>>> topNPricedProducts = productsGroupByCategory.flatMap(lambda p: getTopNPricedProductsPerCategoryId(p,3))

>>> for i in topN

topNPrices topNPriceProducts topNPricedProducts

>>> for i in topNPricedProducts.collect(): print(i)

Sets Operations::

Let us get into set operations using Spark APIs. Spark support

* union
* intersection
* distinct
* minus – subtract
* When a union is performed, data will not be unique
* Typically we have to use distinct after union to eliminate duplicates

Create a dataset::

F:/pySpark/data-master/data\_folder/retail\_db/order\_items

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

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

>>> orders201312 = orders.filter(lambda o: o.split(",")[1][:7] == "2013-12").map(lambda o: (int(o.split(",")[0]), o))

>>> orders201401 = orders.filter(lambda o: o.split(",")[1][:7] == "2014-01").map(lambda o: (int(o.split(",")[0]), o))

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

>>> orderItems201312 = orders201312.join(orderItemsMap)

>>> for i in orderitems201312.take(10): print(i)

...

Traceback (most recent call last):

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

NameError: name 'orderitems201312' is not defined

>>> for i in orderItems201312.take(10): print(i)

...

(20916, ('20916,2013-12-01 00:00:00.0,11503,CLOSED', '52252,20916,957,1,299.98,299.98'))

(20916, ('20916,2013-12-01 00:00:00.0,11503,CLOSED', '52253,20916,365,2,119.98,59.99'))

(20916, ('20916,2013-12-01 00:00:00.0,11503,CLOSED', '52254,20916,897,5,124.95,24.99'))

(20920, ('20920,2013-12-01 00:00:00.0,4799,PROCESSING', '52264,20920,1073,1,199.99,199.99'))

(20920, ('20920,2013-12-01 00:00:00.0,4799,PROCESSING', '52265,20920,502,1,50.0,50.0'))

(20924, ('20924,2013-12-01 00:00:00.0,417,PENDING', '52273,20924,627,3,119.97,39.99'))

(20924, ('20924,2013-12-01 00:00:00.0,417,PENDING', '52274,20924,502,1,50.0,50.0'))

(20928, ('20928,2013-12-01 00:00:00.0,1516,COMPLETE', '52283,20928,957,1,299.98,299.98'))

(20928, ('20928,2013-12-01 00:00:00.0,1516,COMPLETE', '52284,20928,502,4,200.0,50.0'))

(20928, ('20928,2013-12-01 00:00:00.0,1516,COMPLETE', '52285,20928,403,1,129.99,129.99'))

>>> orderItems201312 = orders201312.join(orderItemsMap).map(lambda oi: oi[1][1])

>>> orderItems201401 = orders201401.join(orderItemsMap).map(lambda oi: oi[1][1])

>>>

Set operations union and intersection

Set operations let us get common elements between two data sets or all the elements from the two data sets.

* A union will get all the elements from both the data sets
* intersect will get all the elements common in both the datasets
* distinct will get all the distinct elements in a data set
* In the case of a union, it will not get distinct elements. Apply distinct, if you only want to get distinct elements after union.
* When we use set operations such as union and intersect, data should have a similar structure
* Diff and complement are not available on top of RDDs

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

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

>>> orders201312 = orders.filter(lambda o: o.split(",")[1][:7] == "2013-12").map(lambda o: (int(o.split(",")[0]), o))

>>> orders201401 = orders.filter(lambda o: o.split(",")[1][:7] == "2014-01").map(lambda o: (int(o.split(",")[0]), o))

|  |
| --- |
| orderItemsMap = orderItems. \ |
|  | map(lambda oi: (int(oi.split(",")[1]), oi)) |

>>> for i in orders201312.take(10): print(i)

...

(20916, '20916,2013-12-01 00:00:00.0,11503,CLOSED')

(20917, '20917,2013-12-01 00:00:00.0,10441,PENDING\_PAYMENT')

(20918, '20918,2013-12-01 00:00:00.0,1664,PENDING')

(20919, '20919,2013-12-01 00:00:00.0,383,COMPLETE')

(20920, '20920,2013-12-01 00:00:00.0,4799,PROCESSING')

(20921, '20921,2013-12-01 00:00:00.0,4712,PROCESSING')

(20922, '20922,2013-12-01 00:00:00.0,9720,COMPLETE')

(20923, '20923,2013-12-01 00:00:00.0,10118,COMPLETE')

(20924, '20924,2013-12-01 00:00:00.0,417,PENDING')

(20925, '20925,2013-12-01 00:00:00.0,6416,PAYMENT\_REVIEW')

>>> orderItems201312 = orders201312.join(orderItemsMap)

>>> for i in ordersItems201312.take(10): print(i)

...

Traceback (most recent call last):

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

NameError: name 'ordersItems201312' is not defined

>>> for i in orderItems201312.take(10): print(i)

...

(20916, ('20916,2013-12-01 00:00:00.0,11503,CLOSED', '52252,20916,957,1,299.98,299.98'))

(20916, ('20916,2013-12-01 00:00:00.0,11503,CLOSED', '52253,20916,365,2,119.98,59.99'))

(20916, ('20916,2013-12-01 00:00:00.0,11503,CLOSED', '52254,20916,897,5,124.95,24.99'))

(20920, ('20920,2013-12-01 00:00:00.0,4799,PROCESSING', '52264,20920,1073,1,199.99,199.99'))

(20920, ('20920,2013-12-01 00:00:00.0,4799,PROCESSING', '52265,20920,502,1,50.0,50.0'))

(20924, ('20924,2013-12-01 00:00:00.0,417,PENDING', '52273,20924,627,3,119.97,39.99'))

(20924, ('20924,2013-12-01 00:00:00.0,417,PENDING', '52274,20924,502,1,50.0,50.0'))

(20928, ('20928,2013-12-01 00:00:00.0,1516,COMPLETE', '52283,20928,957,1,299.98,299.98'))

(20928, ('20928,2013-12-01 00:00:00.0,1516,COMPLETE', '52284,20928,502,4,200.0,50.0'))

(20928, ('20928,2013-12-01 00:00:00.0,1516,COMPLETE', '52285,20928,403,1,129.99,129.99'))

>>> > orderItems201312 = orders201312.join(orderItemsMap).map(lambda oi: oi[1][1])

>>> orderItems201401 = orders201401.join(orderItemsMap).map(lambda oi: oi[1][1]) orderItems201312 = orders201312.join(orderItemsMap).map(lambda oi: oi[1][1])

>>> orderItems201401 = orders201401.join(orderItemsMap).map(lambda oi: oi[1][1])

>>> products201312 = orderItems201312.map(lambda p: int(p.split(",")[2]))

>>> products201401 = orderItems201401.map(lambda p: int(p.split(",")[2]))

>>> for i in products201312.take(10): print(i)

...

957

365

897

1073

502

627

502

957

502

403

>>> allproducts = products201312.union(products201401)

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

...

*#Set operations - Intersection - Get product ids sold in both 2013-12 and 2014-01*

products201312 = orderItems201312. \

map(**lambda** p: int(p.split(",")[2]))

products201401 = orderItems201401. \

map(**lambda** p: int(p.split(",")[2]))

commonproducts = products201312.intersection(products201401)

>>> allproducts.count()

29395

>>> products201312.count()

14729

>>> products201401.count()

commonProducts = products201312.intersection(products201401)

>>> commonProducts.count()

98

Saving data into hdfs::

compression

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

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

>>> from operator import add

>>> reveneuPerOrderID = orderItemsMap.reduceBy

orderItemsMap.reduceByKey( orderItemsMap.reduceByKeyLocally(

>>> reveneuPerOrderID = orderItemsMap.reduceByKey(add).map(lambda r: str(r[0]) + "\t" + str(r[1]))

>>> for i in reve

reveneuPerOrderID reversed(

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

...

2 579.98

4 699.85

8 729.8399999999999

10 651.9200000000001

12 1299.8700000000001

14 549.94

16 419.93

18 449.96000000000004

20 879.8599999999999

24 829.97

>>> saveFileRevenue = reveneuPerOrderID.save

reveneuPerOrderID.saveAsHadoopDataset( reveneuPerOrderID.saveAsNewAPIHadoopDataset( reveneuPerOrderID.saveAsPickleFile( reveneuPerOrderID.saveAsTextFile(

reveneuPerOrderID.saveAsHadoopFile( reveneuPerOrderID.saveAsNewAPIHadoopFile( reveneuPerOrderID.saveAsSequenceFile(

>>> saveFileRevenue = reveneuPerOrderID.saveAsTextFile("file:///F:/pySpark/data-master/data\_folder/retail\_db/revenue")

>>>

|  |
| --- |
| revenuePerOrderId. \ |
|  | saveAsTextFile("/user/dgadiraju/revenue\_per\_order\_compressed", |
|  | compressionCodecClass="org.apache.hadoop.io.compress.SnappyCodec") |

 Saving data into HDFS using Data Frames -  
json

**ave data in different file formats**

* Supported file formats
  + orc
  + json
  + parquet
  + avro (with databricks plugin)
* Steps to save into different file formats
  + Make sure data is represented as Data Frame
  + Use write or save API to save Data Frame into different file formats
  + Use compression algorithm if required

|  |
| --- |
| Saving as JSON - Get revenue per order id |
|  | orderItems = sc.textFile("/public/retail\_db/order\_items") |
|  | orderItemsMap = orderItems. \ |
|  | map(lambda oi: (int(oi.split(",")[1]), float(oi.split(",")[4]))) |
|  |  |
|  | from operator import add |
|  | revenuePerOrderId = orderItemsMap. \ |
|  | reduceByKey(add). \ |
|  | map(lambda r: (r[0], round(r[1], 2))) |
|  |  |
|  | revenuePerOrderIdDF = revenuePerOrderId. \ |
|  | toDF(schema=["order\_id", "order\_revenue"]) |
|  |  |
|  | revenuePerOrderIdDF.save("/user/dgadiraju/revenue\_per\_order\_json", "json") |
|  | revenuePerOrderIdDF.write.json("/user/dgadiraju/revenue\_per\_order\_json") |
|  |  |
|  | sqlContext.read.json("/user/dgadiraju/revenue\_per\_order\_json").show() |