Spark Cheat Sheet

Spark Initialization in Scala

SparkContext

```
import org.apache.spark.SparkContext

val sc = new SparkContext("local[*]","app1"
```

SparkSession

```
import org.apache.spark.SparkConf
import org.apache.spark.sql.SparkSession

val sparkConf = new SparkConf()
 sparkConf.set("spark.app.name","my first app")
 sparkConf.set("spark.master","local[2]")

 val spark=SparkSession.builder()
 .config(sparkConf)
 .getOrCreate()
```

Read files in Scala

```
 val ordersDf=spark.read
          .format("csv")
          .option("header",true)
          .option("inferSchema",true)
.option("path","C:/Users/Lenovo/Documents/BIG DATA/WEEK11/orders.csv")
           .load

      ordersDf.show()
```

Read files in Python

```
df=spark.read.format("csv") \
 .option"header",”true”) \
 .option("inferSchema",”true”)\
 .option("sep",",") \
 .option("path","/FileStore/tables/Employees-3.csv") \
 .load()

display(df)
```

Read Modes in Scala

```
 val ordersDf=spark.read
          .format("csv")
          .option("header",true)
           .option("mode", "FAILFAST")
          .option("inferSchema",true)
.option("path","C:/Users/Lenovo/Documents/BIG DATA/WEEK11/orders.csv")
           .load
```

PERMISSIVE
Sets all fields to null when it encounters a corrupted record and places all corrupted records in a string column called _corrupt_record
DROPMALFORMED
Drops the row that contains malformed records
FAILFAST
Fails immediately upon encountering malformed records
The default is permissive.

Read Modes in Python

```
df=spark.read.format("csv") \
 .option"header",”true”) \
 .option("inferSchema",”true”)\
 .option("mode", "FAILFAST") \
 .option("sep",",") \
 .option("path","/FileStore/tables/Employees-3.csv") \
 .load()

display(df)
```

Write to Sink in Scala	Write to sink in Python

import org.apache.spark.sql.SaveMode
 	
ordersDf.write
.format("json") //default format is parquet if not specified
.mode(SaveMode.Overwrite) //4 modes:- Append, overwrite, Errorifexists, ignore
.option("path","C:/Users/Lenovo/Documents/BIG DATA/WEEK11/newfolder")
.save()
 	
Default is Errorifexists
	df.write.format("csv") \
	.mode("overwrite") \
	.csv('/FileStore/tables_output/data.csv')

Impose Schema in Scala(StructType)	**Impose Schema in Python**
import org.apache.spark.sql.types.IntegerType import org.apache.spark.sql.types.StringType import org.apache.spark.sql.types.StructType import org.apache.spark.sql.types.StructField import org.apache.spark.sql.types.TimestampType val ordersSchema= StructType(List(StructField("orderid",IntegerType), StructField("orderdate",TimestampType), StructField("customerid",IntegerType), StructField("status",StringType))) val ordersDf=spark.read .format("csv") .schema(ordersSchema) .option("path","C:/Users/Lenovo/Documents/BIG DATA/WEEK11/orders.csv") .load ordersDf.show()	from pyspark.sql.types import StructType,StructField,StringType,IntegerType empSchema=StructType((StructField("empid",IntegerType()), StructField("empname",StringType()), StructField("city",StringType()), StructField("salary",IntegerType()))) df = spark.read.format("csv") \ .option("header","false") \ .schema(empSchema) \ .option("path","/FileStore/tables/EmployeesN.csv") \ .load() df.printSchema() df.show()
Impose Schema in Scala(DDL string)	**Impose Schema in Scala(DDL string)**
val ordersSchema="orderid int, orderdate string, custid int, orderstatus string" val ordersDf=spark.read .format("csv") .schema(ordersSchema) .option("path","C:/Users/Lenovo/Documents/BIG DATA/WEEK11/orders.csv") .load ordersDf.show()	empschema="empid int,empname string,city string,salary double" df=spark.read.format("csv") \ .option("header","false") \ .schema(empschema) \ .option("path","/FileStore/tables/EmployeesN.csv") \ .load() df.printSchema() df.show()
Rename columns in Scala	**Rename columns in Pyspark**
val newDf= ordersDf.withColumnRenamed("order_customer_id", "customer_id")	df=df.withColumnRenamed("id","id_new")

Rename Multiple columns in Scala	Rename Multiple columns in Pyspark
val newDf= ordersDf.withColumnRenamed("order_id", "id") .withColumnRenamed("order_date", "date") .withColumnRenamed("order_customer_id", customer_id") .withColumnRenamed("order_status", "status")	df=df.withColumnRenamed("id","id_new") .withColumnRenamed("name","name_New") .withColumnRenamed("City","City_New")
Rename Multiple columns in Scala(SelectExpr)	**Rename Multiple columns in Pyspark(SelectExpr)**
ordersDf.selectExpr("order_id as id","order_date as date")	df.selectExpr("id as NewId","Name as NewName")
Add columns in Scala	**Add columns in Pyspark**
ordersDf.withColumn("country", lit("india"))	df.withColumn("Country",lit("India"))
ordersDf.withColumn("dblid", col("order_id")*2)	df.withColumn("Incentive",col("salary")*0.2)
Drop column in Scala	**Drop column in Pyspark**
val newDf =countriesDf.drop("REGION")	newdf2=countriesDf2.drop("REGION")
val newDf =countriesDf.drop("ID","REGION")	newdf3=countriesDf2.drop("ID","REGION")
Select columns in Scala	**Select columns in Pyspark**
import org.apache.spark.sql.functions.{col, column,expr} ordersDf.select("order_id”,” order_customer_id", "order_status").show	df.select("id","name","salary")
ordersDf.select(column("order_id"),col("order_date") ,$"order_customer_id",'order_status).show	df.select(col("id"),col("name"))
ordersDf.select(column("order_id"), expr("concat(order_status,'_STATUS')")).show(false)	df.select(col("id"), expr("concat(name,'_STATUS')"))
ordersDf.selectExpr("order_id","order_date" ,"concat(order_status,'_STATUS')")	df.selectExpr("id","name" ,"concat(name,'_STATUS')")
Filter in Scala	**Filter in Pyspark**
ordersDf.filter("weeknum==50")	df.filter(df.id==1)
ordersDf.filter("weeknum>45")	df.filter(df.id>5)
ordersDf.filter("country=='India'")	df.filter(df.city=="PUNE")
ordersDf.filter("country='India' OR country='Italy'")	df.filter((df.id==1) | (df.id==3))
ordersDf.filter(ordersDf("country")==="India" && ordersDf("totalqty")>1000)	df.filter((df.city=="PUNE") & (df.salary>50000))
ordersDf.filter("weeknum!=50")	df.filter(df.id!=1)
ordersDf.filter("country!='India'")	df.filter(df.city!="PUNE")
df.filter(df("salary")>=30000 && df("salary")<=60000).show	df[df["salary"].between(30000,60000)].show()
Sort in Scala	**Sort in Pyspark**
ordersDf.sort("invoicevalue")	df.sort(df.salary)
ordersDf.sort(col("invoicevalue").desc)	df.sort(df.salary.desc())
ordersDf.sort("country","invoicevalue")	df.sort(df.city,df.salary)
ordersDf.sort(col("country").asc,col("invoicevalue").desc)	df.sort(df.city,df.salary.desc())
Remove duplicates in Scala	**Remove duplicates in Pyspark**

ordersDf.distinct()	df.distinct()
ordersDf.dropDuplicates()	df.dropDuplicates()
ordersDf.dropDuplicates("city")	df.dropDuplicates(["city"])
ordersDf.dropDuplicates("name","city")	df.dropDuplicates(["city","salary"])
Union in Scala	**Union in Pyspark**
ordersDf.union(ordersDf)	df.union(df2)
When in Scala	**When in Pyspark**
ordersDf.withColumn("Tier", when(col("city")==="MUMBAI",1).when(col("city")==="PUNE",2).otherwise(0))	df3.withColumn("CityTier",when(col("city")=="Pune",3).when(col("city")=="Delhi",1). when(col("city")=="Mumbai",2).otherwise('na'))
ordersDf.select(col("*"), when(col("city")==="MUMBAI",1).when(col("city")==="PUNE",2).otherwise(0).as("Tier"))	df3.select(col("*"),when(col("city")=="Pune",3) .when(col("city")=="Delhi",1). when(col("city")=="Mumbai",2). otherwise('na').alias("CityTier"))
Contains in Scala	**Contains in Pyspark**
import org.apache.spark.sql.functions.col val filteredDf= countriesDf.where(col("REGION").contains("ST"))	from pyspark.sql.functions import col filteredDf2=countriesDf2.where(col("REGION").contains("ST"))
df.filter(col("empname").like("A%")).show df.filter(col("empname").like("%N")).show df.filter(col("empname").like("%A%")).show	df.filter(col("empname").like("A%")).show df.filter(col("empname").like("%N")).show df.filter(col("empname").like("%A%")).show
Summary in Scala	**Summary in Pyspark**
countriesDf2.describe().show()	countriesDf2.describe().show()
Case Conversion in Scala	**Case Conversion in Pyspark**
import org.apache.spark.sql.functions.{initcap,upper,lower,col} val df2=df.select(initcap(col("data"))) val df2=df.select(upper(col("data"))) val df2=df.select(lower(col("data")))	from pyspark.sql.functions import initcap,col df4.select(initcap(col("data"))).show(truncate=0) df4.select(upper(col("data"))).show(truncate=0) df4.select(lower(col("data"))).show(truncate=0)
Trim in Scala	**Trim in Pyspark**
import org.apache.spark.sql.functions.{lit, ltrim, rtrim, rpad, lpad, trim} countriesDf.select(ltrim(lit(" HELLO ")).as("ltrim"), rtrim(lit(" HELLO ")).as("rtrim"), trim(lit(" HELLO ")).as("trim"), lpad(lit("HELLO"), 3, " ").as("lp"), rpad(lit("HELLO"), 10, " ").as("rp")).show(2) val df2=df.select(upper(col("data"))) val df2=df.select(lower(col("data")))	from pyspark.sql.functions import lit, ltrim, rtrim, rpad, lpad, trim countriesDf2.select(ltrim(lit(" HELLO ")).alias("ltrim"), rtrim(lit(" HELLO ")).alias("rtrim"), trim(lit(" HELLO ")).alias("trim"), lpad(lit("HELLO"), 3, " ").alias("lp"), rpad(lit("HELLO"), 10, " ").alias("rp")).show(2)

Round in Scala	**Round in Pyspark**
import org.apache.spark.sql.functions.{round, bround,col} val roundedDf =countriesDf.select(round(col("SALES"), 1).alias("rounded")) countriesDf.select(round(lit("2.5")), bround(lit("2.5"))).show(2)	from pyspark.sql.functions import lit,round, bround countriesDf2.select(round(lit("2.5")), bround(lit("2.5"))).show(2)
Split in Scala	**Split in Pyspark**
import org.apache.spark.sql.functions.{split,col} newdf.select(split(col("data")," ").alias("words_array")).show splitnewdf.selectExpr("words_array[0]").show	from pyspark.sql.functions import split,col newdf2.select(split(col("data")," ").alias("words_array")).show() splitnewdf.selectExpr("words_array[0]").show()
Size of array in Scala	**Size of array in Pyspark**
import org.apache.spark.sql.functions.{size,col} splitnewdf.select(size(col("words_array"))).show	from pyspark.sql.functions import size,col splitnewdf.select(size(col("words_array"))).show( )
Array contains in Scala	**Array contains in Pyspark**
import org.apache.spark.sql.functions.{array_contains,col } splitnewdf.select(array_contains(col("words_arra y"),"big")).show	from pyspark.sql.functions import array_contains,col splitnewdf.select(array_contains(col("words_arra y"),"big")).show()
Explode in Scala	**Explode in Pyspark**
import org.apache.spark.sql.functions.{explode,col} splitnewdf.withColumn("exploded_words",explod e(col("words_array"))).show(false)	from pyspark.sql.functions import explode,col splitnewdf.withColumn("exploded_words",explo de(col("words_array"))).show(truncate=0)
UDF in Scala	**UDF in Pyspark**
def power3(number:Double):Double = number * number * number spark.udf.register("power3", power3(_:Double):Double) udfExampleDF.selectExpr("power3(num)").show	def power3(double_value): return double_value ** 3
Joins in Scala	**Joins in Pyspark**
val joincondition = ordersDf.col("order_customer_id")===customers Df.col("customer_id")	df1.join(df2,df1.id==df2.id,"inner").show() df1.join(df2,df1.id==df2.id,"left").show() df1.join(df2,df1.id==df2.id,"right").show() df1.join(df2,df1.id==df2.id,"outer").show()

val joinedDf= ordersDf.join(customersDf,joincondition,"inner"). sort("order_customer_id")	
Collect set & list in Scala	**Collect set & list in Pyspark**
import org.apache.spark.sql.functions.{collect_set, collect_list} selectDf.agg(collect_set("Country")).show(false) selectDf.agg(collect_list("Country")).show()	from pyspark.sql.functions import collect_set, collect_list selectDf2.agg(collect_set("Country")).show() selectDf2.agg(collect_list("Country")).show()
Aggregate in Scala	**Aggregate in Pyspark**
ordersDf.select(count("*").as("Rowcount"), sum("Quantity").as("TotalQty"), avg("UnitPrice").as("AvgPrice"), countDistinct("InvoiceNo").as("DistinctInvoices") //method1:- column object expression).show	
ordersDf.selectExpr("count(*) as Rowcount", "sum(Quantity) as TotalQty", "avg(UnitPrice) as AvgPrice", "count(Distinct(InvoiceNo)) as DistinctInvoices" //method2:- string expression).show	ordersdf.selectExpr("count(*) as Rowcount", "sum(Quantity) as TotalQty", "avg(UnitPrice) as AvgPrice", "count(Distinct(InvoiceNo)) as DistinctInvoices").show()
ordersDf.createOrReplaceTempView("sales") //method 3:- spark sql spark.sql("select count(*) as Rowcount,sum(Quantity) as TotalQty,avg(UnitPrice) as AvgPrice,count(Distinct(InvoiceNo)) as DistinctInvoices from sales").show	ordersdf.createOrReplaceTempView("sales") \ spark.sql("select count(*) as Rowcount,sum(Quantity) as TotalQty,avg(UnitPrice) as AvgPrice,count(Distinct(InvoiceNo)) as DistinctInvoices from sales").show()
Grouping Aggregate in Scala	**Grouping Aggregate in Pyspark**
ordersDf.groupBy("country").sum("Quantity").show	df.groupby('city').sum('salary')
ordersDf.groupBy("country","InvoiceNo") .agg(sum("Quantity").as("TotalQty"), sum(expr("Quantity * UnitPrice")).as("InvoiceValue")).show //method1	df.groupby('city').agg(sum('salary').alias('TotalSalary'), max('salary').alias('MaxSalary'),min('salary') ,min('salary').alias('MinSalary'), avg('salary').alias('AvgSalary'))

ordersDf.groupBy("country","InvoiceNo")
.agg(expr("sum(Quantity) as TotalQty"),
expr("sum(Quantity * UnitPrice") as InvoiceValue") //method2
).show	

ordersDf.createOrReplaceTempView("sales") spark.sql("""select country,InvoiceNo,sum(Quantity) as TotalQty, sum(Quantity * UnitPrice) as InvoiceValue from sales group by country,InvoiceNo""").show //method3	
Window Aggregate in Scala	**Window Aggregate in Pyspark**
val RowWindow = Window.partitionBy().orderBy("TotalQty") ordersDf.withColumn("Rownum",row_number().over(RowWindow)).show	window = Window.partitionBy().orderBy("salary") df.withColumn("Rownum",row_number().over(window)).show()
val RowWindow2 = Window.partitionBy().orderBy(col("TotalQty").desc) ordersDf.withColumn("Rownum",row_number().over(RowWindow2)).show	window = Window.partitionBy().orderBy(col("salary").desc()) df.withColumn("Rownum",row_number().over(window)).show()
val RowWindow3 = Window.partitionBy("country").orderBy(col("TotalQty").desc) ordersDf.withColumn("Rownum",row_number().over(RowWindow3)).show	window = Window.partitionBy("city").orderBy(col("salary").desc()) df.withColumn("Rownum",row_number().over(window)).show()
val RowWindow4 = Window.partitionBy("country","weeknum").orderBy(col("TotalQty").desc) ordersDf.withColumn("Rownum",row_number().over(RowWindow4)).show(100)	window = Window.partitionBy("state","city").orderBy(col("salary").desc()) df.withColumn("Rownum",row_number().over(window)).show()
Running Total in Scala	**Running Total in Pyspark**
val RunningWindow = Window.partitionBy().orderBy("country") .rowsBetween(Window.unboundedPreceding,Window.currentRow) ordersDf.withColumn("RunningTotal",sum("invoicevalue").over(RunningWindow)).show	RunningWindow = Window.partitionBy().orderBy("city") \ .rowsBetween(Window.unboundedPreceding,Window.currentRow) df.withColumn("RunningTotal",sum("salary").over(RunningWindow)).show()
val myWindow = Window.partitionBy("country") .orderBy("weeknum") .rowsBetween(Window.unboundedPreceding,Window.currentRow) val myDf = ordersDf.withColumn("RunningTotal",sum("invoicevalue").over(myWindow))	RunningWindow = Window.partitionBy("city").orderBy("city") \ .rowsBetween(Window.unboundedPreceding,Window.currentRow) df.withColumn("RunningTotal",sum("salary").over(RunningWindow)).show()
val myWindow2 = Window.partitionBy() .orderBy("weeknum")	RunningWindow = Window.partitionBy().orderBy("city") \

.rowsBetween(-2,Window.currentRow) ordersDf.withColumn("RunningTotal",sum("invoicevalue").over(myWindow2)).show	.rowsBetween(-2,Window.currentRow) df.withColumn("RunningTotal",sum("salary").over(RunningWindow)).show()
Rank in Scala	**Rank in Pyspark**
val RunningWindow = Window.partitionBy().orderBy("invoicevalue") ordersDf.withColumn("Ranks",rank().over(RunningWindow)).show	RunningWindow = Window.partitionBy().orderBy("salary") df.withColumn("Ranks",rank().over(RunningWindow)).show()
val RunningWindow2 = Window.partitionBy().orderBy(col("invoicevalue").desc) ordersDf.withColumn("Ranks",rank().over(RunningWindow2)).show	RunningWindow = Window.partitionBy().orderBy(col("salary").desc()) df.withColumn("Ranks",rank().over(RunningWindow)).show()
val RunningWindow3 = Window.partitionBy("country").orderBy(col("invoicevalue").desc) ordersDf.withColumn("Ranks",rank().over(RunningWindow3)).show	RunningWindow = Window.partitionBy("city").orderBy(col("salary").desc()) df.withColumn("Ranks",rank().over(RunningWindow)).show()
Dense Rank in Scala	**Dense Rank in Pyspark**
val RunningWindow = Window.partitionBy().orderBy("invoicevalue") ordersDf.withColumn("Ranks",dense_rank().over(RunningWindow)).show	RunningWindow = Window.partitionBy().orderBy("salary") df.withColumn("Ranks",dense_rank().over(RunningWindow)).show()
val RunningWindow2 = Window.partitionBy().orderBy(col("invoicevalue").desc) ordersDf.withColumn("Ranks", dense_rank ().over(RunningWindow2)).show	RunningWindow = Window.partitionBy().orderBy(col("salary").desc()) df.withColumn("Ranks", dense_rank().over(RunningWindow)).show()
val RunningWindow3 = Window.partitionBy("country").orderBy(col("invoicevalue").desc) ordersDf.withColumn("Ranks", dense_rank ().over(RunningWindow3)).show	RunningWindow = Window.partitionBy("city").orderBy(col("salary").desc()) df.withColumn("Ranks", dense_rank().over(RunningWindow)).show()
Repartition in Scala	**Repartition in Pyspark**
val newRdd=inputRDD.repartition(6)	df.repartition(6).write.format("parquet").mode("overwrite").save('/FileStore/tables/Repart')
Coalesce in Scala	**Coalesce in Pyspark**
val newRdd=inputRDD. Coalesce (6)	df. Coalesce (6).write.format("parquet").mode("overwrite").save('/FileStore/tables/Repart')
Partition in Scala	**Partition in Pyspark**
ordersDf.write .format("csv") .partitionBy("order_status") .mode(SaveMode.Overwrite)	df.write.option("header","true").partitionBy("COUNTRY").mode("overwrite").csv("/FileStore/tables/Sample_Partition_op")

.option("path","C:/Users/Lenovo/Documents/BIG DATA/WEEK11/newfolder") .save()	
ordersDf.write .format("csv") .partitionBy("country","order_status") .mode(SaveMode.Overwrite) .option("path","C:/Users/Lenovo/Documents/BIG DATA/WEEK11/newfolder") .save()	df.write.option("header","true").partitionBy("COUNTRY" ,"CITY").mode("overwrite").csv("/FileStore/tables/Sample_Partition_op")
Bucketing in Scala	**Bucketing in Pyspark**
ordersDf.write .format("csv") .mode(SaveMode.Overwrite) .bucketBy(4, "order_customer_id") .sortBy("order_customer_id") .saveAsTable("orders")	df.write.format("csv") \ .mode("overwrite") \ .bucketBy(4, "id") \ .sortBy("id") \ .saveAsTable("orders_bucketed")
Cast Column in Scala	**Cast Column in Pyspark**
val df= ordersDf.withColumn("id", ordersDf("id").cast(IntegerType))	df.withColumn("id",df.id.cast('integer')).withColumn("salary",df.salary.cast('integer'))
ordersDf.select(col("id").cast("int").as("id"),col("name").cast("string").as("name"))	df2.select(col("id").cast('int'),col("name"),col("salary").cast('int'))
ordersDf.selectExpr("cast(id as int)","name","cast(salary as int)")	df3.selectExpr('cast(id as int)','name','cast(salary as int)')
Fill nulls in Scala	**Fill nulls in Pyspark**
df.na.fill(0)	df.na.fill(0)
df.na.fill("none")	df.na.fill("none")
ordersDf.withColumn("order_id",expr("coalesce(order_id,-1)"))	df.withColumn("salary",expr("coalesce(salary,-1)"))
Read directly in Scala	**Read Directly in Pyspark**
spark.sql("select * from csv.`C:/Users/Lenovo/Documents/Employees.csv`")	spark.sql("SELECT * FROM csv.`/user/hive/warehouse/orders_bucketed/part-00000-tid-3984408860399578289-17a5aa99-d1f9-4500-88cf-1adde09ef7fb-19-1_00000.c000.csv`")
Literal in Scala	**Literal in Pyspark**
import org.apache.spark.sql.functions.{lit,expr} val limitCountriesDf=countriesDf.select(expr("*"),lit(1).as("Literalcol")) limitCountriesDf.show(10)	from pyspark.sql.functions import lit,expr limitCountriesDf2=countriesDf2.select(expr("*"),lit(1).alias("Literalcol")) limitCountriesDf2.show(10)

CDU
Cloud And Data Universe