Flatten nested structures and explode arrays with Apache Spark in synapse

Note:

You are not required to complete the processes, tasks, activities, or steps presented in this example. The various samples provided are for illustrative purposes only and it’s likely that if you try this out you will encounter issues in your system.

Here you will learn how to work with complex data structure and use functions to view data more easily.

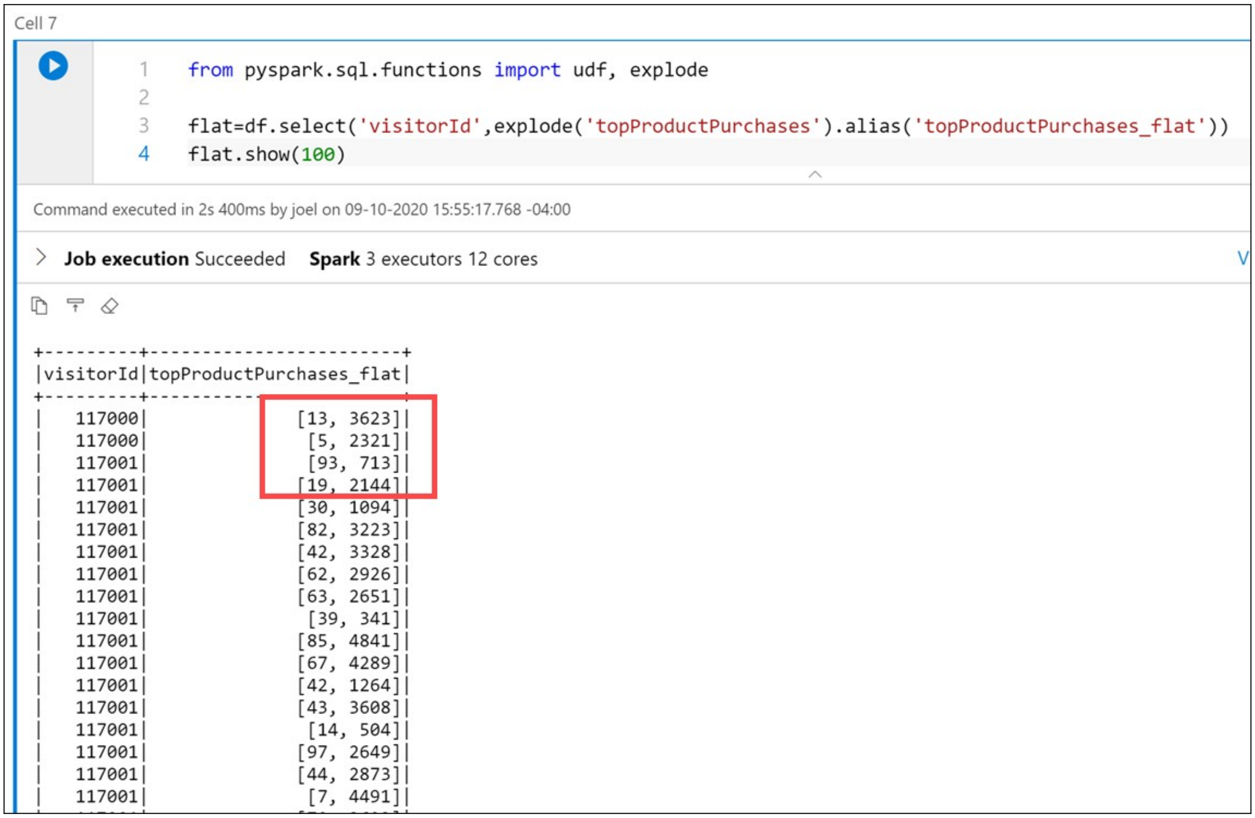
1. PySpark contains a special explode function which returns a new row for each element of the array. The new row helps to flatten the *topProductPurchases* column for better readability or for easier querying. Execute the code below in a new cell:

```
from pyspark.sql.functions import udf, explode

flat=df.select('visitorId',explode('topProductPurchases').alias('topProductPurchases_flat'))
flat.show(100)
```

In this cell, we created a new DataFrame named *flat* that includes the *visitorId* field and a new aliased field named *topProductPurchases_flat*.

As you can see, the output is a bit easier to read and, by extension, easier to query.



Cell 7

```
from pyspark.sql.functions import udf, explode

flat=df.select('visitorId',explode('topProductPurchases').alias('topProductPurchases_flat'))
flat.show(100)
```

Command executed in 2s 400ms by joel on 09-10-2020 15:55:17.768 -04:00

Job execution Succeeded Spark 3 executors 12 cores

```
+---------+-----------------------+
|visitorId|topProductPurchases_flat|
+---------+-----------------------+
|   117000|             [13, 3623]|
|   117000|              [5, 2321]|
|   117001|              [93, 713]|
|   117001|             [19, 2144]|
|   117001|             [30, 1094]|
|   117001|             [82, 3223]|
|   117001|             [42, 3328]|
|   117001|             [62, 2926]|
|   117001|             [63, 2651]|
|   117001|              [39, 341]|
|   117001|             [85, 4841]|
|   117001|             [67, 4289]|
|   117001|             [42, 1264]|
|   117001|             [43, 3608]|
|   117001|              [14, 504]|
|   117001|             [97, 2649]|
|   117001|             [44, 2873]|
|   117001|              [7, 4491]|
```

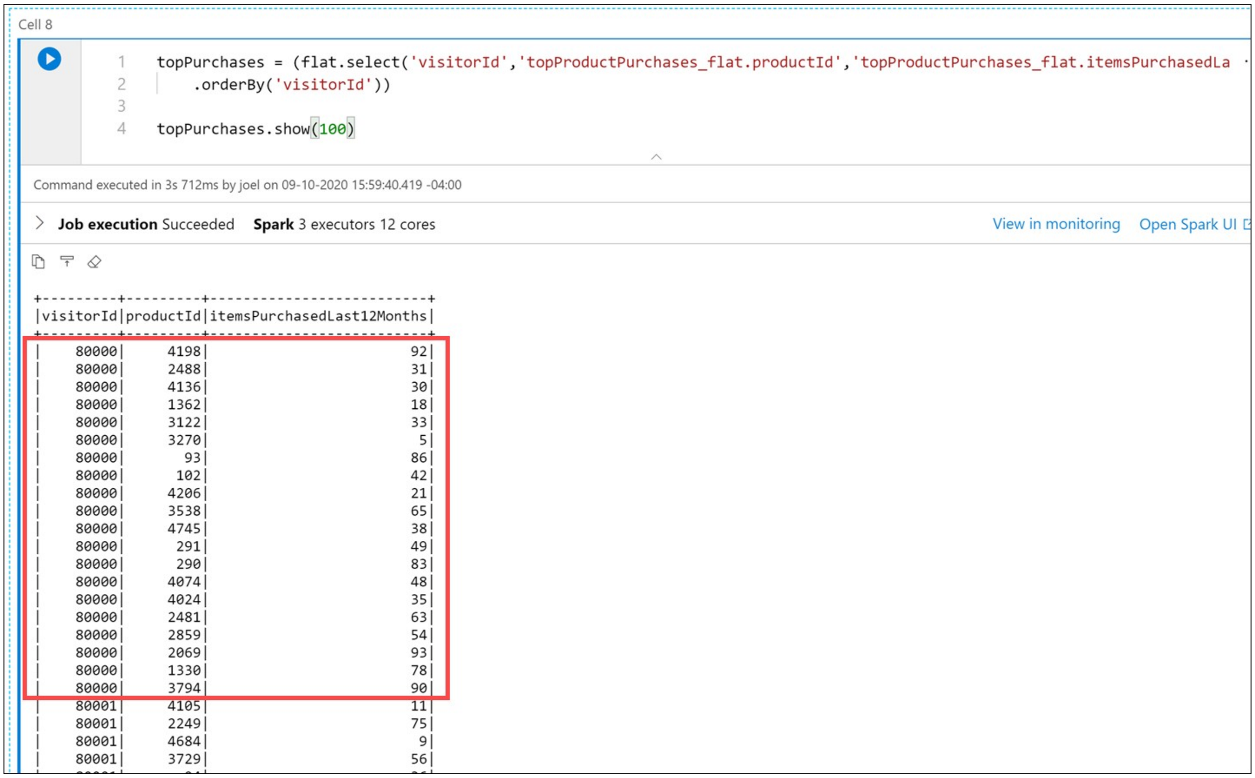
The improved output is displayed.

2. Next, create a new cell and then execute the following code in order to create a new flattened version of the DataFrame that extracts the *topProductPurchases_flat.productId* and *topProductPurchases_flat.itemsPurchasedLast12Months* fields to create new rows for each data combination:

```
topPurchases = (flat.select('visitorId','topProductPurchases_flat.productId','topProductPurchases_flat.itemsPurchasedLast12Months')
    .orderBy('visitorId'))

topPurchases.show(100)
```

In the output, notice that you now have multiple rows for each *visitorId*.



```
topPurchases = (flat.select('visitorId','topProductPurchases_flat.productId','topProductPurchases_flat.itemsPurchasedLa
    .orderBy('visitorId'))

topPurchases.show(100)
```

Command executed in 3s 712ms by joel on 09-10-2020 15:59:40.419 -04:00

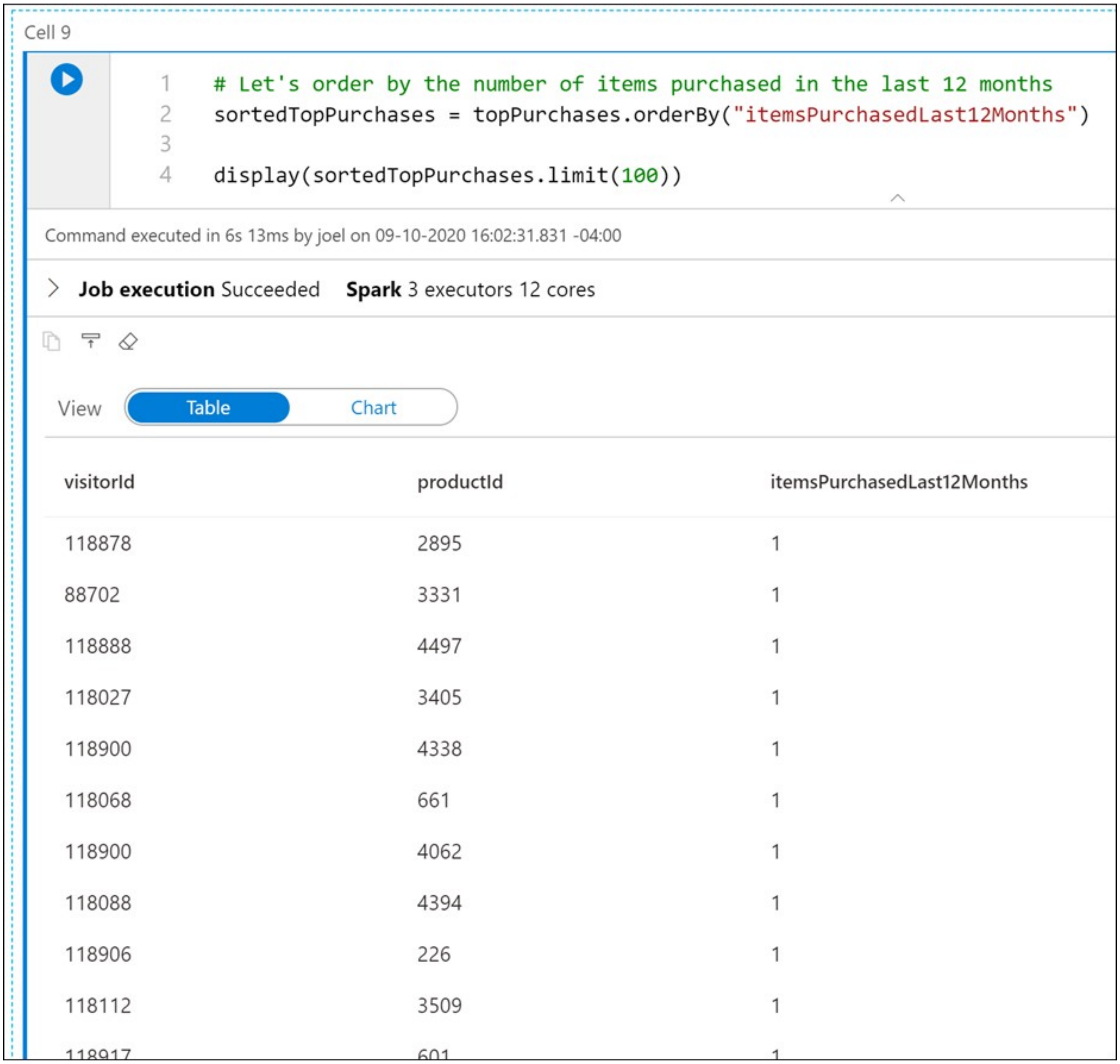
```
+---------+---------+------------------------+
|visitorId|productId|itemsPurchasedLast12Months|
+---------+---------+------------------------+
|    80000|     4198|                      92|
|    80000|     2488|                      31|
|    80000|     4136|                      30|
|    80000|     1362|                      18|
|    80000|     3122|                      33|
|    80000|     3270|                       5|
|    80000|       93|                      86|
|    80000|      102|                      42|
|    80000|     4206|                      21|
|    80000|     3538|                      65|
|    80000|     4745|                      38|
|    80000|      291|                      49|
|    80000|      290|                      83|
|    80000|     4074|                      48|
|    80000|     4024|                      35|
|    80000|     2481|                      63|
|    80000|     2859|                      54|
|    80000|     2069|                      93|
|    80000|     1330|                      78|
|    80000|     3794|                      90|
|    80001|     4105|                      11|
|    80001|     2249|                      75|
|    80001|     4684|                       9|
|    80001|     3729|                      56|
```

The vistorId rows are highlighted.

3. Order the rows by the number of items purchased in the last 12 months. Create a new cell and execute the following code:

```
# Let's order by the number of items purchased in the last 12 months
sortedTopPurchases = topPurchases.orderBy("itemsPurchasedLast12Months")

display(sortedTopPurchases.limit(100))
```



```python
# Let's order by the number of items purchased in the last 12 months
sortedTopPurchases = topPurchases.orderBy("itemsPurchasedLast12Months")

display(sortedTopPurchases.limit(100))
```

Command executed in 6s 13ms by joel on 09-10-2020 16:02:31.831 -04:00

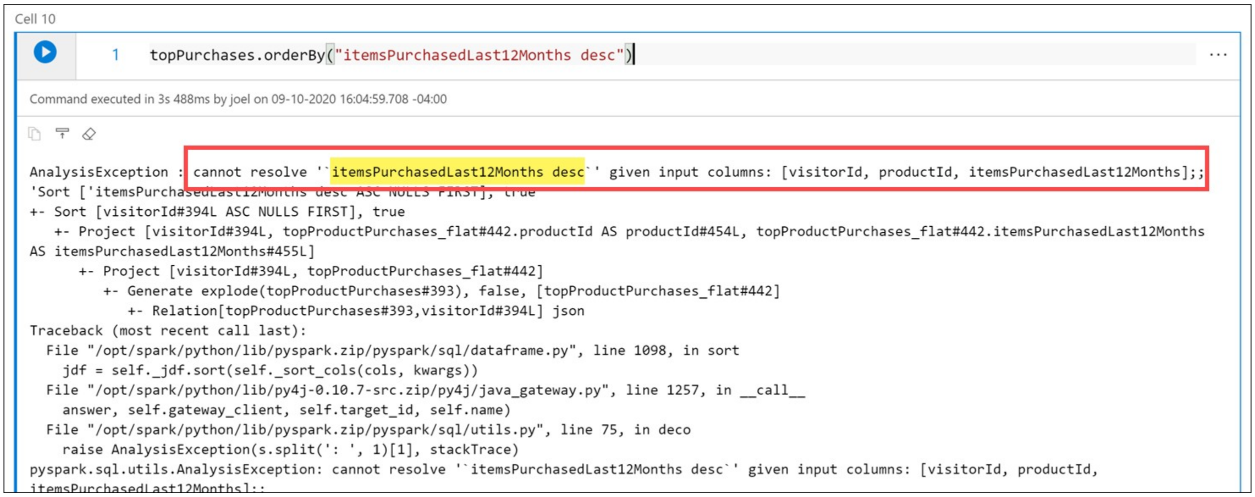
Job execution Succeeded **Spark** 3 executors 12 cores

visitorId	productId	itemsPurchasedLast12Months
118878	2895	1
88702	3331	1
118888	4497	1
118027	3405	1
118900	4338	1
118068	661	1
118900	4062	1
118088	4394	1
118906	226	1
118112	3509	1
118917	601	1

The result is displayed.

4. In order to sort in reverse order, you might conclude that you could make a call like this: *topPurchases.orderBy("itemsPurchasedLast12Months desc")*. Try it in a new cell:

```python
topPurchases.orderBy("itemsPurchasedLast12Months desc")
```



Cell 10

```
topPurchases.orderBy("itemsPurchasedLast12Months desc")
```

Command executed in 3s 488ms by joel on 09-10-2020 16:04:59.708 -04:00

```
AnalysisException : cannot resolve '`itemsPurchasedLast12Months desc`' given input columns: [visitorId, productId, itemsPurchasedLast12Months];;
'Sort ['itemsPurchasedLast12Months desc ASC NULLS FIRST], true
+- Sort [visitorId#394L ASC NULLS FIRST], true
   +- Project [visitorId#394L, topProductPurchases_flat#442.productId AS productId#454L, topProductPurchases_flat#442.itemsPurchasedLast12Months
AS itemsPurchasedLast12Months#455L]
      +- Project [visitorId#394L, topProductPurchases_flat#442]
         +- Generate explode(topProductPurchases#393), false, [topProductPurchases_flat#442]
            +- Relation[topProductPurchases#393,visitorId#394L] json
Traceback (most recent call last):
  File "/opt/spark/python/lib/pyspark.zip/pyspark/sql/dataframe.py", line 1098, in sort
    jdf = self._jdf.sort(self._sort_cols(cols, kwargs))
  File "/opt/spark/python/lib/py4j-0.10.7-src.zip/py4j/java_gateway.py", line 1257, in __call__
    answer, self.gateway_client, self.target_id, self.name)
  File "/opt/spark/python/lib/pyspark.zip/pyspark/sql/utils.py", line 75, in deco
    raise AnalysisException(s.split(': ', 1)[1], stackTrace)
pyspark.sql.utils.AnalysisException: cannot resolve '`itemsPurchasedLast12Months desc`' given input columns: [visitorId, productId,
itemsPurchasedLast12Months];;
```

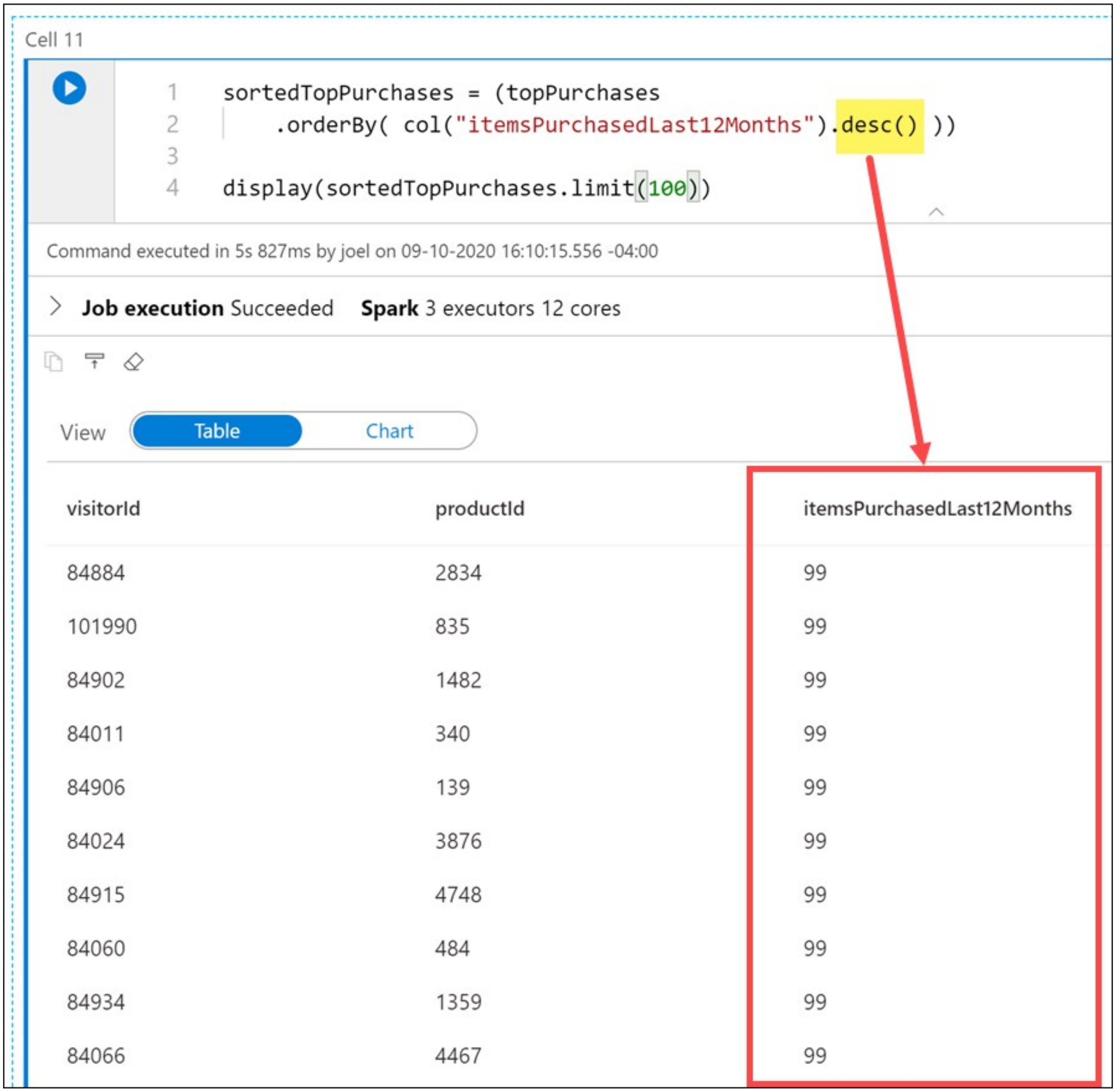
An error is displayed.

Notice that there is an *AnalysisException* error, because *itemsPurchasedLast12Months desc* does not match up with a column name. 5. The **Column** class is an object that encompasses not just the name of the column, but also column-level-transformations, such as sorting in a descending order. Execute the following code in a new cell:

```
sortedTopPurchases = (topPurchases
    .orderBy( col("itemsPurchasedLast12Months").desc() ))

display(sortedTopPurchases.limit(100))
```

Notice that the results are now sorted by the *itemsPurchasedLast12Months* column in descending order, thanks to the *desc()* method on the *col* object.



Cell 11

```
sortedTopPurchases = (topPurchases
    .orderBy( col("itemsPurchasedLast12Months").desc() ))

display(sortedTopPurchases.limit(100))
```

Command executed in 5s 827ms by joel on 09-10-2020 16:10:15.556 -04:00

Job execution Succeeded **Spark** 3 executors 12 cores

visitorId	productId	itemsPurchasedLast12Months
84884	2834	99
101990	835	99
84902	1482	99
84011	340	99
84906	139	99
84024	3876	99
84915	4748	99
84060	484	99
84934	1359	99
84066	4467	99

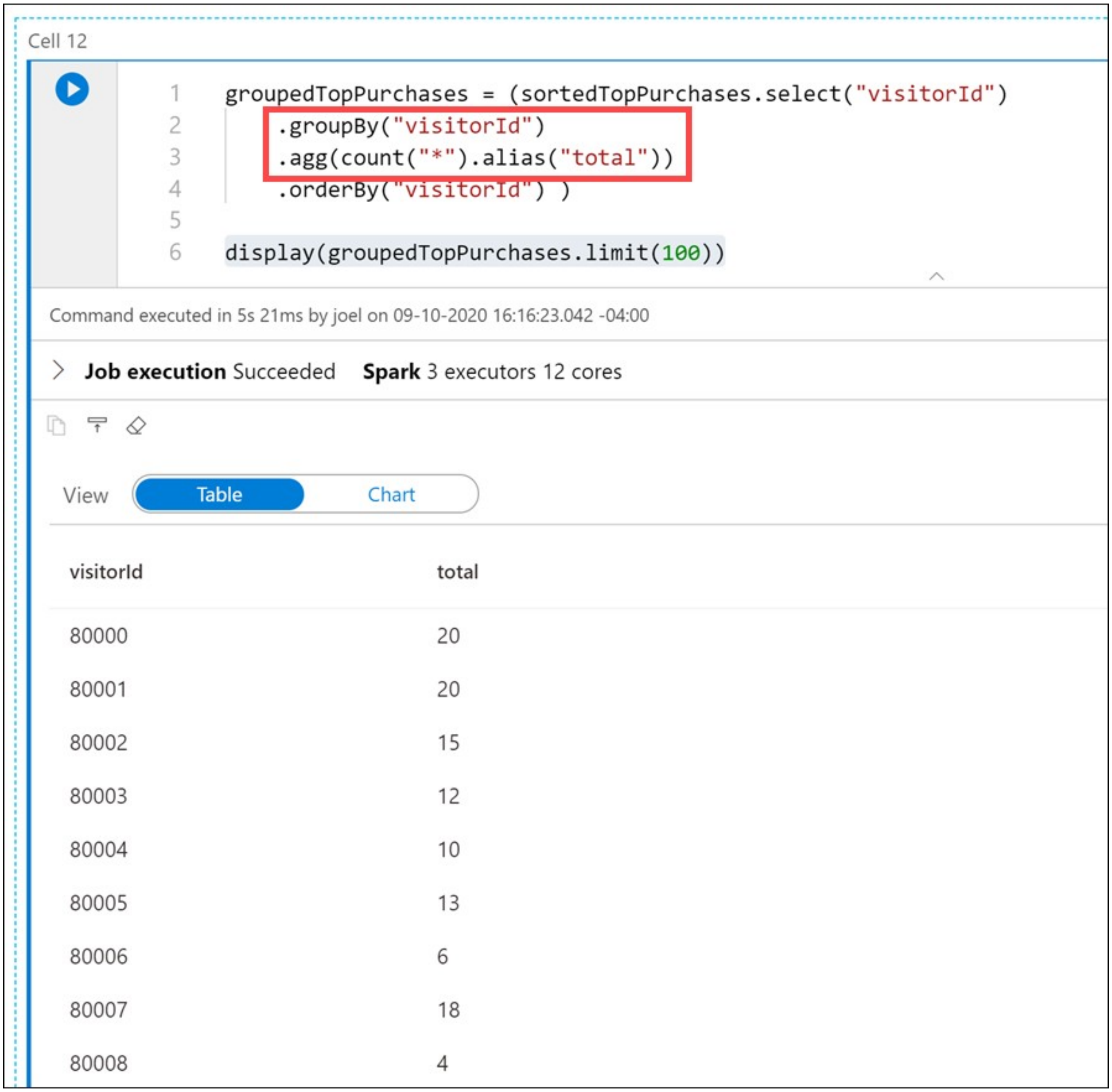
The results are sorted in descending order.

6. How many *types* of products did each customer purchase? To find the answer, group by *visitorId* and aggregate on the number of rows per customer. Execute the following code in a new cell:

```
groupedTopPurchases = (sortedTopPurchases.select("visitorId")
    .groupBy("visitorId")
    .agg(count("*").alias("total"))
    .orderBy("visitorId") )

display(groupedTopPurchases.limit(100))
```

Notice how you can use the *groupby* method on the *visitorId* column, and the *agg* method over a count of records to display the total for each customer.



```
groupedTopPurchases = (sortedTopPurchases.select("visitorId")
    .groupBy("visitorId")
    .agg(count("*").alias("total"))
    .orderBy("visitorId") )

display(groupedTopPurchases.limit(100))
```

Command executed in 5s 21ms by joel on 09-10-2020 16:16:23.042 -04:00

Job execution Succeeded **Spark** 3 executors 12 cores

visitorId	total
80000	20
80001	20
80002	15
80003	12
80004	10
80005	13
80006	6
80007	18
80008	4

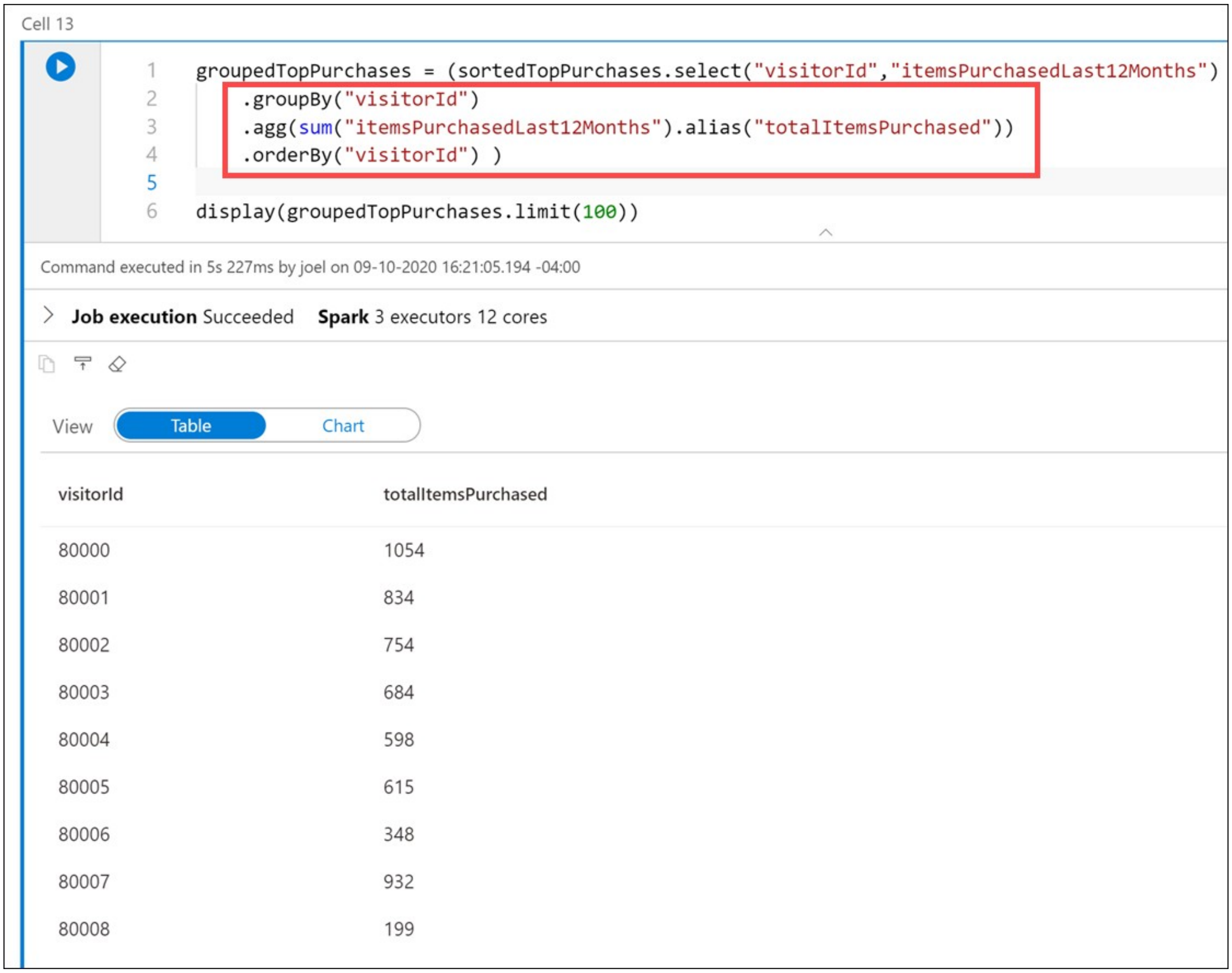
The query output is displayed.

7. How many *total items* did each customer purchase? To find the answer, group by *visitorId* and aggregate on the sum of *itemsPurchasedLast12Months* values per customer. Execute the following code in a new cell:

```
groupedTopPurchases = (sortedTopPurchases.select("visitorId","itemsPurchasedLast12Months")
    .groupBy("visitorId")
    .agg(sum("itemsPurchasedLast12Months").alias("totalItemsPurchased"))
    .orderBy("visitorId") )

groupedTopPurchases.show(100)
```

Group by *visitorId* once again, but now use a *sum* over the *itemsPurchasedLast12Months* column in the *agg* method. Notice that this includes the *itemsPurchasedLast12Months* column in the *select* statement so that it can be used in the *sum*.



Cell 13

```
groupedTopPurchases = (sortedTopPurchases.select("visitorId","itemsPurchasedLast12Months")
    .groupBy("visitorId")
    .agg(sum("itemsPurchasedLast12Months").alias("totalItemsPurchased"))
    .orderBy("visitorId") )

display(groupedTopPurchases.limit(100))
```

Command executed in 5s 227ms by joel on 09-10-2020 16:21:05.194 -04:00

Job execution Succeeded **Spark** 3 executors 12 cores

visitorId	totalItemsPurchased
80000	1054
80001	834
80002	754
80003	684
80004	598
80005	615
80006	348
80007	932
80008	199