

Python For Data Science Cheat Sheet

PySpark Basics



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 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 `sc.textFile()`, or read in a directory of text files with `sc.wholeTextFiles()`

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

Retrieving RDD Information

Basic Information

<pre>>>> rdd.getNumPartitions() >>> rdd.count() 3 >>> rdd.countByKey() defaultdict(<type 'int'>, {'a':2, 'b':1}) >>> rdd.countByValue() defaultdict(<type 'int'>, {'b':2}:1, {'a':2}:1, {'a':1}:1) >>> rdd.collectAsMap() {'a': 2, 'b': 2} >>> rdd3.sum() 4950 >>> sc.parallelize([]).isEmpty() True</pre>	List the number of partitions Count RDD instances Count RDD instances by key Count RDD instances by value Return (key,value) pairs as a dictionary Sum of RDD elements Check whether RDD is empty
---	--

Summary

<pre>>>> rdd3.max() Maximum value of RDD elements 99 >>> rdd3.min() Minimum value of RDD elements 0 >>> rdd3.mean() Mean value of RDD elements 49.5 >>> rdd3.stdev() Standard deviation of RDD elements 28.866070047722118 >>> rdd3.variance() Compute variance of RDD elements 833.25 >>> rdd3.histogram(3) Compute histogram by bins ([0,33,66,99], [33,33,34]) >>> rdd3.stats() Summary statistics (count, mean, stdev, max & min)</pre>	
--	--

Applying Functions

<pre>>>> rdd.map(lambda x: x+(x[1],x[0])) .collect() [('a',7,7), ('a',2,2), ('b',2,2), ('b',2,2)] >>> rdd5 = rdd.flatMap(lambda x: x+(x[1],x[0])) >>> rdd5.collect() [('a',7,7,'a','a',2,2,'a','b',2,2,'b')] >>> rdd4.flatMapValues(lambda x: x) .collect() [('a','x'), ('a','y'), ('a','z'), ('b','p'), ('b','r')]</pre>	Apply a function to each RDD element Apply a function to each RDD element and flatten the result Apply a flatMap function to each (key,value) pair of RDD without changing the keys
---	--

Selecting Data

<pre>>>> rdd.collect() [('a', 7), ('a', 2), ('b', 2)] >>> rdd.take(2) [('a', 7), ('a', 2)] >>> rdd.first() ('a', 7) >>> rdd.top(2) [('b', 2), ('a', 7)] >>> rdd3.sample(False, 0.15, 81).collect() [3,4,27,31,40,41,42,43,60,76,79,80,86,97]</pre>	Return a list with all RDD elements Take first 2 RDD elements Take first RDD element Take top 2 RDD elements Return sampled subset of RDD
<pre>>>> rdd.filter(lambda x: "a" in x) .collect() [('a',7), ('a',2)] >>> rdd5.distinct().collect() [('a',2,'b',7)] >>> rdd.keys().collect() ['a', 'a', 'b']</pre>	Filter the RDD Return distinct RDD values Return (key,value) RDD's keys

Iterating

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

Reshaping Data

<pre>>>> rdd.reduceByKey(lambda x,y : x+y) .collect() [('a',9), ('b',2)] >>> rdd.reduce(lambda a, b: a + b) ('a',7,'a',2,'b',2) >>> rdd3.groupBy(lambda x: x % 2) .mapValues(list) .collect() >>> rdd.groupByKey() .mapValues(list) .collect() [('a',[7,2]), ('b',[2])]</pre>	Merge the <code>rdd</code> values for each key Merge the <code>rdd</code> values Return RDD of grouped values Group <code>rdd</code> by key
<pre>>>> 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) >>> rdd.aggregateByKey((0,0), seqOp, combOp) .collect() [('a',(9,2)), ('b',(2,1))]</pre>	Aggregate RDD elements of each partition and then the results Aggregate values of each RDD key
<pre>>>> rdd3.fold(0,add) 4950 >>> rdd.foldByKey(0, add) .collect() [('a',9), ('b',2)] >>> rdd3.keyBy(lambda x: x+x) .collect()</pre>	Aggregate the elements of each partition, and then the results Merge the values for each key Create tuples of RDD elements by applying a function

Mathematical Operations

<pre>>>> rdd.subtract(rdd2) .collect() [('b',2), ('a',7)] >>> rdd2.subtractByKey(rdd) .collect() [('d', 1)] >>> rdd.cartesian(rdd2).collect() rdd2</pre>	Return each value not contained in <code>rdd2</code> Return each (key,value) pair of with no matching key in <code>rdd2</code> Return the Cartesian product of <code>rdd</code> and <code>rdd2</code>
---	--

Sort

<pre>>>> rdd2.sortBy(lambda x: x[1]) .collect() [('d',1), ('b',1), ('a',2)] >>> rdd2.sortByKey() .collect() [('a',2), ('b',1), ('d',1)]</pre>	Sort RDD by given function Sort (key, value) RDD by key
---	---

Repartitioning

<pre>>>> rdd.repartition(4) >>> rdd.coalesce(1)</pre>	New RDD with 4 partitions 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
```

