K. Praveen kumar

Pyspark Assignment-3

**Basic Spark Commands:**

1. To start spark shell: $bin/spark-shell
2. Read a file from local system: Val data= sc.textfile(“filename”)
3. Create RDD through Parallelizing:

Scala> val num= Array (1,2,3,4,5,6,7,8)

Scala> val newdata= sc.parallelize(num)

1. To count items in RDD: scala> newdata.count()
2. To collect items in RDD: scala> newdata.collect
3. To read first 3 elements from RDD: newdata.take(3)
4. To save output/process data into text file:

Scala> counts.saveAstextFile(“output”)

Here “output” folder is the current path which is been provided.

**Intermediate Spark Commands:**

1. Filter on RDD:

Let’s create a new RDD for items that contain “yes”.

Scala> val DFdata= data.filter(line=>line.contain(“yes”)

1. Chain Operation:

Scala> data.filter(line=>line.contains(“DataFlair”)).count()

Here filter transformation and count action acted together. So it is a chain operation.

1. To Read the first item from RDD: Data.first()
2. To count RDD partitions: scala> data.partitions.length

By default, the minimum number of partitions is 2.

1. To cache a file: scala> data.cache()

The data will not be cached if you run the above function and this can be proved by visiting:

<http://localhost:4040/storage>

RDD will be cached once the action is done. For example:

Scala> data.count()

Scala> data.collect()

**Persist()** is another function which is similar to cache().

Persist allows user to specify an argument determining where the data will be cached, whether in memory, disk or off-heap memory.

Cache() and persist() both function well without any arguments.

**Basic functions of Spark:**

**1.Read():** we can start loading the files in our dataset using the “spark.read.load” command.

Syntax: cases=spark.read.load(“/home/Rahul/projects/sparkdf/coronavirusdataset/case.csv”,format=”csv”,sep=”,” , inferschema=”true”, header=”true”)

To see a Few rows: cases.show()

.toPandas() is a function used to convert spark dataframe into a pandas version.

Syntax: cases.limit(10).toPandas()

**Change Column Names:**

Command to change a single column name:

cases=cases.withColumnRenamed(“infection\_case”,”infection\_sorce”)

For all columns:

cases= cases.toDF(\*[‘case\_id’,’province’,’city’,’group’])

**Select columns:**

We can also select a subset of columns using the select keyword.

cases=cases.select(‘province’,’city’,’confirmed’)

cases.show()

**Sort:**

**Ascending order:**

cases.sort(“confirmed”).show()

**Descending order:**

from pySpark.sql import functions as F

cases.sort(F.desc(“confirmed”)).show()

**CAST:**

We might find scenarios in which pySpark reads a double as an integer or string. In such cases, you can use the cast function to convert types.

Syntax:

From pySpark.sql.types import DoubleType, IntegerType, StringType

cases=cases.withcolumn(‘confirmed’,F.col(‘confirmed’).cast(IntegerType()))

**Filter:**

We can filter a dataframe using AND(&), OR(|) and NOT(~) conditions.

Syntax:

cases.filter((cases.confirmed>10)&(cases.province=”daegue”)).show()

**GROUPBY:**

We can use groupby function with a spark dataframe too. The process is pretty much same as the pandas groupby version with the exception that you will need to import pyspark.sql.functions

Syntax:

From pyspark import functions as F

cases.groupBy ([“province”,”city”]).agg(F.sum(“confirmed”),F.max(“confirmed”)).show()

To change the columns names, use “Alias”

From pyspark import functions as F

cases.groupBy ([“province”,”city”]).agg(F.sum(“confirmed”).alias(“totalconfimed),\

F.max(“confirmed”).alias(“maxfromoneconfirmedcase”)\).show()

**JOINS:**

To start with joins, will need to introduce one more csv file will go with the region file.

regions=spark.read.load((“/home/Rahul/projects/sparkdf/coronavirusdataset/region.csv”,format=”csv”,sep=”,” , inferschema=”true”, header=”true”)

regions.limit(10).toPandas()

we get this information in our cases file by joining the two dataframes.

Syntax:

cases= cases.join(regions,[‘province’,’city’],how=’left’)

cases.limit(10).toPandas()