Python For Data Science *Cheat Sheet*

PySpark - RDD Basics

Learn Python for data science Interactively at www.DataCamp.com

Spark

PySpark is the Spark Python API that exposes the Spark programming model to Python.

Initializing Spark

SparkContext

```
>>> from pyspark import SparkContext
>>> sc = SparkContext(master = 'local[2]')
```

Inspect SparkContext

`>>> sc.version`	Retrieve SparkContext version
`>>> sc.pythonVer`	Retrieve Python version
`>>> sc.master`	Master URL to connect to
`>>> str(sc.sparkHome)`	Path where Spark is installed on worker nodes
`>>> str(sc.sparkUser())`	Retrieve name of the Spark User running SparkContext
`>>> sc.appName`	Return application name
`>>> sc.applicationId`	Retrieve application ID
`>>> sc.defaultParallelism`	Return default level of parallelism
`>>> sc.defaultMinPartitions`	Default minimum number of partitions for RDDs

Configuration

```
>>> from pyspark import SparkConf, SparkContext
>>> conf = (SparkConf()
             .setMaster("local")
             .setAppName("My app")
             .set("spark.executor.memory", "1g"))
>>> sc = SparkContext(conf = conf)
```

Using The Shell

In the PySpark shell, a special interpreter-aware SparkContext is already created in the variable called `sc`.

```
$ ./bin/spark-shell --master local[2]
$ ./bin/pyspark --master local[4] --py-files code.py
```

Set which master the context connects to with the `--master` argument, and add Python .zip, .egg or .py files to the runtime path by passing a comma-separated list to `--py-files`.

Loading Data

Parallelized Collections

```
>>> rdd = sc.parallelize([('a',7),('a',2),('b',2)])
>>> rdd2 = sc.parallelize([('a',2),('d',1),('b',1)])
>>> rdd3 = sc.parallelize(range(100))
>>> rdd4 = sc.parallelize([("a",["x","y","z"]),
                           ("b",["p", "r"])])
```

External Data

Read either one text file from HDFS, a local file system or or any Hadoop-supported file system URI with `textFile()`, or read in a directory of text files with `wholeTextFiles()`.

```
>>> textFile = sc.textFile("/my/directory/*.txt")
>>> textFile2 = sc.wholeTextFiles("/my/directory/")
```

Retrieving RDD Information

Basic Information

`>>> rdd.getNumPartitions()`	List the number of partitions

`>>> rdd.count()`
`3`
	Count RDD instances

`>>> rdd.countByKey()`
`defaultdict(<type 'int'>,{'a':2,'b':1})`
	Count RDD instances by key

`>>> rdd.countByValue()`
`defaultdict(<type 'int'>,{('b',2):1,('a',2):1,('a',7):1})`
	Count RDD instances by value

`>>> rdd.collectAsMap()`
`{'a': 2,'b': 2}`
	Return (key,value) pairs as a dictionary

`>>> rdd3.sum()`
`4950`
	Sum of RDD elements

`>>> sc.parallelize([]).isEmpty()`
`True`
	Check whether RDD is empty

Summary

`>>> rdd3.max()`
`99`
	Maximum value of RDD elements

`>>> rdd3.min()`
`0`
	Minimum value of RDD elements

`>>> rdd3.mean()`
`49.5`
	Mean value of RDD elements

`>>> rdd3.stdev()`
`28.866070047722118`
	Standard deviation of RDD elements

`>>> rdd3.variance()`
`833.25`
	Compute variance of RDD elements

`>>> rdd3.histogram(3)`
`([0,33,66,99],[33,33,34])`
	Compute histogram by bins

`>>> rdd3.stats()`	Summary statistics (count, mean, stdev, max & min)

Applying Functions

`>>> rdd.map(lambda x: x+(x[1],x[0])).collect()`
`[('a',7,7,'a'),('a',2,2,'a'),('b',2,2,'b')]`
	Apply a function to each RDD element

`>>> rdd5 = rdd.flatMap(lambda x: x+(x[1],x[0]))`
`>>> rdd5.collect()`
`['a',7,7,'a','a',2,2,'a','b',2,2,'b']`
	Apply a function to each RDD element and flatten the result

`>>> rdd4.flatMapValues(lambda x: x).collect()`
`[('a','x'),('a','y'),('a','z'),('b','p'),('b','r')]`
	Apply a flatMap function to each (key,value) pair of `rdd4` without changing the keys

Selecting Data

Getting

`>>> rdd.collect()`
`[('a', 7), ('a', 2), ('b', 2)]`
	Return a list with all RDD elements

`>>> rdd.take(2)`
`[('a', 7), ('a', 2)]`
	Take first 2 RDD elements

`>>> rdd.first()`
`('a', 7)`
	Take first RDD element

`>>> rdd.top(2)`
`[('b', 2), ('a', 7)]`
	Take top 2 RDD elements

Sampling

`>>> rdd3.sample(False, 0.15, 81).collect()`
`[3,4,27,31,40,41,42,43,60,76,79,80,86,97]`
	Return sampled subset of `rdd3`

Filtering

`>>> rdd.filter(lambda x: "a" in x).collect()`
`[('a',7),('a',2)]`
	Filter the RDD

`>>> rdd5.distinct().collect()`
`['a',2,'b',7]`
	Return distinct RDD values

`>>> rdd.keys().collect()`
`['a', 'a', 'b']`
	Return (key,value) RDD's keys

Iterating

`>>> def g(x): print(x)`
`>>> rdd.foreach(g)`
`('a', 7)`
`('b', 2)`
`('a', 2)`
	Apply a function to all RDD elements

Reshaping Data

Reducing

`>>> rdd.reduceByKey(lambda x,y : x+y).collect()`
`[('a',9),('b',2)]`
	Merge the `rdd` values for each key

`>>> rdd.reduce(lambda a, b: a + b)`
`('a',7,'a',2,'b',2)`
	Merge the `rdd` values

Grouping by

`>>> rdd3.groupBy(lambda x: x % 2).mapValues(list).collect()`	Return RDD of grouped values

`>>> rdd.groupByKey().mapValues(list).collect()`
`[('a',[7,2]),('b',[2])]`
	Group `rdd` by key

Aggregating

`>>> seqOp = (lambda x,y: (x[0]+y,x[1]+1))`
`>>> combOp = (lambda x,y:(x[0]+y[0],x[1]+y[1]))`	

`>>> rdd3.aggregate((0,0),seqOp,combOp)`
`(4950,100)`
	Aggregate RDD elements of each partition and then the results

`>>> rdd.aggregateByKey((0,0),seqop,combop).collect()`
`[('a',(9,2)), ('b',(2,1))]`
	Aggregate values of each RDD key

`>>> rdd3.fold(0,add)`
`4950`
	Aggregate the elements of each partition, and then the results

`>>> rdd.foldByKey(0, add).collect()`
`[('a',9),('b',2)]`
	Merge the values for each key

`>>> rdd3.keyBy(lambda x: x+x).collect()`	Create tuples of RDD elements by applying a function

Mathematical Operations

`>>> rdd.subtract(rdd2).collect()`
`[('b',2),('a',7)]`
	Return each `rdd` value not contained in `rdd2`

`>>> rdd2.subtractByKey(rdd).collect()`
`[('d', 1)]`
	Return each (key,value) pair of `rdd2` with no matching key in `rdd`

`>>> rdd.cartesian(rdd2).collect()`	Return the Cartesian product of `rdd` and `rdd2`

Sort

`>>> rdd2.sortBy(lambda x: x[1]).collect()`
`[('d',1),('b',1),('a',2)]`
	Sort RDD by given function

`>>> rdd2.sortByKey().collect()`
`[('a',2),('b',1),('d',1)]`
	Sort (key, value) RDD by key

Repartitioning

`>>> rdd.repartition(4)`	New RDD with 4 partitions
`>>> rdd.coalesce(1)`	Decrease the number of partitions in the RDD to 1

Saving

```
>>> rdd.saveAsTextFile("rdd.txt")
>>> rdd.saveAsHadoopFile("hdfs://namenodehost/parent/child",
                         'org.apache.hadoop.mapred.TextOutputFormat')
```

Stopping SparkContext

```
>>> sc.stop()
```

Execution

```
$ ./bin/spark-submit examples/src/main/python/pi.py
```