pySpark Cheatsheet

Table of Contents

[Command Shell Execution Within iPython / jupyter notebook 2](#_Toc12070696)

[Configure Spark from a jupyter notebook command 3](#_Toc12070697)

[Copy a file from another user’s Directory 3](#_Toc12070698)

[Delete an HDFS directory using jupyter notebook 3](#_Toc12070699)

[Examine Underlying HDFS Directory Structures in a Jupyter Notebook 3](#_Toc12070700)

[Set the spark timeout value 5](#_Toc12070701)

[Transfer a CSV file to HDFS 5](#_Toc12070702)

[DataFrames (pyspark.sql.dataframe.DataFrame) 6](#_Toc12070703)

[Add a Column Based on Another Column 6](#_Toc12070704)

[Append an Array to a pySpark DataFrame 6](#_Toc12070705)

[Assign each Row a new ID 6](#_Toc12070706)

[Calculate Statistics on a DataFrame Column 6](#_Toc12070707)

[Change a DataFrame’s Column Names 7](#_Toc12070708)

[Concatenate 2 pySpark DataFrames, row-wise 7](#_Toc12070709)

[Convert a Pandas DataFrame to a pySpark DataFrame 8](#_Toc12070710)

[Convert a tab-delimited File to a pySpark DataFrame 8](#_Toc12070711)

[Convert a pySpark DataFrame to a Pandas DataFrame 9](#_Toc12070712)

[Convert a Timestamp Column to a Date Column 9](#_Toc12070713)

[Count the number of distinct values for a column 10](#_Toc12070714)

[Count the number of Missing Observations in each Column 10](#_Toc12070715)

[Create a DataFrame from Scratch 11](#_Toc12070716)

[Create a new pySpark DataFrame Column from Existing Columns 11](#_Toc12070717)

[Display DataFrame Columns and Types 12](#_Toc12070718)

[DataFrame Schema 12](#_Toc12070719)

[Delete a Column from a pySpark DataFrame 13](#_Toc12070720)

[Drop Duplicate Rows 13](#_Toc12070721)

[Drop Duplicate Rows which are Identical except for the Index 14](#_Toc12070722)

[Drop Rows which have a certain number of Columns with missing values 14](#_Toc12070723)

[Drop Rows which have a null in Specific Fields 15](#_Toc12070724)

[Execute SQL on a DataFrame 15](#_Toc12070725)

[Execute multiple functions on a DataFrame 15](#_Toc12070726)

[Explode a List inside a DataFrame column 15](#_Toc12070727)

[Explode an Array inside a DataFrame, but rename the new column 16](#_Toc12070728)

[Export a Spark Data Frame to CSV 16](#_Toc12070729)

[Extract a column from a pyspark DataFrame as a List 16](#_Toc12070730)

[Extract a struct (structure) as a DataFrame 17](#_Toc12070731)

[Extract Distinct Values from a DataFrame Column 17](#_Toc12070732)

[Extract Nested JSON Data in Spark 17](#_Toc12070733)

[Fill in missing data in a pySpark DataFrame by imputing the means 17](#_Toc12070734)

[Filter in rows having values contained in a list 18](#_Toc12070735)

[Get distinct values from a column 18](#_Toc12070736)

[groupBy usage with pySpark DataFrames 18](#_Toc12070737)

[groupBy and keep only the max value on the group 19](#_Toc12070738)

[Join two pySpark DataFrames 19](#_Toc12070739)

[Example with mismatched column names 21](#_Toc12070740)

[Loading a Schema while Correcting org.apache.parquet.io.ParquetDecodingException 21](#_Toc12070741)

[Read a DataFrame from a csv file 21](#_Toc12070742)

[Example 1 21](#_Toc12070743)

[Example 2 22](#_Toc12070744)

[Register a DataFrame as a Temporary Table 22](#_Toc12070745)

[Row Count of a DataFrame 23](#_Toc12070746)

[Sample a DataFrame (approximate sample) 23](#_Toc12070747)

[selectExpr on a DataFrame 23](#_Toc12070748)

[Show a DataFrame without Truncating Columns 23](#_Toc12070749)

[Show the percentage of missing observations in each column of a pySpark DataFrame 24](#_Toc12070750)

[Sort a DataFrame 24](#_Toc12070751)

[Specify the schema programmatically 25](#_Toc12070752)

[Statistics on pyspark.sql.dataframe.DataFrame columns 26](#_Toc12070753)

[Calculate Mean and Median of a Column 26](#_Toc12070754)

[Statistics Summary for a pySpark DataFrame 26](#_Toc12070755)

[pyspark.sql 26](#_Toc12070756)

[Recover from a Lost Connection 27](#_Toc12070757)

[SparkSQL 27](#_Toc12070758)

[Case Statements 27](#_Toc12070759)

[DataFrame (pyspark.sql.dataframe.DataFrame) 27](#_Toc12070760)

[Read a Parquet File into a DataFrame 27](#_Toc12070761)

[Store DataFrame as Temp Table for Later Querying 28](#_Toc12070762)

[Select an Array Element 28](#_Toc12070763)

[Invoke a python function from an SQL query 28](#_Toc12070764)

[Parquet 30](#_Toc12070765)

[Explode a list or array data field 30](#_Toc12070766)

[Strings 32](#_Toc12070767)

[Concatenate strings 32](#_Toc12070768)

[Convert a numeric field to a string 32](#_Toc12070769)

[StringIndexer 32](#_Toc12070770)

[Substring 32](#_Toc12070771)

[Timestamps 33](#_Toc12070772)

[Calculate time difference (seconds) between two timestamps 33](#_Toc12070773)

[Convert string to timestamp 33](#_Toc12070774)

[Convert Unix Timestamp to Formatted Date 34](#_Toc12070775)

[Convert Unix Timestamp to ISO Standard Date and Time 35](#_Toc12070776)

[Get Calendar Parts like Year, Month, Day, Hour from a Timestamp 35](#_Toc12070777)

[Appendix 1: pySpark DataTypes: 36](#_Toc12070778)

[Appendix 2 : Identifying Outliers 36](#_Toc12070779)

# Command Shell Execution Within iPython / jupyter notebook



## Configure Spark from a jupyter notebook command

Example: Set the Spark timeout value

# Configure the timeout to allow a long-winded function to execute

spark.conf.set("spark.sql.broadcastTimeout", 43200) # 12 hours

## Copy a file from another user’s Directory

%sx cp /home/eburas/Ocean/march\_april\_linescapemerged\_AIS.csv /home/bbeauchamp/ocean\_models/port\_congestion

## Delete an HDFS directory using jupyter notebook



## Examine Underlying HDFS Directory Structures in a Jupyter Notebook

%sx hdfs dfs -ls ../../lambda

Output:

['Found 6 items',

'drwxr-xr-x - saviops saviops 0 2017-08-09 16:59 ../../lambda/UNKNOWN',

'drwxr-xr-x - saviops saviops 0 2017-08-23 03:44 ../../lambda/definitions-parquet',

'drwxr-xr-x - saviops saviops 0 2017-07-31 13:06 ../../lambda/smf-json',

'drwxr-xr-x - saviops saviops 0 2017-08-23 03:25 ../../lambda/smf-parquet',

'drwxr-xr-x - hdfs saviops 0 2017-06-28 15:44 ../../lambda/summaries-json',

'drwxr-xr-x - saviops saviops 0 2017-08-23 04:11 ../../lambda/summaries-parquet']

Logout[Control Panel](https://10.200.4.13/hub/home)Untitled Last Checkpoint: a few seconds ago (autosaved)

pySpark Basic (Spark 1.4.1)

* [File](https://10.200.4.13/user/bbeauchamp/notebooks/Untitled.ipynb)
* [Edit](https://10.200.4.13/user/bbeauchamp/notebooks/Untitled.ipynb)
* [View](https://10.200.4.13/user/bbeauchamp/notebooks/Untitled.ipynb)
* [Insert](https://10.200.4.13/user/bbeauchamp/notebooks/Untitled.ipynb)
* [Cell](https://10.200.4.13/user/bbeauchamp/notebooks/Untitled.ipynb)
* [Kernel](https://10.200.4.13/user/bbeauchamp/notebooks/Untitled.ipynb)
* [Help](https://10.200.4.13/user/bbeauchamp/notebooks/Untitled.ipynb)

CodeMarkdownRaw NBConvertHeading

Cell Toolbar:NoneEdit MetadataRaw Cell FormatSlideshow

In [1]:

**print** "test"

test

In [17]:

**%**sx ls

Out[17]:

['derby.log',

'Getting Started with SparkSQL.ipynb',

'metastore\_db',

'output2.csv',

'Untitled.ipynb']

In [8]:

**%**sx hdfs dfs **-**ls

Out[8]:

['Found 2 items',

'-rw-r--r-- 3 bbeauchamp bbeauchamp 185038 2016-03-04 14:03 output\_file.json',

'drwxr-xr-x - bbeauchamp bbeauchamp 0 2016-03-04 14:25 test\_parquet.txt']

In [18]:

test = sqlContext.read.json("output\_file.json")

In [19]:

**print** test.take(1)

[Row(CORRIDOR=u'1701-PB340', DRIVER\_ETA\_MINS\_REM=369.0, HOUR\_OF\_DAY=19.0, KM\_REMAINING=139.03697714753346, LATITUDE=33.543056, LAT\_LON\_CORR=u'33.40\_-82.00\_33.58\_-84.52', LONGITUDE=-83.01861099999999, MINS\_REMAINING=95.0, SHIPMENT\_ID=u'304716814', carrier=u'USXI', currentLocationTime=u'2015-02-07T19:51:00.000Z', dest\_lat=33.576, dest\_lon=-84.5154, hour\_of\_day\_local=14.0, plannedArrivalTime=u'2015-02-08T02:00:00.000Z')]

In [6]:

test.printSchema()

root

|-- CORRIDOR: string (nullable = true)

|-- DRIVER\_ETA\_MINS\_REM: double (nullable = true)

|-- HOUR\_OF\_DAY: double (nullable = true)

|-- KM\_REMAINING: double (nullable = true)

|-- LATITUDE: double (nullable = true)

|-- LAT\_LON\_CORR: string (nullable = true)

|-- LONGITUDE: double (nullable = true)

|-- MINS\_REMAINING: double (nullable = true)

|-- SHIPMENT\_ID: string (nullable = true)

|-- carrier: string (nullable = true)

|-- currentLocationTime: string (nullable = true)

|-- dest\_lat: double (nullable = true)

|-- dest\_lon: double (nullable = true)

|-- hour\_of\_day\_local: double (nullable = true)

|-- plannedArrivalTime: string (nullable = true)

In [7]:

test.write.save("test\_parquet.txt", format="parquet")

In [9]:

df = sqlContext.read.parquet("test\_parquet.txt")

In [10]:

df.count()

​

Out[10]:

408

In [11]:

**print** df

DataFrame[CORRIDOR: string, DRIVER\_ETA\_MINS\_REM: double, HOUR\_OF\_DAY: double, KM\_REMAINING: double, LATITUDE: double, LAT\_LON\_CORR: string, LONGITUDE: double, MINS\_REMAINING: double, SHIPMENT\_ID: string, carrier: string, currentLocationTime: string, dest\_lat: double, dest\_lon: double, hour\_of\_day\_local: double, plannedArrivalTime: string]

In [12]:pdf = df.toPandas()

In [13]:type(pdf)

​

Out[13]:pandas.core.frame.DataFrame

In [14]:**import** pandas **as** pd

​

In [15]:pdf.to\_csv("output2.csv", header=0)

​

## Set the spark timeout value

# Configure the timeout to allow a long-winded function to execute

spark.conf.set("spark.sql.broadcastTimeout", 43200) # 12 hours

## Transfer a CSV file to HDFS

file name after upload via jupyter filename in HDFS home

%sx hdfs dfs -put /home/bbeauchamp/pg\_lanes\_export\_2017-05-18.csv pg\_lanes\_export\_2017-05-18.csv

# DataFrames (pyspark.sql.dataframe.DataFrame)

## Add a Column Based on Another Column

Note: timDiff is type pyspark.sql.column.Column

timeFmt = "yyyy-MM-dd'T'HH:mm:ss.SSS"

timeDiff = (unix\_timestamp('port\_end\_ts', format=timeFmt)

- unix\_timestamp('port\_start\_ts', format=timeFmt))

df = df.withColumn("Duration", timeDiff)

df = df.withColumn("Duration\_hrs", timeDiff / 3600.0)

Output:

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

| shipment\_id| port\_start\_ts| port\_end\_ts|Duration| Duration\_hrs|

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

| 27690030954076|2018-03-11T18:26:...|2018-03-13T08:14:...| 136058| 37.79388888888889|

|27690031135535\_YM...|2018-02-20T18:55:...|2018-02-22T01:00:...| 108287|30.079722222222223|

|27690030954052\_YM...|2018-03-03T21:57:...|2018-03-05T08:20:...| 123786| 34.385|

|27690030954069\_YM...|2018-03-05T20:49:...|2018-03-07T18:51:...| 165738|46.038333333333334|

|27690030699786\_YM...|2018-02-04T19:59:...|2018-02-06T09:26:...| 134809| 37.44694444444445|

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

## 

## Append an Array to a pySpark DataFrame

Do this if you want to explode the items in the Array, in a later step

sequential\_dates\_list\_udf = udf(sequential\_date\_list, ArrayType(StringType()))

vessel\_dates\_df = vessel\_dt\_df.withColumn("port\_transit\_dates",

sequential\_dates\_list\_udf(vessel\_dt\_df.mooring\_start\_dt,

vessel\_dt\_df.mooring\_end\_dt) )

## Assign each Row a new ID

df.withColumn('new\_id', fn.monotonically\_increasing\_id()).show()

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

| id|weight|height|age|gender| new\_id|

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

| 5| 133.2| 5.7| 54| F| 25769803776|

| 4| 144.5| 5.9| 33| M| 171798691840|

| 2| 167.2| 5.4| 45| M| 592705486848|

| 3| 124.1| 5.2| 23| F|1236950581248|

| 5| 129.2| 5.3| 42| M|1365799600128|

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

## Calculate Statistics on a DataFrame Column

import numpy as np

df.show()

Output:

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

| shipment\_id| port\_start\_ts| port\_end\_ts|Duration| Duration\_hrs|

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

| 27690030954076|2018-03-11T18:26:...|2018-03-13T08:14:...| 136058| 37.79388888888889|

|27690031135535\_YM...|2018-02-20T18:55:...|2018-02-22T01:00:...| 108287|30.079722222222223|

|27690030954052\_YM...|2018-03-03T21:57:...|2018-03-05T08:20:...| 123786| 34.385|

|27690030954069\_YM...|2018-03-05T20:49:...|2018-03-07T18:51:...| 165738|46.038333333333334|

|27690030699786\_YM...|2018-02-04T19:59:...|2018-02-06T09:26:...| 134809| 37.44694444444445|

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

duration\_list = df.rdd.map(lambda r: r.Duration).collect()

print "median=", np.median(duration\_list) # 37.4 hours

print "sigma=", np.std(duration\_list) # 5.2 hours

Output:

median= 134809.0

sigma= 18853.7800836

## Change a DataFrame’s Column Names

from pyspark.sql.functions import explode

df1.show()

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

| a|col| c|

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

| 1| 1| [7, 8, 9]|

| 1| 2| [7, 8, 9]|

| 1| 3| [7, 8, 9]|

| 2| 4|[10, 11, 12]|

| 2| 5|[10, 11, 12]|

| 2| 6|[10, 11, 12]|

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

df2 = df1.selectExpr("a as alph", "col as exploded", "c as age")

df2.show()

Output:

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

|alph|exploded| age|

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

| 1| 1| [7, 8, 9]|

| 1| 2| [7, 8, 9]|

| 1| 3| [7, 8, 9]|

| 2| 4|[10, 11, 12]|

| 2| 5|[10, 11, 12]|

| 2| 6|[10, 11, 12]|

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

## Concatenate 2 pySpark DataFrames, row-wise

df1 = spark.range(3)

df1.show()

df2 = spark.range(4)

df2.show()

df = df1.union(df2)

df.show()

Output:

+---+

| id|

+---+

| 0|

| 1|

| 2|

+---+

+---+

| id|

+---+

| 0|

| 1|

| 2|

| 3|

+---+

+---+

| id|

+---+

| 0|

| 1|

| 2|

| 0|

| 1|

| 2|

| 3|

+---+

## Convert a Pandas DataFrame to a pySpark DataFrame

import pandas as pd

df\_pd = pd.DataFrame(

data={'integers': [1, 2, 3],

'floats': [-1.0, 0.5, 2.7],

'integer\_arrays': [[1, 2], [3, 4, 5], [6, 7, 8, 9]]}

)

**df = spark.createDataFrame(df\_pd)**

print type(df\_pd)

print type(df)

Output:

<class 'pandas.core.frame.DataFrame'>

<class 'pyspark.sql.dataframe.DataFrame'>

## Convert a tab-delimited File to a pySpark DataFrame

from pyspark.sql.types import \*

import pandas as pd

from pyspark.sql import SQLContext

airports\_pdf = pd.read\_csv("./airport-codes-na.txt", delimiter='\t', )

print type(airports\_pdf)

mySchema = StructType([StructField("City", StringType(), True),

StructField('State', StringType(), True),

StructField('Country', StringType(), True),

StructField('IATA', StringType(), True)])

airports = spark.createDataFrame(airports\_pdf, schema=mySchema)

print type(airports)

airports.show(5)

Output:

<class 'pandas.core.frame.DataFrame'>

<class 'pyspark.sql.dataframe.DataFrame'>

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

| City|State|Country|IATA|

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

|Abbotsford| BC| Canada| YXX|

| Aberdeen| SD| USA| ABR|

| Abilene| TX| USA| ABI|

| Akron| OH| USA| CAK|

| Alamosa| CO| USA| ALS|

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

## Convert a pySpark DataFrame to a Pandas DataFrame

pdf = df.toPandas()

## Convert a Timestamp Column to a Date Column

test\_df.show(10)

test\_df = test\_df.withColumn('date\_utc', test\_df['timestamp\_utc'].cast('date'))

test\_df.show(10)

Output:

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

| vesselId| deviceId| timestamp\_utc| delta\_km|moving|

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

|US--31185|86737803525694|2018-05-07 15:11:...|66.0684173462|moving|

|US--31185|86737803525694|2018-05-07 16:11:...|91.0644685432|moving|

|US--31185|86737803525694|2018-05-07 18:13:...|135.119960516|moving|

|US--31185|86737803525694|2018-05-07 20:13:...|170.303472288|moving|

|US--31185|86737803525694|2018-05-08 05:13:...|343.468741185|moving|

|US--31185|86737803525694|2018-05-08 14:17:...|156.760950786|moving|

|US--31185|86737803525694|2018-05-08 16:17:...|43.6579880144|moving|

|US--31185|86737803525694|2018-05-08 17:18:...|72.4366914182|moving|

|US--31185|86737803525694|2018-05-08 18:18:...|67.8436685098|moving|

|US--31185|86737803525694|2018-05-08 21:19:...|210.997405758|moving|

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

only showing top 10 rows

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

| vesselId| deviceId| timestamp\_utc| delta\_km|moving| date\_utc|

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

|US--31185|86737803525694|2018-05-07 15:11:...|66.0684173462|moving|2018-05-07|

|US--31185|86737803525694|2018-05-07 16:11:...|91.0644685432|moving|2018-05-07|

|US--31185|86737803525694|2018-05-07 18:13:...|135.119960516|moving|2018-05-07|

|US--31185|86737803525694|2018-05-07 20:13:...|170.303472288|moving|2018-05-07|

|US--31185|86737803525694|2018-05-08 05:13:...|343.468741185|moving|2018-05-08|

|US--31185|86737803525694|2018-05-08 14:17:...|156.760950786|moving|2018-05-08|

|US--31185|86737803525694|2018-05-08 16:17:...|43.6579880144|moving|2018-05-08|

|US--31185|86737803525694|2018-05-08 17:18:...|72.4366914182|moving|2018-05-08|

|US--31185|86737803525694|2018-05-08 18:18:...|67.8436685098|moving|2018-05-08|

|US--31185|86737803525694|2018-05-08 21:19:...|210.997405758|moving|2018-05-08|

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

only showing top 10 rows

test\_df.dtypes

Output:

[('vesselId', 'string'),

('deviceId', 'string'),

('timestamp\_utc', 'timestamp'),

('delta\_km', 'double'),

('moving', 'string'),

('date\_utc', 'date')]

## Count the number of distinct values for a column

import pyspark.sql.functions as fn

df.agg(

fn.count('id').alias('count'),

fn.countDistinct('id').alias('distinct')

).show()

Output:

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

|count|distinct|

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

| 5| 4|

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

## Count the number of Missing Observations in each Column

df\_miss = spark.createDataFrame([

(1, 143.5, 5.6, 28, 'M', 100000),

(2, 167.2, 5.4, 45, 'M', None),

(3, None , 5.2, None, None, None),

(4, 144.5, 5.9, 33, 'M', None),

(5, 133.2, 5.7, 54, 'F', None),

(6, 124.1, 5.2, None, 'F', None),

(7, 129.2, 5.3, 42, 'M', 76000),

], ['id', 'weight', 'height', 'age', 'gender', 'income'])

df\_miss.agg(\*[

(1 - (fn.count(c) / fn.count('\*'))).alias(c + '\_missing')

for c in df\_miss.columns

]).show()

Output:

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

|id\_missing| weight\_missing|height\_missing| age\_missing| gender\_missing| income\_missing|

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

| 0.0|0.1428571428571429| 0.0|0.2857142857142857|0.1428571428571429|0.7142857142857143|

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

## Create a DataFrame from Scratch

df = sqlContext.createDataFrame([Row(a=1, b=[1,2,3],c=[7,8,9]), Row(a=2, b=[4,5,6],c=[10,11,12])])

df.show()

Output:

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

| a| b| c|

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

| 1|[1, 2, 3]| [7, 8, 9]|

| 2|[4, 5, 6]|[10, 11, 12]|

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

Note: This is type pyspark.sql.dataframe.DataFrame

Another example from <https://www.mungingdata.com/apache-spark/dates-times>:

import java.sql.Timestamp

sourceDF = spark.createDF(

List(

(1, Timestamp.valueOf("2017-12-02 03:04:00")),

(2, Timestamp.valueOf("1999-01-01 01:45:20"))

), List(

("person\_id", IntegerType, *true*),

("fun\_time", TimestampType, *true*)

)

)

sourceDF.withColumn(

"fun\_minute",

minute(col("fun\_time"))

).withColumn(

"fun\_second",

second(col("fun\_time"))

).show()

Output:

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

|person\_id| fun\_time|fun\_minute|fun\_second|

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

| 1|2017-12-02 03:04:00| 4| 0|

| 2|1999-01-01 01:45:20| 45| 20|

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

## Create a new pySpark DataFrame Column from Existing Columns

See also <http://changhsinlee.com/pyspark-udf/>.

test\_pdf = pd.DataFrame(data={'floats1': [-1.0, 0.5, 2.7],

'floats2': [101.0, 102.0, 103.0]})

df = spark.createDataFrame(test\_pdf)

df.show()

def addem(flt1, flt2):

return(flt1 + flt2)

# end addem

addem\_udf = udf(lambda float1, float2: addem(float1, float2), DoubleType() )

df.select('floats1', 'floats2', addem\_udf('floats1', 'floats2').alias('total')).show()

Output:

root

|-- floats1: double (nullable = true)

|-- floats2: double (nullable = true)

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

|floats1|floats2|

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

| -1.0| 101.0|

| 0.5| 102.0|

| 2.7| 103.0|

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

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

|floats1|floats2|total|

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

| -1.0| 101.0|100.0|

| 0.5| 102.0|102.5|

| 2.7| 103.0|105.7|

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

## Display DataFrame Columns and Types

df = sqlContext.sql("SELECT \* from shipment\_sums limit 5")

display(df)

Output:

DataFrame[carriers: array<string>, computedTimeUtc: string, countDeliveryLocationArrivals: bigint, countDeliveryLocationDepartures: bigint, countGeolocationUpdates: bigint, countPickupLocationArrivals: bigint, countPickupLocationDepartures: bigint, countRestatedPickupLocationArrivals: bigint, countRestatedPickupLocationDepartures: bigint, customers: array<string>, kmsDirectShipment: double, kmsTraversedShipment: double, legCount: bigint, modes: array<string>, poiFromId: bigint, poiFromLatitude: double, poiFromLongitude: double, poiFromName: string, poiToId: bigint, poiToLatitude: double, poiToLongitude: double, poiToName: string, routePoiIds: array<bigint>, routePoiNames: array<string>, secsDurationLifeCycle: double, secsMovingInTransit: double, secsStationaryInTransit: double, secsStationaryMax: double, shipmentDestinationArrivalAnalyticUtc: string, shipmentDestinationArrivalDetectedUtc: string, shipmentDestinationArrivalReportedUtc: string, shipmentDestinationDepartureAnalyticUtc: string, shipmentDestinationDepartureReportedUtc: string, shipmentEndActualUtc: string, shipmentEndLocalDate: string, shipmentEndLocalDow: bigint, shipmentEndLocalHour: bigint, shipmentEndLocalMonth: bigint, shipmentEndLocalTime: string, shipmentEndLocalTimeZone: string, shipmentEndLocalYear: bigint, shipmentId: string, shipmentOriginArrivalAnalyticUtc: string, shipmentOriginArrivalDetectedUtc: string, shipmentOriginArrivalReportedUtc: string, shipmentOriginDepartureAnalyticUtc: string, shipmentOriginDepartureDetectedUtc: string, shipmentOriginDepartureReportedUtc: string, shipmentScheduleStatus: string, shipmentStartActualUtc: string, shipmentStartLocalDate: string, shipmentStartLocalDow: bigint, shipmentStartLocalHour: bigint, shipmentStartLocalMonth: bigint, shipmentStartLocalTime: string, shipmentStartLocalTimeZone: string, shipmentStartLocalYear: bigint, shipmentSummaryId: string, shipper: string]

## DataFrame Schema

smurf\_processed\_df.printSchema()

Output:

root

|-- smfMetaData: struct (nullable = true)

| |-- defaultProcessMode: string (nullable = true)

| |-- deviceId: string (nullable = true)

| |-- endpointReceiptTime: string (nullable = true)

| |-- endpointResponseCode: string (nullable = true)

| |-- ipAddress: string (nullable = true)

## Delete a Column from a pySpark DataFrame

df\_miss\_no\_income = df\_miss.select([c for c in df\_miss.columns if c != 'income'])

or

**port\_vessel\_mooring\_subset\_df = port\_vessel\_mooring\_vesselCounts\_df.select(['hour', 'vessel\_count', 'hrs\_moored', 'hrs\_to\_departure'])**

Output:

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

|hour|vessel\_count| hrs\_moored| hrs\_to\_departure|

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

| 4| 6|0.23083333333333333|119.66055555555556|

| 4| 6|0.31472222222222224|119.57666666666667|

| 4| 6| 0.5136111111111111|119.37777777777778|

## Drop Duplicate Rows

df = df.dropDuplicates()

## Drop Duplicate Rows which are Identical except for the Index

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

| id|weight|height|age|gender|

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

| 1| 144.5| 5.9| 33| M|

| 2| 167.2| 5.4| 45| M|

| 3| 124.1| 5.2| 23| F|

| 4| 144.5| 5.9| 33| M|

| 5| 133.2| 5.7| 54| F|

| 3| 124.1| 5.2| 23| F|

| 5| 129.2| 5.3| 42| M|

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

df = df.dropDuplicates(subset=[c for c in df.columns if c != 'id'])

df.show()

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

| id|weight|height|age|gender|

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

| 5| 133.2| 5.7| 54| F|

| 4| 144.5| 5.9| 33| M|

| 2| 167.2| 5.4| 45| M|

| 3| 124.1| 5.2| 23| F|

| 5| 129.2| 5.3| 42| M|

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

## Drop Rows which have a certain number of Columns with missing values

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

| id|weight|height| age|gender|

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

| 1| 143.5| 5.6| 28| M|

| 2| 167.2| 5.4| 45| M|

| 3| null| 5.2|null| null|

| 4| 144.5| 5.9| 33| M|

| 5| 133.2| 5.7| 54| F|

| 6| 124.1| 5.2|null| F|

| 7| 129.2| 5.3| 42| M|

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

df\_miss\_no\_income.dropna(thresh=3).show()

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

| id|weight|height| age|gender|

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

| 1| 143.5| 5.6| 28| M|

| 2| 167.2| 5.4| 45| M|

| 4| 144.5| 5.9| 33| M|

| 5| 133.2| 5.7| 54| F|

| 6| 124.1| 5.2|null| F|

| 7| 129.2| 5.3| 42| M|

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

## Drop Rows which have a null in Specific Fields

df.na.drop(subset=["dt\_mvmt"])

## Execute SQL on a DataFrame

sqlContext.registerDataFrameAsTable(data, "myTable")

df2 = sqlContext.sql("SELECT Name AS name, askdaosdka as age from myTable")

df2.show()

# Output

#+-------+---+

#| name|age|

#+-------+---+

#|Alberto| 2|

#| Dakota| 2|

#+-------+---+

## Execute multiple functions on a DataFrame

Wrap in parentheses:

from pyspark.sql.functions import col, max as max\_

df\_711 = sc.parallelize([

("2016-04-06 16:36", 1234, 111, 1),

("2016-04-06 17:35", 1234, 111, 5),

("2016-04-06 18:37", 1235, 111, 5),

("2016-04-06 18:37", 1234, 222, 5)

]).toDF(["datetime", "userId", "memberId", "value"])

df\_712 = (df\_711.withColumn("datetime", col("datetime").cast("timestamp"))

.groupBy("userId", "memberId")

.agg(max\_("datetime"))

.withColumnRenamed('max(datetime)', 'most\_recent'))

df\_712.show()

Output:

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

|userId|memberId| most\_recent|

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

| 1235| 111|2016-04-06 18:37:...|

| 1234| 222|2016-04-06 18:37:...|

| 1234| 111|2016-04-06 17:35:...|

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

## Explode a List inside a DataFrame column

df = sqlContext.createDataFrame([Row(a=1, b=[1,2,3],c=[7,8,9]), Row(a=2, b=[4,5,6],c=[10,11,12])])

df.show()

Output:

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

| a| b| c|

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

| 1|[1, 2, 3]| [7, 8, 9]|

| 2|[4, 5, 6]|[10, 11, 12]|

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

df1 = df.select(df.a, explode(df.b), df.c)

df.show()

Output:

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

| a|col| c|

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

| 1| 1| [7, 8, 9]|

| 1| 2| [7, 8, 9]|

| 1| 3| [7, 8, 9]|

| 2| 4|[10, 11, 12]|

| 2| 5|[10, 11, 12]|

| 2| 6|[10, 11, 12]|

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

### Explode an Array inside a DataFrame, but rename the new column

test\_658 = vessel\_dates\_df.selectExpr("\*",

"explode(port\_transit\_dates) as sched\_date")

test\_658.show()

Output:

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

| imo| poi|mooring\_start\_ts|mooring\_end\_ts|hrs\_moored| long\_name|centroid\_lat|centroid\_lng|radius\_km| tz|mooring\_start\_dt|mooring\_end\_dt| port\_transit\_dates|**sched\_date**|

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

|7808188|14536| 1557159154| 1557662590|139.843333|TASUCU [TRTAS]| 36.3167| 33.8833| 2.474|Europe/Istanbul| 2019-05-06| 2019-05-12|[2019-05-06, 2019...|**2019-05-06**|

## Export a Spark Data Frame to CSV

Two ways:

df.toPandas().to\_csv('mycsv.csv')

df.coalesce(1).write.csv('mycsv.csv')

## Extract a column from a pyspark DataFrame as a List

grp\_list = final\_df.rdd.map(lambda r: r.grp).collect()

print grp\_list

Returns:

[u'14979844271498287624', u'14979844271498287624', u'14979844271498287624',...]

## Extract a struct (structure) as a DataFrame

Note: In this example, ‘poi\_state’ is the column name of a column which is a struct.

poi\_state\_df\_2 = test\_result\_df.select('poi\_state.\*')

poi\_state\_df\_2.show(n=3, truncate = 25)

## Extract Distinct Values from a DataFrame Column

1. df.select("columnname").distinct().show()

2)

syngenta\_truck\_df.select('lastSpName').distinct().collect()

Output:

[Row(lastSpName=u'SUDAN, TX TEXAS PRODUCERS COOP [10001309]'),

Row(lastSpName=u'BOYLE, MS HELENA CHEMICAL COMPANY [3930781]'),

Row(lastSpName=u'CROP PRODUCTION SERVICES INC [10000465]'),

...

]

## Extract Nested JSON Data in Spark

<http://bigdatums.net/2016/02/12/how-to-extract-nested-json-data-in-spark/>

## Fill in missing data in a pySpark DataFrame by imputing the means

df\_miss\_no\_income.show()

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

| id|weight|height| age|gender|

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

| 1| 143.5| 5.6| 28| M|

| 2| 167.2| 5.4| 45| M|

| 3| null| 5.2|null| null|

| 4| 144.5| 5.9| 33| M|

| 5| 133.2| 5.7| 54| F|

| 6| 124.1| 5.2|null| F|

| 7| 129.2| 5.3| 42| M|

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

means = df\_miss\_no\_income.agg(

\*[fn.mean(c).alias(c) for c in df\_miss\_no\_income.columns if c != 'gender']

).toPandas().to\_dict('records')[0]

means['gender'] = 'missing'

df\_miss\_no\_income.fillna(means).show()

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

| id| weight|height|age| gender|

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

| 1| 143.5| 5.6| 28| M|

| 2| 167.2| 5.4| 45| M|

| 3|140.283333333| 5.2| 40|missing|

| 4| 144.5| 5.9| 33| M|

| 5| 133.2| 5.7| 54| F|

| 6| 124.1| 5.2| 40| F|

| 7| 129.2| 5.3| 42| M|

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

print means

{'gender': 'missing', 'age': 40.399999999999999, 'id': 4.0, 'weight': 140.28333333333333, 'height': 5.4714285714285706}

## Filter in rows having values contained in a list

port\_visits\_df = one\_leg\_2017\_df.where(one\_leg\_2017\_df['prev\_poi'] == one\_leg\_2017\_df['poi']) \

.where(one\_leg\_2017\_df.poi.isin(target\_poi\_list))

## Get distinct values from a column

from pyspark.sql import functions as F

>>> df1.show()

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

|no\_children|

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

| 0|

| 3|

| 2|

| 4|

| 1|

| 4|

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

>>> df1.select(F.collect\_set('no\_children').alias('no\_children')).first()['no\_children']

[0, 1, 2, 3, 4]

## groupBy usage with pySpark DataFrames

from pyspark.sql.functions import sum

ports\_df = sqlContext.createDataFrame(pd.read\_csv('port\_vessel\_hours\_2017.csv'))

ports\_df.orderBy(['poi', 'month']).show(12)

ports\_year\_sum\_df = ports\_df.groupBy('poi', 'long\_name', 'year' ).agg(sum('vessel\_hours') as vess\_hrs\_2017)

ports\_year\_sum\_df.orderBy(['poi']).show(5)

# ports\_df.show(5)

#+----+--------------------+----+-----+------------+

#| poi| long\_name|year|month|vessel\_hours|

#+----+--------------------+----+-----+------------+

#|4243|PORT LINCOLN [AUPLO]|2017| 1| 375|

#|4243|PORT LINCOLN [AUPLO]|2017| 2| 571|

#|4243|PORT LINCOLN [AUPLO]|2017| 3| 285|

#|4243|PORT LINCOLN [AUPLO]|2017| 4| 464|

#|4243|PORT LINCOLN [AUPLO]|2017| 5| 507|

# ...

#+----+--------------------+----+-----+------------+

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

| poi| long\_name|year|sum(vessel\_hours)|

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

|4243|PORT LINCOLN [AUPLO]|2017| 4125|

|4244|PORT MELBOURNE [A...|2017| 8653|

|4245|PORT OF BRISBANE ...|2017| 55019|

|4246| PORT PIRIE [AUPPI]|2017| 5342|

|4248| PORTLAND [AUPTJ]|2017| 23410|

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

And if you want to re-name the column:

import pyspark.sql.functions as sf

df.groupBy("group")\

.agg(sf.sum('money').alias('money'))\

.show(100)

### groupBy and keep only the max value on the group

from pyspark.sql.functions import col, max as max\_

df\_711 = sc.parallelize([

("2016-04-06 16:36", 1234, 111, 1),

("2016-04-06 17:35", 1234, 111, 5),

("2016-04-06 18:37", 1235, 111, 5),

("2016-04-06 18:37", 1234, 222, 5)

]).toDF(["datetime", "userId", "memberId", "value"])

df\_712 = (df\_711.withColumn("datetime", col("datetime").cast("timestamp"))

.groupBy("userId", "memberId")

.agg(max\_("datetime"))

.withColumnRenamed('max(datetime)', 'most\_recent'))

df\_712.show()

Output:

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

|userId|memberId| most\_recent|

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

| 1235| 111|2016-04-06 18:37:...|

| 1234| 222|2016-04-06 18:37:...|

| 1234| 111|2016-04-06 17:35:...|

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

## Join two pySpark DataFrames

See <http://www.learnbymarketing.com/1100/pyspark-joins-by-example/>.

df\_outliers = spark.createDataFrame([

(1, 143.5, 5.3, 28),

(2, 154.2, 5.5, 45),

(3, 342.3, 5.1, 99),

(4, 144.5, 5.5, 33),

(5, 133.2, 5.4, 54),

(6, 124.1, 5.1, 21),

(7, 129.2, 5.3, 42),

], ['id', 'weight', 'height', 'age'])

outliers = df\_outliers.select(\*['id'] + [

(

(df\_outliers[c] < bounds[c][0]) |

(df\_outliers[c] > bounds[c][1])

).alias(c + '\_o') for c in cols

])

outliers.show()

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

| id|weight\_o|height\_o|age\_o|

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

| 1| false| false|false|

| 2| false| false|false|

| 3| true| false| true|

| 4| false| false|false|

| 5| false| false|false|

| 6| false| false|false|

| 7| false| false|false|

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

df\_outliers = df\_outliers.join(outliers, on='id')

df\_outliers.show()

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

| id|weight|height|age|weight\_o|height\_o|age\_o|

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

| 7| 129.2| 5.3| 42| false| false|false|

| 6| 124.1| 5.1| 21| false| false|false|

| 5| 133.2| 5.4| 54| false| false|false|

| 1| 143.5| 5.3| 28| false| false|false|

| 3| 342.3| 5.1| 99| true| false| true|

| 2| 154.2| 5.5| 45| false| false|false|

| 4| 144.5| 5.5| 33| false| false|false|

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

### Example with mismatched column names

Starting with a vessel\_df like:

# +---------+----+----------------+--------------+----------+

# | imo| poi|mooring\_start\_ts|mooring\_end\_ts|hrs\_moored|

# +---------+----+----------------+--------------+----------+

# |103558307|5567| 1554544859| 1554560286| 4.285278|

# |103558307|5567| 1555121474| 1555175654| 15.050000|

# +---------+----+----------------+--------------+----------+

append a timezone, and local date info for the start of mooring and end of mooring

'''

vessel\_df = vessel\_df.join(pois\_col\_subset\_df, vessel\_df.poi == pois\_col\_subset\_df.id, 'inner')

vessel\_df output:

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

| imo| poi|mooring\_start\_ts|mooring\_end\_ts|hrs\_moored| id| poi\_type| short\_name| long\_name|centroid\_lat|centroid\_lng|radius\_km| fence\_json| tz|

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

|103558307|5567| 1554544859| 1554560286| 4.285278|5567|marinePort|CHANGSHU PT|CHANGSHU PT [CNCGS]| 31.65| 120.7167| 12.586|{"type":"Polygon"...|Asia/Shanghai|

|103558307|5567| 1555121474| 1555175654| 15.050000|5567|marinePort|CHANGSHU PT|CHANGSHU PT [CNCGS]| 31.65| 120.7167| 12.586|{"type":"Polygon"...|Asia/Shanghai|

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

## Loading a Schema while Correcting org.apache.parquet.io.ParquetDecodingException

This throws an Exception:

smf\_parquet = sqlContext.read.parquet( '/lambda/smf-parquet/POC/smurf-processed//quarter=2017-q1/\*',

'/lambda/smf-parquet/POC/smurf-processed//quarter=2017-q2/\*',

'/lambda/smf-parquet/POC/smurf-processed//quarter=2017-q3/\*')

smf\_parquet.registerTempTable('smurf\_processed')

test\_df = sqlContext.sql('''

SELECT \*

FROM smurf\_processed

LIMIT 10''')

test\_df.take(5)

but the following works:

merged\_df = spark.read.option("mergeSchema", "true").parquet( '/lambda/smf-parquet/POC/smurf-processed//quarter=2017-q1/\*',  
'/lambda/smf-parquet/POC/smurf-processed//quarter=2017-q2/\*')

## Read a DataFrame from a csv file

See <https://spark.apache.org/docs/1.6.0/api/java/org/apache/spark/sql/types/DataTypes.html>

### Example 1

import pandas as pd

pandas\_pdf = pd.read\_csv('syngenta\_od\_pairs.csv')

od\_pair\_df = sqlContext.createDataFrame(pandas\_pdf)

### Example 2

flightPerf\_pdf = pd.read\_csv("./departuredelays.csv")

flightPerfSchema = StructType([StructField("date", IntegerType(), True),

StructField("delay", IntegerType(), True),

StructField("distance", IntegerType(), True),

StructField("origin", StringType(), True),

StructField("destination", StringType(), True)])

flightPerf = spark.createDataFrame(flightPerf\_pdf, schema=flightPerfSchema)

flightPerf.createOrReplaceTempView("FlightPerformance")

flightPerf.cache()

Output:

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

| date|delay|distance|origin|destination|

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

|1011245| 6| 602| ABE| ATL|

|1020600| -8| 369| ABE| DTW|

## Register a DataFrame as a Temporary Table

vessel\_poi\_count\_df = sqlContext.sql('''

SELECT mmsi, imo, count(distinct(poi)) as poi\_count

FROM vessel\_poi

GROUP BY mmsi, imo

ORDER BY count(distinct(poi)) DESC

''')

vessel\_poi\_count\_df.createOrReplaceTempView('vessel\_poi\_count')

print vessel\_poi\_count\_df.count()

vessel\_poi\_count\_df.show(5)

# Output # of pois traversed by a ship, in a year

# 25883

#+---------+-------+-------------------+

#| mmsi| imo|count(DISTINCT poi)|

#+---------+-------+-------------------+

#|311913000|9118006| 131|

#|311007600|9073892| 129|

#|314220000|9005742| 129|

#|244890901|9760407| 127|

#|245219000|8915756| 127|

#+---------+-------+-------------------+

## Row Count of a DataFrame

df.count()

## Sample a DataFrame (approximate sample)

od\_sample = od.sample(withReplacement=False, fraction=0.04, seed=13) # Sample 4% of data

## selectExpr on a DataFrame

data = sqlContext.createDataFrame([("Alberto", 2), ("Dakota", 2)],

["Name", "askdaosdka"])

data.show()

data.printSchema()

# Output

#+-------+----------+

#| Name|askdaosdka|

#+-------+----------+

#|Alberto| 2|

#| Dakota| 2|

#+-------+----------+

#root

# |-- Name: string (nullable = true)

# |-- askdaosdka: long (nullable = true)

df = data.selectExpr("Name as name", "askdaosdka as age")

df.show()

df.printSchema()

# Output

#+-------+---+

#| name|age|

#+-------+---+

#|Alberto| 2|

#| Dakota| 2|

#+-------+---+

## Show a DataFrame without Truncating Columns

test\_754\_df.show()

test\_754\_df.show(truncate=False)

Output:

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

| utct| local\_time|

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

|2016-03-01 00:00:...|2016-02-29 14:00:...|

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

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

|utct |local\_time |

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

|2016-03-01 00:00:00.0|2016-02-29 14:00:00.0|

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

## Show the percentage of missing observations in each column of a pySpark DataFrame

df\_miss = spark.createDataFrame([

(1, 143.5, 5.6, 28, 'M', 100000),

(2, 167.2, 5.4, 45, 'M', None),

(3, None , 5.2, None, None, None),

(4, 144.5, 5.9, 33, 'M', None),

(5, 133.2, 5.7, 54, 'F', None),

(6, 124.1, 5.2, None, 'F', None),

(7, 129.2, 5.3, 42, 'M', 76000),

], ['id', 'weight', 'height', 'age', 'gender', 'income'])

df\_miss.agg(\*[(1 - (fn.count(c) / fn.count('\*'))).alias(c + '\_missing') for c in df\_miss.columns]).show()

Output:

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

|id\_missing| weight\_missing|height\_missing| age\_missing| gender\_missing| income\_missing|

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

| 0.0|0.1428571428571429| 0.0|0.2857142857142857|0.1428571428571429|0.7142857142857143|

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

## Sort a DataFrame

od\_sample = od\_sample.sort("od\_pair", ascending=True)

Output:

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

|source\_id|dest\_id|od\_pair|

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

| 10| 1077|10-1077|

| 10| 1171|10-1171|

| 10| 136| 10-136|

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

# Sort potentially by Multiple Columns

vessel\_df = start\_moored\_df[start\_moored\_df['imo'] == vessel\_imo\_str]

print "vessel\_df=", vessel\_df.show(5)

sorted\_desc\_df = vessel\_df.orderBy(['timestamp'], ascending=[0])

print "sorted descending=", sorted\_desc\_df.show(5)

Output:

vessel\_df=

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

| timestamp| mmsi| imo|moored|nav\_status| lat| lon|sog| pois|

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

|1483246223|235102677|9604146| 1| 5|22.4477|113.8881|0.2|[5715, 5576, 5776...|

|1484510320|235102677|9604146| 1| 5| 24.989| 55.0521|0.0|[16417, 16421, 16...|

|1484671995|235102677|9604146| 1| 5|27.0948| 56.0583|0.0|[17044, 11510, 11...|

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

sorted descending=

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

| timestamp| mmsi| imo|moored|nav\_status| lat| lon|sog| pois|

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

|1514413286|235102677|9604146| 1| 5|37.9542| 23.5801|0.0|[10834, 10677, 10...|

|1512546458|235102677|9604146| 1| 5| 1.275|103.7826|0.0|[14299, 14300, 17...|

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

## Specify the schema programmatically

from pyspark.sql.types import \*

# Generate our own CSV data

stringCSVRDD = sc.parallelize([(123, 'Katie', 19, 'brown'),

(234, 'Michael', 22, 'green'),

(345, 'Simone', 23, 'blue')])

# The schema is encoded in a string, using StructType we define the schema using various pyspark.sql.types

schema = StructType([

StructField("id", LongType(), True),

StructField("name", StringType(), True),

StructField("age", LongType(), True),

StructField("eyeColor", StringType(), True)

])

# Apply the schema to the RDD and Create DataFrame

swimmers = spark.createDataFrame(stringCSVRDD, schema)

# Creates a temporary view using the DataFrame

swimmers.createOrReplaceTempView("swimmers")

swimmers.printSchema()

Output:

root

|-- id: long (nullable = true)

|-- name: string (nullable = true)

|-- age: long (nullable = true)

|-- eyeColor: string (nullable = true)

## Statistics on pyspark.sql.dataframe.DataFrame columns

### Calculate Mean and Median of a Column

import numpy as np

vessel\_poi\_count\_df.show(5)

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

| mmsi| imo|poi\_count|

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

|311913000|9118006| 131|

|311007600|9073892| 129|

|314220000|9005742| 129|

|244890901|9760407| 127|

|245219000|8915756| 127|

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

poi\_count\_list = vessel\_poi\_count\_df.rdd.map(lambda r: r.poi\_count).collect()

print "median=", np.median(poi\_count\_list)

print "mean=", np.mean(poi\_count\_list)

Output:

median= 21.0

mean= 24.9121044701

### Statistics Summary for a pySpark DataFrame

vessel\_poi\_count\_df.describe().show()

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

|summary| mmsi| imo| poi\_count|

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

| count| 25883| 24839| 25883|

| mean|4.114968495927829E8|1.4876571290188815E7| 24.91210447011552|

| stddev| 1.37569361042951E8| 6.857711122390723E7|19.900553546011217|

| min| 112| 103558307| 1|

| max| 886092406| 995467000| 131|

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

# pyspark.sql

Select a Field from a Row

print take\_out[0]

take\_out[0].SAPShipmentID

Output:

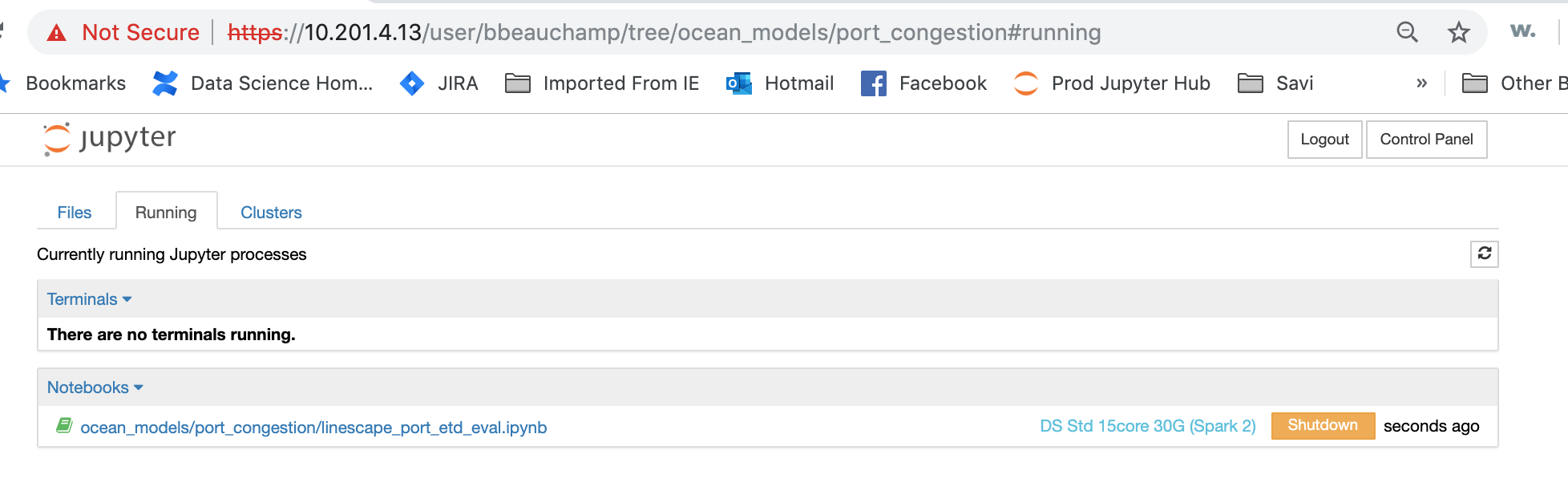
Row(SAPShipmentID=u'0305463905')

u'0305463905'

# Recover from a Lost Connection

Symptom is that a red Error indicator is shown on the jupyter notebook, that doesn’t resolve when you click on the Reconnect button. Also, if you look at the Running tab of the jupyter notebook, not only does it indicate that the desired notebook is running, but the Shutdown button is non-responsive.

To get back to where you were, close the notebook tab in the browser, and also close the direcotry tree tab in the browser. Then simply reconnect anew to the jupyter hub, and when you reonnect you will be able to load the notebook again, and it will resume the previous state.



# SparkSQL

## Case Statements

shipment\_modes\_df = sqlContext.sql(''' SELECT poiFromId, poiToId,

truckCount / (truckCount + intermodalCount + unknownCount) as truckPct

FROM

(SELECT poiFromId, poiToId,

SUM(CASE WHEN mode = 'truck' THEN 1 ELSE 0 END) AS truckCount,

SUM(CASE WHEN mode = 'intermodal' THEN 1 ELSE 0 END) AS intermodalCount,

SUM(CASE WHEN mode = 'unknown' THEN 1 ELSE 0 END) AS unknownCount

FROM

(SELECT shipmentId, poiFromId, poiToId,

explode(modes) as mode

FROM shipment\_sums\_table

WHERE legCount = 1)

GROUP BY poiFromId, poiToId

ORDER BY poiFromId, poiToId

)

order by truckPct

''')

## DataFrame (pyspark.sql.dataframe.DataFrame)

### Read a Parquet File into a DataFrame

shipment\_summaries\_path = '/lambda/summaries-parquet/PG/shipment-summaries/\*'

shipment\_sums = sqlContext.read.parquet(shipment\_summaries\_path) # pyspark.sql.dataframe.DataFrame

### Store DataFrame as Temp Table for Later Querying

shipment\_sums.createOrReplaceTempView("shipment\_sums\_table")

Query from the Temp Table:

test\_df = sqlContext.sql('''SELECT shipmentId FROM shipment\_sums\_table limit 1''')

test\_df.head()

Output:

Row(shipmentId=u'0306098015')

### Select an Array Element

df.printSchema()

## root

## |-- stuff: array (nullable = true)

## | |-- element: struct (containsNull = true)

## | | |-- a: long (nullable = true)

## | | |-- b: long (nullable = true)

## | | |-- c: long (nullable = true)

sqlContext.sql("SELECT stuff[0].a FROM df").show()

## +---+

## |\_c0|

## +---+

## | 1|

## +---+

# Invoke a python function from an SQL query

**def** simple\_function(v):

**return** int(v \* 10)

*#test the function*

**print** simple\_function(3)

sqlContext.registerFunction("simple\_function", simple\_function)

query = """

select

ID,

VAL1,

VAL2,

simple\_function(cast(VAL1 as int)) as s\_VAL1,

simple\_function(cast(VAL2 as int)) as s\_VAL2

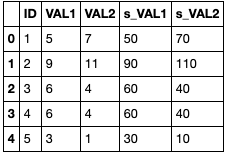
from

example2

"""

sqlContext.sql(query).toPandas()

Output:



# Parquet

## Explode a list or array data field

shipment\_df = sqlContext.sql('''SELECT shipmentId, modes

FROM shipment\_sums\_table

where legCount = 2

limit 5

''')

shipment\_df.show()

Output:

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

|shipmentId| modes|

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

|0306080481|[truck, truck]|

|0306120361|[truck, truck]|

|0306120411|[truck, truck]|

|0306103320|[truck, truck]|

|0306078031|[truck, truck]|

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

# What modes are available?

modes\_df = sqlContext.sql('''SELECT shipmentId,

explode(distinct modes)as mode

FROM shipment\_sums\_table

WHERE legCount = 2

limit 6

''')

modes\_df.show()

Output:

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

|shipmentId| mode|

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

|0306080481|truck|

|0306080481|truck|

|0306120361|truck|

|0306120361|truck|

|0306120411|truck|

|0306120411|truck|

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

**Exploding an Array stored in a Column**

Unexploded:

start\_moored = sqlContext.sql('''

SELECT timestamp, mmsi, pois

FROM (

SELECT \*

FROM pos\_data

WHERE moored = 1 and prev\_moored = 0 and prev2\_moored = 0 and prev3\_moored = 0 and prev4\_moored = 0 and prev5\_moored = 0 and prev6\_moored = 0 and prev7\_moored = 0

) a

'''

)

Output:

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

| timestamp| mmsi| pois|

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

|1493653149|212370000| [12223]|

|1493847898|212370000|[14952, 15249]|

|1493949978|212370000| [12223]|

|1494075847|212370000| [12275]|

|1494258522|212370000| [12223]|

Exploded:

start\_moored = sqlContext.sql('''

SELECT timestamp, mmsi, poi

FROM (

SELECT \*

FROM pos\_data

WHERE moored = 1 and prev\_moored = 0 and prev2\_moored = 0 and prev3\_moored = 0 and prev4\_moored = 0 and prev5\_moored = 0 and prev6\_moored = 0 and prev7\_moored = 0

) a

LATERAL VIEW OUTER EXPLODE(pois) zzz AS poi

'''

Output:

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

| timestamp| mmsi| poi|

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

|1493653149|212370000|12223|

|1493847898|212370000|14952| # These were

|1493847898|212370000|15249| # exploded

|1493949978|212370000|12223|

|1494075847|212370000|12275|

|1494258522|212370000|12223|

|1494419315|212370000|14952|

|1494419315|212370000|15249|

|1494621881|212370000|12275|

|1494913457|212370000|12223|

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

# Strings

## Concatenate strings

CONCAT(SUBSTR(smfPayloadData.timestamp.asOf,1,10), ' ',

SUBSTR(smfPayloadData.timestamp.asOf, 12, 8)) as test

## Convert a numeric field to a string

test = sqlContext.sql('''

SELECT DISTINCT format\_string("%s", poiFromId) as originId,

format\_string("%s", poiToId) as destId,

format\_string("%s-%s", poiFromId, poiToId) as od\_pair

FROM shipment\_sums

LIMIT 5

''')

print type(test)

test.show(5)

Output:

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

|originId|destId|od\_pair|

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

| 39| 981| 39-981|

| 31| 1056|31-1056|

| 38| 547| 38-547|

| 33| 174| 33-174|

| 37| 40| 37-40|

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

## StringIndexer

## Substring

SUBSTR(smfPayloadData.timestamp.asOf,1,10)

output: “2016-03-01”

# Timestamps

## Calculate time difference (seconds) between two timestamps

from pyspark.sql import functions as F

from pyspark.sql import Row

df1 = sqlContext.createDataFrame([Row(dt\_str='2019-01-04 00:00:00', dt\_str\_2='2019-01-05 00:00:00'),

Row(dt\_str='2019-01-05 00:00:00', dt\_str\_2='2019-01-10 00:00:00')])

df1.show()

format = "yyyy-MM-dd"

df2 = df1.withColumn('timestamp\_1', F.unix\_timestamp('dt\_str', format).cast('timestamp'))

df2 = df2.withColumn('timestamp\_2', F.unix\_timestamp('dt\_str\_2', format).cast('timestamp'))

df2.show()

timeDiff = (F.unix\_timestamp('dt\_str\_2', format=format)

- F.unix\_timestamp('dt\_str', format=format))

df2 = df2.withColumn("Duration\_sec", timeDiff)

df2.show()

Output:

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

| dt\_str| dt\_str\_2|

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

|2019-01-04 00:00:00|2019-01-05 00:00:00|

|2019-01-05 00:00:00|2019-01-10 00:00:00|

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

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

| dt\_str| dt\_str\_2| timestamp\_1| timestamp\_2|

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

|2019-01-04 00:00:00|2019-01-05 00:00:00|2019-01-04 00:00:...|2019-01-05 00:00:...|

|2019-01-05 00:00:00|2019-01-10 00:00:00|2019-01-05 00:00:...|2019-01-10 00:00:...|

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

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

| dt\_str| dt\_str\_2| timestamp\_1| timestamp\_2|Duration\_sec|

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

|2019-01-04 00:00:00|2019-01-05 00:00:00|2019-01-04 00:00:...|2019-01-05 00:00:...| 86400|

|2019-01-05 00:00:00|2019-01-10 00:00:00|2019-01-05 00:00:...|2019-01-10 00:00:...| 432000|

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

## Convert string to timestamp

shipment\_carr = sqlContext.sql('''

SELECT smfPayloadData.paired.shipmentId,

smfPayloadData.paired.shipper,

smfPayloadData.paired.carrier,

smfPayloadData.timestamp.asOf as dest\_arr\_utc,

to\_utc\_timestamp('2016-03-01 00:00:00','UTC') as example\_timestamp

FROM mapped

where smfPayloadData.events.DeliveryLocationArrival is not null

and smfPayloadData.timestamp.asOf >= to\_utc\_timestamp('2016-03-01 00:00:00','UTC')

and smfPayloadData.timestamp.asOf < to\_utc\_timestamp('2016-03-02 00:00:00','UTC')

limit 1

''')

printResultsAsJson(shipment\_carr)

output:

{

**"example\_timestamp": "2016-03-01 00:00:00.0"**,

"carrier": "SCNN",

"shipper": "PG",

"dest\_arr\_utc": "2016-03-01T23:51:00.000Z",

"shipmentId": "Jackie will add"

}

## Convert Unix Timestamp to Formatted Date

mooring\_exp\_df = sqlContext.sql('''

SELECT \*,

from\_unixtime(timestamp, 'YYYY-MM-dd') as timestamp\_dt

FROM start\_moored

ORDER BY mmsi, timestamp

LIMIT 10

'''

)

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

| timestamp| mmsi| poi|timestamp\_dt|

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

|1493942222|173313120|12052| 2017-05-04|

|1494196603|173313120|12043| 2017-05-07|

|1494472207|173313120|12052| 2017-05-11|

...

## Convert Unix Timestamp to ISO Standard Date and Time

port\_pairs = sqlContext.sql('''

select mmsi,

orig\_timestamp, from\_unixtime(orig\_timestamp, "YYYY-MM-dd'T'HH-mm-SS'Z'") as orig\_dt, orig\_poi,

dest\_timestamp, from\_unixtime(dest\_timestamp, "YYYY-MM-dd'T'HH-mm-SS'Z'") as orig\_dt, dest\_poi”””)

## Get Calendar Parts like Year, Month, Day, Hour from a Timestamp

# year\_udf = udf(lambda timestamp: time.gmtime(timestamp).tm\_year, IntegerType())

# month\_udf = udf(lambda timestamp: time.gmtime(timestamp).tm\_mon, IntegerType())

# day\_udf = udf(lambda timestamp: time.gmtime(timestamp).tm\_mday, IntegerType())

# hour\_udf = udf(lambda timestamp: time.gmtime(timestamp).tm\_hour, IntegerType())

# Appendix 1: pySpark DataTypes:

<http://spark.apache.org/docs/1.5.1/api/java/org/apache/spark/sql/types/package-summary.html>

# Appendix 2 : Identifying Outliers

This is from the book “Learning pySpark” [Drabas, Lee], chapter 4.

df\_outliers = spark.createDataFrame([

(1, 143.5, 5.3, 28),

(2, 154.2, 5.5, 45),

(3, 342.3, 5.1, 99),

(4, 144.5, 5.5, 33),

(5, 133.2, 5.4, 54),

(6, 124.1, 5.1, 21),

(7, 129.2, 5.3, 42),

], ['id', 'weight', 'height', 'age'])

df\_outliers.show()

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

| id|weight|height|age|

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

| 1| 143.5| 5.3| 28|

| 2| 154.2| 5.5| 45|

| 3| 342.3| 5.1| 99|

| 4| 144.5| 5.5| 33|

| 5| 133.2| 5.4| 54|

| 6| 124.1| 5.1| 21|

| 7| 129.2| 5.3| 42|

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

**First, we calculate the lower and upper *cut off* points for each feature.**

cols = ['weight', 'height', 'age']

bounds = {}

for col in cols:

quantiles = df\_outliers.approxQuantile(col, [0.25, 0.75], 0.05)

IQR = quantiles[1] - quantiles[0]

bounds[col] = [quantiles[0] - 1.5 \* IQR, quantiles[1] + 1.5 \* IQR]

The bounds dictionary holds the lower and upper bounds for each feature.

bounds

{'age': [-11.0, 93.0],

'height': [4.499999999999999, 6.1000000000000005],

'weight': [91.69999999999999, 191.7]}

Let's now use it to flag our outliers.

outliers = df\_outliers.select(\*['id'] + [

(

(df\_outliers[c] < bounds[c][0]) | (df\_outliers[c] > bounds[c][1])

).alias(c + '\_o') for c in cols

]

)

outliers.show()

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

| id|weight\_o|height\_o|age\_o|

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

| 1| false| false|false|

| 2| false| false|false|

| 3| true| false| true|

| 4| false| false|false|

| 5| false| false|false|

| 6| false| false|false|

| 7| false| false|false|

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

We have an outlier in the weight feature and the age feature.

df\_outliers = df\_outliers.join(outliers, on='id')

df\_outliers.show()

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

| id|weight|height|age|weight\_o|height\_o|age\_o|

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

| 7| 129.2| 5.3| 42| false| false|false|

| 6| 124.1| 5.1| 21| false| false|false|

| 5| 133.2| 5.4| 54| false| false|false|

| 1| 143.5| 5.3| 28| false| false|false|

| 3| 342.3| 5.1| 99| true| false| true|

| 2| 154.2| 5.5| 45| false| false|false|

| 4| 144.5| 5.5| 33| false| false|false|

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