

CS 649 Big Data: Tools and Methods
Fall Semester, 2021
Doc 15 Spark 2
Mar 11, 2021

Copyright ©, All rights reserved. 2021 SDSU & Roger Whitney,
5500 Campanile Drive, San Diego, CA 92182-7700 USA.
OpenContent (<http://www.opencontent.org/opl.shtml>) license
defines the copyright on this document.

Transformations & Actions

Transformations

Done on worker machines

Lazy

Actions

Bring results to master machine

Triggers transformations

Transformations

Add rows or columns

Remove rows or columns

Change row into column and column into row

Change order of rows

PySpark Actions

`collect()`

`count()`

`describe(*cols)`

`first()`

`foreach(f)`

`foreachPartition(f)`

`head(n=None)`

`show(n=20, truncate=True, vertical=False)`

`summary(*statistics)`

PySpark Transactions

agg(*exprs)
alias(alias)
coalesce(numPartitions)
colRegex(colName)
crossJoin(other)
cube(*cols)
distinct()
dropDuplicates(subset=None)
exceptAll(other)
filter(condition)
groupby(*cols)
intersect(other)
intersectAll(other)
join(other, on=None, how=None)
limit(num)
na

orderBy(*cols, **kwargs)
sample(withReplacement=None, fraction=None, seed=None)
sampleBy(col, fractions, seed=None)
select(*cols)
selectExpr(*expr)
sort(*cols, **kwargs)
sortWithinPartitions(*cols, **kwargs)
stat
union(other)
unionAll(other)
unionByName(other)
where(condition)
withColumn(colName, col)
withColumnRenamed(existing, new)

Basic Dataset functions

approxQuantile(col, probabilities, relativeError)
cache()
checkpoint(eager=True)
columns
corr(col1, col2, method=None)
cov(col1, col2)
createGlobalTempView(name)
createOrReplaceGlobalTempView(name)
createOrReplaceTempView(name)
createTempView(name)
crosstab(col1, col2)
drop(*cols)
dropna(how='any', thresh=None, subset=None)
dtypes
explain(extended=False)
fillna(value, subset=None)
freqItems(cols, support=None)
hint(name, *parameters)
isLocal()
isStreaming

localCheckpoint(eager=True)
persist(storageLevel=StorageLevel(True, True, True))
printSchema()
randomSplit(weights, seed=None)
rdd
registerTempTable(name)
repartition(numPartitions, *cols)
repartitionByRange(numPartitions, *cols)
replace(to_replace, value=<no value>, subset=None)
rollup(*cols)
schema
storageLevel
subtract(other)
take(num)
toDF(*cols)
toJSON(use_unicode=True)
toLocalIterator()
toPandas()
unpersist(blocking=False)
withWatermark(eventTime, delayThreshold)
write
writeStream

Data For Example

United States Bureau of Transportation statistics

The Definitive Guide, Zaharia & Chambers, O'Reilly Media

```
df = spark.read.json("2015-summary.json")
df.show(2)
```

```
+-----+-----+-----+
|DEST_COUNTRY_NAME|ORIGIN_COUNTRY_NAME|count|
+-----+-----+-----+
|    United States|          Romania   |    15|
|    United States|          Croatia   |     1|
+-----+-----+-----+
```

2015-summary.json

```
{"ORIGIN_COUNTRY_NAME":"Romania","DEST_COUNTRY_NAME":"United States","count":15}
{"ORIGIN_COUNTRY_NAME":"Croatia","DEST_COUNTRY_NAME":"United States","count":1}
{"ORIGIN_COUNTRY_NAME":"Ireland","DEST_COUNTRY_NAME":"United States","count":344}
...
```

Select & SelectExpr

```
SELECT * FROM dataFrameTable
```

```
SELECT columnName FROM dataFrameTable
```

```
SELECT columnName * 10, otherColumn, someOtherCol as c FROM dataFrameTable
```

```
from pyspark.sql.functions import column, col, expr
```

```
newDf = df.select("DEST_COUNTRY_NAME","ORIGIN_COUNTRY_NAME")
```

```
newDf.show(2)
```

```
+-----+-----+
|DEST_COUNTRY_NAME|ORIGIN_COUNTRY_NAME|
+-----+-----+
|    United States|             Romania|
|    United States|             Croatia|
+-----+-----+
```


Different Syntax

```
import org.apache.spark.sql.functions.{expr, col, column}
```

```
df.select(  
df.col("DEST_COUNTRY_NAME"),  
  col("DEST_COUNTRY_NAME"),  
  column("DEST_COUNTRY_NAME"),  
'DEST_COUNTRY_NAME,  
 $"DEST_COUNTRY_NAME",  
  expr("DEST_COUNTRY_NAME")  
)
```

Columns & expo

`col("someCol") - 5`

`expr("someCol - 5")`

`(((col("someCol") + 5) * 200) - 6) < col("otherCol")`

`# Boolean column`

`expr("((someCol + 5) * 200) - 6) < otherCol")`

`# Boolean column`

Col example

```
import pyspark.sql.functions as F
newDf = df.select(
    F.col("DEST_COUNTRY_NAME"),
    F.col("ORIGIN_COUNTRY_NAME"),
    F.col("count")*2 - F.sin("count"))
newDf.show(2)
```

```
+-----+-----+-----+
|DEST_COUNTRY_NAME|ORIGIN_COUNTRY_NAME|((count * 2) - SIN(count))|
+-----+-----+-----+
|    United States|          Romania|          29.349712159842884|
|    United States|          Croatia|          1.1585290151921035|
+-----+-----+-----+
```

expr

```
from pyspark.sql.functions import expr
```

```
newDf = df.select(  
    expr("DEST_COUNTRY_NAME"),  
    expr("ORIGIN_COUNTRY_NAME"),  
    expr("count *2 - sin(count)").alias("Goofy"))  
newDf.show(2)
```

```
+-----+-----+-----+  
|DEST_COUNTRY_NAME|ORIGIN_COUNTRY_NAME|          Goofy|  
+-----+-----+-----+  
|    United States|          Romania|29.349712159842884|  
|    United States|          Croatia|1.1585290151921035|  
+-----+-----+-----+
```

select + expr

```
newDf = df.selectExpr("DEST_COUNTRY_NAME",  
                      "ORIGIN_COUNTRY_NAME",  
                      "count *2 - sin(count) as Goofier")
```

```
newDf.show(2)
```

```
+-----+-----+-----+  
| DEST_COUNTRY_NAME | ORIGIN_COUNTRY_NAME |          Goofier |  
+-----+-----+-----+  
|   United States |          Romania | 29.349712159842884 |  
|   United States |          Croatia | 1.1585290151921035 |  
+-----+-----+-----+
```

Adding to Existing

```
df.selectExpr(  
    "*",  
    "(DEST_COUNTRY_NAME = ORIGIN_COUNTRY_NAME) as withinCountry"  
).show(2)
```

```
+-----+-----+-----+-----+  
|DEST_COUNTRY_NAME|ORIGIN_COUNTRY_NAME|count|withinCountry|  
+-----+-----+-----+-----+  
|    United States|          Romania|    15|         false|  
|    United States|          Croatia|     1|         false|  
+-----+-----+-----+-----+
```

Aggregate functions

```
newDf = df.selectExpr("sum(count) as `Total Flights`",  
                      "count(DEST_COUNTRY_NAME) as `Country Pairs`",  
                      "count(Distinct(DEST_COUNTRY_NAME)) as Destinations",  
                      "count(Distinct(ORIGIN_COUNTRY_NAME)) as Origins"  
)  
newDf.show()
```

```
+-----+-----+-----+-----+  
|Total Flights|Country Pairs|Destinations|Origins|  
+-----+-----+-----+-----+  
|      453316|          256|          132|      125|  
+-----+-----+-----+-----+
```

So What Functions can we use?

See `pyspark.sql.functions`

`pyspark.sql.DataFrameStatFunctions`

`pyspark.sql.DataFrameNaFunctions`

Adding Columns with withColumn

```
import pyspark.sql.functions as F
df.withColumn("numberOne", F.lit(1)).show(2)
```

```
+-----+-----+-----+-----+
|DEST_COUNTRY_NAME|ORIGIN_COUNTRY_NAME|count|numberOne|
+-----+-----+-----+-----+
|    United States|          Romania|    15|         1|
|    United States|          Croatia|     1|         1|
+-----+-----+-----+-----+
```

```
df.withColumn("Random", F.rand()).show(2)
```

```
+-----+-----+-----+-----+
|DEST_COUNTRY_NAME|ORIGIN_COUNTRY_NAME|count|          Random|
+-----+-----+-----+-----+
|    United States|          Romania|    15|0.6922143025538695|
|    United States|          Croatia|     1|0.4291601163044023|
+-----+-----+-----+-----+
```

WithColumn & Expr

```
import pyspark.sql.functions as F
```

```
in_country = df.withColumn(  
    "withinCountry",  
    F.expr("ORIGIN_COUNTRY_NAME == DEST_COUNTRY_NAME")  
)
```

```
in_country.show(2)
```

```
+-----+-----+-----+-----+  
|DEST_COUNTRY_NAME|ORIGIN_COUNTRY_NAME|count|withinCountry|  
+-----+-----+-----+-----+  
|    United States|          Romania|    15|         false|  
|    United States|          Croatia|     1|         false|  
+-----+-----+-----+-----+
```

Rename

```
df.withColumnRenamed("DEST_COUNTRY_NAME", "dest").columns
```

```
['dest', 'ORIGIN_COUNTRY_NAME', 'count']
```

to_date

```
reader = spark.read
reader.option("header", True)
reader.option("inferSchema", True)
reader.option("sep", "\t")
dwell_df = reader.csv("smallDwell.tsv")
dwell_df.show(2)
```

	date	dwell-time
	2014-12-31 16:03:43	74
	2014-12-31 16:32:12	109

```
import pyspark.sql.functions as F
dwellDf.withColumn("Date", F.to_date(F.col("date"))).show(2)
```

	Date	dwell-time
	2014-12-31	74
	2014-12-31	109

Date, Hour, format

```
import pyspark.sql.functions as F
with_hour = dwell_df.withColumnRenamed("date", "TimeStamp"). \
    withColumn("Date", F.to_date(col("TimeStamp"))). \
    withColumn("Hour", F.hour(col("TimeStamp"))). \
    withColumn("Month", F.date_format(col("Date"),"MMMM"))
with_hour.show(2)
```

TimeStamp dwell-time	Date Hour	Month
2014-12-31 16:03:43	74 2014-12-31 16	December
2014-12-31 16:32:12	109 2014-12-31 16	December

Single Pass

```
import pyspark.sql.functions as F
with_hour = dwell_dff.withColumnRenamed("date", "TimeStamp"). \
    withColumn("Date", F.to_date(col("TimeStamp"))). \
    withColumn("Hour", F.hour(col("TimeStamp"))). \
    withColumn("Month", F.date_format(col("Date"),"MMMM"))
with_hour.explain()
```

== Physical Plan ==

```
*Project [date#1116 AS TimeStamp#1656, dwell-time#1117,
  to_date(cast(date#1116 as date)) AS Date#1660,
  hour(date#1116, Some(America/Los_Angeles)) AS Hour#1665,
  date_format(cast(to_date(cast(date#1116 as date)) as timestamp), MMMM, Some(America/
  Los_Angeles)) AS Month#1671]
+- *FileScan csv [date#1116,dwell-time#1117]
    Batched: false, Format: CSV,
    Location: InMemoryFileIndex[file:/Users/whitney/Courses/696/Fall17/notebookExamples/
smallDwell.tsv],
    PartitionFilters: [], PushedFilters: [], ReadSchema: struct<date:timestamp,dwell-time:int>
```

Dropping Columns

```
df = spark.read.json(flight_file)
df.show(2)
```

```
+-----+-----+-----+
|DEST_COUNTRY_NAME|ORIGIN_COUNTRY_NAME|count|
+-----+-----+-----+
|    United States|          Romania|    15|
|    United States|          Croatia|     1|
+-----+-----+-----+
```

```
df.drop("ORIGIN_COUNTRY_NAME").columns
```

```
['DEST_COUNTRY_NAME', 'count']
```

Selecting Rows filter = where

```
import pyspark.sql.functions as F
```

```
col_condition = df.filter(F.col("count") < 2).take(2)
```

```
conditional = df.where("count < 2").take(2)
```

```
df.where(F.col("count") < 2) \
```

```
  .where(F.col("ORIGIN_COUNTRY_NAME") != "Croatia") \
```

```
  .show(2)
```

```
+-----+-----+-----+
|DEST_COUNTRY_NAME|ORIGIN_COUNTRY_NAME|count|
+-----+-----+-----+
|      United States|           Singapore|    1|
|             Moldova|      United States|    1|
+-----+-----+-----+
```


Plan

```
df.where(F.col("count") < 2) \
  .where(F.col("ORIGIN_COUNTRY_NAME") != "Croatia") \
  .explain()
```

== Physical Plan ==

```
*(1) Filter (((isnotnull(count#318L) AND isnotnull(ORIGIN_COUNTRY_NAME#317)) AND
(count#318L < 2)) AND NOT (ORIGIN_COUNTRY_NAME#317 = Croatia))
```

```
+-- FileScan json [DEST_COUNTRY_NAME#316,ORIGIN_COUNTRY_NAME#317,count#318L]
Batched: false,
```

```
DataFilters: [
```

```
  isnotnull(count#318L),
```

```
  isnotnull(ORIGIN_COUNTRY_NAME#317),
```

```
  (count#318L < 2),
```

```
  NOT (ORIGIN_COUNTRY_..., Format: JSON, Location: InMemoryFileIndex[file:/Users/
whitney/Courses/696/Fall17/SparkBookData/flight-data/json/2015-summ..., PartitionFilters:
[], PushedFilters: [IsNotNull(count), IsNotNull(ORIGIN_COUNTRY_NAME),
LessThan(count,2), Not(EqualTo(ORIGIN_COUNTRY_...,
```

```
ReadSchema:
```

```
struct<DEST_COUNTRY_NAME:string,ORIGIN_COUNTRY_NAME:string,count:bigint>
```

Another where

```
in_country = df.withColumn("withinCountry",  
    expr("ORIGIN_COUNTRY_NAME == DEST_COUNTRY_NAME")) \  
    .where(F.col("withinCountry"))
```

```
in_country.show(2)
```

```
+-----+-----+-----+-----+  
|DEST_COUNTRY_NAME|ORIGIN_COUNTRY_NAME|count|withinCountry|  
+-----+-----+-----+-----+  
|    United States|    United States|370002|          true|  
+-----+-----+-----+-----+
```

where's are anded

```
import pyspark.sql.functions as F
```

```
frequent = F.col("count") > 100
```

```
to_USA = F.col("DEST_COUNTRY_NAME").contains("United States")
```

```
frequent_to_USA = df.where(frequent).where(to_USA)
```

```
frequent_to_USA.show(5)
```

```
+-----+-----+-----+
|DEST_COUNTRY_NAME|ORIGIN_COUNTRY_NAME|count|
+-----+-----+-----+
|    United States|          Ireland   |  344 |
|    United States|    Sint Maarten   |  325 |
|    United States|          Russia    |  161 |
|    United States|    Netherlands    |  660 |
|    United States|          Ecuador   |  300 |
+-----+-----+-----+
```

or

```
import org.apache.spark.sql.functions.col
```

```
val frequent = col("count") > 100
```

```
val toUSA = col("DEST_COUNTRY_NAME").contains("United States")
```

```
val frequentToUSA = df.where(frequent.or(toUSA))
```

```
frequentToUSA.show(7)
```

or

```
import pyspark.sql.functions as F
```

```
frequent_to_USA = df.where(F.expr("(count > 100) or (DEST_COUNTRY_NAME == 'United States')"))
```

```
frequent_to_USA.show(7)here(frequent.or(toUSA))
```

```
frequentToUSA.show(7)
```

DEST_COUNTRY_NAME	ORIGIN_COUNTRY_NAME	count
United States	Romania	15
United States	Croatia	1
United States	Ireland	344
United States	India	62
United States	Singapore	1
United States	Grenada	62
Costa Rica	United States	588

!, not

```
val inCountry = df.withColumn(  
  "withinCountry",  
  expr("ORIGIN_COUNTRY_NAME == DEST_COUNTRY_NAME")  
).where(!col("withinCountry"))
```

```
inCountry.show(5)
```

```
).where(not(col("withinCountry")))
```

!, not

```
import pyspark.sql.functions as F
in_country = df.withColumn(
    "withinCountry",
    expr("ORIGIN_COUNTRY_NAME == DEST_COUNTRY_NAME")
).where(F.col("withinCountry") == False)
```

```
in_country.show(5)
```

```
+-----+-----+-----+-----+
|DEST_COUNTRY_NAME|ORIGIN_COUNTRY_NAME|count|withinCountry|
+-----+-----+-----+-----+
|    United States|          Romania|    15|         false|
|    United States|          Croatia|     1|         false|
|    United States|          Ireland|   344|         false|
|           Egypt|    United States|    15|         false|
|    United States|           India|    62|         false|
+-----+-----+-----+-----+
```

Distinct Rows - distinct

```
df.select("ORIGIN_COUNTRY_NAME").\
  distinct().\
  sort(col("ORIGIN_COUNTRY_NAME")).\
  take(10)
```

```
[Row(ORIGIN_COUNTRY_NAME='Angola'),  
Row(ORIGIN_COUNTRY_NAME='Anguilla'),  
Row(ORIGIN_COUNTRY_NAME='Antigua and Barbuda'),  
Row(ORIGIN_COUNTRY_NAME='Argentina'),  
Row(ORIGIN_COUNTRY_NAME='Aruba'),  
Row(ORIGIN_COUNTRY_NAME='Australia'),  
Row(ORIGIN_COUNTRY_NAME='Austria'),  
Row(ORIGIN_COUNTRY_NAME='Azerbaijan'),  
Row(ORIGIN_COUNTRY_NAME='Bahrain'),  
Row(ORIGIN_COUNTRY_NAME='Barbados')]
```


Sort === orderBy

```
df.sort("count").show(5)
```

```
df.orderBy("count", "DEST_COUNTRY_NAME").show(5)
```

```
df.orderBy(F.col("count"), F.col("DEST_COUNTRY_NAME")).show(5)
```

```
import pyspark.sql.functions as F
```

```
df.orderBy(expr("count desc")).show(2)
```

```
df.orderBy(F.desc("count"), F.asc("DEST_COUNTRY_NAME")).show(2)
```

WTF?

```
topTenDF = df.orderBy(expr("count asc"))  
topTenDF.show(3)
```

```
+-----+-----+-----+  
| DEST_COUNTRY_NAME | ORIGIN_COUNTRY_NAME | count |  
+-----+-----+-----+  
|           Moldova |           United States |      1 |  
|   United States |           Singapore |      1 |  
|   United States |           Croatia |      1 |  
+-----+-----+-----+
```

```
topTenDF = df.orderBy(expr("count desc"))  
topTenDF.show(3)
```

```
+-----+-----+-----+  
| DEST_COUNTRY_NAME | ORIGIN_COUNTRY_NAME | count |  
+-----+-----+-----+  
|           Moldova |           United States |      1 |  
|   United States |           Singapore |      1 |  
|   United States |           Croatia |      1 |  
+-----+-----+-----+
```

Limit

```
import pyspark.sql.functions as F
```

```
topTenDF = df.orderBy(F.desc("count")).limit(10)
```

```
topTenDF.show()
```

```
topTenDF.count()
```

```
+-----+-----+-----+
|DEST_COUNTRY_NAME|ORIGIN_COUNTRY_NAME|count|
+-----+-----+-----+
|    United States|    United States|370002|
|    United States|           Canada|  8483|
|           Canada|    United States|  8399|
|    United States|           Mexico|  7187|
|           Mexico|    United States|  7140|
|    United Kingdom|    United States|  2025|
|    United States|    United Kingdom|  1970|
|           Japan|    United States|  1548|
|    United States|           Japan|  1496|
|           Germany|    United States|  1468|
+-----+-----+-----+
```

Take, Collect

Return DF to master node as Array

`take(n)`

`collect` returns all

`on_master = df.take(3)`

`on_master`

```
[Row(DEST_COUNTRY_NAME='United States', ORIGIN_COUNTRY_NAME='Romania', count=15),  
Row(DEST_COUNTRY_NAME='United States', ORIGIN_COUNTRY_NAME='Croatia', count=1),  
Row(DEST_COUNTRY_NAME='United States', ORIGIN_COUNTRY_NAME='Ireland', count=344)]
```

`on_master[0]`

```
Row(DEST_COUNTRY_NAME='United States', ORIGIN_COUNTRY_NAME='Romania', count=15)
```

`on_master[0][0]`

```
'United States'
```

`on_master[0]['DEST_COUNTRY_NAME']`

```
'United States'
```

toPanda

```
as_panda_df = df.toPandas()  
as_panda_df.head(3)
```

	DEST_COUNTRY_NAME	ORIGIN_COUNTRY_NAME	count
0	United States	Romania	15
1	United States	Croatia	1
2	United States	Ireland	344

Appending Rows to a DataFrame

```
import pyspark.sql as sql
import pyspark.sql.functions as F
```

union add DataFrame to end

```
schema = df.schema
```

Schemas of the two DataFrames
must match

```
newRows = [sql.Row("New Country", "Other Country", 5),
           sql.Row("New Country 2", "Other Country 3", 1)]
```

```
parallelizedRows = spark.sparkContext.parallelize(newRows)
```

```
newDF = spark.createDataFrame(parallelizedRows, schema)
added = df.union(newDF)
```

```
added.where("count = 1"). \
  where(F.col("ORIGIN_COUNTRY_NAME") != "United States"). \
  show()
```

Output

```
+-----+-----+-----+
| DEST_COUNTRY_NAME | ORIGIN_COUNTRY_NAME | count |
+-----+-----+-----+
| United States | Croatia | 1 |
| United States | Singapore | 1 |
| United States | Gibraltar | 1 |
| United States | Cyprus | 1 |
| United States | Estonia | 1 |
| United States | Lithuania | 1 |
| United States | Bulgaria | 1 |
| United States | Georgia | 1 |
| United States | Bahrain | 1 |
| United States | Papua New Guinea | 1 |
| United States | Montenegro | 1 |
| United States | Namibia | 1 |
| New Country 2 | Other Country 3 | 1 |
+-----+-----+-----+
```

Random Samples

```
tenDF = df.limit(10)
tenDF.show
```

```
seed = 5
withReplacement = False
fraction = 0.5
```

```
smallSample = tenDF.sample(withReplacement, fraction, seed)
smallSample.show()
```

DEST_COUNTRY_NAME	ORIGIN_COUNTRY_NAME	count
United States	Romania	15
United States	Croatia	1
United States	Ireland	344
Egypt	United States	15
United States	India	62
United States	Singapore	1
United States	Grenada	62
Costa Rica	United States	588
Senegal	United States	40
Moldova	United States	1

DEST_COUNTRY_NAME	ORIGIN_COUNTRY_NAME	count
United States	Romania	15
Egypt	United States	15
United States	India	62
Costa Rica	United States	588
Senegal	United States	40

Random Splits

Split DF into two disjoint parts randomly

One dataframe for training

One for validation

randomSplit

seed = 5

twoDF = tenDF.randomSplit((0.25, 0.75), seed)

twoDF[0].show()

twoDF[1].show()

DEST_COUNTRY_NAME	ORIGIN_COUNTRY_NAME	count
United States	Romania	15
United States	Croatia	1
United States	Ireland	344
Egypt	United States	15
United States	India	62
United States	Singapore	1
United States	Grenada	62
Costa Rica	United States	588
Senegal	United States	40
Moldova	United States	1

DEST_COUNTRY_NAME	ORIGIN_COUNTRY_NAME	count
Costa Rica	United States	588
United States	Ireland	344
United States	Romania	15

DEST_COUNTRY_NAME	ORIGIN_COUNTRY_NAME	count
Egypt	United States	15
Moldova	United States	1
Senegal	United States	40
United States	Croatia	1
United States	Grenada	62
United States	India	62
United States	Singapore	1

Weights Normalized to 1

```
twoDF = tenDF.randomSplit([0.25, 0.50])
```

```
twoDF[0].show()
```

```
twoDF[1].show()
```

DEST_COUNTRY_NAME	ORIGIN_COUNTRY_NAME	count
Costa Rica	United States	588
United States	Ireland	344
United States	Romania	15

DEST_COUNTRY_NAME	ORIGIN_COUNTRY_NAME	count
Egypt	United States	15
Moldova	United States	1
Senegal	United States	40
United States	Croatia	1
United States	Grenada	62
United States	India	62
United States	Singapore	1

pyspark.sql.functions

283 functions

```
from pyspark.sql.functions import sin
```

```
df = spark.createDataFrame([(1, "John Doe", 21)], ("id", "name", "age"))  
df.select("id","name",sin("age")).show()
```

```
+---+-----+-----+  
| id | name | SIN(age) |  
+---+-----+-----+  
|  1 | John Doe | 0.8366556385360561 |  
+---+-----+-----+
```

User Defined Functions on DataFrames

```
def to_upper(s):  
    if s is not None:  
        return s.upper()
```

```
df = spark.createDataFrame([(1, "John Doe", 21)], ("id", "name", "age"))  
df.select("id", to_upper("name"), "age").show()
```

```
+----+-----+----+  
| id |    NAME | age |  
+----+-----+----+  
|  1 | John Doe |  21 |  
+----+-----+----+
```

Function is applied to the column name not the data!

User Defined Functions on DataFrames

```
from pyspark.sql.types import IntegerType, StringType
from pyspark.sql.functions import udf
```

```
slen = udf(lambda s: len(s), IntegerType())
```

```
def add_one(x):
    if x is not None:
        return x + 1
```

```
add_one_udf = udf(add_one, IntegerType())
```

```
def to_upper(s):
    if s is not None:
        return s.upper()
```

```
to_upper_udf = udf(to_upper, StringType())
```

User Defined Functions on DataFrames

```
df = spark.createDataFrame([(1, "John Doe", 21)], ("id", "name", "age"))  
df.select(slen("name").alias("slen(name)"), to_upper_udf("name"), add_one_udf("age")).show()
```

```
+-----+-----+-----+  
|slen(name)|to_upper(name)|add_one(age)|  
+-----+-----+-----+  
|          8|      JOHN DOE|           22|  
+-----+-----+-----+
```

Type Mismatch

```
df = spark.createDataFrame([(1, "John Doe", 21)], ("id", "name", "age"))  
df.select("name", add_one_udf("name"), "age").show()
```

Python

Exception

Scala

Null values

Aggregations

Summarize

groupBy

roll up

cube

window

Aggregation Functions

count

countDistinct

approx_count_distinct

first, last

min, max

sum

sumDistinct

avg, mean

variance, var_samp, var_pop

stddev, stddev_samp, stddev_pop

skewness, kurtosis

Covariance & Correlation

corr, covar_samp, covar_pop

Example

```
flight_file = "/Users/whitney/Courses/696/Fall17/SparkBookData/flight-data/json/2015-summary.json"  
flight_df = spark.read.json(flight_file)  
flight_df.show(3)
```

```
+-----+-----+-----+  
|DEST_COUNTRY_NAME|ORIGIN_COUNTRY_NAME|count|  
+-----+-----+-----+  
|    United States|          Romania|    15|  
|    United States|          Croatia|     1|  
|    United States|          Ireland|   344|  
+-----+-----+-----+
```

Example

```
import pyspark.sql.functions as F
newDf = flight_df.select(F.sum("count").alias('Sum'),
                        F.mean("count").alias('Mean'),
                        F.max("count").alias("Max"),
                        F.stddev_samp("count").alias("Sample StdDev"),
                        F.stddev_pop("count").alias("Pop StdDev"),
                        F.count("DEST_COUNTRY_NAME").alias("Count"))
newDf.show()
```

```
+-----+-----+-----+-----+-----+-----+
|   Sum|      Mean|    Max|  Sample StdDev|  Pop StdDev|Count|
+-----+-----+-----+-----+-----+-----+
|453316|1770.765625|370002|23126.516918551915|23081.30374350104|  256|
+-----+-----+-----+-----+-----+-----+
```

countDistinct

```
import pyspark.sql.functions as F
newDf = flight_df.select(
    F.countDistinct(F.col("DEST_COUNTRY_NAME")).alias("Distinct Dest"),
    F.countDistinct(F.col("ORIGIN_COUNTRY_NAME")).alias("Distinct Origin"))

newDf.show()
```

```
+-----+-----+
|Distinct Dest|Distinct Origin|
+-----+-----+
|           132|           125|
+-----+-----+
```

countDistinct

```
import pyspark.sql.functions as F
newDf = flight_df.select(
    F.countDistinct(
        F.col("DEST_COUNTRY_NAME"),
        F.col("ORIGIN_COUNTRY_NAME")).alias("Distinct Pair"))

newDf.show()
```

```
+-----+
|Distinct Pair|
+-----+
|           256|
+-----+
```

countDistinct

```
import pyspark.sql.functions as F
newDf = flight_df.select(
    F.countDistinct(
        "DEST_COUNTRY_NAME",
        "ORIGIN_COUNTRY_NAME",
        "count").alias("Distinct Rows"))

newDf.show()
```

```
+-----+
|Distinct Rows|
+-----+
|                256|
+-----+
```

countDistinct

```
pyspark.sql.functions.countDistinct(col, *cols)
```

```
pyspark.sql.functions.count(col)
```


Group By Data

```
reader = spark.read  
reader.option("header",True).option("inferSchema",True)  
ordersDF = reader.csv("orders.csv")  
ordersDF.show()
```

```
+-----+-----+  
|customer|amount|  
+-----+-----+  
|         a |      2 |  
|         b |      8 |  
|         a |      3 |  
|         c |      9 |  
|         a |      4 |  
|         b |     16 |  
|         c |     11 |  
|         b |     24 |  
|         c |     30 |  
+-----+-----+
```

groupBy

```
import pyspark.sql.functions as F

amountGrouped = ordersDF.groupBy("customer") \
    .agg(
        F.sum("amount").alias("Total"),
        F.mean("amount").alias("Average"),
        F.count("amount").alias("Number of Orders"))
amountGrouped.sort("customer").show()
```

customer	amount
a	2
b	8
a	3
c	9
a	4
b	16
c	11
b	24
c	30

customer	Total	Average	Number of Orders
a	9	3.0	3
b	48	16.0	3
c	50	16.666666666666668	3

How does this work?

```
import pyspark.sql.functions as F

amountGrouped = ordersDF.groupBy("customer") \
    .agg(F.sum("amount").alias("Total"))
amountGrouped.sort("customer").show()
```

```
+-----+-----+
|customer|Total|
+-----+-----+
|      a |    9|
|      b |   48|
|      c |   50|
+-----+-----+
```

groupBy

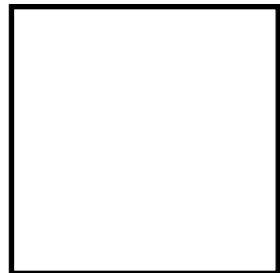
Workers

a	2
b	8
a	3

c	9
a	4
b	16

c	11
b	24
c	30

Master



customer	amount
a	2
b	8
a	3
c	9
a	4
b	16
c	11
b	24
c	30

groupBy

Workers

a	5
b	8

c	9
a	4
b	16

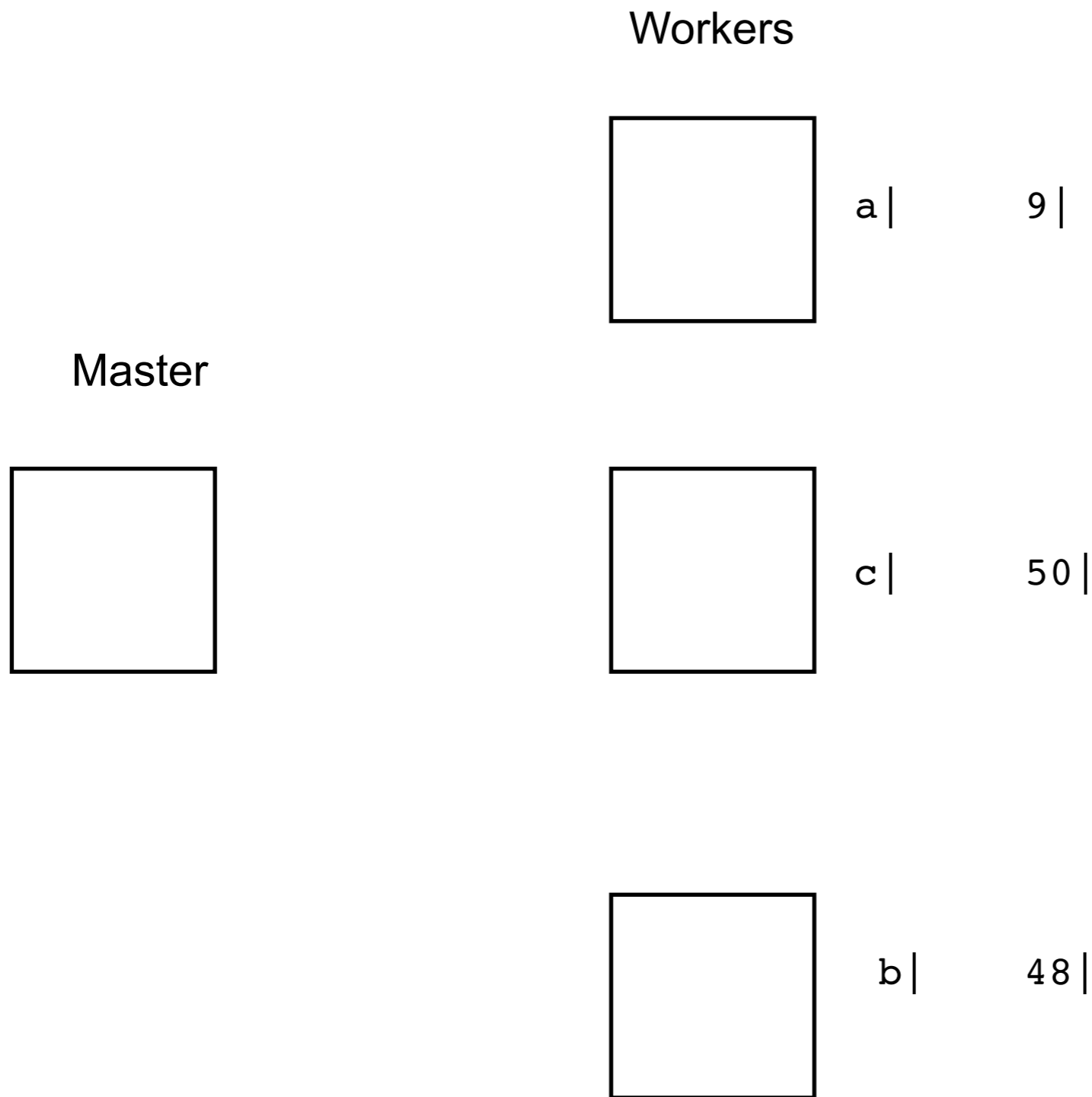
c	41
b	24

Master

--	--

customer	amount
a	2
b	8
a	3
c	9
a	4
b	16
c	11
b	24
c	30

groupBy



customer	amount
a	2
b	8
a	3
c	9
a	4
b	16
c	11
b	24
c	30

groupBy
Transformation