Introduction to Spark

This notebook was designed with Spark 2.0.1 in October 2016 and parts were updated with 2.1.0 in mind in July 2017.

Objectives

- Understand the relationship between Hadoop, MapReduce, and Spark
  - Spark as a higher-level alternative to MapReduce for distributed computation
  - Spark as a happy friend of HadoopDFS, S3, etc. for distributed data
  - Describe the advantages/disadvantages of Spark compared to Hadoop MapReduce
- RDD
  - Define what an RDD is, by its properties and operations
  - Explain the different operation types on an RDD
    - transformations
    - actions
  - Apply different transformations through use cases
  - Describe what persisting/caching an RDD means, and situations where this is useful
- DataFrame
  - Define what a DataFrame is, by its properties and operations
    - Schema, Syntax
  - Compare DataFrames to RDDs
    - Operation Types (same)
    - Cacheability (same)
    - Syntax (different)

Overview and Context

What is Big Data?

Too big for local. What do we do?

- Use a SQL or NoSQL DB
  - Sometimes these databases themselves are distributed
- Use a distributed system

Hadoop, MapReduce, Spark

Local versus Distributed Systems

A local system uses the resources of a single machine.

A distributed system uses the resources of multiple machines.

- After a certain point, it's easier to scale/add resources to a distributed system than it is to add them to a single machine system. Distributed systems include fault tolerance. If a single task or even a whole machine has a failure, the whole system can still go on by re-running that task or running it on a different machine.

Overview of Hadoop Ecosystem

Storage and Computation

Hadoop for *storage and replication* and the Hadoop Distributed Filesystem, HDFS

MapReduce for *computations* across the distributed dataset stored in HDFS

- Job tracker sends code to run on the Task tracker
- Task trackers then allocate resources (CPU, RAM) for the tasks on the worker nodes, monitors the tasks, reruns if necessary...

Overview of Spark

Spark improves on the *compute* side of things.

Spark doesn't compete with Hadoop; in fact, Spark can use data in Hadoop.

People sometimes talk about "Hadoop MapReduce", and contrast that to "Spark". Sometimes they shorten the former to just "Hadoop" and compare "Hadoop versus Spark". But really they mean to contrast "Hadoop & MapReduce" with "Possibly-Hadoop & Spark". The contrast is between MapReduce and Spark, as ways to do computation over distributed datasets.

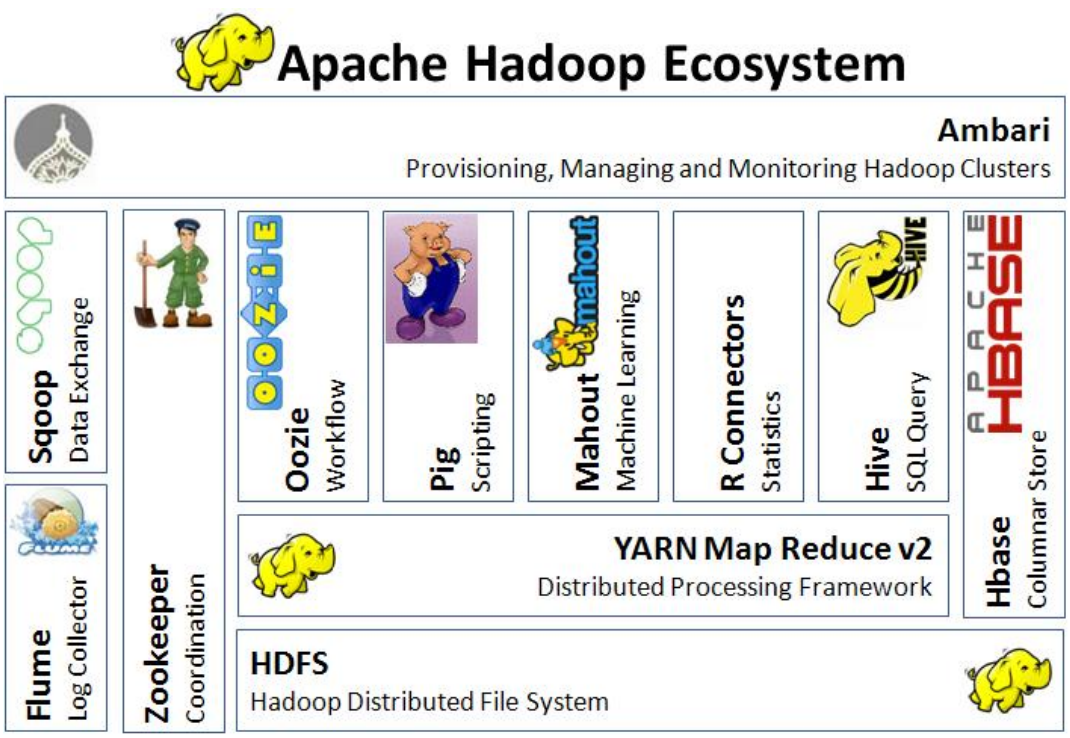
What does Spark do that's so great?

- Speed
  - Spark can perform operations up to 100x faster than MapReduce. How?
    - MapReduce writes data to disk after each map and reduce op. That's slow.
    - Spark keeps most of the data in memory after each transformation, spilling over to disk if necessary. As RAM has gotten cheaper, it makes sense to keep things in it more.
- RDDs
  - Immutable
  - Lazily Evaluated

    - Cacheable
- Two types of operations, for working with large datasets
    - Transformations
        - Recipe to follow ("describe it")
    - Actions
        - Call to action; follow the recipe ("do it")
- Syntax
    - RDD
    - DataFrame
        - Now the standard for Spark's ML capabilities; RDD support is in 'maintenance mode' now, will be deprecated, then removed
- Framework for dealing with large data
    - Not itself a language
        - Written in Scala, which itself is written in Java
    - Clients in Scala, Java, Python, R...
        - Scala and Java are sort of "first class" clients
    - PySpark is how we work with Spark from Python

1. Key Concepts

1.1. MapReduce vs Spark

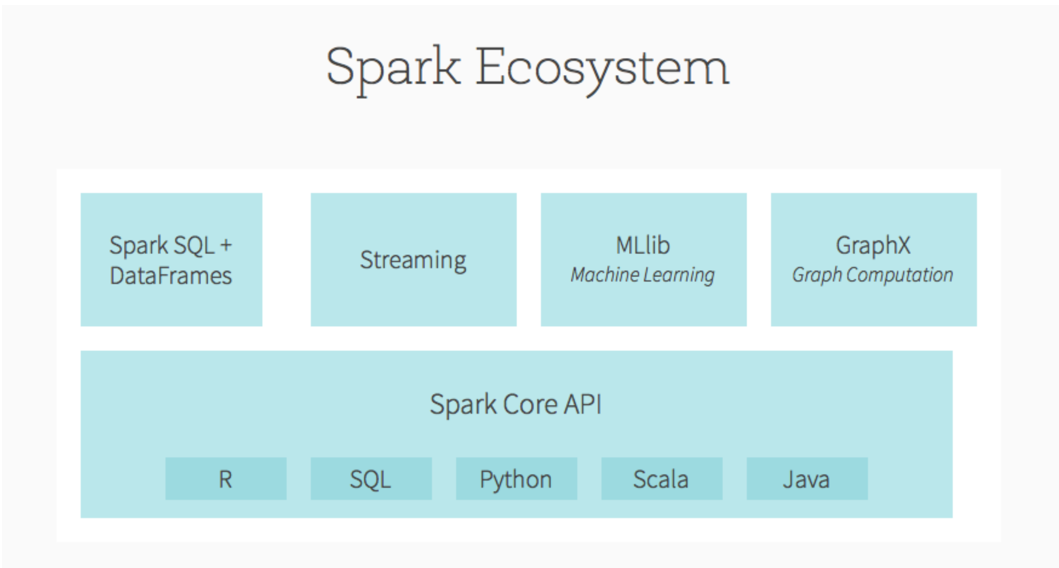


Hadoop MapReduce limits:

- your job has to fit the `<key,value>` paradigm
- no interactions (except by programming)
- each job read from disk: problem with iterative algorithms (machine learning)

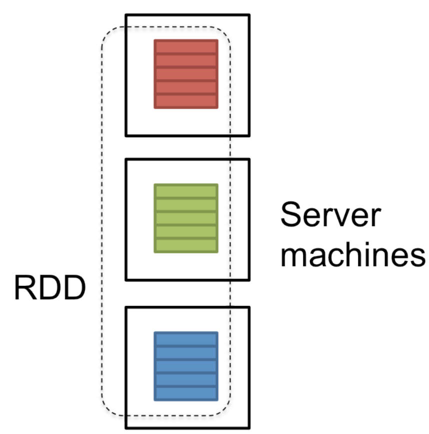
How Spark answers this:

- Spark proposes other processing workflows than MapReduce
- highly efficient distributed operations
- Spark runs in memory and on disk
- Can be up to 100x faster than Hadoop MapReduce in memory, and 10x faster on disk.
- Spark keeps everything in memory when possible, uses lots of it.



1.2. Resilient Distributed Datasets (RDD)

[Image Source (http://horicky.blogspot.com/2015/02/big-data-processing-in-spark.html)]

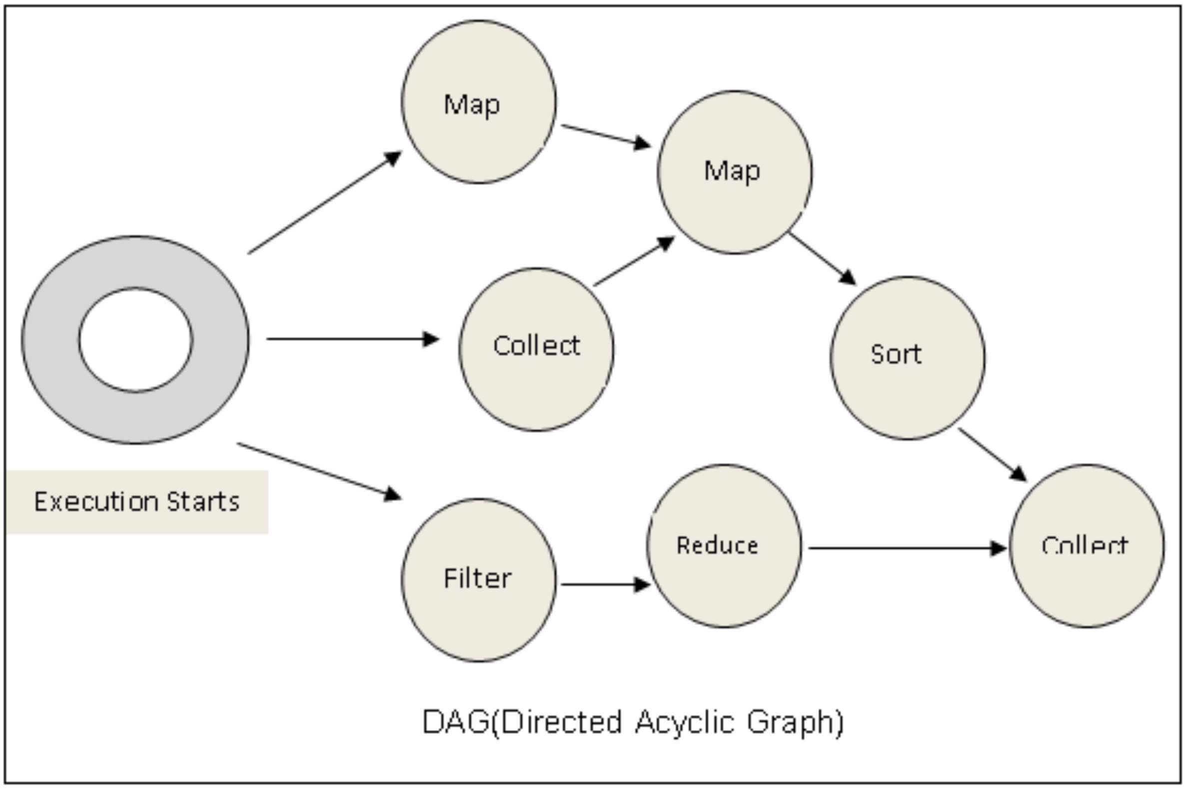


- created from HDFS, S3, HBase, JSON, text, local... or transformed from another RDD
- distributed accross the cluster, partitioned (atomic chunks of data)
- can recover from errors (node failure, slow process)
- traceability of each partition, can re-run the processing
- **immutable** : you cannot *modify* an RDD in place

1.3. A "functional programming paradigm" and DAGs

RDDs are **immutable** ! You can only **transform** an existing RDD into another one.

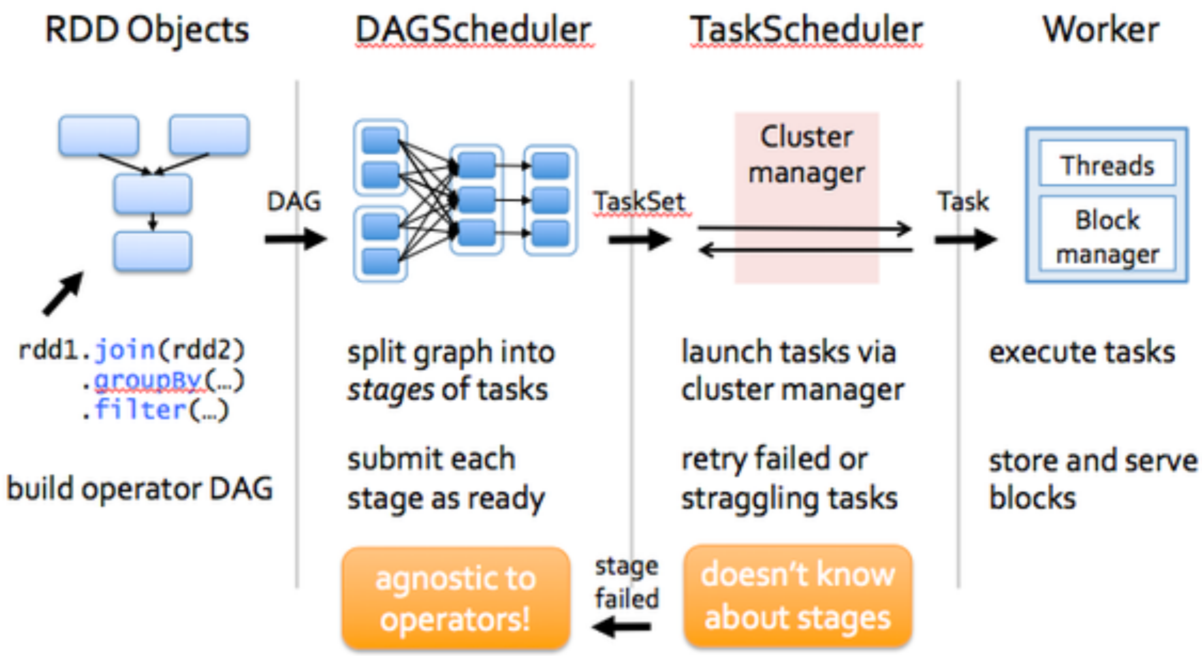
Spark provides many transformations functions. By programming these functions, you construct a **Directed Acyclic Graph** (DAG).



[Image Source ()]

When you use them, these functions are passed from the **client** to the **master**, who then distributes them to workers, who apply them accross their partitions of the RDD.

1.4. Spark architecture : from your coding hands to the cluster



[Image Source ()]

You construct your sequence of transformations in python. Spark functional programming interface builds up a **DAG** This DAG is sent by the **driver** for execution to the **cluster manager**.

1.5. Jargon

Excerpt taken from [Arush Kharbanda (https://www.quora.com/What-exactly-is-Apache-Spark-and-how-does-it-work) on Quora]

Job: A piece of code which reads some input from HDFS or local, performs some computation on the data and writes some output data.

Stages: Jobs are divided into stages. Stages are classified as a Map or reduce stages(Its easier to understand if you have worked on Hadoop and want to correlate). Stages are divided based on computational boundaries, all computations(operators) cannot be Updated in a single Stage. It happens over many stages.

Tasks: Each stage has some tasks, one task per partition. One task is executed on one partition of data on one executor(machine).

DAG: DAG stands for Directed Acyclic Graph, in the present context its a DAG of operators.

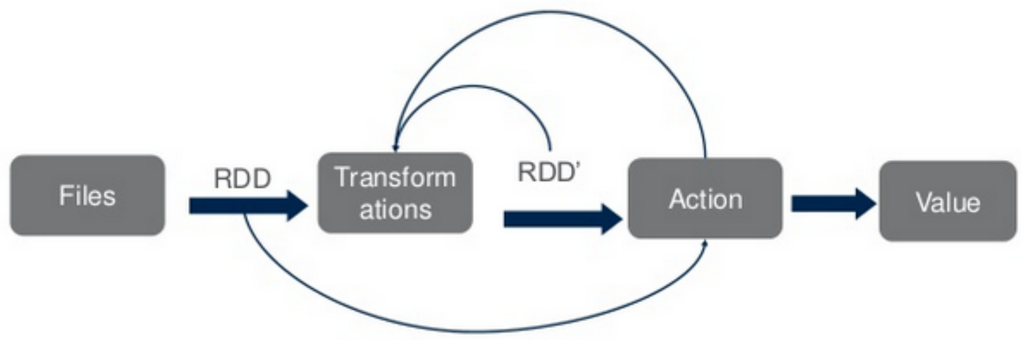
Executor: The process responsible for executing a task.

Driver: The program/process responsible for running the Job over the Spark Engine

Master: The machine on which the Driver program runs

Slave/Worker: The machine on which the Executor program runs

2. Operational Spark in Python



We'll proceed along the usual spark flow (see above).

1. create the enviromnent to run spark from python
2. extract RDDs from files
3. run some transformations
4. execute actions to obtain values (local objects in python)

Brainstorming: So, let's suppose you have this thing called an RDD, which is just basically a dataset made of rows and values. What are all the operations you'd like to do to that RDD ?

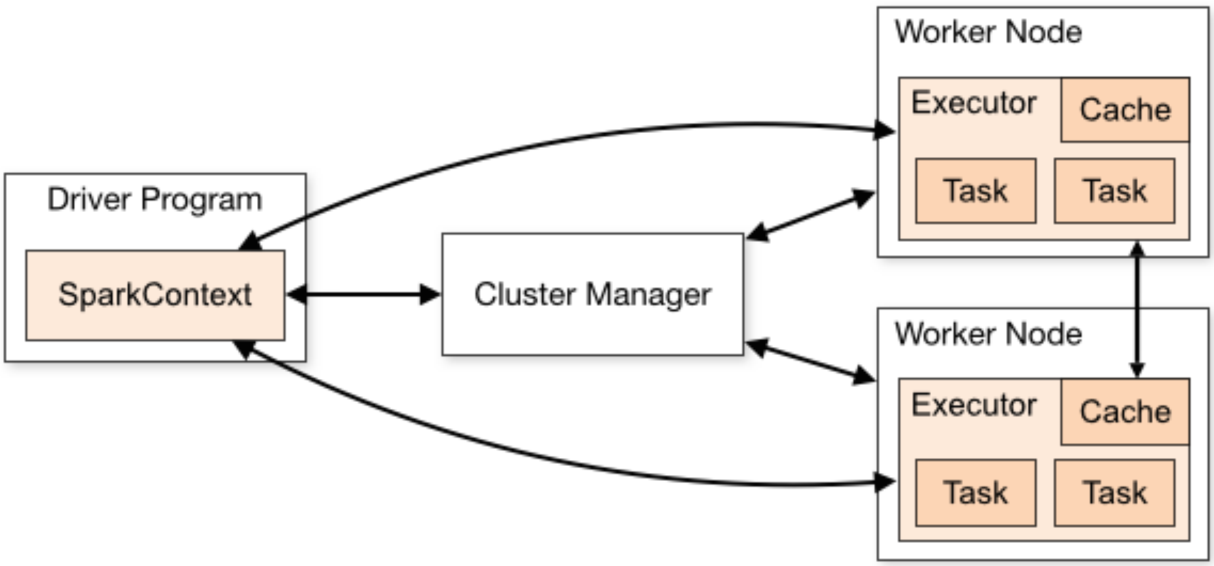
In [140]:
```
# your ideas here...
```

2.1. Initializing a `SparkContext` in Python

IPython / IPython notebook can be a *client* to interact with the *master*.

The client will have a `SparkContext` that..

1. Acts as a gateway between the client and Spark master
2. Sends code/data from IPython to the master (who then sends it to the workers)



Using:

```
import pyspark as ps
sc = ps.SparkContext('local[4]')
```

will create a "*local*" cluster made of the driver using all 4 cores.

In [141]:
```
import findspark
findspark.init('/home/sparkles/spark-2.1.0-bin-hadoop2.7')
import pyspark as ps    # for the pyspark suite
import warnings         # for displaying warning

try:
    # we try to create a SparkContext to work locally on all cpus available
    sc = ps.SparkContext('local[4]')
    print("Just created a SparkContext")
except ValueError:
    # give a warning if SparkContext already exists (for use inside pyspark)
    warnings.warn("SparkContext already exists in this scope")
```

```
/home/sparkles/.local/lib/python3.5/site-packages/ipykernel_launcher
.py:12: UserWarning: SparkContext already exists in this scope
  if sys.path[0] == '':
```

2.2. Creating an RDD (from files)

RDDs are **immutable**. Once created, you cannot modify them directly. You can only transform them into another RDD.

Functions for creating an RDD from an external source are methods of the SparkContext object `sc`.

Method	Description
`sc.parallelize(array)` ()	Create an RDD from a python array or list
`sc.textFile(path)` ()	Create an RDD from a text file
`sc.pickleFile(path)` ()	Create an RDD from a pickle file

2.2.1. Creating RDDs from local files

`sc.parallelize()` : create an RDD from a python iterable

In [142]:
```
# creating an adhoc list
data_array = [['matthew', 4],
              ['jorge', 8],
              ['josh', 15],
              ['evangeline', 16],
              ['emilie', 23],
              ['yunjin', 42]]

# reading the array/list using SparkContext
rdd = sc.parallelize(data_array)

# to output the content in python [irl, use with great care]
rdd.collect()
```

Out[142]:
```
[['matthew', 4],
 ['jorge', 8],
 ['josh', 15],
 ['evangeline', 16],
 ['emilie', 23],
 ['yunjin', 42]]
```

`sc.textFile()` : from a text file !

The import will give you an rdd made of **strings which are lines of the text file**.

In [143]:
```
# displaying the content of the file in stdout
with open('data/toy_data.txt', 'r') as fin:
    print(fin.read())

# reading the file using SparkContext
rdd = sc.textFile('data/toy_data.txt')

# to output the content in python [irl, use collect() with great care]
rdd.collect()
```

```
matthew,4
jorge,8
josh,15
evangeline,16
emilie,23
yunjin,42
```

Out[143]:
```
['matthew,4', 'jorge,8', 'josh,15', 'evangeline,16', 'emilie,23', 'y
unjin,42']
```

`sc.pickeFile()` : from a HDFS pickle file

The import will give you an rdd composed of whatever table was stored into that file.

In [144]:
```
%ls data/
```

```
aapl.csv*          input.txt*       sales.csv*   sales.txt*      toy_
data.pkl/
cookie_data.txt*  sales2.json.gz*  sales.json*  toy_dataB.txt*  toy_
data.txt*
```

In [145]:
```
# reading the file using SparkContext
rdd = sc.pickleFile('data/toy_data.pkl')

# to output the content in python [irl, use with great care]
rdd.collect()
```

Out[145]:
```
['emilie,23', 'yunjin,42', 'matthew,4', 'jorge,8', 'josh,15', 'evang
eline,16']
```

2.2.2. Creating RDDs from S3

These two functions above can perform loading from an s3 repository too ! Effortless.

Warning: don't .collect() that, or you'll break the internet !

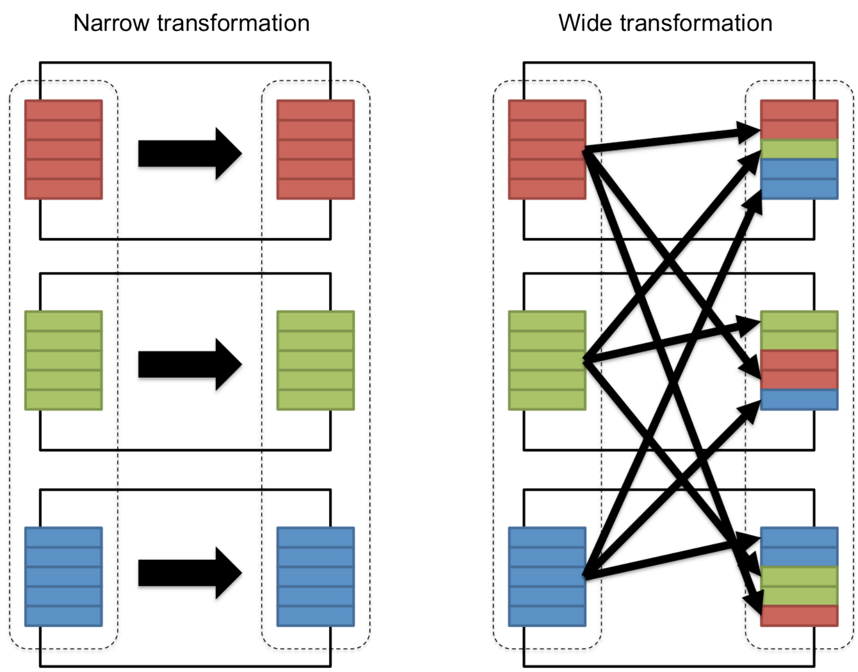
Note: in order to do that, you need to have launched jupyter with the `--packages` options for aws and hadoop.

In [146]:
```
import os

# obtaining your credentials from your environment variables
ACCESS_KEY = os.getenv('AWS_ACCESS_KEY_ID')
SECRET_KEY = os.getenv('AWS_SECRET_ACCESS_KEY')

# link to the S3 repository
link = 's3n://mortar-example-data/airline-data'

# creating an RDD...
rdd = sc.textFile(link)
```

In [147]:
```
# requires s3n packages to be configured #todo
# rdd.getNumPartitions()
# rdd.count()
```

2.3. Transformations : transforming an RDD into another

- They are **lazy**: Spark doesn't apply the transformation right away, it just builds on the **DAG**
- They transform an RDD into another RDD because RDD are **immutable**.
- They can be **wide** or **narrow** (whether they shuffle partitions or not).



[Image Source (http://horicky.blogspot.com/2013/12/spark-low-latency-massively-parallel.html)]

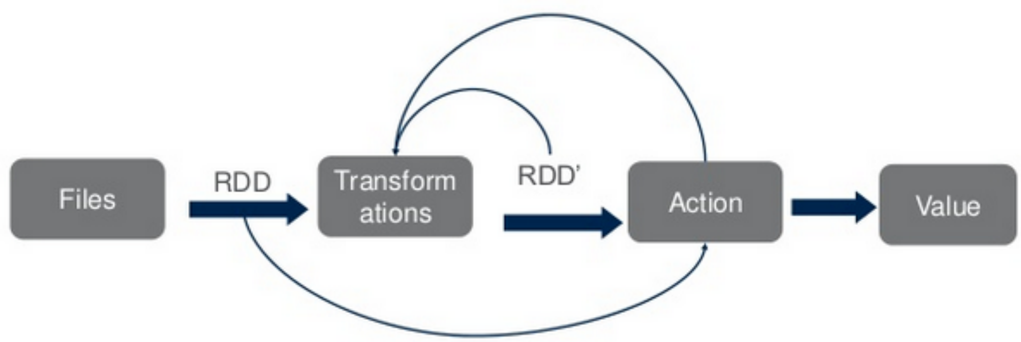
Method	Type	C
.map(func) (http://spark.apache.org/docs/latest/api/python/pyspark.html#pyspark.RDD.map)	transformation	m

[`.flatMap(func)`](http://spark.apache.org/docs/latest/api/python/pyspark.html#pyspark.RDD.flatMap) (http://spark.apache.org/docs/latest/api/python/pyspark.html#pyspark.RDD.flatMap)	transformation	m
[`.filter(func)`](http://spark.apache.org/docs/latest/api/python/pyspark.html#pyspark.RDD.filter) (http://spark.apache.org/docs/latest/api/python/pyspark.html#pyspark.RDD.filter)	transformation	re
[`.sample()`](http://spark.apache.org/docs/latest/api/python/pyspark.html#pyspark.RDD.sample) (http://spark.apache.org/docs/latest/api/python/pyspark.html#pyspark.RDD.sample)	transformation	re
[`.distinct()`](http://spark.apache.org/docs/latest/api/python/pyspark.html#pyspark.RDD.distinct) (http://spark.apache.org/docs/latest/api/python/pyspark.html#pyspark.RDD.distinct)	transformation	re
[`.keys()`](http://spark.apache.org/docs/latest/api/python/pyspark.html#pyspark.RDD.keys) (http://spark.apache.org/docs/latest/api/python/pyspark.html#pyspark.RDD.keys)	transformation	<
[`.values()`](http://spark.apache.org/docs/latest/api/python/pyspark.html#pyspark.RDD.values) (http://spark.apache.org/docs/latest/api/python/pyspark.html#pyspark.RDD.values)	transformation	<
`.join(rddB)`	transformation	<

(http://spark.apache.org/docs/latest/api/python/pyspark.html#pyspark.RDD.join)		
`.reduceByKey()` (http://spark.apache.org/docs/latest/api/python/pyspark.html#pyspark.RDD.reduceByKey)	transformation	<
`.groupByKey()` (http://spark.apache.org/docs/latest/api/python/pyspark.html#pyspark.RDD.groupByKey)	transformation	<
`.sortBy(keyfunc)` (http://spark.apache.org/docs/latest/api/python/pyspark.html#pyspark.RDD.sortBy)	transformation	s
`.sortByKey()` (http://spark.apache.org/docs/latest/api/python/pyspark.html#pyspark.RDD.sortByKey)	transformation	s

2.3.1. Applying transformations and chaining them

Recall the spark flow:



In the sequence below, we will in one sequence:

1. read an RDD from a text file
2. transform by applying `split`
3. transform by filtering
4. transform by casting some columns to their corresponding type.
5. use an action to output the results

Each transformation is a method of an RDD, and returns another RDD.

In [148]:
```
# displaying the content of the file in stdout
with open('data/sales.txt', 'r') as fin:
    print(fin.read())
```

```
#ID     Date            Store   State   Product     Amount
101     11/13/2014      100     WA      331         300.00
104     11/18/2014      700     OR      329         450.00
102     11/15/2014      203     CA      321         200.00
106     11/19/2014      202     CA      331         330.00
103     11/17/2014      101     WA      373         750.00
105     11/19/2014      202     CA      321         200.00
```

In [149]:
```
# Recall: Input functions, reading RDDs from files,
# are functions of the SparkContext.

# reads a text file line by line
rdd1 = sc.textFile('data/sales.txt')

rdd1.collect()  # beware collect() in practice!
```

Out[149]:
```
['#ID    Date          Store   State  Product    Amount',
 '101    11/13/2014    100     WA     331        300.00',
 '104    11/18/2014    700     OR     329        450.00',
 '102    11/15/2014    203     CA     321        200.00',
 '106    11/19/2014    202     CA     331        330.00',
 '103    11/17/2014    101     WA     373        750.00',
 '105    11/19/2014    202     CA     321        200.00']
```

In [150]:
```
rdd1.take(4)
```

Out[150]:
```
['#ID    Date          Store   State  Product    Amount',
 '101    11/13/2014    100     WA     331        300.00',
 '104    11/18/2014    700     OR     329        450.00',
 '102    11/15/2014    203     CA     321        200.00']
```

In [151]:
```
# applies split() to each row
rdd2 = rdd1.map(lambda rowstr : rowstr.split())

rdd2.collect()  # beware collect() in practice!
```

Out[151]:
```
[['#ID', 'Date', 'Store', 'State', 'Product', 'Amount'],
 ['101', '11/13/2014', '100', 'WA', '331', '300.00'],
 ['104', '11/18/2014', '700', 'OR', '329', '450.00'],
 ['102', '11/15/2014', '203', 'CA', '321', '200.00'],
 ['106', '11/19/2014', '202', 'CA', '331', '330.00'],
 ['103', '11/17/2014', '101', 'WA', '373', '750.00'],
 ['105', '11/19/2014', '202', 'CA', '321', '200.00']]
```

In [152]:
```
# filters rows
rdd3 = rdd2.filter(lambda row: not row[0].startswith('#'))

# rdd3.collect()  # beware collect() in practice!
```

In [153]:
```
rdd3 = rdd1.map(lambda rowstr : rowstr.split()).filter(lambda row: not
row[0].startswith('#'))
rdd3.collect()
```

Out[153]:
```
[['101', '11/13/2014', '100', 'WA', '331', '300.00'],
 ['104', '11/18/2014', '700', 'OR', '329', '450.00'],
 ['102', '11/15/2014', '203', 'CA', '321', '200.00'],
 ['106', '11/19/2014', '202', 'CA', '331', '330.00'],
 ['103', '11/17/2014', '101', 'WA', '373', '750.00'],
 ['105', '11/19/2014', '202', 'CA', '321', '200.00']]
```

In [154]:
```
def casting_function(row):
    _id, date, store, state, product, amount = row
    return (int(_id), date, int(store), state, int(product), float(amo
unt))

# applies casting_function to rows
rdd4 = rdd3.map(casting_function)

# shows the result
rdd4.collect()
```

Out[154]:
```
[(101, '11/13/2014', 100, 'WA', 331, 300.0),
 (104, '11/18/2014', 700, 'OR', 329, 450.0),
 (102, '11/15/2014', 203, 'CA', 321, 200.0),
 (106, '11/19/2014', 202, 'CA', 331, 330.0),
 (103, '11/17/2014', 101, 'WA', 373, 750.0),
 (105, '11/19/2014', 202, 'CA', 321, 200.0)]
```

Now, let's see the canonical way to write that in Python...

In [155]:
```
# v0
rdd_sales = sc.textFile('data/sales.txt')

rdd_sales.collect()
```

Out[155]:
```
['#ID    Date          Store   State  Product    Amount',
 '101    11/13/2014    100     WA     331        300.00',
 '104    11/18/2014    700     OR     329        450.00',
 '102    11/15/2014    203     CA     321        200.00',
 '106    11/19/2014    202     CA     331        330.00',
 '103    11/17/2014    101     WA     373        750.00',
 '105    11/19/2014    202     CA     321        200.00']
```

In [156]:
```
# v1
rdd_sales = sc.textFile('data/sales.txt')\
        .map(lambda rowstr : rowstr.split())   # <= JUST ADDED THIS HE
RE

rdd_sales.collect()
```

Out[156]:
```
[['#ID', 'Date', 'Store', 'State', 'Product', 'Amount'],
 ['101', '11/13/2014', '100', 'WA', '331', '300.00'],
 ['104', '11/18/2014', '700', 'OR', '329', '450.00'],
 ['102', '11/15/2014', '203', 'CA', '321', '200.00'],
 ['106', '11/19/2014', '202', 'CA', '331', '330.00'],
 ['103', '11/17/2014', '101', 'WA', '373', '750.00'],
 ['105', '11/19/2014', '202', 'CA', '321', '200.00']]
```

In [157]:
```
# v2
rdd_sales = sc.textFile('data/sales.txt')\
        .map(lambda rowstr : rowstr.split())\
        .filter(lambda row: not row[0].startswith('#'))    # <= JUST A
DDED THIS HERE

rdd_sales.collect()
```

Out[157]:
```
[['101', '11/13/2014', '100', 'WA', '331', '300.00'],
 ['104', '11/18/2014', '700', 'OR', '329', '450.00'],
 ['102', '11/15/2014', '203', 'CA', '321', '200.00'],
 ['106', '11/19/2014', '202', 'CA', '331', '330.00'],
 ['103', '11/17/2014', '101', 'WA', '373', '750.00'],
 ['105', '11/19/2014', '202', 'CA', '321', '200.00']]
```

In [158]:
```
# v3
def casting_function(row):
    _id, date, store, state, product, amount = row
    return (int(_id), date, int(store), state, int(product), float(amo
unt))


rdd_sales = sc.textFile('data/sales.txt')\
        .map(lambda rowstr : rowstr.split())\
        .filter(lambda row: not row[0].startswith('#'))\
        .map(casting_function)   # <= JUST ADDED THIS HERE

rdd_sales.collect()
```

Out[158]:
```
[(101, '11/13/2014', 100, 'WA', 331, 300.0),
 (104, '11/18/2014', 700, 'OR', 329, 450.0),
 (102, '11/15/2014', 203, 'CA', 321, 200.0),
 (106, '11/19/2014', 202, 'CA', 331, 330.0),
 (103, '11/17/2014', 101, 'WA', 373, 750.0),
 (105, '11/19/2014', 202, 'CA', 321, 200.0)]
```

Nice. We did that all at once! Just so we can have a fresh plate, let's create our two RDDs again for more exploration of RDD functionality

In [159]:
```
# let's make a names rdd again...

# creating an adhoc list
data_array = [['matthew', 4],
              ['jorge', 8],
              ['josh', 15],
              ['evangeline', 16],
              ['emilie', 23],
              ['yunjin', 42]]

# reading the array/list using SparkContext
rdd_names = sc.parallelize(data_array)

# to output the content in python [irl, use with great care]
rdd_names.collect()
```

Out[159]:
```
[['matthew', 4],
 ['jorge', 8],
 ['josh', 15],
 ['evangeline', 16],
 ['emilie', 23],
 ['yunjin', 42]]
```

In [160]:
```
# and let's make a sales rdd again...

def casting_function(row):
    _id, date, store, state, product, amount = row
    return (int(_id), date, int(store), state, int(product), float(amo
unt))


rdd_sales = sc.textFile('data/sales.txt')\
        .map(lambda rowstr : rowstr.split())\
        .filter(lambda row: not row[0].startswith('#'))\
        .map(casting_function)

rdd_sales.collect()
```

Out[160]:
```
[(101, '11/13/2014', 100, 'WA', 331, 300.0),
 (104, '11/18/2014', 700, 'OR', 329, 450.0),
 (102, '11/15/2014', 203, 'CA', 321, 200.0),
 (106, '11/19/2014', 202, 'CA', 331, 330.0),
 (103, '11/17/2014', 101, 'WA', 373, 750.0),
 (105, '11/19/2014', 202, 'CA', 321, 200.0)]
```

2.3.2. Mapping

`.map(func)` : applying a function on every row

In [161]:
```
# applying a lambda function to an rdd
rddout = rdd_names.map(lambda x : len(x[0]))

# print out the original rdd
print("before: {}".format(rdd_names.collect()))

# print out the new rdd generated
print("after: {}".format(rddout.collect()))
```

```
before: [['matthew', 4], ['jorge', 8], ['josh', 15], ['evangeline',
16], ['emilie', 23], ['yunjin', 42]]
after: [7, 5, 4, 10, 6, 6]
```

How readable was that?

When using lambda functions, we used to be able to use **argument unpacking** to provide a more readable transformation.

... but after fierce debate, it was taken away with Python 3. See: https://www.python.org/dev/peps/pep-3113/ (https://www.python.org/dev/peps/pep-3113/)

In [162]:
```
# applying a lambda function to an rdd
rddout = rdd_names.map(lambda (name,number) : len(name))  # no-no in P
ython 3, see PEP 3113

# print out the original rdd
print("before: {}".format(rdd_names.collect()))

# print out the new rdd generated
print("after: {}".format(rddout.collect()))
```

```
  File "<ipython-input-162-9c790a899f8d>", line 2
    rddout = rdd_names.map(lambda (name,number) : len(name))  # no-n
o in Python 3, see PEP 3113
                                  ^

SyntaxError: invalid syntax
```

`.flatMap(func)` : applying a function on every row and flattening the resulting lists

In [163]:
```
# applying a lambda function to an rdd (because why not)
rddout = rdd_names.flatMap(lambda x : [x[1], x[1]+2, x[1]+len(x[0])])

# print out the original rdd
print("before: {}".format(rdd_names.collect()))

# print out the new rdd generated
print("after: {}".format(rddout.collect()))
```

```
before: [['matthew', 4], ['jorge', 8], ['josh', 15], ['evangeline',
16], ['emilie', 23], ['yunjin', 42]]
after: [4, 6, 11, 8, 10, 13, 15, 17, 19, 16, 18, 26, 23, 25, 29, 42,
44, 48]
```

2.3.3. Row reduction

`.filter(func)`: filters an RDD using a function that returns boolean values

In [164]:
```
# filtering an rdd
rddout = rdd_sales.filter(lambda x: (x[3] == 'CA'))

# print out the original rdd
print("before: {}".format(rdd_sales.collect()))

# print out the new rdd generated
print("after: {}".format(rddout.collect()))
```

```
before: [(101, '11/13/2014', 100, 'WA', 331, 300.0), (104, '11/18/20
14', 700, 'OR', 329, 450.0), (102, '11/15/2014', 203, 'CA', 321, 200
.0), (106, '11/19/2014', 202, 'CA', 331, 330.0), (103, '11/17/2014',
101, 'WA', 373, 750.0), (105, '11/19/2014', 202, 'CA', 321, 200.0)]
after: [(102, '11/15/2014', 203, 'CA', 321, 200.0), (106, '11/19/201
4', 202, 'CA', 331, 330.0), (105, '11/19/2014', 202, 'CA', 321, 200.
0)]
```

`.sample(withReplacement, fraction, seed)`: sampling an RDD !!

In [165]:
```
# sampling an rdd
rddout = rdd_sales.sample(True, 0.4)

# print out the original rdd
print("before: {}".format(rdd_sales.collect()))

# print out the new rdd generated
print("after: {}".format(rddout.collect()))
```

```
before: [(101, '11/13/2014', 100, 'WA', 331, 300.0), (104, '11/18/20
14', 700, 'OR', 329, 450.0), (102, '11/15/2014', 203, 'CA', 321, 200
.0), (106, '11/19/2014', 202, 'CA', 331, 330.0), (103, '11/17/2014',
101, 'WA', 373, 750.0), (105, '11/19/2014', 202, 'CA', 321, 200.0)]
after: [(102, '11/15/2014', 203, 'CA', 321, 200.0), (106, '11/19/201
4', 202, 'CA', 331, 330.0), (106, '11/19/2014', 202, 'CA', 331, 330.
0), (103, '11/17/2014', 101, 'WA', 373, 750.0)]
```

`.distinct()`: obtaining distinct rows

In [166]:
```
# obtaining distinct values of the "state" column of rdd_sales
rddout = rdd_sales.map(lambda x: x[3]).distinct()

# print out the original rdd
print("before: {}".format(rdd_sales.collect()))

# print out the new rdd generated
print("after: {}".format(rddout.collect()))
```

```
before: [(101, '11/13/2014', 100, 'WA', 331, 300.0), (104, '11/18/20
14', 700, 'OR', 329, 450.0), (102, '11/15/2014', 203, 'CA', 321, 200
.0), (106, '11/19/2014', 202, 'CA', 331, 330.0), (103, '11/17/2014',
101, 'WA', 373, 750.0), (105, '11/19/2014', 202, 'CA', 321, 200.0)]
after: ['CA', 'WA', 'OR']
```

2.3.4. Methods with a <`k,v`> paradigm

`.values()`: returns the values of a RDD made of <`k,v`> pairs

In [167]:
```
rddout = rdd_names.values()

# print out the original rdd
print("before: {}".format(rdd_names.collect()))

# print out the new rdd generated
print("after: {}".format(rddout.collect()))
```

```
before: [['matthew', 4], ['jorge', 8], ['josh', 15], ['evangeline',
16], ['emilie', 23], ['yunjin', 42]]
after: [4, 8, 15, 16, 23, 42]
```

`.keys()`: returns the keys of a RDD made of <`k,v`> pairs

In [168]:
```
rddout = rdd_names.keys()

# print out the original rdd
print("before: {}".format(rdd_names.collect()))

# print out the new rdd generated
print("after: {}".format(rddout.collect()))
```

```
before: [['matthew', 4], ['jorge', 8], ['josh', 15], ['evangeline',
16], ['emilie', 23], ['yunjin', 42]]
after: ['matthew', 'jorge', 'josh', 'evangeline', 'emilie', 'yunjin'
]
```

`rddA.join(rddB)`: join another RDD

In [169]:
```
rdd_salesperstate = rdd_sales.map(lambda x: (x[3],x[5]))

rdd_salesperstate.collect()
```

Out[169]:
```
[('WA', 300.0),
 ('OR', 450.0),
 ('CA', 200.0),
 ('CA', 330.0),
 ('WA', 750.0),
 ('CA', 200.0)]
```

In [170]:
```
# creating an adhoc list of managers for each state
data_array = [['CA', 'matthew'],
              ['OR', 'jorge'],
              ['WA','matthew'],
              ['TX', 'emilie']]

# reading the array/list using SparkContext
rdd_managers = sc.parallelize(data_array)

# to output the content in python [irl, use with great care]
rdd_salesperstate.join(rdd_managers).collect()
```

Out[170]:
```
[('CA', (200.0, 'matthew')),
 ('CA', (330.0, 'matthew')),
 ('CA', (200.0, 'matthew')),
 ('OR', (450.0, 'jorge')),
 ('WA', (300.0, 'matthew')),
 ('WA', (750.0, 'matthew'))]
```

`.reduceByKey(func)`: reduce `v`s by their `k` by applying func (what ?)

The `func` here needs to be associative and commutative... can you guess why ?

In [171]:
```
# creating an adhoc list
data_array = [['CA', 1],
              ['WA', 1],
              ['CA', 2],
              ['OR', 1],
              ['CA', 5],
              ['OR', 1]]

# reading the array/list using SparkContext
rdd = sc.parallelize(data_array)

# to output the content in python [irl, use with great care]
rdd.collect()
```

Out[171]:
```
[['CA', 1], ['WA', 1], ['CA', 2], ['OR', 1], ['CA', 5], ['OR', 1]]
```

In [172]:
```
rdd.reduceByKey(lambda v1,v2 : v1+v2).collect()
```

Out[172]:
```
[('CA', 8), ('WA', 1), ('OR', 2)]
```

`.groupByKey(func)`: reduce `v`s by their `k` by applying func (again ?)

This can use any function non-commutative

In [173]:
```
# creating an adhoc list
data_array = [['CA', 1],
              ['WA', 1],
              ['CA', 2],
              ['OR', 1],
              ['CA', 5],
              ['OR', 1]]

# reading the array/list using SparkContext
rdd = sc.parallelize(data_array)

# to output the content in python [irl, use with great care]
rdd.collect()
```

Out[173]:
```
[['CA', 1], ['WA', 1], ['CA', 2], ['OR', 1], ['CA', 5], ['OR', 1]]
```

In [174]:
```
def mean(iterator):
    total = 0.0; count = 0
    for x in iterator:
        total += x; count += 1
    return total / count

rdd.groupByKey()\
    .map(lambda x: (x[0], mean(x[1])))\
    .collect()

# rdd2 = rdd.groupByKey().map(lambda x: (x[0], mean(x[1])))
```

Out[174]:
```
[('CA', 2.6666666666666665), ('WA', 1.0), ('OR', 1.0)]
```

2.3.5. Sorting methods

`.sortBy(keyfunc)`: sorting by the value of a function on rows

In [175]:
```
# sorting by any function (because why not?)
rddout = rdd_names.sortBy(lambda x : (13-x[1])**2, ascending=True)

# print out the original rdd
print(rdd_names.collect())

# print out the new rdd generated
print(rddout.collect())
```

```
[['matthew', 4], ['jorge', 8], ['josh', 15], ['evangeline', 16], ['e
milie', 23], ['yunjin', 42]]
[['josh', 15], ['evangeline', 16], ['jorge', 8], ['matthew', 4], ['e
milie', 23], ['yunjin', 42]]
```

`.sortByKey()`: sorting by key on a `<k,v>` RDD

In [176]:
```
# sorting k,v pairs by key
rddout = rdd_names.sortByKey(ascending=False)

# print out the original rdd
print(rdd_names.collect())

# print out the new rdd generated
print(rddout.collect())
```

```
[['matthew', 4], ['jorge', 8], ['josh', 15], ['evangeline', 16], ['e
milie', 23], ['yunjin', 42]]
[('yunjin', 42), ('matthew', 4), ('josh', 15), ('jorge', 8), ('evang
eline', 16), ('emilie', 23)]
```

2.4. Actions : turning your RDD into something else (local object)

Actions are specific methods of an RDD object, they are usually designed to transform an RDD into something else (a python object, or a statistic).

When used/executed in IPython or in a notebook, they **launch the processing of the DAG**. This is where Spark stops being **lazy**. This is where your script will take time to execute.

Method	Type	Description
[`.collect()` (http://spark.apache.org/docs/latest/api/python/pyspark.html#pyspark.RDD.collect)](http://spark.apache.org/docs/latest/api/python/pyspark.html#pyspark.RDD.collect)	action	Return a list that contains all of the elements in this RDD. Note that this method should only be used if the resulting array is expected to be small, as all the data is loaded into the driver's memory.
`.count()`	action	Return the number of

(http://spark.apache.org/docs/latest/api/python/pyspark.html#pyspark.RDD.count)		elements in this RDD.
`.take(n)` (http://spark.apache.org/docs/latest/api/python/pyspark.html#pyspark.RDD.take)	action	Take the first `n` elements of the RDD.
`.top(n)` (http://spark.apache.org/docs/latest/api/python/pyspark.html#pyspark.RDD.top)	action	Get the top `n` elements from a RDD. It returns the list sorted in descending order.
`.first()` (http://spark.apache.org/docs/latest/api/python/pyspark.html#pyspark.RDD.first)	action	Return the first element in a RDD.
`.sum()` (http://spark.apache.org/docs/latest/api/python/pyspark.html#pyspark.RDD.sum)	action	Add up the elements in this RDD.
`.mean()` (http://spark.apache.org/docs/latest/api/python/pyspark.html#pyspark.RDD.mean)	action	Compute the mean of this RDD's elements.
`.stdev()` (http://spark.apache.org/docs/latest/api/python/pyspark.html#pyspark.RDD.stdev)	action	Compute the standard deviation of this RDD's elements.

In [177]:

```
# creating an adhoc list
data_array = [['matthew', 4],
              ['jorge', 8],
              ['josh', 15],
              ['evangeline', 16],
              ['emilie', 23],
              ['yunjin', 42]]

# reading the array/list using SparkContext
rdd_names = sc.parallelize(data_array)
```

2.4.1. Actions that return portions of an RDD

`.collect()` : returning the *full* content of an RDD to "python space"

Returns the rows of an RDD as a list. Can be a bad idea if your RDD is gigantic, cause `.collect()` will return everything and put it in memory for python to process.

In [178]:
```
# to output the content in python
collected = rdd_names.collect()

# let's check the type of RDD
print("type of rdd: {}".format(type(rdd_names)))

# let's check the type of what's collected
print("type of rdd_collected: {}".format(type(collected)))

# let's print the collected content
print(collected)
```

```
type of rdd: <class 'pyspark.rdd.RDD'>
type of rdd_collected: <class 'list'>
[['matthew', 4], ['jorge', 8], ['josh', 15], ['evangeline', 16], ['e
milie', 23], ['yunjin', 42]]
```

`.take(n)` : returning (any) n lines of an RDD

Returns `n` the rows of an RDD as a list. These `n` are not randomly selected. They are Spark's own internal mechanism for obtaining the lines that can be collected first.

In [179]:
```
# to output the content in python
taken = rdd_names.take(2)

# let's check the type of what's collected
print("type of rdd_taken: {}".format(type(taken)))

# let's print the collected content
print(taken)
```

```
type of rdd_taken: <class 'list'>
[['matthew', 4], ['jorge', 8]]
```

`.first()` : returning the first line of an RDD

In [180]:
```
print(rdd_names.first())
```

```
['matthew', 4]
```

2.4.2. Actions that compute some statistics

`.count()` : count the number of lines

In [181]:
```
print(rdd_names.count())
```

```
6
```

`.sum()`: summing every line in an RDD

(The RDD needs to be containing summable values)

In [182]:
```
print(rdd_names.values().sum())
```

```
108
```

`.mean()`: averaging every line in an RDD

(The RDD needs to be containing summable values)

In [183]:
```
print(rdd_names.values().mean())
```

```
18.0
```

`.stdev()`: you get that right ?

In [184]:
```
print(rdd_names.values().stdev())
```

```
12.3153021346
```

3. Let's design chains of transformations together !

3.1. Computing sales per state

Input RDD

In [185]:
```
def casting_function(row):
    _id, date, store, state, product, amount = row
    return (int(_id), date, int(store), state, int(product), float(amo
unt))


rdd_sales = sc.textFile('data/sales.txt')\
        .map(lambda x : x.split())\
        .filter(lambda x: not x[0].startswith('#'))\
        .map(casting_function)

rdd_sales.collect()
```

Out[185]:
```
[(101, '11/13/2014', 100, 'WA', 331, 300.0),
 (104, '11/18/2014', 700, 'OR', 329, 450.0),
 (102, '11/15/2014', 203, 'CA', 321, 200.0),
 (106, '11/19/2014', 202, 'CA', 331, 330.0),
 (103, '11/17/2014', 101, 'WA', 373, 750.0),
 (105, '11/19/2014', 202, 'CA', 321, 200.0)]
```

Task

You want to obtain a sorted RDD of the states in which you have most sales done (amount).

What transformations do you need to apply ? If you had to draw a workflow of the transformations to apply ?

Code

In [186]:
```
rddout = rdd_sales # apply transformation here...

rddout.collect()
```

Out[186]:
```
[(101, '11/13/2014', 100, 'WA', 331, 300.0),
 (104, '11/18/2014', 700, 'OR', 329, 450.0),
 (102, '11/15/2014', 203, 'CA', 321, 200.0),
 (106, '11/19/2014', 202, 'CA', 331, 330.0),
 (103, '11/17/2014', 101, 'WA', 373, 750.0),
 (105, '11/19/2014', 202, 'CA', 321, 200.0)]
```

Solution (use your mouse to uncover)

```
rddout = rdd_sales.map(lambda x: (x[3],x[5]))\
.reduceByKey(lambda amount1,amount2: amount1+amount2)\
.sortBy(lambda state_amount:state_amount[1],ascending=False)

rddout.collect()
```

In [187]:
```
# revealed solution here...
```

3.2. Word count (again)

Input RDD

In [188]:
```
# displaying the content of the file in stdout
with open('data/input.txt', 'r') as fin:
    print(fin.read())

# reading the file using SparkContext

rdd = sc.textFile('data/input.txt')
```

```
hello world
another line
yet another line
yet another another line
```

Task

What transformations do you need to apply ? If you had to draw a workflow of the transformations to apply ?

Code

In [189]:
```
rddout = rdd # apply transformation here...

# collect the result
rddout.collect()
```

Out[189]:
```
['hello world', 'another line', 'yet another line', 'yet another ano
ther line']
```

Solution (use your mouse to uncover)

```
rddout = rdd.flatMap(lambda str : str.split())\
.map(lambda word: (word,1))\
.reduceByKey(lambda v1,v2: v1+v2)

rddout.collect()
```

In [190]:
```
# revealed solution here...
```

3.3. Find the date on which AAPL's stock price was the highest

Input RDD

In [191]:
```
rdd_aapl_raw = sc.textFile('data/aapl.csv')

print("lines in file: {}".format(rdd_aapl_raw.count()))

rdd_aapl_raw.take(5)
```

```
lines in file: 254
```

Out[191]:
```
['Date,Open,High,Low,Close,Volume,Adj Close',
 '2016-10-25,117.949997,118.360001,117.309998,118.25,39190300,118.25
',
 '2016-10-24,117.099998,117.739998,117.00,117.650002,23538700,117.65
0002',
 '2016-10-21,116.809998,116.910004,116.279999,116.599998,23192700,11
6.599998',
 '2016-10-20,116.860001,117.379997,116.330002,117.059998,24125800,11
7.059998']
```

Task

Now, design a pipeline that would :

1. filter out headers
2. split each line based on comma
3. keep only fields for Date (col 0) and Close (col 4)
4. order by Close in descending order

Code

In [192]:
```
rddout = rdd_aapl_raw # apply transformation here...

rddout.take(5)
```

Out[192]:
```
['Date,Open,High,Low,Close,Volume,Adj Close',
 '2016-10-25,117.949997,118.360001,117.309998,118.25,39190300,118.25
',
 '2016-10-24,117.099998,117.739998,117.00,117.650002,23538700,117.65
0002',
 '2016-10-21,116.809998,116.910004,116.279999,116.599998,23192700,11
6.599998',
 '2016-10-20,116.860001,117.379997,116.330002,117.059998,24125800,11
7.059998']
```

Solution

```
rddout = rdd_aapl_raw.filter(lambda line: not line.startswith("Date"))\
.map(lambda line: line.split(","))\
.map(lambda fields: (float(fields[4]),fields[0]))\
.sortBy(lambda (close, date): close, ascending=False)
rddout.collect()
```

In [193]:
```
# revealed solution here
```

4. Caching / Persistency

- The RDD does no work until an action is called. And then when an action is called it figures out the answer and then throws away all the data.
- If you have an RDD that you are going to reuse in your computation you can use cache() to make Spark cache the RDD.
- This is especially useful if you have to run the same computation over and over again on one RDD: one use case ? oh I don't know maybe... **MACHINE LEARNING !!!**

4.1. Caching

Consider the following job...

In [194]:
```
import random
num_count = 500*1000
num_list = [random.random() for i in range(num_count)]
rdd1 = sc.parallelize(num_list)
rdd2 = rdd1.sortBy(lambda num: num)
```

In [195]:
```
%time rdd2.count()
%time rdd2.count()
%time rdd2.count()
```

```
CPU times: user 4 ms, sys: 4 ms, total: 8 ms
Wall time: 603 ms
CPU times: user 12 ms, sys: 0 ns, total: 12 ms
Wall time: 406 ms
CPU times: user 4 ms, sys: 4 ms, total: 8 ms
Wall time: 388 ms
```

Out[195]: 500000

In [196]:
```
rdd2.cache()
%time rdd2.count()
%time rdd2.count()
%time rdd2.count()
```

```
CPU times: user 12 ms, sys: 4 ms, total: 16 ms
Wall time: 384 ms
CPU times: user 4 ms, sys: 0 ns, total: 4 ms
Wall time: 43.7 ms
CPU times: user 4 ms, sys: 0 ns, total: 4 ms
Wall time: 68.7 ms
```

Out[196]:
```
500000
```

- Caching the RDD speeds up the job because the RDD does not have to be computed from scratch again.
- Calling cache() flips a flag on the RDD.
- The data is not cached until an action is called.
- You can uncache an RDD using unpersist()

4.2. Persist

- Persist RDD to disk instead of caching it in memory.
- You can cache RDDs at different levels.

Level	Meaning
MEMORY_ONLY	Same as cache()
MEMORY_AND_DISK	Cache in memory then overflow to disk
MEMORY_AND_DISK_SER	Like above; in cache keep objects serialized instead of live
DISK_ONLY	Cache to disk not to memory

In [ ]:
```
```

In [ ]:
```
```

In [ ]:
```
```

The following parts of this doc were created for use with Spark 2.1.0 in June 2017

Spark 2.1 Environment Setup

In [197]:

```
# Get pyspark, spark
import findspark
findspark.init('/home/sparkles/spark-2.1.0-bin-hadoop2.7') # your spar
k dir
import pyspark
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName('Lecture').getOrCreate()  # calle
d 'spark' by convention
```

Get Some Data from the Internet

In [198]:
```
# Grab some data from the web, save it to disk as csv
# Air Traffic Passenger Statistics for San Francisco:
# Data as CSV: https://data.sfgov.org/api/views/rkru-6vcg/rows.csv
# Source: https://catalog.data.gov/dataset/air-traffic-passenger-stati
stics
import requests
import contextlib
import csv

csv_path = 'https://data.sfgov.org/api/views/rkru-6vcg/rows.csv'
response = requests.get(csv_path)
with open('rawdata.csv', 'w') as f:
    f.write(response.text)

# Let's look at the beginning of the data:
response.text[:1000]
```

Out[198]:
```
'Activity Period,Operating Airline,Operating Airline IATA Code,Publi
shed Airline,Published Airline IATA Code,GEO Summary,GEO Region,Acti
vity Type Code,Price Category Code,Terminal,Boarding Area,Passenger 
Count\n,,,,,,,,,,,\n,,,,,,,,,,,\n,Operating Airline,Operating Airlin
e IATA Code,Published Airline,Published Airline IATA Code,GEO Summar
y,GEO Region,Activity Type Code,Price Category Code,Terminal,Boardin
g Area,\n200507,ATA Airlines,TZ,ATA Airlines,TZ,Domestic,US,Deplaned
,Low Fare,Terminal 1,B,27271\n200507,ATA Airlines,TZ,ATA Airlines,TZ
,Domestic,US,Enplaned,Low Fare,Terminal 1,B,29131\n200507,ATA Airlin
es,TZ,ATA Airlines,TZ,Domestic,US,Thru / Transit,Low Fare,Terminal 1
,B,5415\n200507,Air Canada,AC,Air Canada,AC,International,Canada,Dep
laned,Other,Terminal 1,B,35156\n200507,Air Canada,AC,Air Canada,AC,I
nternational,Canada,Enplaned,Other,Terminal 1,B,34090\n200507,Air Ch
ina,CA,Air China,CA,International,Asia,Deplaned,Other,International,
G,6263\n200507,Air China,CA,Air China,CA,International,Asia'
```

Internet Bleach

In [199]:
```
import pandas as pd
pandas_df = pd.read_csv("rawdata.csv", header=0, skiprows=[1,2,3])
# pandas_df = pd.read_csv("rawdata.csv")
pandas_df
# pandas_df.to_csv('data.csv')
# pandas_df = pd.read_csv('data.csv')
# pandas_df
```

Out[199]:

	Activity	Operating	Operating Airline	Published	Published Airline	GEO	GEO	A

	Period	Airline	IATA Code	Airline	IATA Code	Summary	Region	
0	200507.0	ATA Airlines	TZ	ATA Airlines	TZ	Domestic	US	De
1	200507.0	ATA Airlines	TZ	ATA Airlines	TZ	Domestic	US	En
2	200507.0	ATA Airlines	TZ	ATA Airlines	TZ	Domestic	US	Th Tra
3	200507.0	Air Canada	AC	Air Canada	AC	International	Canada	De
4	200507.0	Air Canada	AC	Air Canada	AC	International	Canada	En
5	200507.0	Air China	CA	Air China	CA	International	Asia	De
6	200507.0	Air China	CA	Air China	CA	International	Asia	En
7	200507.0	Air France	AF	Air France	AF	International	Europe	De
8	200507.0	Air France	AF	Air France	AF	International	Europe	En
9	200507.0	Air New Zealand	NZ	Air New Zealand	NZ	International	Australia / Oceania	De
10	200507.0	Air New Zealand	NZ	Air New Zealand	NZ	International	Australia / Oceania	En
11	200507.0	AirTran Airways	FL	AirTran Airways	FL	Domestic	US	De
12	200507.0	AirTran Airways	FL	AirTran Airways	FL	Domestic	US	En
13	200507.0	Alaska Airlines	AS	Alaska Airlines	AS	Domestic	US	De
14	200507.0	Alaska Airlines	AS	Alaska Airlines	AS	Domestic	US	En
15	200507.0	Alaska Airlines	AS	Alaska Airlines	AS	Domestic	US	Th Tra
16	200507.0	Alaska Airlines	AS	Alaska Airlines	AS	International	Canada	De

17	200507.0	Alaska Airlines	AS	Alaska Airlines	AS	International	Canada	En
18	200507.0	Alaska Airlines	AS	Alaska Airlines	AS	International	Mexico	De
19	200507.0	Alaska Airlines	AS	Alaska Airlines	AS	International	Mexico	En
20	200507.0	Alaska Airlines	AS	Alaska Airlines	AS	International	Mexico	Th Tra
21	200507.0	All Nippon Airways	NH	All Nippon Airways	NH	International	Asia	De
22	200507.0	All Nippon Airways	NH	All Nippon Airways	NH	International	Asia	En
23	200507.0	American Airlines	AA	American Airlines	AA	Domestic	US	De
24	200507.0	American Airlines	AA	American Airlines	AA	Domestic	US	En
25	200507.0	American Eagle Airlines	MQ	American Airlines	AA	Domestic	US	De
26	200507.0	American Eagle Airlines	MQ	American Airlines	AA	Domestic	US	En
27	200507.0	Asiana Airlines	OZ	Asiana Airlines	OZ	International	Asia	De
28	200507.0	Asiana Airlines	OZ	Asiana Airlines	OZ	International	Asia	En
29	200507.0	Atlantic Southeast Airlines	EV	Delta Air Lines	DL	Domestic	US	De
...	...	...	...	...	...	...	...	...
16189	201612.0	United Airlines	UA	United Airlines	UA	Domestic	US	Th Tra
16190	201612.0	United Airlines	UA	United Airlines	UA	International	Asia	De
16191	201612.0	United	UA	United	UA	International	Asia	En

		Airlines		Airlines				
16192	201612.0	United Airlines	UA	United Airlines	UA	International	Asia	En
16193	201612.0	United Airlines	UA	United Airlines	UA	International	Asia	En
16194	201612.0	United Airlines	UA	United Airlines	UA	International	Australia / Oceania	De
16195	201612.0	United Airlines	UA	United Airlines	UA	International	Australia / Oceania	En
16196	201612.0	United Airlines	UA	United Airlines	UA	International	Canada	En
16197	201612.0	United Airlines	UA	United Airlines	UA	International	Canada	De
16198	201612.0	United Airlines	UA	United Airlines	UA	International	Canada	En
16199	201612.0	United Airlines	UA	United Airlines	UA	International	Canada	De
16200	201612.0	United Airlines	UA	United Airlines	UA	International	Canada	En
16201	201612.0	United Airlines	UA	United Airlines	UA	International	Europe	De
16202	201612.0	United Airlines	UA	United Airlines	UA	International	Europe	En
16203	201612.0	United Airlines	UA	United Airlines	UA	International	Mexico	De
16204	201612.0	United Airlines	UA	United Airlines	UA	International	Mexico	En
16205	201612.0	United Airlines	UA	United Airlines	UA	International	Mexico	En
16206	201612.0	United Airlines	UA	United Airlines	UA	International	Mexico	En
16207	201612.0	Virgin America	VX	Virgin America	VX	Domestic	US	De

16208	201612.0	Virgin America	VX	Virgin America	VX	Domestic	US	En
16209	201612.0	Virgin America	VX	Virgin America	VX	International	Mexico	De
16210	201612.0	Virgin America	VX	Virgin America	VX	International	Mexico	En
16211	201612.0	Virgin Atlantic	VS	Virgin Atlantic	VS	International	Europe	De
16212	201612.0	Virgin Atlantic	VS	Virgin Atlantic	VS	International	Europe	En
16213	201612.0	Volaris Airlines	Y4	Volaris Airlines	Y4	International	Mexico	De
16214	201612.0	Volaris Airlines	Y4	Volaris Airlines	Y4	International	Mexico	En
16215	201612.0	WOW Air	WW	WOW Air	WW	International	Europe	En
16216	201612.0	WOW Air	WW	WOW Air	WW	International	Europe	De
16217	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
16218	NaN	NaN	NaN	NaN	- 1 -	NaN	NaN	Na

16219 rows × 12 columns

Load Data into Spark

In [200]:

```
df = spark.read.format("csv").option("header", True).load("data.csv")
# rdd = sc.textFile("/FileStore/tables/2esy8tnj1455052720017/")
# diamonds = sqlContext.read.format('csv').options(header='true', infe
rSchema='true').load('/databricks-datasets/Rdatasets/data-001/csv/ggpl
ot2/diamonds.csv')
df.show()
```

```
+---+---------------+-----------------+---------------------------+-
----------------+---------------------------+-------------+---------
----------+-------------------+-------------------+-------------+----
---------+---------------+
|_c0|Activity Period|Operating Airline|Operating Airline IATA Code|P
ublished Airline|Published Airline IATA Code|  GEO Summary|
GEO Region|Activity Type Code|Price Category Code|     Terminal|Boar
```

```
ding Area|Passenger Count|
+---+---------------+------------------+---------------------------+-
----------------+----------------------------+-------------+---------
----------+-------------------+-------------------+-------------+----
---------+---------------+
|  0|       200507.0|     ATA Airlines|                         TZ|
ATA Airlines|                         TZ|     Domestic|
US|           Deplaned|           Low Fare|   Terminal 1|
B|        27271.0|
|  1|       200507.0|     ATA Airlines|                         TZ|
ATA Airlines|                         TZ|     Domestic|
US|           Enplaned|           Low Fare|   Terminal 1|
B|        29131.0|
|  2|       200507.0|     ATA Airlines|                         TZ|
ATA Airlines|                         TZ|     Domestic|
US|     Thru / Transit|           Low Fare|   Terminal 1|
B|         5415.0|
|  3|       200507.0|       Air Canada|                         AC|
Air Canada|                         AC|International|           Ca
nada|           Deplaned|              Other|   Terminal 1|
B|        35156.0|
|  4|       200507.0|       Air Canada|                         AC|
Air Canada|                         AC|International|           Ca
nada|           Enplaned|              Other|   Terminal 1|
B|        34090.0|
|  5|       200507.0|        Air China|                         CA|
Air China|                         CA|International|             A
sia|           Deplaned|              Other|International|
G|         6263.0|
|  6|       200507.0|        Air China|                         CA|
Air China|                         CA|International|             A
sia|           Enplaned|              Other|International|
G|         5500.0|
|  7|       200507.0|       Air France|                         AF|
Air France|                         AF|International|           Eu
rope|           Deplaned|              Other|International|
A|        12050.0|
|  8|       200507.0|       Air France|                         AF|
Air France|                         AF|International|           Eu
rope|           Enplaned|              Other|International|
A|        11638.0|
|  9|       200507.0|  Air New Zealand|                         NZ|
Air New Zealand|                         NZ|International|Australia
/ Oceania|           Deplaned|              Other|International|
G|         4998.0|
| 10|       200507.0|  Air New Zealand|                         NZ|
Air New Zealand|                         NZ|International|Australia
/ Oceania|           Enplaned|              Other|International|
G|         4962.0|
| 11|       200507.0|  AirTran Airways|                         FL|
```

```
AirTran Airways|                         FL|     Domestic|
US|          Deplaned|           Low Fare|International|
A|         8055.0|
| 12|       200507.0|  AirTran Airways|                         FL|
AirTran Airways|                         FL|     Domestic|
US|          Enplaned|           Low Fare|International|
A|         7984.0|
| 13|       200507.0|  Alaska Airlines|                         AS|
Alaska Airlines|                         AS|     Domestic|
US|          Deplaned|              Other|International|
A|        36641.0|
| 14|       200507.0|  Alaska Airlines|                         AS|
Alaska Airlines|                         AS|     Domestic|
US|          Enplaned|              Other|International|
A|        39379.0|
| 15|       200507.0|  Alaska Airlines|                         AS|
Alaska Airlines|                         AS|     Domestic|
US|    Thru / Transit|              Other|International|
A|         3678.0|
| 16|       200507.0|  Alaska Airlines|                         AS|
Alaska Airlines|                         AS|International|
Canada|          Deplaned|              Other|International|
A|         7977.0|
| 17|       200507.0|  Alaska Airlines|                         AS|
Alaska Airlines|                         AS|International|
Canada|          Enplaned|              Other|International|
A|         8837.0|
| 18|       200507.0|  Alaska Airlines|                         AS|
Alaska Airlines|                         AS|International|
Mexico|          Deplaned|              Other|International|
A|         6969.0|
| 19|       200507.0|  Alaska Airlines|                         AS|
Alaska Airlines|                         AS|International|
Mexico|          Enplaned|              Other|International|
A|        10046.0|
+---+---------------+-----------------+--------------------------+-
----------------+--------------------------+-------------+---------
----------+------------------+------------------+-------------+----
---------+---------------+
only showing top 20 rows
```

DF Basics

In [201]:
```
df.printSchema()
```

```
root
 |-- _c0: string (nullable = true)
 |-- Activity Period: string (nullable = true)
 |-- Operating Airline: string (nullable = true)
 |-- Operating Airline IATA Code: string (nullable = true)
 |-- Published Airline: string (nullable = true)
 |-- Published Airline IATA Code: string (nullable = true)
 |-- GEO Summary: string (nullable = true)
 |-- GEO Region: string (nullable = true)
 |-- Activity Type Code: string (nullable = true)
 |-- Price Category Code: string (nullable = true)
 |-- Terminal: string (nullable = true)
 |-- Boarding Area: string (nullable = true)
 |-- Passenger Count: string (nullable = true)
```

In [202]:
```
df.describe()
```

Out[202]:
```
DataFrame[summary: string, _c0: string, Activity Period: string, Ope
rating Airline: string, Operating Airline IATA Code: string, Publish
ed Airline: string, Published Airline IATA Code: string, GEO Summary
: string, GEO Region: string, Activity Type Code: string, Price Cate
gory Code: string, Terminal: string, Boarding Area: string, Passenge
r Count: string]
```

In [203]:
```
df.describe().show()
```

```
+-------+-----------------+------------------+-----------------+----
-----------------------+-----------------+-------------------------
-+-------------+----------+------------------+-------------------+--
-----------+-------------+-----------------+
|summary|              _c0|   Activity Period|Operating Airline|Oper
ating Airline IATA Code|Published Airline|Published Airline IATA Cod
e|  GEO Summary|GEO Region|Activity Type Code|Price Category Code|
Terminal|Boarding Area|  Passenger Count|
+-------+-----------------+------------------+-----------------+----
-----------------------+-----------------+-------------------------
-+-------------+----------+------------------+-------------------+--
-----------+-------------+-----------------+
|  count|            16219|             16217|            16217|
16159|            16217|                     16160|         16217|
16217|            16217|              16218|          16217|         1
6217|            16218|
|   mean|           8109.0| 201087.0767712894|             null|
null|             null|                      null|          null|
null|             null|               null|           null|          n
ull|59252.92082870884|
| stddev|4682.166343335814|335.76330796754615|             null|
null|             null|                      null|          null|
null|             null|               null|           null|          n
ull|3773158.085088906|
|    min|                0|          200507.0|     ATA Airlines|
4T|     ATA Airlines|                    - 1 - |      Domestic|
Asia|          Deplaned|         1:32:36 PM|International|
A|              1.0|
|    max|             9999|          201612.0|     Xtra Airways|
YX|     Xtra Airways|                        YX|International|
US|    Thru / Transit|              Other|   Terminal 3|        Othe
r|           9999.0|
+-------+-----------------+------------------+-----------------+----
-----------------------+-----------------+-------------------------
-+-------------+----------+------------------+-------------------+--
-----------+-------------+-----------------+
```

That wasn't super useful, yet, was it? Next up, properly setting up our schemas and working with the super-awesome DataFrame syntax!

In [ ]: