

# 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
```

### 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)
```

#### 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.

Write to Sink in Scala	Write to sink in Python
<pre>import org.apache.spark.sql.SaveMode  ordersDf.write     .format("json") //default format is <u>parquet</u> if not specified     .mode(SaveMode.Overwrite) //4 modes:- Append, overwrite, <u>Errorifexists</u>, ignore .option("path","C:/Users/Lenovo/Documents/BIG DATA/WEEK11/newfolder")     .save()  Default is <u>Errorifexists</u></pre>	<pre>df.write.format("csv") \     .mode("overwrite") \     .csv('/FileStore/tables_output/data.csv')</pre>
Impose Schema in Scala(StructType)	Impose Schema in Python
<pre>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()</pre>	<pre>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()</pre>
Impose Schema in Scala(DDL string)	Impose Schema in Scala(DDL string)
<pre>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()</pre>	<pre>empschema="empid int,empname string,city string,salary double"  df=spark.read.format("csv") \     .option("header","false") \     .schema(empschema) \ .option("path","/FileStore/tables/EmployeesN.cs v") \     .load()  df.printSchema() df.show()</pre>
Rename columns in Scala	Rename columns in Pyspark
<pre>val newDf= ordersDf.withColumnRenamed("order_customer_ id", "customer_id")</pre>	<pre>df=df.withColumnRenamed("id","id_new")</pre>

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

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

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

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

<pre>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</pre>	
Window Aggregate in Scala	Window Aggregate in Pyspark
<pre>val RowWindow = Window.partitionBy().orderBy("TotalQty")  ordersDf.withColumn("Rownum",row_number().o ver(RowWindow)).show</pre>	<pre>window = Window.partitionBy().orderBy("salary") df.withColumn("Rownum",row_number().over(wi ndow)).show()</pre>
<pre>val RowWindow2 = Window.partitionBy().orderBy(col("TotalQty").des c)  ordersDf.withColumn("Rownum",row_number().o ver(RowWindow2)).show</pre>	<pre>window = Window.partitionBy().orderBy(col("salary").desc() )  df.withColumn("Rownum",row_number().over(wi ndow)).show()</pre>
<pre>val RowWindow3 = Window.partitionBy("country").orderBy(col("Tota lQty").desc)  ordersDf.withColumn("Rownum",row_number().o ver(RowWindow3)).show</pre>	<pre>window = Window.partitionBy("city").orderBy(col("salary"). desc())  df.withColumn("Rownum",row_number().over(wi ndow)).show()</pre>
<pre>val RowWindow4 = Window.partitionBy("country","weeknum").order By(col("TotalQty").desc)  ordersDf.withColumn("Rownum",row_number().o ver(RowWindow4)).show(100)</pre>	<pre>window = Window.partitionBy("state","city").orderBy(col(" salary").desc())  df.withColumn("Rownum",row_number().over(wi ndow)).show()</pre>
Running Total in Scala	Running Total in Pyspark
<pre>val RunningWindow = Window.partitionBy().orderBy("country") .rowsBetween(Window.unboundedPreceding,Wi ndow.currentRow)  ordersDf.withColumn("RunningTotal",sum("invoic evalue").over(RunningWindow)).show</pre>	<pre>RunningWindow = Window.partitionBy().orderBy("city") \  .rowsBetween(Window.unboundedPreceding,Wi ndow.currentRow)  df.withColumn("RunningTotal",sum("salary").ove r(RunningWindow)).show()</pre>
<pre>val myWindow = Window.partitionBy("country") .orderBy("weeknum")  .rowsBetween(Window.unboundedPreceding,Wi ndow.currentRow)  val myDf = ordersDf.withColumn("RunningTotal",sum("invoic evalue").over(myWindow))</pre>	<pre>RunningWindow = Window.partitionBy("city").orderBy("city") \  .rowsBetween(Window.unboundedPreceding,Wi ndow.currentRow)  df.withColumn("RunningTotal",sum("salary").ove r(RunningWindow)).show()</pre>
<pre>val myWindow2 = Window.partitionBy() .orderBy("weeknum")</pre>	<pre>RunningWindow = Window.partitionBy().orderBy("city") \</pre>



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



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

