CSC 735 – Data Analytics

Chapter 7

Aggregations

Aggregations

- Aggregation is an important feature for big data analytics
- It allow us to summarize the data in order to extract patterns or insights
- Ex: sum, average, stdev, count

Aggregations (cont.)

- Spark has many functions for aggregations

Aggregations (cont.)

- Spark has many functions for aggregations
- For big data analysis, it can be expensive to get an exact answer
- There are many aggregate functions that provide an answer with a **reasonable** degree of accuracy

Aggregations (cont.)

- Spark has many functions for aggregations
- For big data analysis, it can be expensive to get an exact answer
- There are many aggregate functions that provide an answer with a reasonable degree of accuracy
- The aggregation functions are designed to perform aggregation on a set of rows in a DataFrame
- The set of rows can be all or some of the rows

Reading the Dataset

```
val df = spark.read.format("csv")
  .option("header", "true")
  .option("inferSchema", "true")
  .load("/data/retail-data/all/*.csv")
  .coalesce(5)

df.cache()

df.createOrReplaceTempView("dfTable")
```

A Sample of the Data

Command Prompt - spark-shell

```
scala> df.sample(0.000005).show(false)
+---------+---------+----------------------------------+--------+----------------+---------+----------+--------------+
|InvoiceNo|StockCode|Description                       |Quantity|InvoiceDate     |UnitPrice|CustomerID|Country       |
+---------+---------+----------------------------------+--------+----------------+---------+----------+--------------+
|550134   |21937    |STRAWBERRY   PICNIC BAG           |5       |4/14/2011 13:50 |2.95     |16249     |United Kingdom|
|565122   |21976    |PACK OF 60 MUSHROOM CAKE CASES    |1       |9/1/2011 11:48  |0.55     |13069     |United Kingdom|
|574511   |20956    |PORCELAIN T-LIGHT HOLDERS ASSORTED|12      |11/4/2011 13:26 |1.25     |15597     |United Kingdom|
|577480   |23102    |SILVER HEARTS TABLE DECORATION    |12      |11/20/2011 11:37|0.83     |14525     |United Kingdom|
+---------+---------+----------------------------------+--------+----------------+---------+----------+--------------+


scala>
```

Aggregate Functions

- All aggregations are available as functions
- Most aggregation functions are in the package org.apache.spark.sql.functions

count

- In this example, count performs as a transformation instead of an action

```
import org.apache.spark.sql.functions.count
df.select(count("StockCode")).show() // 541909
```

count

- In this example, count performs as a transformation instead of an action

```
import org.apache.spark.sql.functions.count
df.select(count("StockCode")).show() // 541909
```

- Using count, we can do one of 2 things:
  - specify a certain column to count
  - or count all the columns by using count("*")

count

- In this example, count performs as a transformation instead of an action

```
import org.apache.spark.sql.functions.count
df.select(count("StockCode")).show() // 541909
```

- Using count, we can do one of 2 things:
  - specify a certain column to count
  - or count all the columns by using count("*")
  - The first one (count over individual column) ignore null values; count(*) does not

countDistinct

- It counts only the unique values under a given column

```
import org.apache.spark.sql.functions.countDistinct
df.select(countDistinct("StockCode")).show() // 4070
```

approx_count_distinct

- Counting the exact number of unique items in each group in a large dataset can take long time
- Sometimes, it is sufficient to have an approximation to a certain degree of accuracy
- In such case, we can use `approx_count_distinct`

```
import org.apache.spark.sql.functions.approx_count_distinct
df.select(approx_count_distinct("StockCode", 0.05)).show() //  3804
```

approx_count_distinct

- Counting the exact number of unique items in each group in a large dataset can take long time
- Sometimes, it is sufficient to have an approximation to a certain degree of accuracy
- In such case, we can use `approx_count_distinct`

```
import org.apache.spark.sql.functions.approx_count_distinct
df.select(approx_count_distinct("StockCode", 0.05)).show() //  3804

df.select(countDistinct("StockCode")).show() // 4070
```

- Return the maximum age of customers per gender

```
scala> customerDF.show()
+---+--------+---+------+
|cId|    name|age|gender|
+---+--------+---+------+
|  1|   James| 21|     M|
|  2|     Liz| 25|     F|
|  3|    John| 31|     M|
|  4|Jennifer| 45|     F|
|  5|  Robert| 41|     M|
|  6|  Sandra| 45|     F|
+---+--------+---+------+
```

Grouping

- Return the maximum age of customers per gender

```
scala> customerDF.show()
+---+--------+---+------+
|cId|    name|age|gender|
+---+--------+---+------+
|  1|   James| 21|     M|
|  2|     Liz| 25|     F|
|  3|    John| 31|     M|
|  4|Jennifer| 45|     F|
|  5|  Robert| 41|     M|
|  6|  Sandra| 45|     F|
+---+--------+---+------+
```

```
scala> customerDF.groupBy("gender").agg(max("age")).show()
+------+--------+
|gender|max(age)|
+------+--------+
|     F|      45|
|     M|      41|
+------+--------+
```

More examples

- Return the maximum and minimum age of customers per gender

```
scala> customerDF.show()
+---+--------+---+------+
|cId|    name|age|gender|
+---+--------+---+------+
|  1|   James| 21|     M|
|  2|     Liz| 25|     F|
|  3|    John| 31|     M|
|  4|Jennifer| 45|     F|
|  5|  Robert| 41|     M|
|  6|  Sandra| 45|     F|
+---+--------+---+------+
```

More examples

- Return the maximum and minimum age of customers per gender

```
scala> customerDF.show()
+---+--------+---+------+
|cId|    name|age|gender|
+---+--------+---+------+
|  1|   James| 21|     M|
|  2|     Liz| 25|     F|
|  3|    John| 31|     M|
|  4|Jennifer| 45|     F|
|  5|  Robert| 41|     M|
|  6|  Sandra| 45|     F|
+---+--------+---+------+
```

```
scala> customerDF.groupBy("gender").agg(max("age"), min("age")).show()
+------+--------+--------+
|gender|max(age)|min(age)|
+------+--------+--------+
|     F|      45|      25|
|     M|      41|      21|
+------+--------+--------+
```

More examples

Command Prompt - spark-shell

```
scala> df.groupBy("InvoiceNo", "CustomerId").agg(count("Quantity")).show(5)
+---------+----------+---------------+
|InvoiceNo|CustomerId|count(Quantity)|
+---------+----------+---------------+
|   536846|     14573|             76|
|   537026|     12395|             12|
|   537883|     14437|              5|
|   538068|     17978|             12|
|   538279|     14952|              7|
+---------+----------+---------------+
only showing top 5 rows
```

Grouping with Expressions – Renaming Columns

- Rename inside agg using .alias and .as

Command Prompt - spark-shell

```
scala> customerDF.groupBy("gender").agg(max("age").alias("maxAge"),
min("age").as("minAge")).show()
+------+------+------+
|gender|maxAge|minAge|
+------+------+------+
|     F|    45|    25|
|     M|    41|    21|
+------+------+------+
```

Grouping with Maps

- We can specify the arguments of the agg function as a series of key-value maps
  - the key is the column name, and
  - the value is an aggregation function to apply to the key

Command Prompt - spark-shell

```
scala> customerDF.groupBy("gender").agg(
     |     "age" -> "max",
     |     "age" -> "min").show()
+------+--------+--------+
|gender|max(age)|min(age)|
+------+--------+--------+
|     F|      45|      25|
|     M|      41|      21|
+------+--------+--------+
```

Grouping Sets – Examples

```
val items_sold = Seq( ("Shirt", "L", 10), ("Shirt", "M", 20),
                      ("Coat", "M", 15), ("Coat", "L", 5)
                    ).toDF("item", "size", "sales")
```

Grouping Sets – Examples

```
val items_sold = Seq( ("Shirt", "L", 10), ("Shirt", "M", 20),
                      ("Coat", "M", 15), ("Coat", "L", 5)
                    ).toDF("item", "size", "sales")

items_sold.createOrReplaceTempView("items_sold_Table")
```

Grouping Sets – Examples

```
val items_sold = Seq( ("Shirt", "L", 10), ("Shirt", "M", 20),
                      ("Coat", "M", 15), ("Coat", "L", 5)
                    ).toDF("item", "size", "sales")

items_sold.createOrReplaceTempView("items_sold_Table")
```

```
scala> items_sold.show
+-----+----+-----+
| item|size|sales|
+-----+----+-----+
|Shirt|   L|   10|
|Shirt|   M|   20|
| Coat|   M|   15|
| Coat|   L|    5|
+-----+----+-----+
```

Grouping Sets – Examples

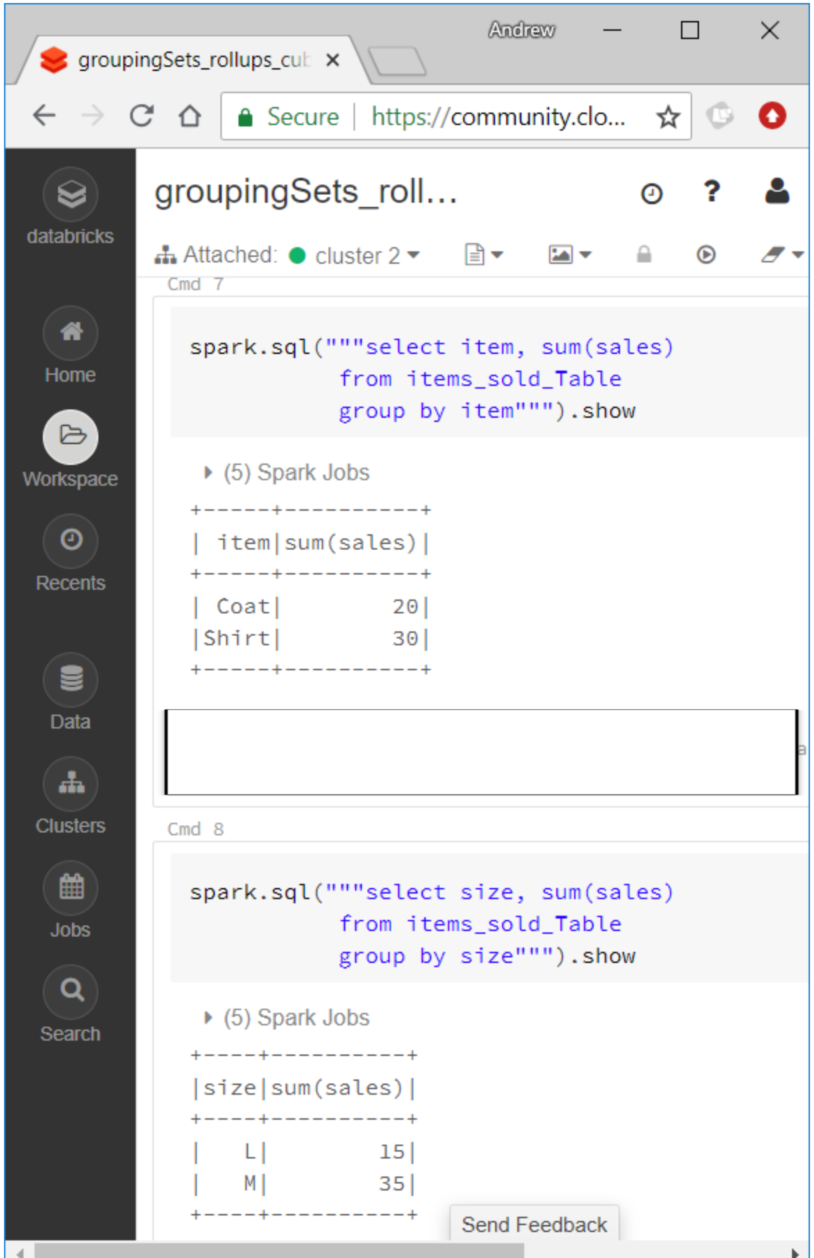
```
val items_sold = Seq( ("Shirt", "L", 10), ("Shirt", "M", 20),
                      ("Coat", "M", 15), ("Coat", "L", 5)
                    ).toDF("item", "size", "sales")

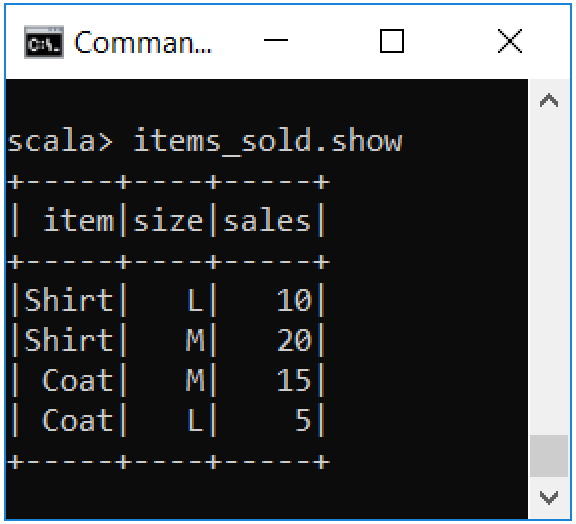
items_sold.createOrReplaceTempView("items_sold_Table")
```

```
scala> items_sold.show
+-----+----+-----+
| item|size|sales|
+-----+----+-----+
|Shirt|   L|   10|
|Shirt|   M|   20|
| Coat|   M|   15|
| Coat|   L|    5|
+-----+----+-----+
```

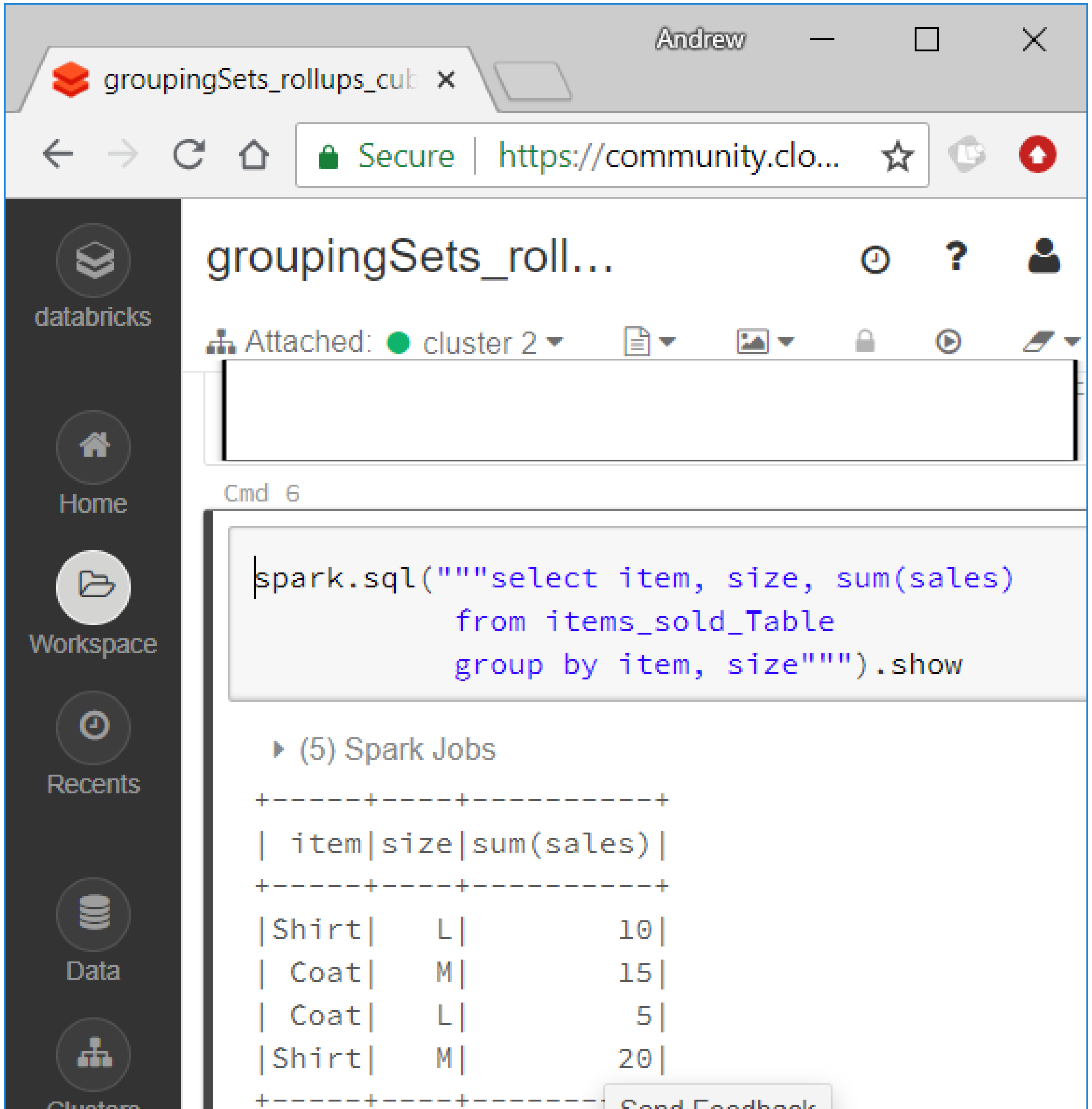
```
scala> items_sold.printSchema
root
 |-- item: string (nullable = true)
 |-- size: string (nullable = true)
 |-- sales: integer (nullable = false)
```

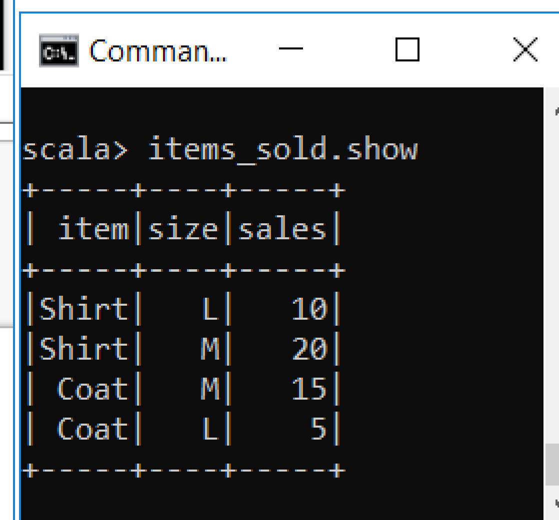
Grouping Sets – Examples





Grouping Sets – Examples





Grouping Sets – Examples

groupingSets_rollups_cubes (Scala)

```
//using grouping sets
spark.sql("""select item, size, sum(sales)
           from items_sold_Table
           group by item, size GROUPING SETS ((item), (size))""").show
```

(5) Spark Jobs

```
+-----+----+----------+
| item|size|sum(sales)|
+-----+----+----------+
| null|   M|        35|
| null|   L|        15|
| Coat|null|        20|
|Shirt|null|        30|
+-----+----+----------+
```

```
scala> items_sold.show
+-----+----+-----+
| item|size|sales|
+-----+----+-----+
|Shirt|   L|   10|
|Shirt|   M|   20|
| Coat|   M|   15|
| Coat|   L|    5|
+-----+----+-----+
```

groupingSets_rollups_cubes (Scala)

Cmd 10

```
//using grouping sets
spark.sql("""select item, size, sum(sales)
             from items_sold_Table
             group by item, size GROUPING SETS ((item), (size), ())""").show
```

(5) Spark Jobs

```
+-----+----+----------+
| item|size|sum(sales)|
+-----+----+----------+
| null|null|        50|
| null|   M|        35|
| null|   L|        15|
| Coat|null|        20|
|Shirt|null|        30|
+-----+----+----------+
```

Comman...

```
scala> items_sold.show
+-----+----+-----+
| item|size|sales|
+-----+----+-----+
|Shirt|   L|   10|
|Shirt|   M|   20|
| Coat|   M|   15|
| Coat|   L|    5|
+-----+----+-----+
```

groupingSets_rollups_cubes (Scala)

Attached: cluster 2

Cmd 11

```
//order results so that nulls are shown last
spark.sql("""select item, size, sum(sales)
          from items_sold_Table
          group by item, size GROUPING SETS ((item), (size), ())
          order by item desc, size desc""").show
```

(1) Spark Jobs

```
+-----+----+----------+
| item|size|sum(sales)|
+-----+----+----------+
|Shirt|null|        30|
| Coat|null|        20|
| null|   M|        35|
| null|   L|        15|
| null|null|        50|
+-----+----+----------+
```

Comman...

```
scala> items_sold.show
+-----+----+-----+
| item|size|sales|
+-----+----+-----+
|Shirt|   L|   10|
|Shirt|   M|   20|
| Coat|   M|   15|
| Coat|   L|    5|
+-----+----+-----+
```

Grouping Sets

- Aggregate functions are computed for each group, and then the results are added to the output
- Each sublist of grouping sets may specify zero or more columns
- An **empty grouping set** means that **all** rows are aggregated down to a single group
- References to the grouping columns are replaced by null values in result rows

Aggregations with Rollups and Cubes

- Grouping sets are only available in SQL
- To do the same thing using DataFrames, we use **rollup** and **cube** operations

Grouping Sets – Rollups

- A clause of the form

```
ROLLUP ( e1, e2, e3, ... )
```

is equivalent to

```
GROUPING SETS (
    ( e1, e2, e3, ... ),
    ...
    ( e1, e2 ),
    ( e1 ),
    ( )
)
```

a GROUPING SETS operation that represents the given list of expressions and **all prefixes** of the list including the empty list

Grouping Sets – Rollups

- A clause of the form

```
ROLLUP(warehouse, product)
```

is equivalent to

```
?
```

a GROUPING SETS operation that represents the given list of expressions and all prefixes of the list including the empty list

Grouping Sets – Rollups

- A clause of the form

```
ROLLUP(warehouse, product)
```

is equivalent to

```
GROUPING SETS(
(warehouse, product),
(warehouse),
()
)
```

The N elements of a ROLLUP specification results in ? GROUPING SETS

Grouping Sets – Rollups

- A clause of the form

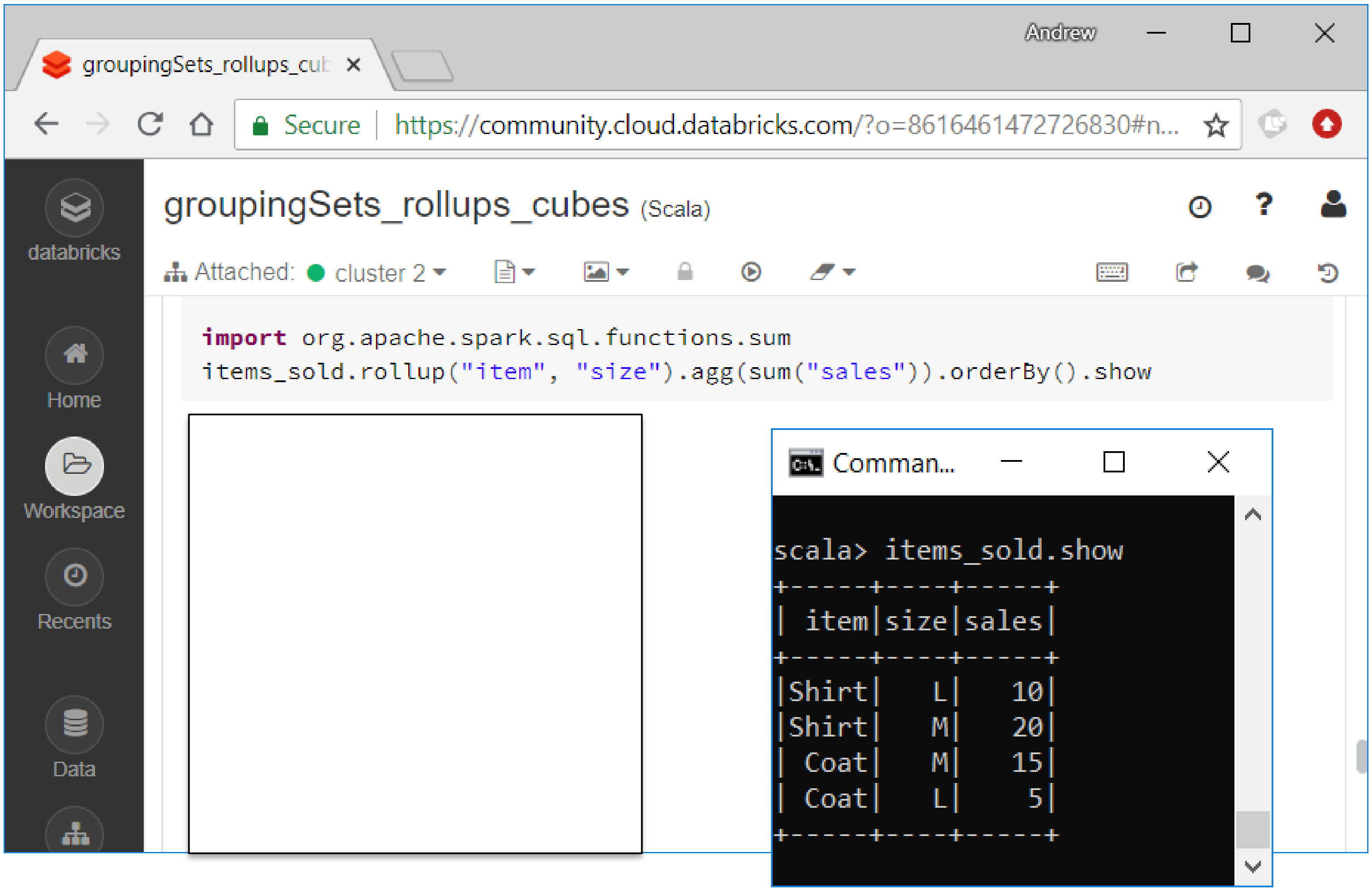
```
ROLLUP(warehouse, product)
```

is equivalent to

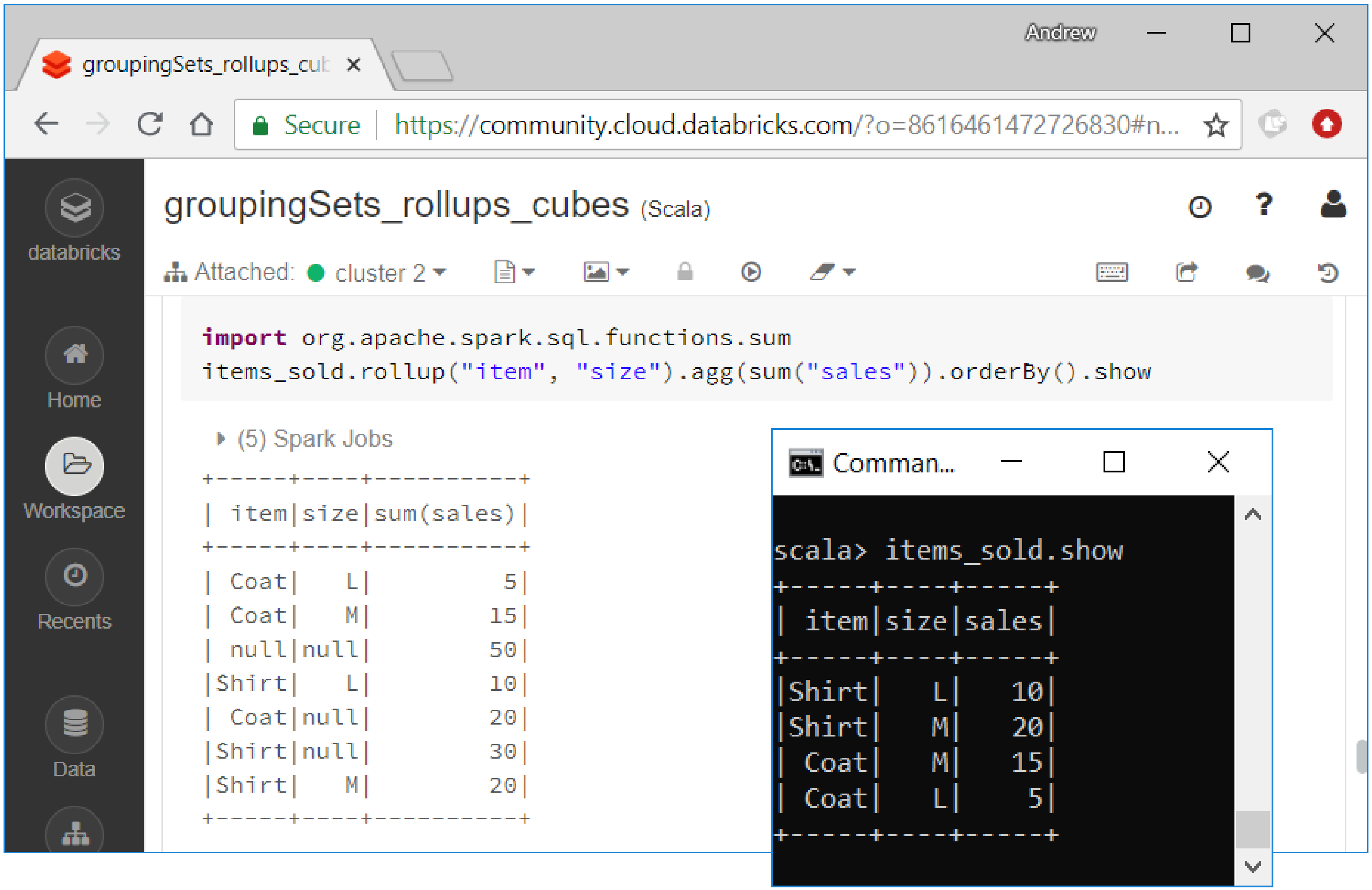
```
GROUPING SETS(
(warehouse, product),
(warehouse),
()
)
```

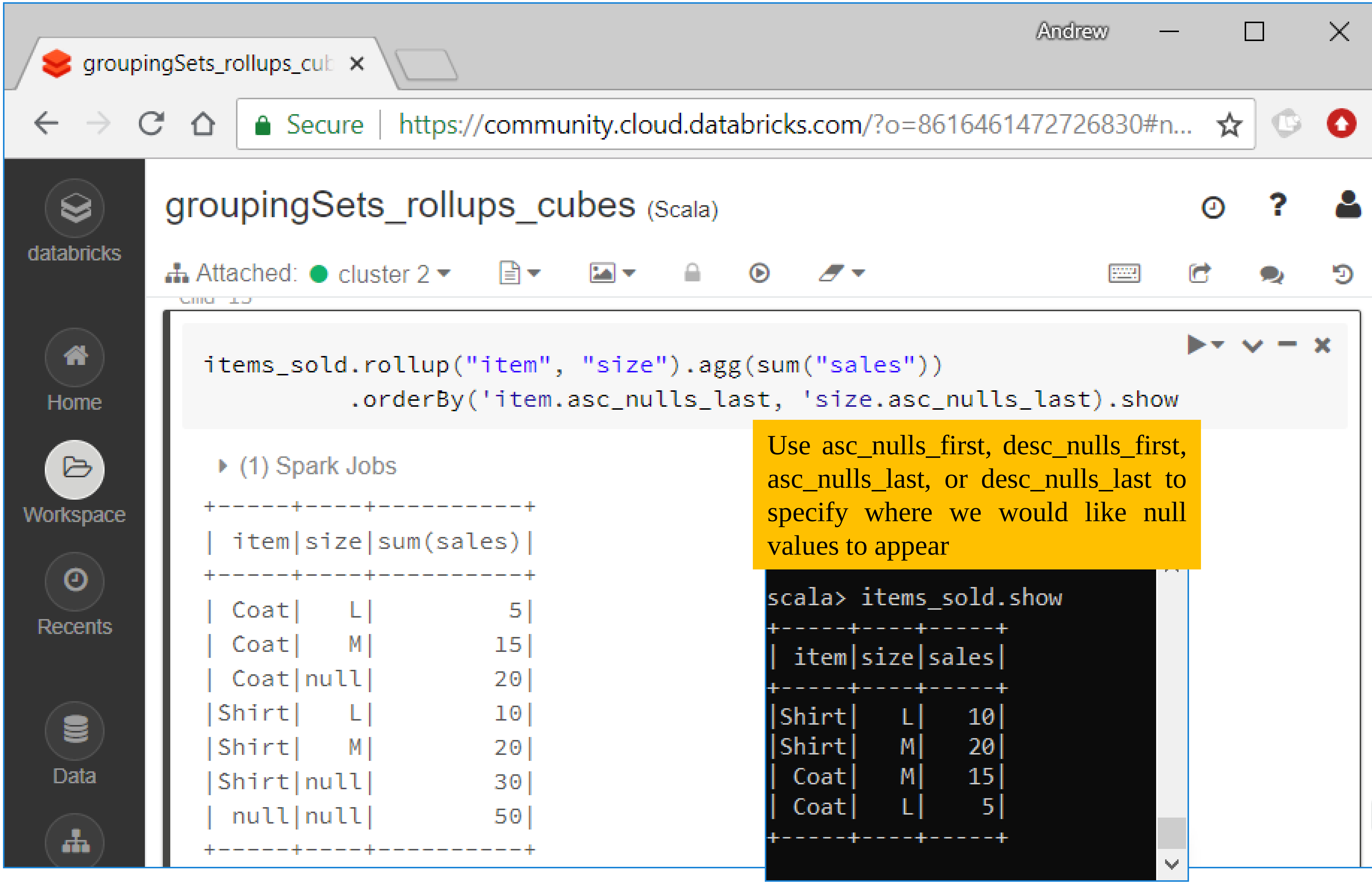
The N elements of a ROLLUP specification results in N+1 GROUPING SETS

Rollups – Example



Rollups – Example





Use asc_nulls_first, desc_nulls_first, asc_nulls_last, or desc_nulls_last to specify where we would like null values to appear

Grouping Sets – Cubes

- A clause of the form

```
CUBE ( e1, e2, e3, ... )
```

is equivalent to

```
GROUPING SETS (
    ( a, b, c ),
    ( a, b    ),
    ( a,    c ),
    ( a       ),
    (    b, c ),
    (    b    ),
    (       c ),
    (         )
)
```

Grouping Sets – Cubes

- A clause of the form

  CUBE(warehouse, product)

  is equivalent to

  ?

Grouping Sets – Cubes

- A clause of the form

```
CUBE(warehouse, product)
```

is equivalent to

```
GROUPING SETS(
(warehouse, product),
(warehouse),
(product),
())
)
```

The N elements of a CUBE specification results in ? GROUPING SETS

Grouping Sets – Cubes

- A clause of the form

```
CUBE(warehouse, product)
```

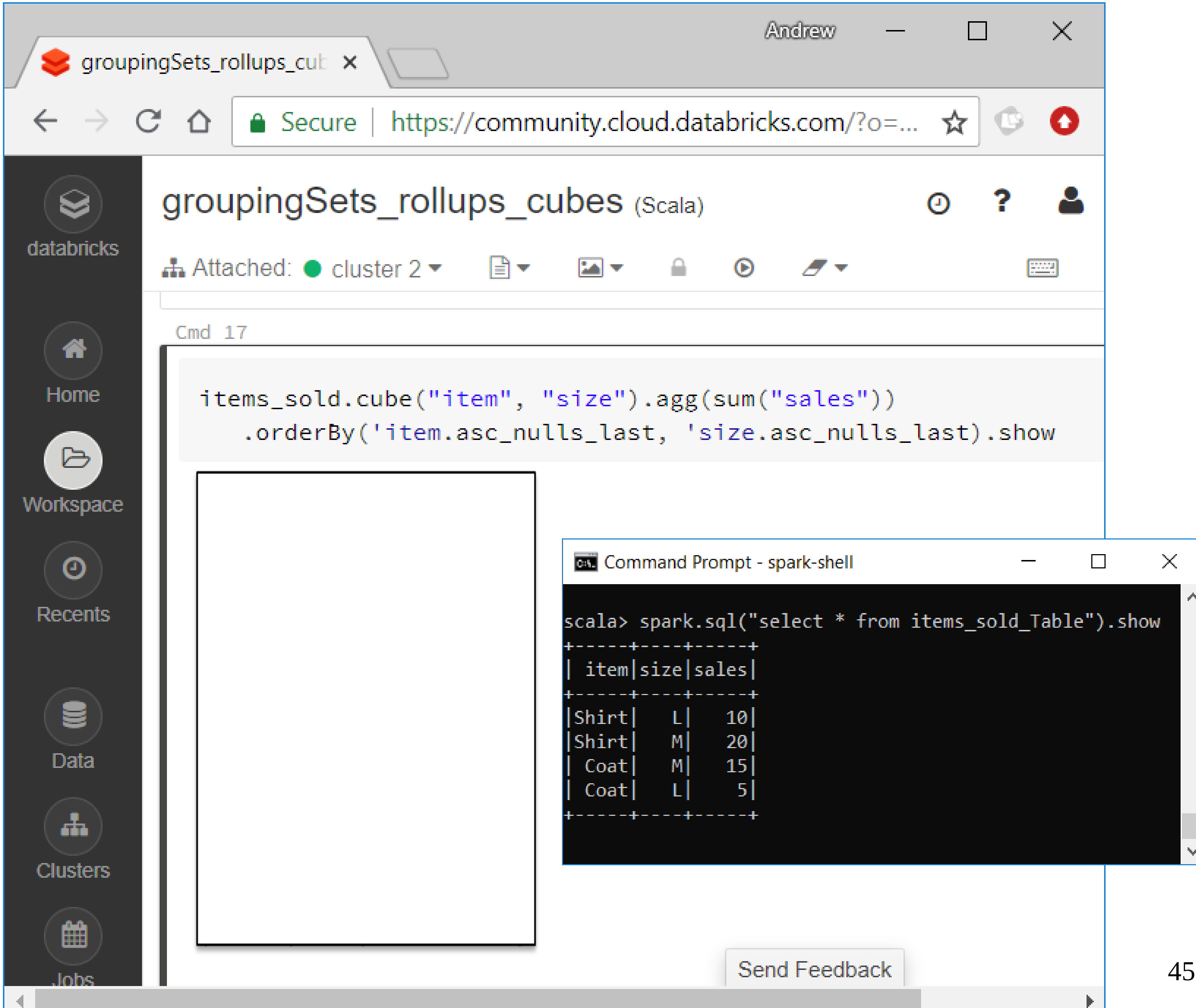
is equivalent to

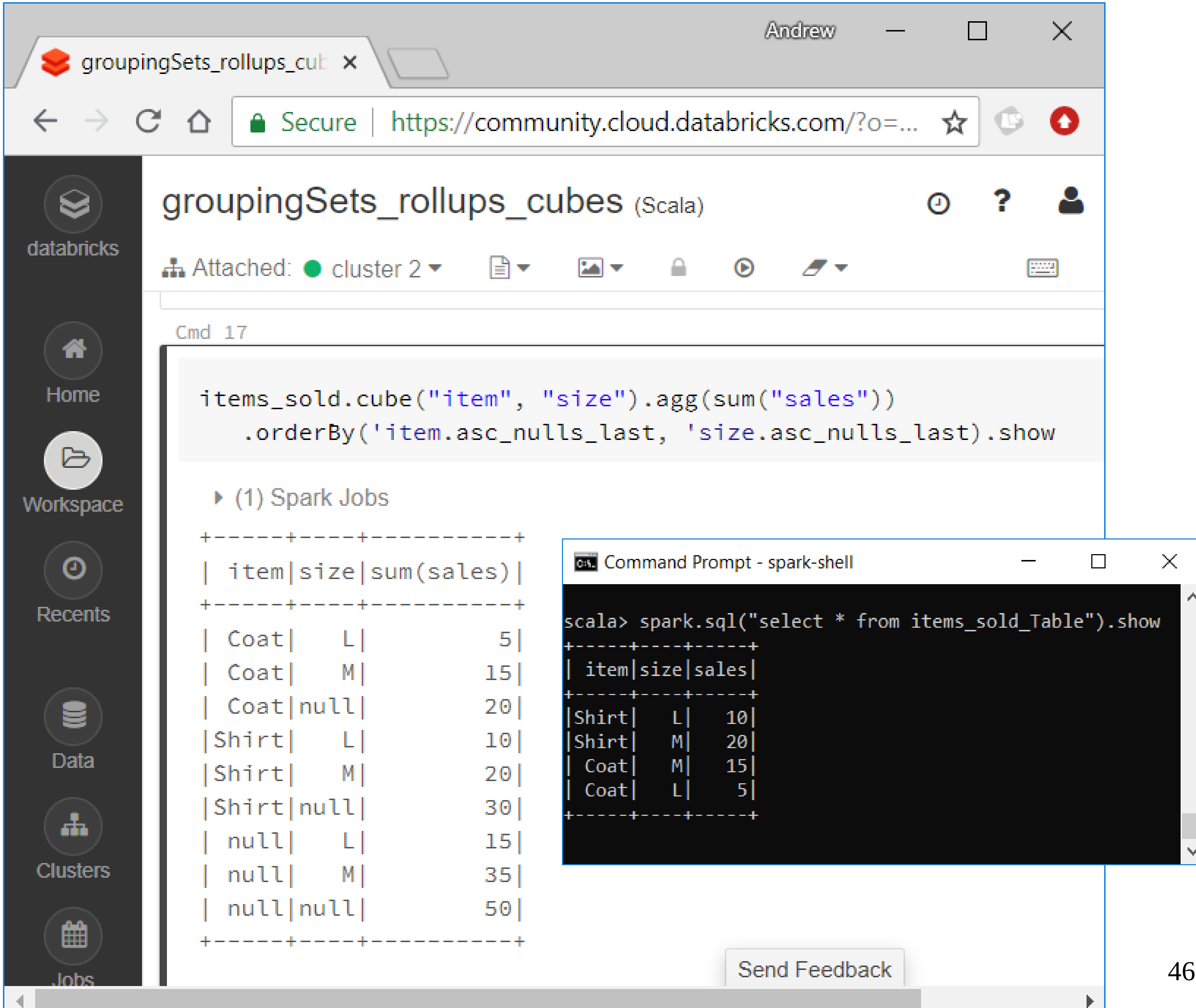
```
GROUPING SETS(
(warehouse, product),
(warehouse),
(product),
())
)
```

The N elements of a CUBE specification results in 2^N GROUPING SETS

Cubes

- A cube is a more advanced version of a rollup
- It performs the aggregations across all the possible combinations of the grouping columns
- Therefore, the result includes what a rollup provides as well as other combinations.


groupingSets_rollups_cul
Andrew
Secure
https://community.cloud.databricks.com/?o=...
databricks
Home
Workspace
Recents
Data
Clusters
Jobs
groupingSets_rollups_cubes (Scala)
Attached: cluster 2
Cmd 17
items_sold.cube("item", "size").agg(sum("sales"))
  .orderBy('item.asc_nulls_last, 'size.asc_nulls_last).show
Command Prompt - spark-shell
scala> spark.sql("select * from items_sold_Table").show
+-----+----+-----+
| item|size|sales|
+-----+----+-----+
|Shirt| L| 10|
|Shirt| M| 20|
| Coat| M| 15|
| Coat| L| 5|
+-----+----+-----+
Send Feedback


groupingSets_rollups_cub
Andrew
Secure
https://community.cloud.databricks.com/?o=...
databricks
Home
Workspace
Recents
Data
Clusters
Jobs
groupingSets_rollups_cubes (Scala)
Attached: cluster 2
Cmd 17
items_sold.cube("item", "size").agg(sum("sales"))
.orderBy('item.asc_nulls_last, 'size.asc_nulls_last).show
(1) Spark Jobs
+-----+----+----------+
| item|size|sum(sales)|
+-----+----+----------+
Coat	L	5
Coat	M	15
Coat	null	20
Shirt	L	10
Shirt	M	20
Shirt	null	30
null	L	15
null	M	35
null	null	50
+-----+----+----------+
Command Prompt - spark-shell
scala> spark.sql("select * from items_sold_Table").show
+-----+----+-----+
| item|size|sales|
+-----+----+-----+
|Shirt| L| 10|
|Shirt| M| 20|
| Coat| M| 15|
| Coat| L| 5|
+-----+----+-----+
Send Feedback

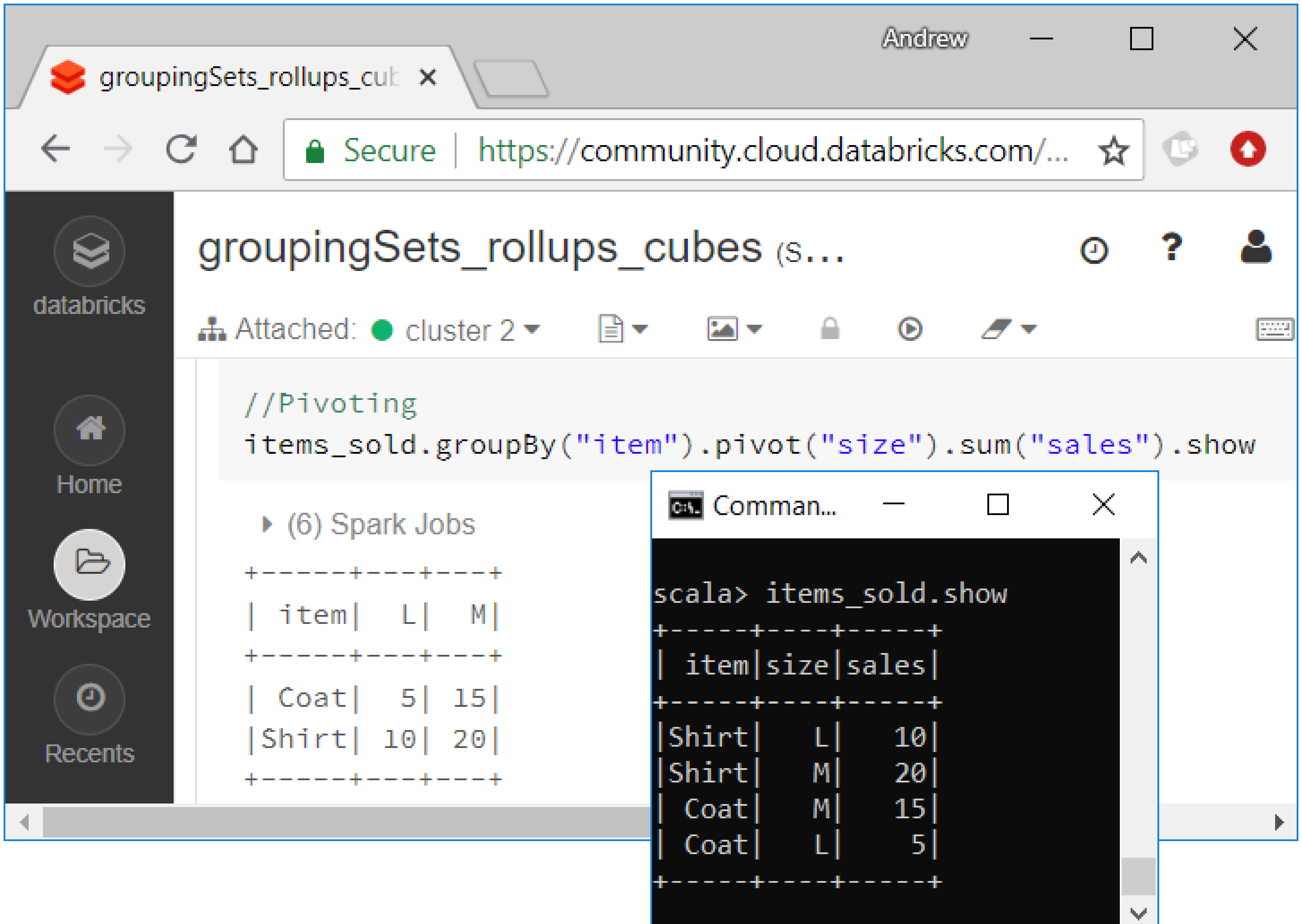
Pivot

- Pivoting is a technique to convert rows into columns
- to create a different view of a table
- Pivoting starts with grouping over one or more columns, then pivoting on a column, and ends with applying one or more aggregations on one or more columns

Pivot - Example

```
scala> items_sold.show
+-----+----+-----+
| item|size|sales|
+-----+----+-----+
|Shirt|   L|   10|
|Shirt|   M|   20|
| Coat|   M|   15|
| Coat|   L|    5|
+-----+----+-----+
```

Pivot - Example



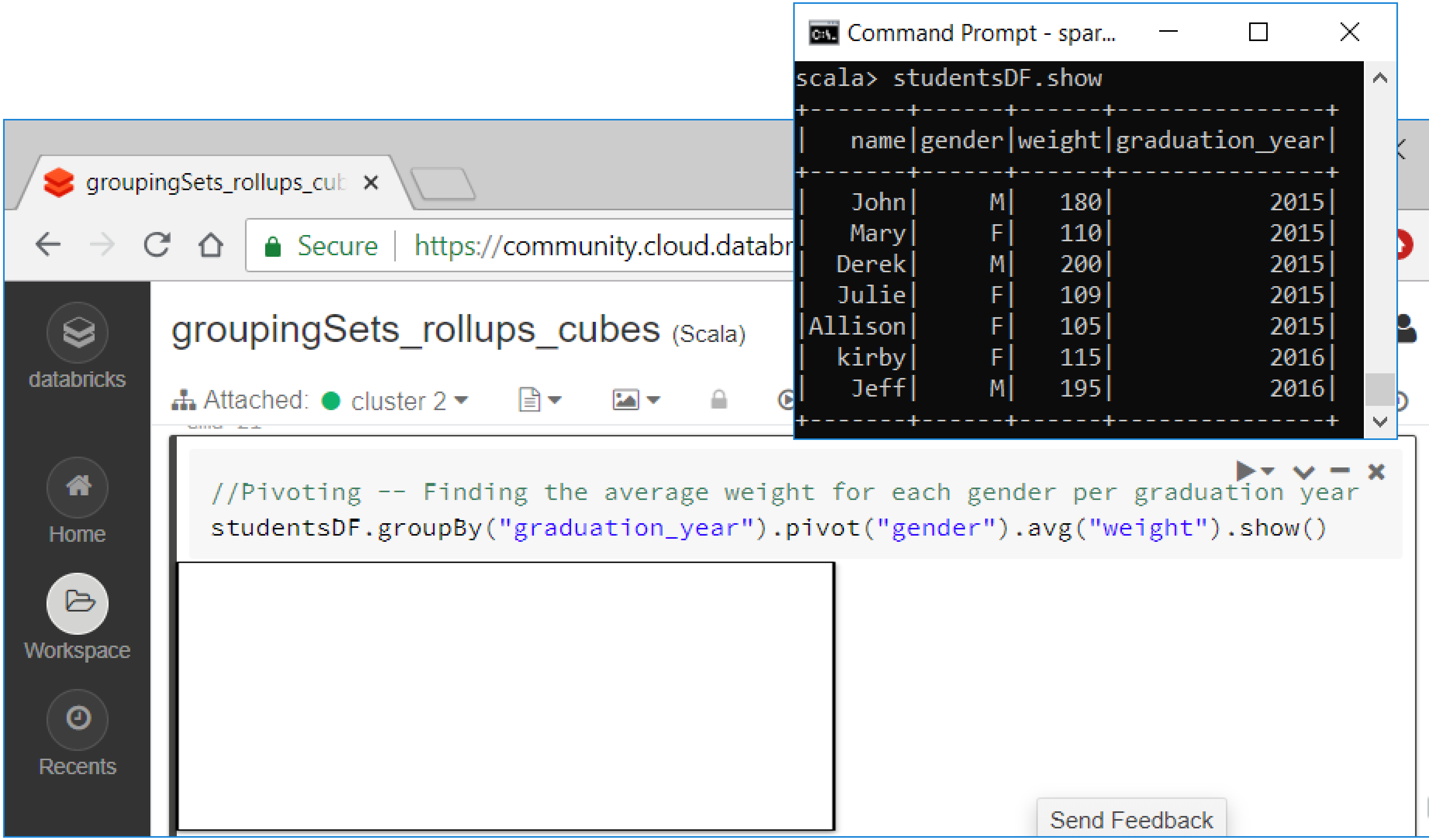
Pivot - Example

- ```
  case class Student(name:String, gender:String, weight:Int,
  graduation_year:Int)
  val studentsDF = Seq(Student("John", "M", 180, 2015),
                       Student("Mary", "F", 110, 2015),
                       Student("Derek", "M", 200, 2015),
                       Student("Julie", "F", 109, 2015),
                       Student("Allison", "F", 105, 2015),
                       Student("kirby", "F", 115, 2016),
                       Student("Jeff", "M", 195, 2016)).toDF
  ```

Command Prompt - spar...

```
scala> studentsDF.show
+-------+------+------+---------------+
|   name|gender|weight|graduation_year|
+-------+------+------+---------------+
|   John|     M|   180|           2015|
|   Mary|     F|   110|           2015|
|  Derek|     M|   200|           2015|
|  Julie|     F|   109|           2015|
|Allison|     F|   105|           2015|
|  kirby|     F|   115|           2016|
|   Jeff|     M|   195|           2016|
+-------+------+------+---------------+
```
```

Pivot - Example



Pivot - Example

Command Prompt - spar...

```
scala> studentsDF.show
+-------+------+------+---------------+
|   name|gender|weight|graduation_year|
+-------+------+------+---------------+
|   John|     M|   180|           2015|
|   Mary|     F|   110|           2015|
|  Derek|     M|   200|           2015|
|  Julie|     F|   109|           2015|
|Allison|     F|   105|           2015|
|  kirby|     F|   115|           2016|
|   Jeff|     M|   195|           2016|
+-------+------+------+---------------+
```

groupingSets_rollups_cub ×

Secure | https://community.cloud.databr

databricks

Home

Workspace

Recents

groupingSets_rollups_cubes (Scala)

Attached: cluster 2

```
//Pivoting -- Finding the average weight for each gender per graduation year
studentsDF.groupBy("graduation_year").pivot("gender").avg("weight").show()
```

(6) Spark Jobs

```
+---------------+-----+-----+
|graduation_year|    F|    M|
+---------------+-----+-----+
|           2015|108.0|190.0|
|           2016|115.0|195.0|
+---------------+-----+-----+
```

Send Feedback