**Problem Statement**

* Use retail\_db data set
* Problem Statement
  + Get daily revenue by product considering completed and closed orders.
  + Data need to be sorted in ascending order by date and then descending  
    order by revenue computed for each product for each day.
* Data for orders and order\_items is available in HDFS  
  /public/retail\_db/orders and /public/retail\_db/order\_items
* Data for products is available locally under /data/retail\_db/products
* Final output need to be stored under
  + HDFS location – avro format  
    /user/YOUR\_USER\_ID/daily\_revenue\_avro\_python
  + HDFS location – text format  
    /user/YOUR\_USER\_ID/daily\_revenue\_txt\_python
  + Local location /home/YOUR\_USER\_ID/daily\_revenue\_python
  + Solution need to be stored under  
    /home/YOUR\_USER\_ID/daily\_revenue\_python.txt

As part of this topic, we will look into the introduction of Spark

* Spark is Distributed computing framework
* Bunch of APIs to process data
* Higher level modules such as Data Frames/SQL, Streaming, MLLib and more
* Well integrated with Python, Scala, Java etc
* Spark uses HDFS API to deal with file system
* It can run on any distributed or cloud file systems – HDFS, s3, Azure Blob etc
* Only Core Spark and Spark SQL (including Data Frames) is part of the curriculum for CCA
* Spark and Hadoop Developer. CCA also requires some knowledge of Spark Streaming.
* Pre-requisites – Programming Language (Scala or Python)

As part of this topic, we will set up Spark in Windows Machine

* Go to the link [Spark Download 8](https://spark.apache.org/downloads.html)
* Choose the relevant version of Spark(this time it is 1.6.3)
* Choose the Hadoop version(this time it is 2.6)
* Click on the download link which will take to the downloads page
* Copy the link and paste it in the Cygwin
* wget [http://www-us.apache.org/dist/spark/spark-1.6.3/spark-1.6.3-bin-hadoop2.6.tgz 2](http://www-us.apache.org/dist/spark/spark-1.6.3/spark-1.6.3-bin-hadoop2.6.tgz)
* Copy the Spark folder from Cygwin to Local Directory
* Download winutils from [here 1](http://www.eaiesb.com/blogs/?tag=apache-spark-installation-on-windows) , save the link into your local machine by creating a hadoop/bin folder
* Set the path in the system variables
* Change the permissions if still the spark-shell is not launched
* Set the Spark path in system variables

As part of this topic, we will cover how to connect to the environment

* Once we set up the spark we can launch it with launch terminal in mac or ubuntu and command prompt in windows
* If you have Quickstart VM or Hortonworks Sandbox we can do ssh or we can directly launch console and use pyspark in it.

**Initializing the job**

* Initialize using pyspark
* Running in yarn mode (client or cluster mode)
* Control arguments
* Deciding on the number of executors
* Setting up additional properties
* Programmatic initialization of job
* Create configuration object
* Create spark context object

Pyspark –master yarn –conf spark.ui.port=54334

**Reading data using SparkContext**

Data can be read from files using textFile of Spark Context object sc. Following are the operations on top of RDD that can be performed to preview the data

* + take
  + first
  + count
  + and more

**Create RDD using data from HDFS**

* RDD is an extension to Python list
* RDD – Resilient Distributed Dataset
* In-memory
* Distributed
* Resilient
* Reading files from HDFS
* A quick overview of Transformations and Actions
* DAG and lazy evaluation
* Previewing the data using Actions

Help(sc)

With SC we can invoke an API using sc.

hadoopFile,textFile,addFile and sequenceFile API for reading data.

Press q to break

help(sc.textFile)

Help on method textFile in module pyspark.context:

textFile(name, minPartitions=None, use\_unicode=True) method of pyspark.context.SparkContext instance

Read a text file from HDFS, a local file system (available on all

nodes), or any Hadoop-supported file system URI, and return it as an

RDD of Strings.

If use\_unicode is False, the strings will be kept as `str` (encoding

as `utf-8`), which is faster and smaller than unicode. (Added in

Spark 1.2)

>>> path = os.path.join(tempdir, "sample-text.txt")

>>> with open(path, "w") as testFile:

... \_ = testFile.write("Hello world!")

>>> textFile = sc.textFile(path)

>>> textFile.collect()

['Hello world!']

RDD::

>>> type(orderItems)

<class 'pyspark.rdd.RDD'>

Path ::

>>> orderItems = sc.textFile("file:///F:/pySpark/data-master/data\_folder/retail\_db/order\_items")

>>> orderItems.first()

'1,1,957,1,299.98,299.98'

More than one rows::

for i in orderItems.ta

orderItems.take( orderItems.takeOrdered( orderItems.takeSample(

>>> for i in orderItems.take(10): pr

print( property(

>>> for i in orderItems.take(10): print(i);

...

1,1,957,1,299.98,299.98

2,2,1073,1,199.99,199.99

3,2,502,5,250.0,50.0

4,2,403,1,129.99,129.99

5,4,897,2,49.98,24.99

6,4,365,5,299.95,59.99

7,4,502,3,150.0,50.0

8,4,1014,4,199.92,49.98

9,5,957,1,299.98,299.98

10,5,365,5,299.95,59.99

[**Apache Spark**](http://data-flair.training/blogs/apache-spark-tutorial-quickstart-introduction/) being an open-source framework for [**Bigdata**](http://data-flair.training/blogs/why-learn-big-data-use-cases/) has a various advantage over other big data solutions like Apache Spark is Dynamic in Nature, it supports in-memory Computation of RDDs. It provides a provision of reusability, Fault Tolerance, real-time stream processing and many more. In this tutorial on features of Apache Spark, we will discuss various advantages of Spark which give us the answer for – Why we should learn Apache Spark? Why is Spark better than [**Hadoop**](http://data-flair.training/blogs/hadoop-introduction-tutorial-quick-guide/)MapReduce and why is Spark called 3G of Big data?

**Apache Spark** is lightning fast, in-memory data processing engine. Spark mainly designs for data science and the abstractions of Spark make it easier. Apache Spark provides high-level APIs in Java, [**Scala**](http://data-flair.training/blogs/why-you-should-learn-scala-introductory-tutorial/), Python and [**R**.](http://data-flair.training/blogs/r-programming-tutorial/) It also has an optimized engine for general execution graph. In data processing, Apache Spark is the largest open source project

## Features of Apache Spark

Let’s discuss sparkling features of Apache Spark:

### **a. Swift Processing**

Using Apache Spark, we achieve a high data processing speed of about 100x faster in memory and 10x faster on the disk. This is made possible by reducing the number of read-write to disk.

### **b. Dynamic in Nature**

We can easily develop a parallel application, as Spark provides 80 high-level operators.

### **c. In-Memory Computation in Spark**

With [**in-memory processing**](http://data-flair.training/blogs/apache-spark-in-memory-computing/), we can increase the processing speed. Here the data is being cached so we need not fetch data from the disk every time thus the time is saved. Spark has[**DAG**](http://data-flair.training/blogs/directed-acyclic-graph-dag-in-apache-spark/)execution engine which facilitates in-memory computation and acyclic data flow resulting in high speed.

### **d. Reusability**

we can reuse the Spark code for batch-processing, join stream against historical data or run ad-hoc queries on stream state.

### **e. Fault Tolerance in Spark**

[Apache Spark provides fault tolerance](http://data-flair.training/blogs/apache-spark-streaming-fault-tolerance/) through Spark abstraction-RDD. [**Spark RDDs**](http://data-flair.training/blogs/rdd-in-apache-spark/)are designed to handle the failure of any worker node in the cluster. Thus, it ensures that the loss of data reduces to zero. Learn different [ways to create RDD in](http://data-flair.training/blogs/how-to-create-rdds-in-apache-spark/) [Apache Spark](http://data-flair.training/blogs/how-to-create-rdds-in-apache-spark/).

### **Real-Time Stream Processing**

Spark has a provision for real-time stream processing. Earlier the problem with Hadoop [**MapReduce**](http://data-flair.training/blogs/hadoop-mapreduce-introduction-tutorial-comprehensive-guide/)was that it can handle and process data which is already present, but not the real-time data. but with [**Spark Streaming**](http://data-flair.training/blogs/apache-spark-streaming-comprehensive-guide/)we can solve this problem

### **Lazy Evaluation in Apache Spark**

All the [**transformations**](http://data-flair.training/blogs/rdd-transformations-actions-apis-apache-spark/) we make in Spark RDD are Lazy in nature, that is it does not give the result right away rather a new RDD is formed from the existing one. Thus, this increases the efficiency of the system. Follow this guide to learn more about[**Spark Lazy Evaluation**](http://data-flair.training/blogs/lazy-evaluation-in-apache-spark-guide/) in great detail.

### **h. Support Multiple Languages**

In Spark, there is Support for multiple languages like **Java, R, Scala, Python**. Thus, it provides dynamicity and overcomes the [**limitation of Hadoop**](http://data-flair.training/blogs/limitations-of-hadoop/)that it can build applications only in Java.  
Get the [best Scala Books To become an expert in Scala programming language](http://data-flair.training/blogs/best-scala-books-list/).

### **i. Active, Progressive and Expanding Spark Community**

Developers from over 50 companies were involved in making of [**Apache Spark**](http://data-flair.training/blogs/apache-spark-introduction-spark-comprehensive-tutorial/). This project was initiated in the year 2009 and is still expanding and now there are about 250 developers who contributed to its expansion. It is the most important project of Apache Community.

### **j. Support for Sophisticated Analysis**

Spark comes with dedicated tools for streaming data, interactive/declarative queries, machine learning which add-on to map and reduce.

### **k. Integrated with Hadoop**

Spark can run independently and also on [**Hadoop YARN Cluster Manager**](http://data-flair.training/blogs/hadoop-yarn-tutorial/) and thus it can read existing [**Hadoop**](http://data-flair.training/blogs/hadoop-features-design-principles-tutorial/) data. Thus, Spark is flexible.

### **l. Spark GraphX**

Spark has **GraphX**, which is a component for graph and graph-parallel computation. It simplifies the graph analytics tasks by the collection of graph algorithm and builders.

### **m. Cost Efficient**

Apache Spark is cost effective solution for [**Big data**](http://data-flair.training/blogs/big-data-history-use-cases/)problem as in Hadoop large amount of storage and the large data center is required during replication.

**RDD** stands for “**Resilient Distributed Dataset”**. It is the fundamental data structure of Apache Spark. RDD in Apache Spark is an immutable collection of objects which computes on the different node of the cluster.  
Decomposing the name RDD:

* **Resilient**, i.e. fault-tolerant with the help of RDD lineage graph([**DAG**](http://data-flair.training/blogs/directed-acyclic-graph-dag-in-apache-spark/)) and so able to recompute missing or damaged partitions due to node failures.
* **Distributed**,since Data resides on multiple nodes.
* **Dataset**represents records of the data you work with. The user can load the data set externally which can be either JSON file, CSV file, text file or database via JDBC with no specific data structure.

There are three [**ways to create RDDs in Spark**](https://techvidvan.com/tutorials/ways-to-create-rdd-in-spark/) such as – Data in stable storage, other RDDs, and parallelizing already existing collection in driver program. One can also operate Spark RDDs in parallel with a low-level API that offers transformations and actions. We will study these Spark RDD Operations later in this section.

Spark RDD can also be **cached** and **manually partitioned**. Caching is beneficial when we use RDD several times. And manual partitioning is important to correctly balance partitions. Generally, smaller partitions allow distributing RDD data more equally, among more executors. Hence, fewer partitions make the work easy.

Programmers can also call a **persist** method to indicate which RDDs they want to reuse in future operations. Spark keeps persistent RDDs [**in memory**](http://data-flair.training/blogs/apache-spark-in-memory-computing/) by default, but it can spill them to disk if there is not enough RAM. Users can also request other persistence strategies, such as storing the RDD only on disk or replicating it across machines, through flags to persist.

The key motivations behind the concept of RDD are-

* Iterative algorithms.
* Interactive data mining tools.
* **DSM**(Distributed Shared Memory) is a very general abstraction, but this generality makes it harder to implement in an efficient and fault tolerant manner on commodity clusters. Here the need of RDD comes into the picture.
* In distributed computing system data is stored in intermediate stable distributed store such as [**HDFS**](http://data-flair.training/blogs/comprehensive-hdfs-guide-introduction-architecture-data-read-write-tutorial/) or Amazon S3. This makes the computation of job slower since it involves many IO operations, replications, and serializations in the process.

In first two cases we keep data in-memory, it can improve performance by an order of magnitude.  
The main challenge in designing RDD is defining a program interface that provides fault tolerance efficiently. To achieve fault tolerance efficiently, RDDs provide a restricted form of shared memory, based on **coarse-grained transformation** rather than **fine-grained** updates to shared state.

Spark exposes RDD through language integrated API. In integrated API each data set is represented as an object and transformation is involved using the method of these objects

Spark RDD Operations

RDD in Apache Spark supports two types of operations:

* Transformation
* Actions

### **Transformations**

Spark **RDD Transformations** are functions that take an RDD as the input and produce one or many RDDs as the output. They do not change the input RDD (since RDDs are immutable and hence one cannot change it), but always produce one or more new RDDs by applying the computations they represent e.g. Map(), filter(), reduceByKey() etc.

Transformations are **lazy** operations on an RDD in Apache Spark. It creates one or many new RDDs, which executes when an Action occurs. Hence, Transformation creates a new dataset from an existing one.  
Certain transformations can be pipelined which is an optimization method, that Spark uses to improve the performance of computations. There are two kinds of transformations: narrow transformation, wide transformation.

#### **Narrow Transformations**

It is the result of map, filter and such that the data is from a single partition only, i.e. it is self-sufficient. An output RDD has partitions with records that originate from a single partition in the parent RDD. Only a limited subset of partitions used to calculate the result.

Spark groups narrow transformations as a stage known as **pipelining**.

#### **Wide Transformations**

It is the result of groupByKey() and reduceByKey() like functions. The data required to compute the records in a single partition may live in many partitions of the parent RDD. Wide transformations are also known as shuffle transformations because they may or may not depend on a shuffle.

### **Actions**

An**Action** in Spark returns final result of RDD computations. It triggers execution using lineage graph to load the data into original RDD, carry out all intermediate transformations and return final results to Driver program or write it out to file system. Lineage graph is dependency graph of all parallel RDDs of RDD.

**Actions** are RDD operations that produce non-RDD values. They materialize a value in a Spark program. An Action is one of the ways to send result from executors to the driver. First(), take(), reduce(), collect(), the count() is some of the Actions in spark.

Using transformations, one can create RDD from the existing one. But when we want to work with the actual dataset, at that point we use Action. When the Action occurs it does not create the new RDD, unlike transformation. Thus, actions are RDD operations that give no RDD values. Action stores its value either to drivers or to the external storage system. It brings laziness of RDD into motion.

## Limitation of Spark RDD

There is also some limitation of Apache Spark RDD. Let’s discuss them one by one-

*Limitations of Apache Spark RDD*

**i. No inbuilt optimization engine**

When working with structured data, RDDs cannot take advantages of Spark’s advanced optimizers including [**catalyst optimizer**](http://data-flair.training/blogs/spark-sql-optimization-catalyst-optimizer/) and**Tungsten execution engine**. Developers need to optimize each RDD based on its attributes.

Unlike [**Dataframe**](http://data-flair.training/blogs/apache-spark-sql-dataframe-tutorial/) and datasets, RDDs don’t infer the schema of the ingested data and requires the user to specify it.

**iii. Performance limitation**

Being in-memory JVM objects, RDDs involve the overhead of Garbage Collection and Java Serialization which are expensive when data grows.

**iv. Storage limitation**

RDDs degrade when there is not enough memory to store them. One can also store that partition of RDD on disk which does not fit in RAM. As a result, it will provide similar performance to current data-parallel systems.

So, this was all in Spark RDD Tutorial. Hope you like our explanation

Creating RDD from collections using parallelize ::

l = range(1, 10000)

>>> type(l)

<class 'range'>

>>> l = range(1, 1000)

>>> type(l)

<class 'range'>

>>> lRDD = sc.parallelize(l)

>>> type(lRDD)

<class 'pyspark.rdd.PipelinedRDD'>

productsRaw = open("F:/pySpark/data-master/data\_folder/retail\_db/products/part-00000").read()

>>> productsRaw = open("F:/pySpark/data-master/data\_folder/retail\_db/products/part-00000").read().splitlines()

>>> type(pr

productsRaw print( property(

>>> type(productsRaw)

<class 'list'>

>>> productsRDD = sc.parallelize(pro

productsRaw property(

>>> productsRDD = sc.parallelize(productsRaw)

>>> type(productsRDD)

<class 'pyspark.rdd.RDD'>

>>> productsRDD.first()

'1,2,Quest Q64 10 FT. x 10 FT. Slant Leg Instant U,,59.98,http://images.acmesports.sports/Quest+Q64+10+FT.+x+10+FT.+Slant+Leg+Instant+Up+Canopy'

>>>

productsRaw productsRDD

>>> productsRDD.count()

1345

Read data form different formats:

>>> sc

<SparkContext master=local[\*] appName=PySparkShell>

>>> sqlContext

<pyspark.sql.context.SQLContext object at 0x0000020CBF06EEB8>

Two API’s load and read

Help on method load in module pyspark.sql.readwriter:

load(path=None, format=None, schema=None, \*\*options) method of pyspark.sql.readwriter.DataFrameReader instance

Loads data from a data source and returns it as a :class`DataFrame`.

:param path: optional string or a list of string for file-system backed data sources.

:param format: optional string for format of the data source. Default to 'parquet'.

:param schema: optional :class:`pyspark.sql.types.StructType` for the input schema

or a DDL-formatted string (For example ``col0 INT, col1 DOUBLE``).

:param options: all other string options

>>> df = spark.read.format("parquet").load('python/test\_support/sql/parquet\_partitioned',

... opt1=True, opt2=1, opt3='str')

>>> df.dtypes

[('name', 'string'), ('year', 'int'), ('month', 'int'), ('day', 'int')]

>>> df = spark.read.format('json').load(['python/test\_support/sql/people.json',

... 'python/test\_support/sql/people1.json'])

>>> df.dtypes

[('age', 'bigint'), ('aka', 'string'), ('name', 'string')]

.. versionadded:: 1.4

Help on DataFrameReader in module pyspark.sql.readwriter object:

class DataFrameReader(OptionUtils)

| DataFrameReader(spark)

|

| Interface used to load a :class:`DataFrame` from external storage systems

| (e.g. file systems, key-value stores, etc). Use :func:`spark.read`

| to access this.

|

| .. versionadded:: 1.4

|

| Method resolution order:

| DataFrameReader

| OptionUtils

| builtins.object

|

| Methods defined here:

|

| \_\_init\_\_(self, spark)

| Initialize self. See help(type(self)) for accurate signature.

|

| csv(self, path, schema=None, sep=None, encoding=None, quote=None, escape=None, comment=None, header=None, inferSchema=None, ignoreLeadingWhiteSpace=None, ignoreTrailingWhiteSpace=None, nullValue=None, nanValue=None, positiveInf=None, negativeInf=None, dateFormat=None, timestampFormat=None, maxColumns=None, maxCharsPerColumn=None, maxMalformedLogPerPartition=None, mode=None, columnNameOfCorruptRecord=None, multiLine=None, charToEscapeQuoteEscaping=None)

| Loads a CSV file and returns the result as a :class:`DataFrame`.

|

| This function will go through the input once to determine the input schema if

| ``inferSchema`` is enabled. To avoid going through the entire data once, disable

sqlContext.read("F:/pySpark/data-master/data\_folder/retail\_db\_json/order\_items", "json").show()

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

TypeError: 'DataFrameReader' object is not callable

>>> sqlContext.read.format("json").load("F:/pySpark/data-master/data\_folder/retail\_db\_json/order\_items").show()

20/08/01 16:23:23 WARN ObjectStore: Version information not found in metastore. hive.metastore.schema.verification is not enabled so recording the schema version 1.2.0

20/08/01 16:23:24 WARN ObjectStore: Failed to get database default, returning NoSuchObjectException

20/08/01 16:23:27 WARN ObjectStore: Failed to get database global\_temp, returning NoSuchObjectException

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

|order\_item\_id|order\_item\_order\_id|order\_item\_product\_id|order\_item\_product\_price|order\_item\_quantity|order\_item\_subtotal|

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

| 1| 1| 957| 299.98| 1| 299.98|

| 2| 2| 1073| 199.99| 1| 199.99|

| 3| 2| 502| 50.0| 5| 250.0|

| 4| 2| 403| 129.99| 1| 129.99|

| 5| 4| 897| 24.99| 2| 49.98|

| 6| 4| 365| 59.99| 5| 299.95|

| 7| 4| 502| 50.0| 3| 150.0|

| 8| 4| 1014| 49.98| 4| 199.92|

| 9| 5| 957| 299.98| 1| 299.98|

| 10| 5| 365| 59.99| 5| 299.95|

| 11| 5| 1014| 49.98| 2| 99.96|

| 12| 5| 957| 299.98| 1| 299.98|

| 13| 5| 403| 129.99| 1| 129.99|

| 14| 7| 1073| 199.99| 1| 199.99|

| 15| 7| 957| 299.98| 1| 299.98|

| 16| 7| 926| 15.99| 5| 79.95|

| 17| 8| 365| 59.99| 3| 179.97|

| 18| 8| 365| 59.99| 5| 299.95|

| 19| 8| 1014| 49.98| 4| 199.92|

| 20| 8| 502| 50.0| 1| 50.0|

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

only showing top 20 rows

sqlContext.read.jdbc( sqlContext.read.json(

>>> sqlContext.read.json("F:/pySpark/data-master/data\_folder/retail\_db\_json/order\_items").show()

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

|order\_item\_id|order\_item\_order\_id|order\_item\_product\_id|order\_item\_product\_price|order\_item\_quantity|order\_item\_subtotal|

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

| 1| 1| 957| 299.98| 1| 299.98|

| 2| 2| 1073| 199.99| 1| 199.99|

| 3| 2| 502| 50.0| 5| 250.0|

| 4| 2| 403| 129.99| 1| 129.99|

| 5| 4| 897| 24.99| 2| 49.98|

| 6| 4| 365| 59.99| 5| 299.95|

| 7| 4| 502| 50.0| 3| 150.0|

| 8| 4| 1014| 49.98| 4| 199.92|

| 9| 5| 957| 299.98| 1| 299.98|

| 10| 5| 365| 59.99| 5| 299.95|

| 11| 5| 1014| 49.98| 2| 99.96|

| 12| 5| 957| 299.98| 1| 299.98|

| 13| 5| 403| 129.99| 1| 129.99|

| 14| 7| 1073| 199.99| 1| 199.99|

| 15| 7| 957| 299.98| 1| 299.98|

| 16| 7| 926| 15.99| 5| 79.95|

| 17| 8| 365| 59.99| 3| 179.97|

| 18| 8| 365| 59.99| 5| 299.95|

| 19| 8| 1014| 49.98| 4| 199.92|

| 20| 8| 502| 50.0| 1| 50.0|

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

only showing top 20 rows

String Manipulation::

>>> orders = sc.textFile("file:///F:/pySpark/data-master/data\_folder/retail\_db/orders")

>>> type(orders)

<class 'pyspark.rdd.RDD'>

>>> orders.take()

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

TypeError: take() missing 1 required positional argument: 'num'

>>> orders.fi

orders.filter( orders.first(

>>> orders.first()

'1,2013-07-25 00:00:00.0,11599,CLOSED'

>>> orders = sc.textFile("file:///F:/pySpark/data-master/data\_folder/retail\_db/orders")

>>> type(orders)

<class 'pyspark.rdd.RDD'>

>>> orders.take()

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

TypeError: take() missing 1 required positional argument: 'num'

>>> orders.fi

orders.filter( orders.first(

>>> orders.first()

'1,2013-07-25 00:00:00.0,11599,CLOSED'

>>> s = orders.first()

>>> type(s)

<class 'str'>

>>> s[0]

'1'

>>> len(s)

36

>>> s[2]

'2'

>>> s[:12]

'1,2013-07-25'

>>> s[:10]

'1,2013-07-'

>>> s[2:12]

'2013-07-25'

>>> s.spl

s.split( s.splitlines(

>>> s.split(",")

['1', '2013-07-25 00:00:00.0', '11599', 'CLOSED']

>>> type(s.split(","))

<class 'list'>

>>> s.split(",")[0]

Help(str)

Type Cast ::

>>> int(s.split(",")[0])

1

print("Prining " + str(1))

Prining 1

Covert date yyy-mm-dd yyyymmdd

MAP FUNCTIONS

SparkSession available as 'spark'.

>>> orders = sc.textFile("file:///F:/pySpark/data-master/data\_folder/retail\_db/orders")

>>> help(orders.map)

Help on method map in module pyspark.rdd:

map(f, preservesPartitioning=False) method of pyspark.rdd.RDD instance

Return a new RDD by applying a function to each element of this RDD.

>>> rdd = sc.parallelize(["b", "a", "c"])

>>> sorted(rdd.map(lambda x: (x, 1)).collect())

[('a', 1), ('b', 1), ('c', 1)]

orders.first()

'1,2013-07-25 00:00:00.0,11599,CLOSED'

orders.map(lambda o: o.split(",")[3]).first()

'CLOSED'

>>> orders.map(lambda o: o.split(",")[1]).first()

'2013-07-25 00:00:00.0'

orders.map(lambda o: int(o.split(",")[1].split(" ")[0].replace("-",""))).first()

20130725

orders.map(lambda o: int(o.split(",")[1].split(" ")[0].replace("-",""))).take(10)

[20130725, 20130725, 20130725, 20130725, 20130725, 20130725, 20130725, 20130725, 20130725, 20130725]

orders.map(lambda o: int(o.split(",")[1].split(" ")[0].replace("-",""))).count()

68883

Tuples ::

>>> orders.map(lambda o: int(o.split(",")[1].split(" ")[0].replace("-",""))).count()

68883

>>> orders.map( lambda o: (o.split(",")[3], 1)).first()

('CLOSED', 1)

>>> orders.map( lambda o: (o.split(",")[3], 1)).take(10)

[('CLOSED', 1), ('PENDING\_PAYMENT', 1), ('COMPLETE', 1), ('CLOSED', 1), ('COMPLETE', 1), ('COMPLETE', 1), ('COMPLETE', 1), ('PROCESSING', 1), ('PENDING\_PAYMENT', 1), ('PENDING\_PAYMENT', 1)]

>>>

>>> orders.map(lambda o: int(o.split(",")[1].split(" ")[0].replace("-",""))).first()

20130725

>>> orders.map(lambda o: int(o.split(",")[1].split(" ")[0].replace("-",""))).take()

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

TypeError: take() missing 1 required positional argument: 'num'

>>> orders.map(lambda o: int(o.split(",")[1].split(" ")[0].replace("-",""))).take(10)

[20130725, 20130725, 20130725, 20130725, 20130725, 20130725, 20130725, 20130725, 20130725, 20130725]

>>> orders.map(lambda o: int(o.split(",")[1].split(" ")[0].replace("-",""))).count()

68883

>>> orders.map( lambda o: (o.split(",")[3], 1)).first()

('CLOSED', 1)

>>> orders.map( lambda o: (o.split(",")[3], 1)).take(10)

[('CLOSED', 1), ('PENDING\_PAYMENT', 1), ('COMPLETE', 1), ('CLOSED', 1), ('COMPLETE', 1), ('COMPLETE', 1), ('COMPLETE', 1), ('PROCESSING', 1), ('PENDING\_PAYMENT', 1), ('PENDING\_PAYMENT', 1)]

>>> order\_items = sc.textFile("file:///F:/pySpark/data-master/data\_folder/retail\_db/order\_items")

>>> order\_items.first()

'1,1,957,1,299.98,299.98'

>>> for i in order\_items.take(10): print(i)

...

1,1,957,1,299.98,299.98

2,2,1073,1,199.99,199.99

3,2,502,5,250.0,50.0

4,2,403,1,129.99,129.99

5,4,897,2,49.98,24.99

6,4,365,5,299.95,59.99

7,4,502,3,150.0,50.0

8,4,1014,4,199.92,49.98

9,5,957,1,299.98,299.98

10,5,365,5,299.95,59.99

>>> orderitemsMap = order\_items.map(lambda oi: (int(oi.split(",")[1]), float(oi.split(",")[4])))

>>> order

orders order\_items orderitemsMap

>>> orderitemsMap.first()

(1, 299.98)

FLATMAP::

orderitemsMap = order\_items.map(lambda oi: (int(oi.split(",")[1]), float(oi.split(",")[4])))

>>> order

orders order\_items orderitemsMap

>>> orderitemsMap.first()

(1, 299.98)

>>> lineList = ["how are you?", "let us perform", "word count using flatmap", "to understand flatmap in details"]

>>> lines = sc.parallelize(lineList)

>>> words = lines.fla

lines.flatMap( lines.flatMapValues(

>>> words = lines.flatMap(lambda l:l.split(" "))

>>> tuples = words.map(lambda word: (word, 1))

>>> for i in tuples: print(i)

...

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

TypeError: 'PipelinedRDD' object is not iterable

>>> for i in tuples.count: print(i)

tuples.count( tuples.countApprox( tuples.countApproxDistinct( tuples.countByKey( tuples.countByValue(

>>> for i in tuples.countB: print(i)

tuples.countByKey( tuples.countByValue(

>>> for i in tuples.countBy: print(i)

...

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

AttributeError: 'PipelinedRDD' object has no attribute 'countBy'

>>> for i in tuples.countByKey: print(i)

...

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

TypeError: 'method' object is not iterable

>>> for i in tuples.countByKey(): print(i)

...

how

are

you?

let

us

perform

Spark map() vs flatMap(), it is clear that Spark map function expresses a one-to-one transformation. It transforms each element of a collection into one element of the resulting collection. While Spark flatMap function expresses a one-to-many transformation. It transforms each element to 0 or more elements.

Filtering ::

Horizontally and veritically

As we know, spark filter is a [***transformation operation***](https://backtobazics.com/big-data/spark/apache-spark-rdd-operations-transformation-and-action/) of RDD which accepts a predicate as an argument. Predicate is function which accepts some parameter and returns boolean value true or false. Spark filter method will pass this predicate in argument and operates on the source RDD. It will filter all the elements of the source RDD for which predicate is not satisfied and creates new RDD with the elements which are passed by the predicate function. Let’s understand this by following example.

As you can see in above image RDD X is the source RDD and contains elements 1 to 5 and has two partitions. Operation filter is take predicate f(x) as an argument which is some thing like x % 2 == 0 it means it will return true for even elements and false for odd elements. RDD Y is a resulting RDD which will have the filtered (i.e. even elements).

Important points to note are,

* filter is a transformation operation in Spark hence it is lazily evaluated
* It is a narrow operation as it is not shuffling data from one partition to multiple partitions
* filter accepts predicate as an argument and will filter the elements from source RDD which are not satisfied by predicate function

<https://sparkbyexamples.com/pyspark/pyspark-dataframe-filter/>

Using Python version 3.7.3 (default, Apr 24 2019 15:29:51)

SparkSession available as 'spark'.

>>> orders = sc.textFile("file:///F:/pySpark/data-master/data\_folder/retail\_db/orders")

>>> orders.first()

'1,2013-07-25 00:00:00.0,11599,CLOSED'

>>> odersComplete = orders.filter(lambda o: o.split(",")[3] in ["COMPLETE", "CLOSED"] and o.split(",")[1][:7] == "2014-01")

>>> ordersComplete.first()

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

NameError: name 'ordersComplete' is not defined

>>> odersComplete.first()

'25882,2014-01-01 00:00:00.0,4598,COMPLETE'

>>> arrayStructureData = [ (("James","","Gordon"),["Java","scala","C++"],"OH","M"),(("Anna","Rose",""),["Spark","Python","C"],"IR","F"), (("Gaurav","","Malik"),["Python","C","NET"],"WS","M"), (("saurabh","","Bhanwala"),["Test","Full stack","Complete"],"CA","M")]

>>> arrayStructureData.first()

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

AttributeError: 'list' object has no attribute 'first'

>>> arrayStructureData

[(('James', '', 'Gordon'), ['Java', 'scala', 'C++'], 'OH', 'M'), (('Anna', 'Rose', ''), ['Spark', 'Python', 'C'], 'IR', 'F'), (('Gaurav', '', 'Malik'), ['Python', 'C', 'NET'], 'WS', 'M'), (('saurabh', '', 'Bhanwala'), ['Test', 'Full stack', 'Complete'], 'CA', 'M')]

>>> arrayStructureSchema = StructType([ StructField('name', StructType([ StructField('firstName', StringType(), True), StructField('middleName', StringType(), True), StructField('lastName', StringType(), True)])), StructField('langauage', ArrayType(StringType()), True), StructField('state', StringType(), True), StructField('gender', StringType(), True)])

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

NameError: name 'StructType' is not defined

>>> from pyspark.sql.types import StructType

>>> arrayStructureSchema = StructType([ StructField('name', StructType([ StructField('firstName', StringType(), True), StructField('middleName', StringType(), True), StructField('lastName', StringType(), True)])), StructField('langauage', ArrayType(StringType()), True), StructField('state', StringType(), True), StructField('gender', StringType(), True)])

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

NameError: name 'StructField' is not defined

>>> from pyspark.sql.types import \*

>>> arrayStructureSchema = StructType([ StructField('name', StructType([ StructField('firstName', StringType(), True), StructField('middleName', StringType(), True), StructField('lastName', StringType(), True)])), StructField('langauage', ArrayType(StringType()), True), StructField('state', StringType(), True), StructField('gender', StringType(), True)])

df = spark.createDataFrame(data = arrayStructureData, schema = arrayStructureSchema)

20/08/05 15:40:05 WARN ObjectStore: Failed to get database global\_temp, returning NoSuchObjectException

>>> df.printSchema()

root

|-- name: struct (nullable = true)

| |-- firstName: string (nullable = true)

| |-- middleName: string (nullable = true)

| |-- lastName: string (nullable = true)

|-- langauage: array (nullable = true)

| |-- element: string (containsNull = true)

|-- state: string (nullable = true)

|-- gender: string (nullable = true)

>>> df.show(truncate=False)

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

|name |langauage |state|gender|

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

|[James, , Gordon] |[Java, scala, C++] |OH |M |

|[Anna, Rose, ] |[Spark, Python, C] |IR |F |

|[Gaurav, , Malik] |[Python, C, NET] |WS |M |

|[saurabh, , Bhanwala]|[Test, Full stack, Complete]|CA |M |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |

df.show(truncate=False)

from pyspark.sql.functions import col

from pyspark.sql.functions import col

>>> df.filter(col("state") == "CA").show(truncate=False)

df.filter("gender == 'M'").show(truncate=False)

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

|name |langauage |state|gender|

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

|[James, , Gordon] |[Java, scala, C++] |OH |M |

|[Gaurav, , Malik] |[Python, C, NET] |WS |M |

|[saurabh, , Bhanwala]|[Test, Full stack, Complete]|CA |M |

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

df.filter( (df.state == "CA") & (df.gender == "M") ).show(truncate=False)

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

|name |langauage |state|gender|

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

|[saurabh, , Bhanwala]|[Test, Full stack, Complete]|CA |M |

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

ROW LEVEL TRANSFORMATIONS -MAP function4

## Filtering on an Array column

DAG and LAZY EVALUATIONS

df.filter(array\_contains(df.langauage, "JAVA")).show(truncate=False)

## Filtering on Nested Struct columns

df.filter(df.name.firstName == "Bhanwala").show(truncate=False)

COUNT ::

from pyspark import SparkContext

>>> words = sc.parallelize(["scala",

... "java",

... "hadoop",

... "spark",

... "akka",

... "spark vs hadoop",

... "pyspark",

... "pyspark and spark"])

>>> words.first()

'scala'

>>> words.count()

8

## collect()

All the elements in the RDD are returned.

words.collect()

['scala', 'java', 'hadoop', 'spark', 'akka', 'spark vs hadoop', 'pyspark', 'pyspark and spark']

# PySpark Join

PySpark provides multiple ways to combine dataframes i.e. join, merge, [union](https://dzone.com/articles/performance-tip-for-tuning-sql-with-union), SQL interface, etc. In this article, we will take a look at how the PySpark join function is similar to SQL join, where two or more tables or [dataframes](https://dzone.com/articles/pyspark-dataframe-tutorial-introduction-to-datafra) can be combined based on conditions.

Let's take a look at some of the join operations supported by PySpark with examples. First, create two dataframes from Python Dictionary, we will be using these two dataframes in this article.

dataset1 = [{ 'key':'abc', 'val11':'1.1', 'val12':'1.2' }, {'key':'def', 'val11':'3.0', 'val12':'4.0'}]

>>> dataset2 = [{ 'key':'abc', 'val21':'2.0', 'val22':'2.1' }, {'key':'def', 'val21':'3.1', 'val22':'3.2'}]

>>> rdd1 = sc.parallelize(dataset1)

>>> df1 = spark.createDataFrame(rdd1)

C:\spark-2.3.4-bin-hadoop2.6\python\pyspark\sql\session.py:360: UserWarning: Using RDD of dict to inferSchema is deprecated. Use pyspark.sql.Row instead

warnings.warn("Using RDD of dict to inferSchema is deprecated. "

>>> df1.show()

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

|key|val11|val12|

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

|abc| 1.1| 1.2|

|def| 3.0| 4.0|

|  |  |  |
| --- | --- | --- |
|  |  |  |

rdd2 = sc.parallelize(dataset2)

>>> df2 = spark.createDataFrame(rdd2)

The following kinds of joins are explained in this article.

* Inner Join.
* Outer Join.
* Left Join.
* Right Join.
* Left Semi Join.
* Left Anti Join.
* Inner Join with advance conditions.

Inner join

df = df1.join(df2, on=['key'], how='inner'

... )

>>> df.show()

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

|key|val21|val22|val21|val22|

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

|abc| 2.0| 2.1| 2.0| 2.1|

|def| 3.1| 3.2| 3.1| 3.2|

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |

### Outer Join

Outer join combines data from both dataframes, irrespective of 'on' column matches or not. If there is a match combined, one row is created if there is no match missing columns for that row are filled with null.

df= df1.join(df2, on=['key'], how="outer")

>>> df.show()

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

|key|val21|val22|val21|val22|

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

|abc| 2.0| 2.1| 2.0| 2.1|

|def| 3.1| 3.2| 3.1| 3.2|

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |

### Left Join

Left join will choose all the data from the left dataframe (i.e. df1 in this example) and perform matches on column name key. If a match is found, values are filled from the matching row, and if not found, unavailable values are filled with null.

df = df1.join(df2, on=['key'], how='left')

df.show()

### Right Join

This is the same as the left join operation performed on right side dataframe, i.e df2 in this example.

df = df1.join(df2, on=['key'], how='right')

df.show()

RDD join ::

|  |
| --- |
| orders = sc.textFile("/public/retail\_db/orders") |
|  | orderItems = sc.textFile("/public/retail\_db/order\_items") |
|  |  |
|  | ordersMap = orders. \ |
|  | map(lambda o:(int(o.split(",")[0]), o.split(",")[3])) |
|  |  |
|  | orderItemsMap = orderItems. \ |
|  | map(lambda oi:(int(oi.split(",")[1]), float(oi.split(",")[4]))) |
|  |  |
|  | ordersLeftOuterJoin = ordersMap.leftOuterJoin(orderItemsMap) |
|  |  |
|  | ordersLeftOuterJoinFilter = ordersLeftOuterJoin. \ |
|  | filter(lambda o: o[1][1] == None) |
|  |  |
|  | for i in ordersLeftOuterJoin.take(10): print(i) |
|  |  |
|  | ordersRightOuterJoin = orderItemsMap.rightOuterJoin(ordersMap) |
|  | ordersRightOuterJoinFilter = ordersRightOuterJoin. \ |
|  | filter(lambda o: o[1][0] == None) |
|  |  |
|  | for i in ordersRightOuterJoinFilter.take(10): print(i) |

# **Pyspark: GroupBy and Aggregate Functions**

GroupBy allows you to group rows together based off some column value, for example, you could group together sales data by the day the sale occured, or group repeast customer data based off the name of the customer.

Once you've performed the GroupBy operation you can use an aggregate function off that data. An aggregate function aggregates multiple rows of data into a single output, such as taking the sum of inputs, or counting the number of inputs.

Count::

>>> orderItems = sc.textFile("file:///F:/pySpark/data-master/data\_folder/retail\_db/order\_items")

>>> order\_items.first()

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

NameError: name 'order\_items' is not defined

>>> orderItems.first()

'1,1,957,1,299.98,299.98'

>>> orderItems.count()

172198

>>>

Aggregation:

from operator import add

>>> orderIt

orderItems orderItemsFiltered

>>> oderItemSubtotals.reduce(add)

199.99

>>> orderItemsFiltered = orderItems.filter(lambda oi: int(oi.split(",")[1]) == 2)

>>> oderItemSubtotals = orderItemsFiltered.map(lambda oi: float(oi.split(",")[4]))

>>> oderItemSubtotals.reduce(add)

579.98

oderItemSubtotals.reduce(lambda x, y: x + y)

> ## get least value pf subtotal for order id

...

>>> orderItemsFiltered = orderItems.filter(lambda oi: int(oi.split(",")[1]) == 2)

>>> oderItemSubtotalsMin.reduce(lambda x, y: x if(float(x.split(",")[4]) < float(y.split(",")[4])) else y )

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

NameError: name 'oderItemSubtotalsMin' is not defined

>>> orderIte.reduce(lambda x, y: x if(float(x.split(",")[4]) < float(y.split(",")[4])) else y )

orderItems orderItemsFiltered

>>> orderItemsFiltered.reduce(lambda x, y: x if(float(x.split(",")[4]) < float(y.split(",")[4])) else y )

'4,2,403,1,129.99,129.99'

>>>

##### spark-get-count-by-status.py

*#Get count by status - countByKey*

orders = sc.textFile("/public/retail\_db/orders")

ordersStatus = orders. \

map(**lambda** o: (o.split(",")[3], 1))

countByStatus = ordersStatus.countByKey()

**for** i **in** countByStatus: print(i)

>>> countByStatus

defaultdict(<class 'int'>, {'CLOSED': 7556, 'PENDING\_PAYMENT': 15030, 'COMPLETE': 22899, 'PROCESSING': 8275, 'PAYMENT\_REVIEW': 729, 'PENDING': 7610, 'ON\_HOLD': 3798, 'CANCELED': 1428, 'SUSPECTED\_FRAUD': 1558})

>>>

Combiner::

The concept of the combiner is relevant in understanding APIs used for aggregations

* Computing intermediate values and then using intermediate values to compute final values is called combiner
* Aggregations such as sum, min, max, average etc can perform better-using combiner

Groupby key

Need to change your RDD into new RDD with key value pair

**lambda** x, y: x **if**(float(x.split(",")[4]) < float(y.split(",")[4])) **else** y

>>> orderItems = sc.textFile("file:///F:/pySpark/data-master/data\_folder/retail\_db/order\_items")

>>> for i in orderItems.take(10): print(i)

...

1,1,957,1,299.98,299.98

2,2,1073,1,199.99,199.99

3,2,502,5,250.0,50.0

4,2,403,1,129.99,129.99

5,4,897,2,49.98,24.99

6,4,365,5,299.95,59.99

7,4,502,3,150.0,50.0

8,4,1014,4,199.92,49.98

9,5,957,1,299.98,299.98

10,5,365,5,299.95,59.99

>>> orderItemMap = orderItems.map(lambda oi: (int(oi.split(",")[1], float(oi.split(",")[4])))

...

... ;

File "<stdin>", line 3

;

^

SyntaxError: invalid syntax

>>> orderItemMap = orderItems.map(lambda oi: (int(oi.split(",")[1]), float(oi.split(",")[4])))

>>> orderItemsgroupById = orderItemMap.groupByKey()

>>> for i in orderItemMap.take(10): print(i)

...

(1, 299.98)

(2, 199.99)

(2, 250.0)

(2, 129.99)

(4, 49.98)

(4, 299.95)

(4, 150.0)

(4, 199.92)

(5, 299.98)

(5, 299.95)

>>> for i in orderItemsgroupById.take(10): print(i)

...

(2, <pyspark.resultiterable.ResultIterable object at 0x00000226214F8128>)

(4, <pyspark.resultiterable.ResultIterable object at 0x00000226214F8EB8>)

(8, <pyspark.resultiterable.ResultIterable object at 0x00000226214F8F98>)

(10, <pyspark.resultiterable.ResultIterable object at 0x00000226214F8048>)

(12, <pyspark.resultiterable.ResultIterable object at 0x00000226214F80F0>)

(14, <pyspark.resultiterable.ResultIterable object at 0x00000226214F8FD0>)

(16, <pyspark.resultiterable.ResultIterable object at 0x00000226214F8F28>)

(18, <pyspark.resultiterable.ResultIterable object at 0x00000226214F8D68>)

(20, <pyspark.resultiterable.ResultIterable object at 0x00000226201F7C50>)

(24, <pyspark.resultiterable.ResultIterable object at 0x00000226215180F0>)

>>> revenuePerOrderID = orderItemsgroupById.map(lambda oi: (oi[0], round(sum(oi[1]), 2)))

>>> for in reve

revenuePerOrderID reversed(

>>> for in revenuePerOrderID.take(10): print(i)

File "<stdin>", line 1

for in revenuePerOrderID.take(10): print(i)

^

SyntaxError: invalid syntax

>>> for i in revenuePerOrderID.take(10): print(i)

...

(2, 579.98)

(4, 699.85)

(8, 729.84)

(10, 651.92)

(12, 1299.87)

(14, 549.94)

(16, 419.93)

(18, 449.96)

(20, 879.86)

(24, 829.97)

Aggregate byKey

> order\_items = sc.textFile("file:///F:/pySpark/data-master/data\_folder/retail\_db/order\_items")

>>> for i in order\_items.take(10): print(i)

...

1,1,957,1,299.98,299.98

2,2,1073,1,199.99,199.99

3,2,502,5,250.0,50.0

4,2,403,1,129.99,129.99

5,4,897,2,49.98,24.99

6,4,365,5,299.95,59.99

7,4,502,3,150.0,50.0

8,4,1014,4,199.92,49.98

9,5,957,1,299.98,299.98

10,5,365,5,299.95,59.99

>>> orderItemMap = order\_items.map(lambda oi: (int(oi.split(",")[1]), float(oi.split(",")[4])))

>>> for i in orderItemMap.take(10): print(i)

...

(1, 299.98)

(2, 199.99)

(2, 250.0)

(2, 129.99)

(4, 49.98)

(4, 299.95)

(4, 150.0)

(4, 199.92)

(5, 299.98)

(5, 299.95)

>>> ## get sum and count items

...

>>> revenue\_count\_Items = orderItemMap.aggre

orderItemMap.aggregate( orderItemMap.aggregateByKey(

>>> revenue\_count\_Items = orderItemMap.aggregateByKey((0.0, 0), lambda x, y: (x[0] + y, x[1] +1), lambda x, y: (x[0] + y[0], x[1] + y[1]))

>>> for i in revenue\_count\_Items.take(10): print(i)

...

(2, (579.98, 3))

(4, (699.85, 4))

(8, (729.8399999999999, 4))

(10, (651.9200000000001, 5))

(12, (1299.8700000000001, 5))

(14, (549.94, 3))

(16, (419.93, 2))

(18, (449.96000000000004, 3))

(20, (879.8599999999999, 4))

(24, (829.97, 5))

*#Get order item details in descending order by revenue - groupByKey*

orderItems = sc.textFile("/public/retail\_db/order\_items")

orderItemsMap = orderItems. \

map(**lambda** oi: (int(oi.split(",")[1]), oi))

orderItemsGroupByOrderId = orderItemsMap.groupByKey()

orderItemsSortedBySubtotalPerOrder = orderItemsGroupByOrderId. \

flatMap(**lambda** oi:

sorted(oi[1], key=**lambda** k: float(k.split(",")[4]), reverse=**True**)

AgregateByKey

<https://medium.com/@yesilliali/apache-spark-understanding-zerovalue-in-aggregatebykey-function-3d7df62567ae>

sorting :: product by price

>>> products = sc.textFile("file:///F:/pySpark/data-master/data\_folder/retail\_db/products")

>>> for i in products.take(10): print(i)

...

1,2,Quest Q64 10 FT. x 10 FT. Slant Leg Instant U,,59.98,http://images.acmesports.sports/Quest+Q64+10+FT.+x+10+FT.+Slant+Leg+Instant+Up+Canopy

2,2,Under Armour Men's Highlight MC Football Clea,,129.99,http://images.acmesports.sports/Under+Armour+Men%27s+Highlight+MC+Football+Cleat

3,2,Under Armour Men's Renegade D Mid Football Cl,,89.99,http://images.acmesports.sports/Under+Armour+Men%27s+Renegade+D+Mid+Football+Cleat

4,2,Under Armour Men's Renegade D Mid Football Cl,,89.99,http://images.acmesports.sports/Under+Armour+Men%27s+Renegade+D+Mid+Football+Cleat

5,2,Riddell Youth Revolution Speed Custom Footbal,,199.99,http://images.acmesports.sports/Riddell+Youth+Revolution+Speed+Custom+Football+Helmet

6,2,Jordan Men's VI Retro TD Football Cleat,,134.99,http://images.acmesports.sports/Jordan+Men%27s+VI+Retro+TD+Football+Cleat

7,2,Schutt Youth Recruit Hybrid Custom Football H,,99.99,http://images.acmesports.sports/Schutt+Youth+Recruit+Hybrid+Custom+Football+Helmet+2014

8,2,Nike Men's Vapor Carbon Elite TD Football Cle,,129.99,http://images.acmesports.sports/Nike+Men%27s+Vapor+Carbon+Elite+TD+Football+Cleat

9,2,Nike Adult Vapor Jet 3.0 Receiver Gloves,,50.0,http://images.acmesports.sports/Nike+Adult+Vapor+Jet+3.0+Receiver+Gloves

10,2,Under Armour Men's Highlight MC Football Clea,,129.99,http://images.acmesports.sports/Under+Armour+Men%27s+Highlight+MC+Football+Cleat

>>> productMap = products.filter(lambda p: p.split(",")[4] != "").map(lambda p: (float(p.split(",")[4]), p)

...

... )

>>> for i in productMap.take(10): print(i)

...

(59.98, '1,2,Quest Q64 10 FT. x 10 FT. Slant Leg Instant U,,59.98,http://images.acmesports.sports/Quest+Q64+10+FT.+x+10+FT.+Slant+Leg+Instant+Up+Canopy')

(129.99, "2,2,Under Armour Men's Highlight MC Football Clea,,129.99,http://images.acmesports.sports/Under+Armour+Men%27s+Highlight+MC+Football+Cleat")

(89.99, "3,2,Under Armour Men's Renegade D Mid Football Cl,,89.99,http://images.acmesports.sports/Under+Armour+Men%27s+Renegade+D+Mid+Football+Cleat")

(89.99, "4,2,Under Armour Men's Renegade D Mid Football Cl,,89.99,http://images.acmesports.sports/Under+Armour+Men%27s+Renegade+D+Mid+Football+Cleat")

(199.99, '5,2,Riddell Youth Revolution Speed Custom Footbal,,199.99,http://images.acmesports.sports/Riddell+Youth+Revolution+Speed+Custom+Football+Helmet')

(134.99, "6,2,Jordan Men's VI Retro TD Football Cleat,,134.99,http://images.acmesports.sports/Jordan+Men%27s+VI+Retro+TD+Football+Cleat")

(99.99, '7,2,Schutt Youth Recruit Hybrid Custom Football H,,99.99,http://images.acmesports.sports/Schutt+Youth+Recruit+Hybrid+Custom+Football+Helmet+2014')

(129.99, "8,2,Nike Men's Vapor Carbon Elite TD Football Cle,,129.99,http://images.acmesports.sports/Nike+Men%27s+Vapor+Carbon+Elite+TD+Football+Cleat")

(50.0, '9,2,Nike Adult Vapor Jet 3.0 Receiver Gloves,,50.0,http://images.acmesports.sports/Nike+Adult+Vapor+Jet+3.0+Receiver+Gloves')

(129.99, "10,2,Under Armour Men's Highlight MC Football Clea,,129.99,http://images.acmesports.sports/Under+Armour+Men%27s+Highlight+MC+Football+Cleat")

>>> productSorted = productMap.sortByKey().map(lambda p: p[1])

>>> for i in productSorted.take(10): print(i)

...

38,3,Nike Men's Hypervenom Phantom Premium FG Socc,,0.0,http://images.acmesports.sports/Nike+Men%27s+Hypervenom+Phantom+Premium+FG+Soccer+Cleat

388,18,Nike Men's Hypervenom Phantom Premium FG Socc,,0.0,http://images.acmesports.sports/Nike+Men%27s+Hypervenom+Phantom+Premium+FG+Soccer+Cleat

414,19,Nike Men's Hypervenom Phantom Premium FG Socc,,0.0,http://images.acmesports.sports/Nike+Men%27s+Hypervenom+Phantom+Premium+FG+Soccer+Cleat

517,24,Nike Men's Hypervenom Phantom Premium FG Socc,,0.0,http://images.acmesports.sports/Nike+Men%27s+Hypervenom+Phantom+Premium+FG+Soccer+Cleat

547,25,Nike Men's Hypervenom Phantom Premium FG Socc,,0.0,http://images.acmesports.sports/Nike+Men%27s+Hypervenom+Phantom+Premium+FG+Soccer+Cleat

934,42,Callaway X Hot Driver,,0.0,http://images.acmesports.sports/Callaway+X+Hot+Driver

1284,57,Nike Men's Hypervenom Phantom Premium FG Socc,,0.0,http://images.acmesports.sports/Nike+Men%27s+Hypervenom+Phantom+Premium+FG+Soccer+Cleat

624,29,adidas Batting Helmet Hardware Kit,,4.99,http://images.acmesports.sports/adidas+Batting+Helmet+Hardware+Kit

815,37,Zero Friction Practice Golf Balls - 12 Pack,,4.99,http://images.acmesports.sports/Zero+Friction+Practice+Golf+Balls+-+12+Pack

336,15,"Nike Swoosh Headband - 2""",,5.0,http://images.acmesports.sports/Nike+Swoosh+Headband+-+2%22

Sorted by revenue and category key

## sort the data by product price and then product categgory in descending

...

>>>

>>> for i in products.take(10): print(i)

...

1,2,Quest Q64 10 FT. x 10 FT. Slant Leg Instant U,,59.98,http://images.acmesports.sports/Quest+Q64+10+FT.+x+10+FT.+Slant+Leg+Instant+Up+Canopy

2,2,Under Armour Men's Highlight MC Football Clea,,129.99,http://images.acmesports.sports/Under+Armour+Men%27s+Highlight+MC+Football+Cleat

3,2,Under Armour Men's Renegade D Mid Football Cl,,89.99,http://images.acmesports.sports/Under+Armour+Men%27s+Renegade+D+Mid+Football+Cleat

4,2,Under Armour Men's Renegade D Mid Football Cl,,89.99,http://images.acmesports.sports/Under+Armour+Men%27s+Renegade+D+Mid+Football+Cleat

5,2,Riddell Youth Revolution Speed Custom Footbal,,199.99,http://images.acmesports.sports/Riddell+Youth+Revolution+Speed+Custom+Football+Helmet

6,2,Jordan Men's VI Retro TD Football Cleat,,134.99,http://images.acmesports.sports/Jordan+Men%27s+VI+Retro+TD+Football+Cleat

7,2,Schutt Youth Recruit Hybrid Custom Football H,,99.99,http://images.acmesports.sports/Schutt+Youth+Recruit+Hybrid+Custom+Football+Helmet+2014

8,2,Nike Men's Vapor Carbon Elite TD Football Cle,,129.99,http://images.acmesports.sports/Nike+Men%27s+Vapor+Carbon+Elite+TD+Football+Cleat

9,2,Nike Adult Vapor Jet 3.0 Receiver Gloves,,50.0,http://images.acmesports.sports/Nike+Adult+Vapor+Jet+3.0+Receiver+Gloves

10,2,Under Armour Men's Highlight MC Football Clea,,129.99,http://images.acmesports.sports/Under+Armour+Men%27s+Highlight+MC+Football+Cleat

>>> productsMap = products.filter(lambda p: p.split(",")[4] != "").map(lambda p: ((int(p.split(",")[1]), float(p.split(",")[4])), p))

>>> for i in productsMap.take(10): print(i)

...

((2, 59.98), '1,2,Quest Q64 10 FT. x 10 FT. Slant Leg Instant U,,59.98,http://images.acmesports.sports/Quest+Q64+10+FT.+x+10+FT.+Slant+Leg+Instant+Up+Canopy')

((2, 129.99), "2,2,Under Armour Men's Highlight MC Football Clea,,129.99,http://images.acmesports.sports/Under+Armour+Men%27s+Highlight+MC+Football+Cleat")

((2, 89.99), "3,2,Under Armour Men's Renegade D Mid Football Cl,,89.99,http://images.acmesports.sports/Under+Armour+Men%27s+Renegade+D+Mid+Football+Cleat")

((2, 89.99), "4,2,Under Armour Men's Renegade D Mid Football Cl,,89.99,http://images.acmesports.sports/Under+Armour+Men%27s+Renegade+D+Mid+Football+Cleat")

((2, 199.99), '5,2,Riddell Youth Revolution Speed Custom Footbal,,199.99,http://images.acmesports.sports/Riddell+Youth+Revolution+Speed+Custom+Football+Helmet')

((2, 134.99), "6,2,Jordan Men's VI Retro TD Football Cleat,,134.99,http://images.acmesports.sports/Jordan+Men%27s+VI+Retro+TD+Football+Cleat")

((2, 99.99), '7,2,Schutt Youth Recruit Hybrid Custom Football H,,99.99,http://images.acmesports.sports/Schutt+Youth+Recruit+Hybrid+Custom+Football+Helmet+2014')

((2, 129.99), "8,2,Nike Men's Vapor Carbon Elite TD Football Cle,,129.99,http://images.acmesports.sports/Nike+Men%27s+Vapor+Carbon+Elite+TD+Football+Cleat")

((2, 50.0), '9,2,Nike Adult Vapor Jet 3.0 Receiver Gloves,,50.0,http://images.acmesports.sports/Nike+Adult+Vapor+Jet+3.0+Receiver+Gloves')

((2, 129.99), "10,2,Under Armour Men's Highlight MC Football Clea,,129.99,http://images.acmesports.sports/Under+Armour+Men%27s+Highlight+MC+Football+Cleat")

>>> productsMap = products.filter(lambda p: p.split(",")[4] != "").map(lambda p: ((int(p.split(",")[1]), -float(p.split(",")[4])), p))

>>> for i in productsMap.take(10): print(i)

...

((2, -59.98), '1,2,Quest Q64 10 FT. x 10 FT. Slant Leg Instant U,,59.98,http://images.acmesports.sports/Quest+Q64+10+FT.+x+10+FT.+Slant+Leg+Instant+Up+Canopy')

((2, -129.99), "2,2,Under Armour Men's Highlight MC Football Clea,,129.99,http://images.acmesports.sports/Under+Armour+Men%27s+Highlight+MC+Football+Cleat")

((2, -89.99), "3,2,Under Armour Men's Renegade D Mid Football Cl,,89.99,http://images.acmesports.sports/Under+Armour+Men%27s+Renegade+D+Mid+Football+Cleat")

((2, -89.99), "4,2,Under Armour Men's Renegade D Mid Football Cl,,89.99,http://images.acmesports.sports/Under+Armour+Men%27s+Renegade+D+Mid+Football+Cleat")

((2, -199.99), '5,2,Riddell Youth Revolution Speed Custom Footbal,,199.99,http://images.acmesports.sports/Riddell+Youth+Revolution+Speed+Custom+Football+Helmet')

((2, -134.99), "6,2,Jordan Men's VI Retro TD Football Cleat,,134.99,http://images.acmesports.sports/Jordan+Men%27s+VI+Retro+TD+Football+Cleat")

((2, -99.99), '7,2,Schutt Youth Recruit Hybrid Custom Football H,,99.99,http://images.acmesports.sports/Schutt+Youth+Recruit+Hybrid+Custom+Football+Helmet+2014')

((2, -129.99), "8,2,Nike Men's Vapor Carbon Elite TD Football Cle,,129.99,http://images.acmesports.sports/Nike+Men%27s+Vapor+Carbon+Elite+TD+Football+Cleat")

((2, -50.0), '9,2,Nike Adult Vapor Jet 3.0 Receiver Gloves,,50.0,http://images.acmesports.sports/Nike+Adult+Vapor+Jet+3.0+Receiver+Gloves')

((2, -129.99), "10,2,Under Armour Men's Highlight MC Football Clea,,129.99,http://images.acmesports.sports/Under+Armour+Men%27s+Highlight+MC+Football+Cleat")

>>> productsSorted = productsMap.sortB

productsMap.sortBy( productsMap.sortByKey(

>>> productsSorted = productsMap.sortByKey()

>>> for i in productsSorted.take(10): print(i)

...

((2, -299.99), '16,2,Riddell Youth 360 Custom Football Helmet,,299.99,http://images.acmesports.sports/Riddell+Youth+360+Custom+Football+Helmet')

((2, -209.99), '11,2,Fitness Gear 300 lb Olympic Weight Set,,209.99,http://images.acmesports.sports/Fitness+Gear+300+lb+Olympic+Weight+Set')

((2, -199.99), '5,2,Riddell Youth Revolution Speed Custom Footbal,,199.99,http://images.acmesports.sports/Riddell+Youth+Revolution+Speed+Custom+Football+Helmet')

((2, -199.99), '14,2,Quik Shade Summit SX170 10 FT. x 10 FT. Canop,,199.99,http://images.acmesports.sports/Quik+Shade+Summit+SX170+10+FT.+x+10+FT.+Canopy')

((2, -139.99), "12,2,Under Armour Men's Highlight MC Alter Ego Fla,,139.99,http://images.acmesports.sports/Under+Armour+Men%27s+Highlight+MC+Alter+Ego+Flash+Football...")

((2, -139.99), "23,2,Under Armour Men's Highlight MC Alter Ego Hul,,139.99,http://images.acmesports.sports/Under+Armour+Men%27s+Highlight+MC+Alter+Ego+Hulk+Football...")

((2, -134.99), "6,2,Jordan Men's VI Retro TD Football Cleat,,134.99,http://images.acmesports.sports/Jordan+Men%27s+VI+Retro+TD+Football+Cleat")

((2, -129.99), "2,2,Under Armour Men's Highlight MC Football Clea,,129.99,http://images.acmesports.sports/Under+Armour+Men%27s+Highlight+MC+Football+Cleat")

((2, -129.99), "8,2,Nike Men's Vapor Carbon Elite TD Football Cle,,129.99,http://images.acmesports.sports/Nike+Men%27s+Vapor+Carbon+Elite+TD+Football+Cleat")

((2, -129.99), "10,2,Under Armour Men's Highlight MC Football Clea,,129.99,http://images.acmesports.sports/Under+Armour+Men%27s+Highlight+MC+Football+Cleat")

>>> prdoctsSortedMap = productsSorted.map(lambda p: p[1])

>>> for i in prdoctsSortedMap.take(10): print(i)

...

16,2,Riddell Youth 360 Custom Football Helmet,,299.99,http://images.acmesports.sports/Riddell+Youth+360+Custom+Football+Helmet

11,2,Fitness Gear 300 lb Olympic Weight Set,,209.99,http://images.acmesports.sports/Fitness+Gear+300+lb+Olympic+Weight+Set

5,2,Riddell Youth Revolution Speed Custom Footbal,,199.99,http://images.acmesports.sports/Riddell+Youth+Revolution+Speed+Custom+Football+Helmet

14,2,Quik Shade Summit SX170 10 FT. x 10 FT. Canop,,199.99,http://images.acmesports.sports/Quik+Shade+Summit+SX170+10+FT.+x+10+FT.+Canopy

12,2,Under Armour Men's Highlight MC Alter Ego Fla,,139.99,http://images.acmesports.sports/Under+Armour+Men%27s+Highlight+MC+Alter+Ego+Flash+Football...

23,2,Under Armour Men's Highlight MC Alter Ego Hul,,139.99,http://images.acmesports.sports/Under+Armour+Men%27s+Highlight+MC+Alter+Ego+Hulk+Football...

6,2,Jordan Men's VI Retro TD Football Cleat,,134.99,http://images.acmesports.sports/Jordan+Men%27s+VI+Retro+TD+Football+Cleat

2,2,Under Armour Men's Highlight MC Football Clea,,129.99,http://images.acmesports.sports/Under+Armour+Men%27s+Highlight+MC+Football+Cleat

8,2,Nike Men's Vapor Carbon Elite TD Football Cle,,129.99,http://images.acmesports.sports/Nike+Men%27s+Vapor+Carbon+Elite+TD+Football+Cleat

10,2,Under Armour Men's Highlight MC Football Clea,,129.99,http://images.acmesports.sports/Under+Armour+Men%27s+Highlight+MC+Football+Cleat

>>>

RANKING

Ranking data based on certain criteria is key in decision making for any organization. Ranking can be classified as Global as well as by key/per group.

## Global Ranking – sortByKey and take

> productsRankMap = products.filter(lambda p: p.split(",")[4] != "").map(lambda p: (float(p.split(",")[4]), p))

>>> productrankByPrice = productsRankMap.sortByKey(False).map(lambda p: p[1])

>>> for i in productrankByPrice.take(1): print(i)

...

208,10,SOLE E35 Elliptical,,1999.99,http://images.acmesports.sports/SOLE+E35+Elliptical

>>>

Ranking by key :: get top N products by price per category

Per-key or per group ranking gives valuable insight to make an executive decision for a given organization. Getting ranking per group is one of the advanced transformations.

Per-key or Per group ranking can be achieved using

* Good knowledge of programming languages such as Python, especially manipulating collections
* groupByKey with flatMap
* After groupByKey, we need to process the values as a collection using APIs of underlying programming language
* The logic needs to be invoked using flatMap

>>> for i in products.take(10): print(i)

...

1,2,Quest Q64 10 FT. x 10 FT. Slant Leg Instant U,,59.98,http://images.acmesports.sports/Quest+Q64+10+FT.+x+10+FT.+Slant+Leg+Instant+Up+Canopy

2,2,Under Armour Men's Highlight MC Football Clea,,129.99,http://images.acmesports.sports/Under+Armour+Men%27s+Highlight+MC+Football+Cleat

3,2,Under Armour Men's Renegade D Mid Football Cl,,89.99,http://images.acmesports.sports/Under+Armour+Men%27s+Renegade+D+Mid+Football+Cleat

4,2,Under Armour Men's Renegade D Mid Football Cl,,89.99,http://images.acmesports.sports/Under+Armour+Men%27s+Renegade+D+Mid+Football+Cleat

5,2,Riddell Youth Revolution Speed Custom Footbal,,199.99,http://images.acmesports.sports/Riddell+Youth+Revolution+Speed+Custom+Football+Helmet

6,2,Jordan Men's VI Retro TD Football Cleat,,134.99,http://images.acmesports.sports/Jordan+Men%27s+VI+Retro+TD+Football+Cleat

7,2,Schutt Youth Recruit Hybrid Custom Football H,,99.99,http://images.acmesports.sports/Schutt+Youth+Recruit+Hybrid+Custom+Football+Helmet+2014

8,2,Nike Men's Vapor Carbon Elite TD Football Cle,,129.99,http://images.acmesports.sports/Nike+Men%27s+Vapor+Carbon+Elite+TD+Football+Cleat

9,2,Nike Adult Vapor Jet 3.0 Receiver Gloves,,50.0,http://images.acmesports.sports/Nike+Adult+Vapor+Jet+3.0+Receiver+Gloves

10,2,Under Armour Men's Highlight MC Football Clea,,129.99,http://images.acmesports.sports/Under+Armour+Men%27s+Highlight+MC+Football+Cleat

>>> productsMap = products.filter(lambda p: p.split(",")[4] != "").map(lambda p: (int(p.split(",")[1]), p))

>>> for i in productsMap.take(10): print(i)

...

(2, '1,2,Quest Q64 10 FT. x 10 FT. Slant Leg Instant U,,59.98,http://images.acmesports.sports/Quest+Q64+10+FT.+x+10+FT.+Slant+Leg+Instant+Up+Canopy')

(2, "2,2,Under Armour Men's Highlight MC Football Clea,,129.99,http://images.acmesports.sports/Under+Armour+Men%27s+Highlight+MC+Football+Cleat")

(2, "3,2,Under Armour Men's Renegade D Mid Football Cl,,89.99,http://images.acmesports.sports/Under+Armour+Men%27s+Renegade+D+Mid+Football+Cleat")

(2, "4,2,Under Armour Men's Renegade D Mid Football Cl,,89.99,http://images.acmesports.sports/Under+Armour+Men%27s+Renegade+D+Mid+Football+Cleat")

(2, '5,2,Riddell Youth Revolution Speed Custom Footbal,,199.99,http://images.acmesports.sports/Riddell+Youth+Revolution+Speed+Custom+Football+Helmet')

(2, "6,2,Jordan Men's VI Retro TD Football Cleat,,134.99,http://images.acmesports.sports/Jordan+Men%27s+VI+Retro+TD+Football+Cleat")

(2, '7,2,Schutt Youth Recruit Hybrid Custom Football H,,99.99,http://images.acmesports.sports/Schutt+Youth+Recruit+Hybrid+Custom+Football+Helmet+2014')

(2, "8,2,Nike Men's Vapor Carbon Elite TD Football Cle,,129.99,http://images.acmesports.sports/Nike+Men%27s+Vapor+Carbon+Elite+TD+Football+Cleat")

(2, '9,2,Nike Adult Vapor Jet 3.0 Receiver Gloves,,50.0,http://images.acmesports.sports/Nike+Adult+Vapor+Jet+3.0+Receiver+Gloves')

(2, "10,2,Under Armour Men's Highlight MC Football Clea,,129.99,http://images.acmesports.sports/Under+Armour+Men%27s+Highlight+MC+Football+Cleat")

for i in productsGroupByCategory.take(10): print(i)

...

(2, <pyspark.resultiterable.ResultIterable object at 0x00000183AA7096D8>)

(4, <pyspark.resultiterable.ResultIterable object at 0x00000183AA709080>)

(6, <pyspark.resultiterable.ResultIterable object at 0x00000183AA709208>)

(8, <pyspark.resultiterable.ResultIterable object at 0x00000183AA709400>)

(10, <pyspark.resultiterable.ResultIterable object at 0x00000183AA709278>)

(12, <pyspark.resultiterable.ResultIterable object at 0x00000183AA709358>)

(38, <pyspark.resultiterable.ResultIterable object at 0x00000183AA709E80>)

(16, <pyspark.resultiterable.ResultIterable object at 0x00000183AA709DA0>)

(18, <pyspark.resultiterable.ResultIterable object at 0x00000183AA7090B8>)

(20, <pyspark.resultiterable.ResultIterable object at 0x00000183AA709E48>)

>>> productsGroupByTOPN = productsGroupByCategory.flatM

productsGroupByCategory.flatMap( productsGroupByCategory.flatMapValues(

>>> productsGroupByTOPN = productsGroupByCategory.flatMap(lambda p: sorted(p[1], key = lambda k: float(k.split(",")[4]), reverse=True)[:3]

... )

>>> for i in productsGroupByTOPN.take(10): print(i)

...

16,2,Riddell Youth 360 Custom Football Helmet,,299.99,http://images.acmesports.sports/Riddell+Youth+360+Custom+Football+Helmet

11,2,Fitness Gear 300 lb Olympic Weight Set,,209.99,http://images.acmesports.sports/Fitness+Gear+300+lb+Olympic+Weight+Set

5,2,Riddell Youth Revolution Speed Custom Footbal,,199.99,http://images.acmesports.sports/Riddell+Youth+Revolution+Speed+Custom+Football+Helmet

66,4,SOLE F85 Treadmill,,1799.99,http://images.acmesports.sports/SOLE+F85+Treadmill

60,4,SOLE E25 Elliptical,,999.99,http://images.acmesports.sports/SOLE+E25+Elliptical

71,4,Diamondback Adult Response XE Mountain Bike 2,,349.98,http://images.acmesports.sports/Diamondback+Adult+Response+XE+Mountain+Bike+2014

117,6,YETI Tundra 65 Chest Cooler,,399.99,http://images.acmesports.sports/YETI+Tundra+65+Chest+Cooler

106,6,Teeter Hang Ups NXT-S Inversion Table,,299.99,http://images.acmesports.sports/Teeter+Hang+Ups+NXT-S+Inversion+Table

100,6,Quik Shade Summit SX170 10 FT. x 10 FT. Canop,,199.99,http://images.acmesports.sports/Quik+Shade+Summit+SX170+10+FT.+x+10+FT.+Canopy

162,8,YETI Tundra 65 Chest Cooler,,399.99,http://images.acmesports.sports/YETI+Tundra+65+Chest+Cooler

>>>

Ranking by key – Get top N priced products

t = productsGroupByCategoryId.  
filter(lambda p: p[0] == 59).  
first()

l = sorted(t[1], key=lambda k: float(k.split(",")[4]), reverse=True)

for i in l: print(i)  
Get top 3 priced products  
l\_map = map(lambda p: float(p.split(","[4]), l))

topNPrices = sorted(set(l\_map), reverse=True)[:3]

import itertools as it

topNPricedProducts = it.takewhile(lambda p: float(p.split(",")[4]) in topNPrices, l)

list(topNPricedProducts)

|  |
| --- |
| def getTopNPricedProductsPerCategoryId(productsPerCategoryId, topN): |
|  | productsSorted = sorted(productsPerCategoryId[1], |
|  | key=lambda k: float(k.split(",")[4]), |
|  | reverse=True |
|  | ) |
|  | productPrices = map(lambda p: float(p.split(",")[4]), productsSorted) |
|  | topNPrices = sorted(set(productPrices), reverse=True)[:topN] |
|  | import itertools as it |
|  | return it.takewhile(lambda p: |
|  | float(p.split(",")[4]) in topNPrices, |
|  | productsSorted |
|  | ) |

DAG Directed Acyclic Graph