pySpark Cheatsheet

Table of Contents

[Command Shell Execution Within iPython / jupyter notebook 2](#_Toc4919653)

[Configure Spark from a jupyter notebook command 2](#_Toc4919654)

[Delete an HDFS directory using jupyter notebook 2](#_Toc4919655)

[Examine Underlying HDFS Directory Structures in a Jupyter Notebook 2](#_Toc4919656)

[Set the spark timeout value 4](#_Toc4919657)

[Transfer a CSV file to HDFS 4](#_Toc4919658)

[DataFrames (pyspark.sql.dataframe.DataFrame) 5](#_Toc4919659)

[Add a Column Based on Another Column 5](#_Toc4919660)

[Assign each Row a new ID 5](#_Toc4919661)

[Calculate Statistics on a DataFrame Column 5](#_Toc4919662)

[Change a DataFrame’s Column Names 6](#_Toc4919663)

[Concatenate 2 pySpark DataFrames, row-wise 6](#_Toc4919664)

[Convert a Pandas DataFrame to a pySpark DataFrame 7](#_Toc4919665)

[Convert a tab-delimited File to a pySpark DataFrame 7](#_Toc4919666)

[Convert a pySpark DataFrame to a Pandas DataFrame 8](#_Toc4919667)

[Convert a Timestamp Column to a Date Column 9](#_Toc4919668)

[Count the number of distinct values for a column 10](#_Toc4919669)

[Count the number of Missing Observations in each Column 10](#_Toc4919670)

[Create a DataFrame from Scratch 11](#_Toc4919671)

[Create a new pySpark DataFrame Column from Existing Columns 11](#_Toc4919672)

[Display DataFrame Columns and Types 11](#_Toc4919673)

[DataFrame Schema 12](#_Toc4919674)

[Delete a Column from a pySpark DataFrame 12](#_Toc4919675)

[Drop Duplicate Rows 12](#_Toc4919676)

[Drop Duplicate Rows which are Identical except for the Index 13](#_Toc4919677)

[Drop Rows which have a certain number of Columns with missing values 13](#_Toc4919678)

[Execute SQL on a DataFrame 14](#_Toc4919679)

[Explode a List inside a DataFrame column 14](#_Toc4919680)

[Export a Spark Data Frame to CSV 14](#_Toc4919681)

[Extract a column from a pyspark DataFrame as a List 15](#_Toc4919682)

[Extract a struct (structure) as a DataFrame 15](#_Toc4919683)

[Extract Distinct Values from a DataFrame Column 15](#_Toc4919684)

[Extract Nested JSON Data in Spark 15](#_Toc4919685)

[Filter in rows having values contained in a list 15](#_Toc4919686)

[groupBy usage with pySpark DataFrames 15](#_Toc4919687)

[Loading a Schema while Correcting org.apache.parquet.io.ParquetDecodingException 16](#_Toc4919688)

[Read a DataFrame from a csv file 16](#_Toc4919689)

[Example 1 17](#_Toc4919690)

[Example 2 17](#_Toc4919691)

[Register a DataFrame as a Temporary Table 17](#_Toc4919692)

[Row Count of a DataFrame 18](#_Toc4919693)

[Sample a DataFrame (approximate sample) 18](#_Toc4919694)

[Sort a DataFrame 18](#_Toc4919695)

[Specify the schema programmatically 19](#_Toc4919696)

[Statistics on pyspark.sql.dataframe.DataFrame columns 20](#_Toc4919697)

[Calculate Mean and Median of a Column 20](#_Toc4919698)

[Statistics Summary for a pySpark DataFrame 20](#_Toc4919699)

[pyspark.sql 20](#_Toc4919700)

[SparkSQL 21](#_Toc4919701)

[Case Statements 21](#_Toc4919702)

[DataFrame (pyspark.sql.dataframe.DataFrame) 21](#_Toc4919703)

[Read a Parquet File into a DataFrame 22](#_Toc4919704)

[Store DataFrame as Temp Table for Later Querying 22](#_Toc4919705)

[Select an Array Element 22](#_Toc4919706)

[Parquet 23](#_Toc4919707)

[Explode a list or array data field 23](#_Toc4919708)

[Strings 25](#_Toc4919709)

[Concatenate strings 25](#_Toc4919710)

[Convert a numeric field to a string 25](#_Toc4919711)

[Substring 25](#_Toc4919712)

[Timestamps 26](#_Toc4919713)

[Convert string to timestamp 26](#_Toc4919714)

[Convert Unix Timestamp to Formatted Date 26](#_Toc4919715)

[Convert Unix Timestamp to ISO Standard Date and Time 27](#_Toc4919716)

[Appendix 1: pySpark DataTypes: 28](#_Toc4919717)

# 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

## 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|

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

## 

## 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 tyape pyspark.sql.dataframe.DataFrame

## 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'])

## 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|

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

## 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|

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

## 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]|

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

## 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/>

## 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))

## 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)

## 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

### 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.registerTempTable('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

## 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...|

|1483581689|235102677|9604146| 1| 5| 1.3571|103.5467|0.2|[12320, 12354, 12...|

|1484010564|235102677|9604146| 1| 5| 6.946| 79.8298|0.0| [12094]|

|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...|

|1513895423|235102677|9604146| 1| 5|32.8184| 35.013|0.2| [11380, 11381]|

|1513687285|235102677|9604146| 1| 5| 31.834| 34.6458|0.0| [11377]|

|1512638579|235102677|9604146| 1| 5| 1.365|103.5443|0.0|[12354, 12358, 12...|

|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'

# 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|

## +---+

# 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|

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

## Substring

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

output: “2016-03-01”

# Timestamps

## 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”””)

# Appendix 1: pySpark DataTypes:

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