Apache Spark - AAT Notes

Python, Scala, pyspark scala shell and apps.

Arturo Alatriste Trujillo

**Abstract**: this document is a cookbook of easy quick recipes for using Apache Spark, Scala, python, shells, configurations...

Also, there is a section for some topics for databricks cloud servers.

Contents

[Hadoop File System 5](#_Toc506060562)

[How to copy file to Hadoop File System 5](#_Toc506060563)

[How to list files in Hadoop File System 5](#_Toc506060564)

[Spark Shell 5](#_Toc506060565)

[How to load Spark Shell ( Scala ) 5](#_Toc506060566)

[Configure log4j.properties 5](#_Toc506060567)

[Set verbose level 5](#_Toc506060568)

[Spark Applications - Run locally 7](#_Toc506060569)

[How to run a python script 7](#_Toc506060570)

[How to run a scala and java applications 7](#_Toc506060571)

[Compile 7](#_Toc506060572)

[How to run a jar 7](#_Toc506060573)

[Spark Applications - Run in cluster 7](#_Toc506060574)

[Start spark-master service 7](#_Toc506060575)

[Start spark-worker service 7](#_Toc506060576)

[Stop spark-master service 7](#_Toc506060577)

[Stop spark-worker service 7](#_Toc506060578)

[How to run in cluster a python script 7](#_Toc506060579)

[How to run in cluster a scala / java applications 7](#_Toc506060580)

[Monitoring 8](#_Toc506060581)

[Monitor master in internet browser 8](#_Toc506060582)

[Monitor worker in internet browser 8](#_Toc506060583)

[Monitor in YARN Resource Manager 8](#_Toc506060584)

[Monitor in HUE 8](#_Toc506060585)

[Monitor in Apache Spark UI 8](#_Toc506060586)

[Monitor stages in internet browser 8](#_Toc506060587)

[SparkContext and SQLContext 9](#_Toc506060588)

[Import Libraries for SQL and Dataframes 9](#_Toc506060589)

[Python 9](#_Toc506060590)

[Load Data 9](#_Toc506060591)

[How to convert array to parallel data 9](#_Toc506060592)

[How to load a File from Hadoop File System 10](#_Toc506060593)

[How to load file from Databricks File System 10](#_Toc506060594)

[How to load file from File System 10](#_Toc506060595)

[How to count the number of rows in an RDD 10](#_Toc506060596)

[Data Frames 11](#_Toc506060597)

[Convert from dataFrame to RDD 11](#_Toc506060598)

[Convert from RDD to dataFrame 11](#_Toc506060599)

[Print, show rows from a dataFrame 11](#_Toc506060600)

[Display as table in notebook 11](#_Toc506060601)

[Print Show schema 11](#_Toc506060602)

[Show name of columns 11](#_Toc506060603)

[Rename columns 11](#_Toc506060604)

[SELECT (object syntax) 12](#_Toc506060605)

[SELECT (sql syntax) 12](#_Toc506060606)

[Filter, when 12](#_Toc506060607)

[Load data from file 13](#_Toc506060608)

[load from text file 13](#_Toc506060609)

[load from text file – infer schema 13](#_Toc506060610)

[or 13](#_Toc506060611)

[load from text file – specify schema - scala 13](#_Toc506060612)

[load from text file – specify schema-python 13](#_Toc506060613)

[load table from Hive 14](#_Toc506060614)

[create new dataframe with select statement 14](#_Toc506060615)

[Save as csv to file system 15](#_Toc506060616)

[Save as parquet to file system 15](#_Toc506060617)

[Save as csv to dbfs 15](#_Toc506060618)

[Save as parquet to dbfs 15](#_Toc506060619)

[Save dataframe using columns for partitions 15](#_Toc506060620)

[Set Compression Codec 16](#_Toc506060621)

[filtering 17](#_Toc506060622)

[Aggregation function on DataFrame 18](#_Toc506060623)

[Aggregation function on DataFrame – Count number of rows 18](#_Toc506060624)

[Aggregation function on DataFrame – Sum values in column 18](#_Toc506060625)

[Aggregation function on DataFrame – count number of null values in each column (object syntax) 18](#_Toc506060626)

[Aggregation function on DataFrame – count number of null values in each column (SQL syntax) 19](#_Toc506060627)

[UDF 19](#_Toc506060628)

[Column 20](#_Toc506060629)

[Convert a column from string to category 20](#_Toc506060630)

[Dates 20](#_Toc506060631)

[dayofmonth 20](#_Toc506060632)

[Conversions 20](#_Toc506060633)

[Convert String to Date – object syntax 20](#_Toc506060634)

[Convert String to Date – sql syntax 20](#_Toc506060635)

[Databricks 21](#_Toc506060636)

[Upload a remote file to Databricks File System 21](#_Toc506060637)

[curl 21](#_Toc506060638)

[wget 21](#_Toc506060639)

[Checkout where it saves it 21](#_Toc506060640)

[Copy files in Databricks Cloud 21](#_Toc506060641)

[from file system to dbfs. 21](#_Toc506060642)

[from dbfs to file system. 21](#_Toc506060643)

[Download from databricks to my local computer 21](#_Toc506060644)

[How to load file from databricks file system 21](#_Toc506060645)

[RDD 22](#_Toc506060646)

[Filter only salmon 22](#_Toc506060647)

[Filter an RDD of tuples. Animals with 6 letters. 22](#_Toc506060648)

[references 23](#_Toc506060649)

[sql 23](#_Toc506060650)

[udfs 23](#_Toc506060651)

[Dates 23](#_Toc506060652)

[File Storage 24](#_Toc506060653)

[Data Frames 24](#_Toc506060654)

[Getting Started (include predefined variables) 24](#_Toc506060655)

[Convert column from string to category 24](#_Toc506060656)

[Data Rows 24](#_Toc506060657)

[RDD 24](#_Toc506060658)

[Old Spark Dataframes 24](#_Toc506060659)

# Hadoop File System

## How to copy file to Hadoop File System

$ hadoop fs -copyFromLocal README.txt /user/cloudera/README.txt

## How to list files in Hadoop File System

$ hadoop fs -ls /user/cloudera

# Spark Shell

## How to load Spark Shell ( Scala )

Find the spark-shell path, and type something like this

$ /usr/bin/spark-shell

Or

$ ./usr/bin/spark-shell

You will get something like

scala>

# Configure log4j.properties

## Set verbose level

Set the property rootCategory to any of this levels

* ALL
* TRACE
* DEBUG
* INFO
* WARN
* ERROR
* FATAL
* OFF

See example below

# Set everything to be logged to the console

#log4j.rootCategory=INFO, console

#log4j.rootCategory=DEBUG, console

log4j.rootCategory=WARN, console

log4j.appender.console=org.apache.log4j.ConsoleAppender

log4j.appender.console.target=System.err

log4j.appender.console.layout=org.apache.log4j.PatternLayout

log4j.appender.console.layout.ConversionPattern=%d{yy/MM/dd HH:mm:ss} %p %c{1}: %m%n

# Settings to quiet third party logs that are too verbose

log4j.logger.org.eclipse.jetty=WARN

log4j.logger.org.eclipse.jetty.util.component.AbstractLifeCycle=ERROR

log4j.logger.org.apache.spark.repl.SparkIMain$exprTyper=ERROR

log4j.logger.org.apache.spark.repl.SparkILoop$SparkILoopInterpreter=ERROR

# Spark Applications - Run locally

## How to run a python script

$ spark-submit wordcount.py shakespeare.txt

## How to run a scala and java applications

### Compile

$ mvn package

### How to run a jar

$ spark-submit

--class myFolder.WordCount

Wordcount-1.0.jar shakespeare.txt

# Spark Applications - Run in cluster

To run your spark application in the cluster, include the parameter master

## Start spark-master service

$ sudo service spark-master start

## Start spark-worker service

$ sudo service spark-worker start

## Stop spark-master service

$ sudo service spark-master stop

## Stop spark-worker service

$ sudo service spark-worker stop

## How to run in cluster a python script

$ spark-submit

--master spark://localhost/7077

--name ‘Count JPG images requests’

count\_jpg.py /loudacre/weblogs/\*

## How to run in cluster a scala / java applications

$ spark-submit

--class stub.CountJPGs

--master spark://localhost/7077

--name ‘Count JPG images requests’

target/countjpgs-1.0.jar /loadacre/weblogs/\*

# Monitoring

## Monitor master in internet browser

http://localhost:18080

## Monitor worker in internet browser

http://localhost:18081

## Monitor in YARN Resource Manager

<http://localhost:4040>

## Monitor in HUE

<http://localhost:8888>

## Monitor in Apache Spark UI

http://localhost:4040

## Monitor stages in internet browser

<http://localhost:4040>

<http://localhost:4040/stages>

# SparkContext and SQLContext

|  |  |  |
| --- | --- | --- |
| Create | scala | Python |
| sparkContext | import org.apache.spark.SparkContext  import org.apache.spark.SparkContext.\_  import org.apache.spark.SparkConf  val conf = new SparkConf().setAppName("Simple Application")  val sc = new SparkContext(conf) | from pyspark import SparkContext, SparkConf  # Create configuration object and set App name  conf = SparkConf().setAppName( "CCA 175 Problem 84" )  sc = SparkContext( conf=conf ) |
| sqlContext | // SQLContext entry point for working with structured data  val sqlContext = new org.apache.spark.sql.SQLContext( sc )  // this is used to implicitly convert an RDD to a DataFrame.  import sqlContext.impIicits.\_  // Import Spark SQL data types and Row.  import org.apache.spark.sql.\_  // Import Spark SQL functions.  import org.apache.spark.sql.functions | from pyspark import SparkContext  from pyspark.sql import SQLContext, Row  sqlContext = SQLContext( sc ) |

# Import Libraries for SQL and Dataframes

SCALA

import org.apache.spark.SparkContext

import org.apache.spark.SparkContext.\_

import org.apache.spark.SparkConf

## Python

from pyspark import SparkContext, SparkConf

# Load Data

## How to convert array to parallel data

val data = Array(1, 2, 3, 4, 5)

val distData = sc.parallelize(data)

## How to load a File from Hadoop File System

First copy your file to Hadoop File System, as shown above.

val txt\_df = sc.textFile( “hdfs://quickstart.cloudera:8020 /user/cloudera/README.txt” )

### How to load file from Databricks File System

val log\_file\_path = "dbfs:/databricks-datasets/cs100/lab2/data-001/apache.access.log.PROJECT"

val base\_df = sqlContext.read.text( log\_file\_path )

### How to load file from File System

val log\_file\_path = " file:/databricks/driver/books.txt"

val base\_df = sqlContext.read.text( log\_file\_path )

## How to count the number of rows in an RDD

scala> txt\_df.count()

# Data Frames

## Convert from dataFrame to RDD

df.rdd

## Convert from RDD to dataFrame

import sqlContextimplicits.\_

val df\_1 = rdd.toDF()

val df\_2 = SparkSession.createDataFrame( rdd )

## Print, show rows from a dataFrame

df.show( n = num\_rows, truncate = False )

// do not truncate columns to fit the screen

df.show( truncate = false)

## Display as table in notebook

display( myDataFrame )

## Print Show schema

myDataFrame.printSchema()

## Show name of columns

myDataFrame.columns

## Rename columns

First use printSchema to see the names of the columns

myDataFrame.printSchema()

output:

root:

\_1:string( nullable = true )

\_2:string( nullable = true )

\_3:string( nullable = true )

Then use “as” to rename the column as we use to do in sql.

myDataFrame.selectExpr( “\_1 as name”, “\_2 as email”, “\_3 as favoriteBook” )

next check using printSchema

myDataFrame.printSchema()

output:

root:

name :string( nullable = true )

email :string( nullable = true )

favouriteBook:string( nullable = true )

## SELECT (object syntax)

val df\_1 = df.select( "name" )

val df\_2 = df.select( $"name", $"age" + 1 )

val df\_4 = df.select( $"name".alias( “NAME” ), $"age".alias( “AGE” ) )

## SELECT (sql syntax)

// register table to make queries

df.createOrReplaceTempView( "df" )

// execute query

val df\_1 = sqlContext.sql(

"""

SELECT \*

FROM df

WHERE

city = ‘Bangkok’

""" )

// the number of bad\_rows should be cero

df\_1.show()

## Filter, when

// year-month-day

df.filter( " my\_time\_col = '1995-08-03' " ).show()

## Load data from file

## load from text file

val base\_df = sqlContext.read.text( log\_file\_path )

## load from text file – infer schema

spark.read.csv(

"some\_input\_file.csv", header=True, mode="DROPMALFORMED", schema=schema

)

## or

(spark.read

.schema(schema)

.option("header", "true")

.option("mode", "DROPMALFORMED")

.csv("some\_input\_file.csv"))

## load from text file – specify schema - scala

f

## load from text file – specify schema-python

from pyspark.sql.types import StructType, StructField

from pyspark.sql.types import DoubleType, IntegerType, StringType

schema = StructType([

StructField("A", IntegerType()),

StructField("B", DoubleType()),

StructField("C", StringType())

])

(sqlContext

.read

.format("com.databricks.spark.csv")

.schema(schema)

.option("header", "true")

.option("mode", "DROPMALFORMED")

.load("some\_input\_file.csv"))

## load table from Hive

// use a spark session variable, here is spark.

// then the sql method and write your query

val diamonds = spark.sql("SELECT \* FROM diamonds")

// in databricks you can call directly the display command

display( diamonds )

display(diamonds.select("\*"))

## create new dataframe with select statement

val new\_df = df.select( )

## Save as csv to file system

df.write.format( "com.databricks.spark.csv" ).save( "file:/tmp1/results.csv" )

## Save as parquet to file system

df.write.format( "parquet" ).save( "file:/tmp1/ results.parquet" )

## Save as csv to dbfs

df.write.format( "com.databricks.spark.csv" ).save( "/tmp1/results.csv" )

## Save as parquet to dbfs

df.write.format( "parquet" ).save( " /tmp1/ results.parquet" )

## Save dataframe using columns for partitions

df

.write

.partitionBy( "end\_year", "end\_month" )

.parquet( "/tmp/sample\_table" )

## Set Compression Codec

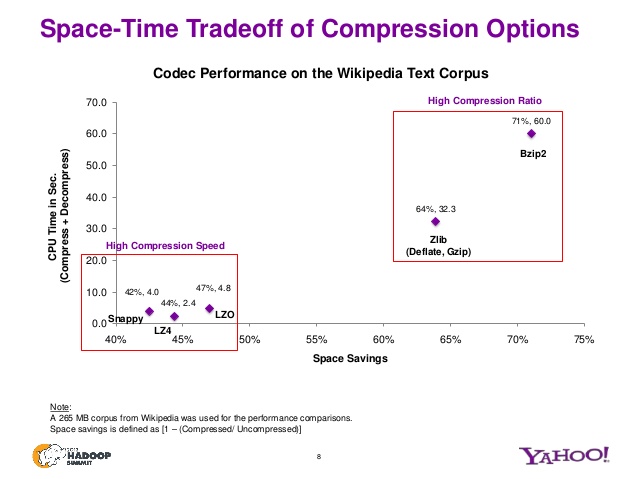
# Accepted Compression Codec values: uncompressed, snappy, gzip, lzo

# set Compression Codec

sqlContext.setConf("spark.sql.parquet.compression.codec", "gzip")

# save data

df.write.format( "parquet" ).save( "/myFolder/data.parquet" )



## filtering

// suppose a dataframe df, with columns name, age, address, tel, sex, etc...

// show all the girls

df.filter( “sex == ‘woman’ ).count()

// count the number of adults

df.filter( “age > 18” ).count()

// show people without address

df.filter( “address is null” ).show

## Aggregation function on DataFrame

val status\_to\_count\_df = logs\_df

.groupBy( "status" )

.count()

.sort( "status" )

.cache()

Found 7 response codes

+------+------+

|status| count|

+------+------+

| 200 |940847|

| 302 | 16244|

| 304 | 79824|

| 403 | 58 |

| 404 | 6185 |

| 500 | 2 |

| 501 | 17 |

+------+------+

## Aggregation function on DataFrame – Count number of rows

val content\_size\_stats = logs\_df.agg(

min( $"content\_size" ),

avg( $"content\_size" ),

max( $"content\_size" ) )

.first()

## Aggregation function on DataFrame – Sum values in column

//

## Aggregation function on DataFrame – count number of null values in each column (object syntax)

import org.apache.spark.sql.functions.{sum,when}

split\_df.agg(

sum( when( $"host" .isNull, 1 ).otherwise( 0 ) ).as( "host" ),

sum( when( $"timestamp" .isNull, 1 ).otherwise( 0 ) ).as( "timestamp" ),

sum( when( $"path" .isNull, 1 ).otherwise( 0 ) ).as( "path" ),

sum( when( $"status" .isNull, 1 ).otherwise( 0 ) ).as( "status" ),

sum( when( $"content\_size" .isNull, 1 ).otherwise( 0 ) ).as( "content\_size" )

).show()

## Aggregation function on DataFrame – count number of null values in each column (SQL syntax)

val bad\_rows\_df3 = sqlContext.sql(

"""

SELECT

sum ( CASE WHEN host is null THEN 1 ELSE 0 END ) as host,

sum ( CASE WHEN timestamp is null THEN 1 ELSE 0 END ) as timestamp,

sum ( CASE WHEN path is null THEN 1 ELSE 0 END ) as path,

sum ( CASE WHEN status is null THEN 1 ELSE 0 END ) as status,

sum ( CASE WHEN content\_size is null THEN 1 ELSE 0 END ) as content\_size

FROM split\_df

""" )

// the number of bad\_rows should be cero

bad\_rows\_df3.show()

## UDF

// create a Scala function (only if you want)

def upper\_fun( s : String ) : String =

{

println( "running upper\_fun" )

val r = s.toUpperCase

return r

}

// create a scala expression

val upper = (s : String) => upper\_fun( s )

// create udf

val upperUDF = udf(upper)

cleaned\_df.withColumn( "HOST\_Upper", upperUDF( $"host" ) ).show( )

# Column

### Convert a column from string to category

import org.apache.spark.ml.feature.StringIndexer

val df = spark.createDataFrame(

Seq((0, "a"), (1, "b"), (2, "c"), (3, "a"), (4, "a"), (5, "c"))

).toDF("id", "category")

val indexer = new StringIndexer()

.setInputCol("category")

.setOutputCol("categoryIndex")

val indexed = indexer.fit(df).transform(df)

indexed.show()

# Dates

### dayofmonth

val day\_to\_host\_pair\_df = logs\_df

.select( $"host", dayofmonth( $"time" ).alias( "day" ) )

## Conversions

The string is in the yyyy-MM-dd format.

### Convert String to Date – object syntax

/// import org.apache.spark.sql.functions.unix\_timestamp

cleaned\_month.select(

$"host" ,

unix\_timestamp( $"string\_column", "dd/MM/yyyy").cast("timestamp").alias( "new\_column\_timestamp" ),

$"path" ,

$"status" ,

$"content\_size"

)

### Convert String to Date – sql syntax

val tmp1 = sqlContext.sql(

"""

SELECT

host,

TO\_DATE(CAST(UNIX\_TIMESTAMP( string\_column, 'dd/MM/yyyy') AS TIMESTAMP)) AS new\_column\_timestamp,

path,

status,

content\_size

FROM cleaned\_month

"""

)

# Databricks

<https://community.cloud.databricks.com>

## Upload a remote file to Databricks File System

### curl

%sh curl -O 'https://github.com/databricks/spark-xml/raw/master/src/test/resources/books.xml'

### wget

%fs wget http://my\_file\_name.txt

### Checkout where it saves it

%fs ls "file:/databricks/driver"

## Copy files in Databricks Cloud

### from file system to dbfs.

dbutils.fs.cp( "file:/foobar/baz.txt", "dbfs:/mydir/baz.txt" )

### from dbfs to file system.

dbutils.fs.cp( "dbfs:/mydir/baz.txt", "file:/foobar/baz.txt" )

## Download from databricks to my local computer

The file must exist in dbfs:FileStore

Open web browser and paste the url

Example:

Databricks file path: dbfs:/FileStore/aat/p1.txt

url in web browser: <https://community.cloud.databricks.com/files/aat/p1.txt>

or

<https://community.cloud.databricks.com/files/aat/p1.txt?o=8566450620527647>

the o= parameter in url is the same that you have in your Community Edition ☺

### How to load file from databricks file system

val log\_file\_path = "dbfs:/databricks-datasets/cs100/lab2/data-001/apache.access.log.PROJECT"

val base\_df = sqlContext.read.text( log\_file\_path )

# RDD

## Filter only salmon

val a = sc.parallelize( List( "dog", "salmon", "salmon", "rat", "elephant") )

a.filter( animal => animal == “salmon” ).collect

a.filter( \_ == “salmon” ).collect

output

Array[String] = Array( salmon, salmon )

## Filter an RDD of tuples. Animals with 6 letters.

val b = a.keyBy( \_.length ) b.take( 10 )

output

Array[(Int, String)] = Array((3,dog), (6,salmon), (6,salmon), (3,rat), (8,elephant))

b.filter( {case (size: Int, animal: String ) => size == 6 } ).take( 20 )

b.filter( \_.\_1 == 6 } ).take( 20 )

res14: Array[(Int, String)] = Array((6,salmon), (6,salmon))

# references

## sql

<http://stackoverflow.com/questions/39727742/how-to-filter-out-a-null-value-from-spark-dataframe>

<http://stackoverflow.com/questions/41765739/count-the-number-of-non-null-values-in-a-spark-dataframe>

<http://stackoverflow.com/questions/5487892/sql-server-case-when-or-then-else-end-the-or-is-not-supported>

<http://stackoverflow.com/questions/30783517/apache-spark-add-an-case-when-else-calculated-column-to-an-existing-d>

## udfs

<https://docs.databricks.com/spark/latest/spark-sql/udf-scala.html>

<https://jaceklaskowski.gitbooks.io/mastering-apache-spark/content/spark-sql-udfs.html>

## Dates

<http://spark.apache.org/docs/latest/api/scala/index.html#org.apache.spark.sql.functions$>

<http://stackoverflow.com/questions/40763796/convert-date-from-string-to-date-format-in-dataframes>

<http://stackoverflow.com/questions/36948012/how-to-change-the-column-type-from-string-to-date-in-dataframes>

<http://stackoverflow.com/questions/40844171/scala-convert-string-to-date-in-apache-spark>

<http://stackoverflow.com/questions/29844144/better-way-to-convert-a-string-field-into-timestamp-in-spark>

<http://stackoverflow.com/questions/34408183/spark-scala-dataframe-timestamp-conversion-sorting>

## File Storage

<https://docs.databricks.com/user-guide/advanced/filestore.html>

<https://docs.databricks.com/user-guide/dbfs-databricks-file-system.html>

## Data Frames

<https://stackoverflow.com/questions/29383578/how-to-convert-rdd-object-to-dataframe-in-spark>

## Getting Started (include predefined variables)

<https://docs.databricks.com/user-guide/getting-started.html>

## Convert column from string to category

<https://spark.apache.org/docs/latest/ml-features.html#stringindexer>

## Data Rows

<https://stackoverflow.com/questions/35720330/getting-specific-field-from-chosen-row-in-pyspark-dataframe>

## RDD

<http://homepage.cs.latrobe.edu.au/zhe/ZhenHeSparkRDDAPIExamples.html>

<https://spark.apache.org/docs/1.6.1/api/python/pyspark.html>

## Old Spark Dataframes

<https://spark.apache.org/docs/1.6.0/sql-programming-guide.html>