show().

**Read and Save various file formats?**

**Create hive tables from file or dataframe ( ORC, External/Internal Tables)**

**Save Dataframe as table in hive:**

df\_writer.partitionBy('col1')\

.saveAsTable('test\_table', format='parquet', mode='overwrite',

path='s3a://bucket/foo')

**Static Table with Partition**

Create table st\_part(id int, name String, dept String, year int, sal int)partitioned by(year int)

Row format delimited fields terminated by ‘,’

Line is terminated by ‘\n’

Stored as ORC;

Load data local inpath ‘/home/pradeep/files/part1’ into table st\_part partition(year=2012);

Load data local inpath ‘/home/pradeep/files/part2’ into table st\_part partition(year=2013);

Once we created a table, files will be in

user/hive/warehouse/st\_part/year=2012

And

user/hive/warehouse/st\_part/year=2013

**External Table:**

Create external table st\_part(id int, name String, dept String, year int, sal int)partitioned by(year int)

Row format delimited fields terminated by ‘,’

Line is terminated by ‘\n’

Stored as ORC;

Load data local inpath ‘/home/pradeep/files/part1’ into table st\_part partition(year=2012);

Load data local inpath ‘/home/pradeep/files/part2’ into table st\_part partition(year=2013);

**Create dynamic partition table**

>set hive exec.dynamic.partition = true

>set hive exec.dynamic.partition.mode = nonstrict;

Create table dy\_part(id int, name String, dept String, year int, sal int) partitioned by(year int)

Row format delimited fields terminated by ‘,’

Line is terminated by ‘\n’

Stored as ORC;

Insert into table dy\_part partition(year) select <list of columns> from table\_name;

Show partitions dy\_part;