



Secondo giorno: pomeriggio

Agenda 4/8

— Aggregazioni

Tipi di aggregazione

Intero DataFrame, *group by*, *window*, *grouping set*, *rollup*, *cube*

Statistiche

Conteggi, somme, media, varianza, ecc.

Aggregazione su tipi complessi

Insiemi e liste

Window

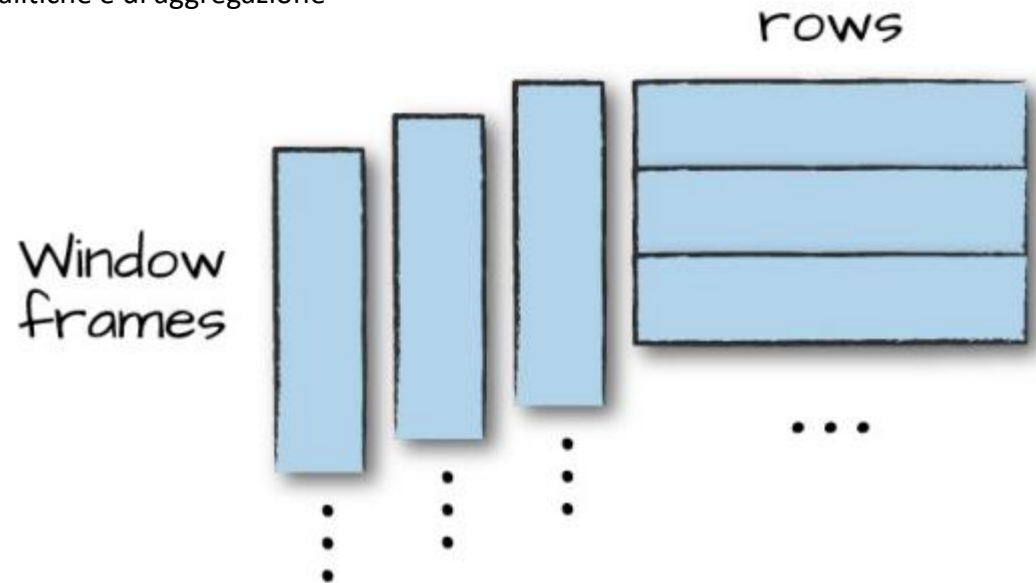
Funzioni su finestre di dati: ranking, funzioni analitiche e di aggregazione

Funzioni di aggregazione definite dall'utente

Esercizi di laboratorio

Esercitazioni sugli argomenti trattati.

$$\left(\sum_{i=1}^n p_i x_i^2 \right) - \mu^2$$



Aggregazioni

La pietra angolare dell'analisi di big data

Nelle aggregazioni bisogna specificare:

- Una **chiave** o un **raggruppamento**
- Una **funzione** di aggregazione che agisce su una o più colonne.

La funzione di aggregazione genera un risultato per ogni Gruppo.

In genere la funzione è numerica (ma non sempre):

- somma,
- prodotto,
- un semplice conteggio,
- Ecc.

Aggregazioni

La pietra angolare dell'analisi di big data

Spark consente diversi tipi di raggruppamenti:

- L'intero *DataFrame* (ad es. per contare il numero di righe)
- “*group by*” : raggruppa in base al valore di una o più colonne
- “*window*” : calcola un valore a partire da un certo insieme di record collegati
- “*grouping set*” : calcola un'aggregazione a più livelli:
 - “*rollup*” : permette di definire una gerarchia di aggregazioni
 - “*cube*” : calcola le aggregazioni con qualunque combinazione di colonne

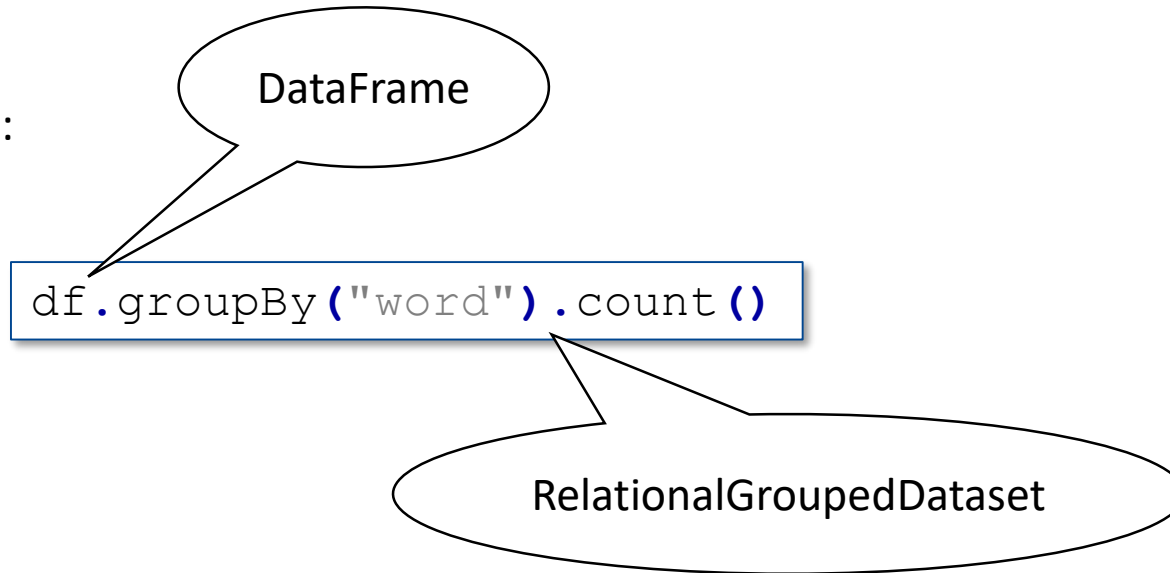
Aggregazioni

La pietra angolare dell'analisi di big data

Dal punto di vista sintattico:

DataFrame => RelationalGroupedDataset => DataFrame (con aggregazione).

Es.:



Aggregazioni

```
earthquakes.show(5)
```

date	city	richter
1966-08-19	mus	4.7
1966-08-19	mus	4.7
1966-12-30	sakarya	4.3
1967-05-22	mugla	4.6
1974-01-26	burdur	4.0

only showing top 5 rows

Took: 0.893s, at 2019-03-05 03:37

```
earthquakes
  .groupBy("richter")
  .count()
  .orderBy(desc("count"))
  .show(5)
```

richter	count
4.6	227
4.9	136
4.3	112
4.2	100
4.1	99

only showing top 5 rows

Took: 1.618s, at 2019-03-05 03:38

Aggregazioni

La pietra angolare dell'analisi di big data

Spesso sono disponibili delle funzioni che restituiscono un risultato approssimato, molto piu' rapidamente delle corrispondenti funzioni esatte.

Tutte le aggregazioni sono disponibili come funzioni nel package

org.apache.spark.sql.functions

Count

`df.count()` : conta il numero di record. È un'azione!

`df.select(count(«word»))`

`df.select(count(«*»))`

`df.select(countDistinct(«word»))`

`df.select(approx_count_distinct(«word», 0.1))`

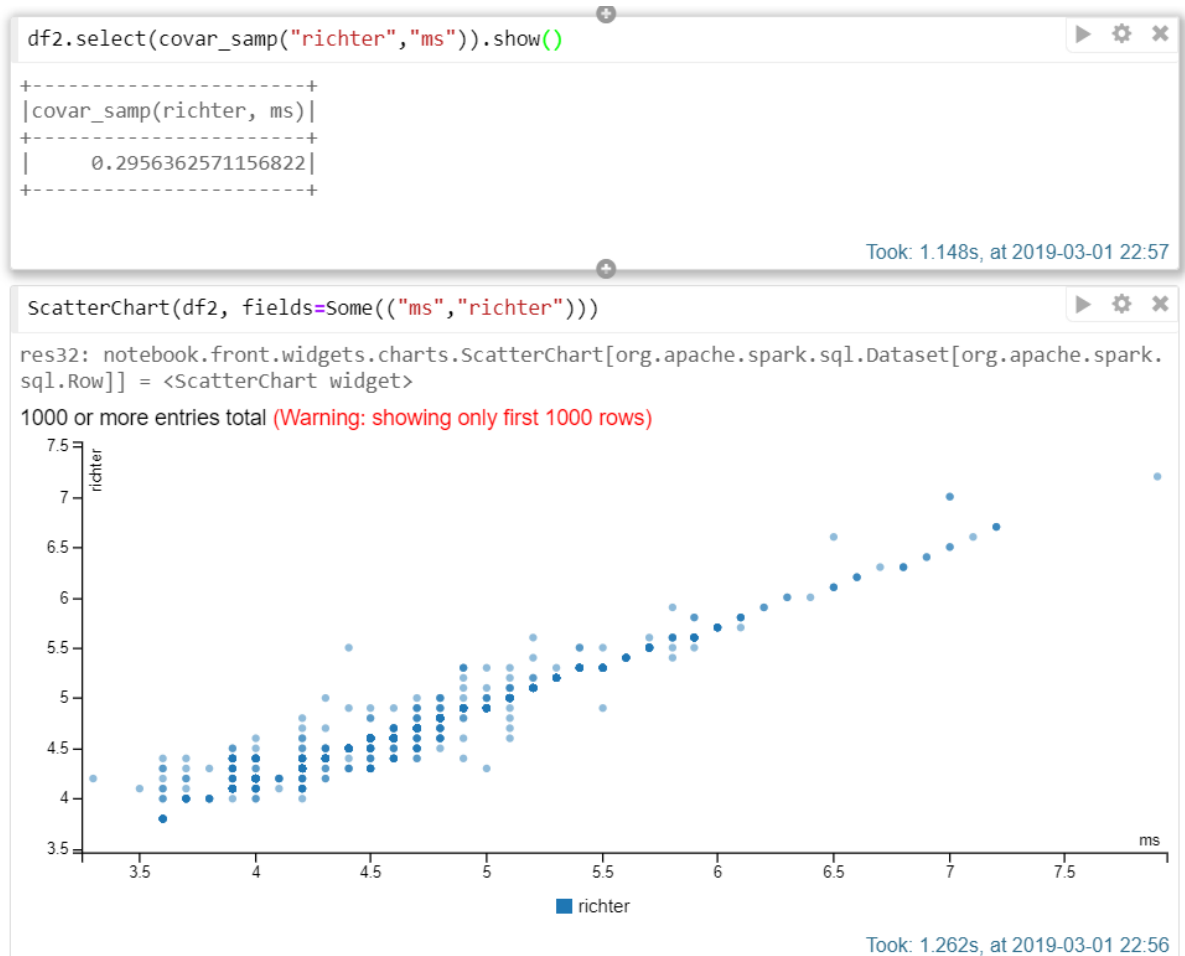
«Raggruppamenti» particolari

`first()`, `last()`, `min()`, `max()`, `take()`

Non generano dataframe: sono azioni che restituiscono oggetti Scala.

Statistiche

- Somma, sumDistinct
- Media, varianza,
- Deviazione standard
- Skewness, Kurtosis
- Covarianza



Un esempio con la somma

Suppongo di avere un dataframe con il numero di terremoti per città e per intensità

```
val df1 = earthquakes
  .filter("richter>4.5")
  .groupBy("richter", "city")
  .count()
  .orderBy($"city", desc("richter"))
df1.show(10)
```

```
+-----+-----+-----+
|richter|    city|count|
+-----+-----+-----+
|    5.7|   adana|    1|
|    5.4|   adana|    1|
|    4.9|   adana|    1|
|    4.8|   adana|    1|
|    4.6|   adana|    2|
|    5.7| adiyaman|    1|
|    5.0| adiyaman|    1|
|    4.9| adiyaman|    2|
|    4.7| adiyaman|    1|
|    4.6| adiyaman|    5|
+-----+-----+-----+
only showing top 10 rows
```

Un esempio con la somma

Voglio ottenere il conteggio totale per ogni città

```
df1
  .groupBy("city")
  .agg(sum("count").as("count"))
  .orderBy(desc("count"))
  .show(10)
```

```
+-----+-----+
|      city|count|
+-----+-----+
|   kutahya|   65|
|   burdur |   49|
|   mugla  |   49|
|    van   |   37|
|  denizli |   35|
|  erzurum |   32|
|  sakarya |   31|
|   manisa |   29|
|   bingol |   29|
| balikesir|   28|
+-----+-----+
```

only showing top 10 rows

Aggregazione su tipi complessi

In Spark le aggregazioni possono produrre tipi complessi, come ad es. liste di valori:

```
import scala.collection.mutable

val cities = df1
  .agg(collect_set("city"))
  .first()
  .getAs[mutable.WrappedArray[String]](0)
```

```
import scala.collection.mutable
cities: scala.collection.mutable.WrappedArray[String] = WrappedArray(
(bitlis, amasya, bartin, isparta, hakkari, nevsehir, artvin, tunceli,
aksaray, bayburt, kocaeli, malatya, mus, van, erzurum, mugla, kirsehi
r, ankara, bilecik, mersin, edirne, bingol, kirikkale, denizli, aydi
n, giresun, mardin, elazig, sakarya, burdur, eskisehir, tekirdag, sii
rt, ardahan, karabuk, batman, istanbul, hatay, agri, bursa, duzce, ka
rs, usak, manisa, balikesir, cankiri, ordu, sanliurfa, sivas, adiyama
n, antalya, corum, gaziantep, afyonkarahisar, erzincan, tokat, yozga
t, diyarbakir, adana, izmir, osmaniye, kutahya, kirkclareli, kastamon
u, konya, bolu, sirnak, canakkale, sinop, kayseri, samsun, trabzon, i
gdir, kahramanmaras)
```

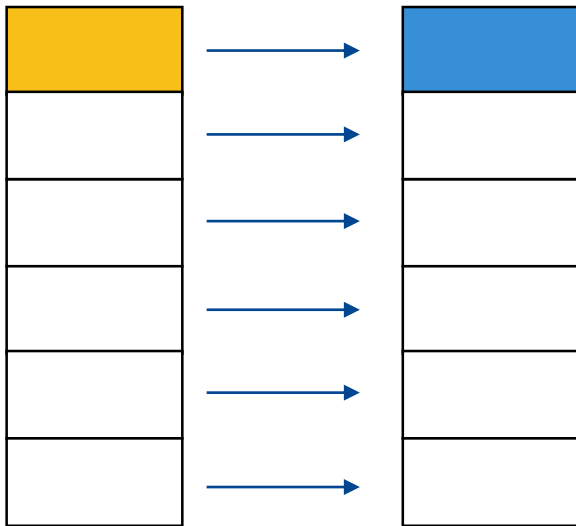
Took: 10.529s, at 2019-03-05 04:05

Window Functions

Fin qui abbiamo visto due tipi di funzioni:

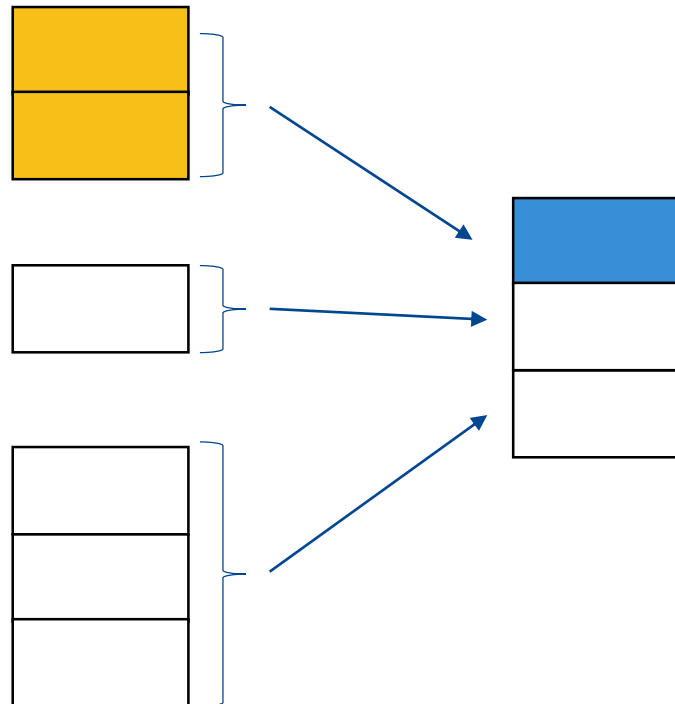
substr(), to_date(), ecc.

Un valore per riga; input = riga



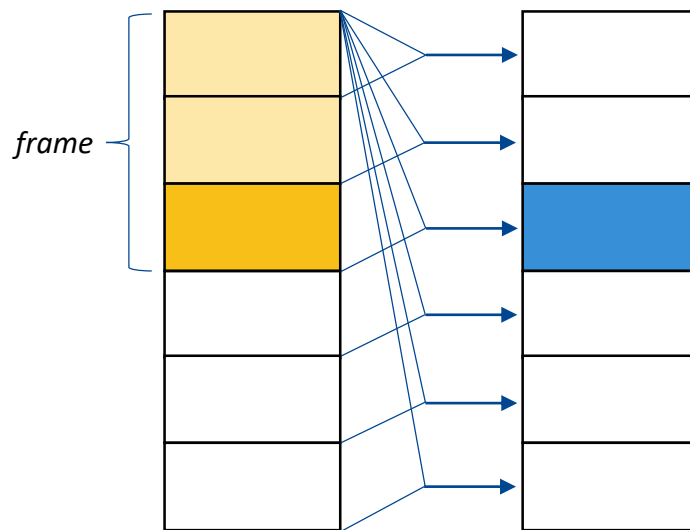
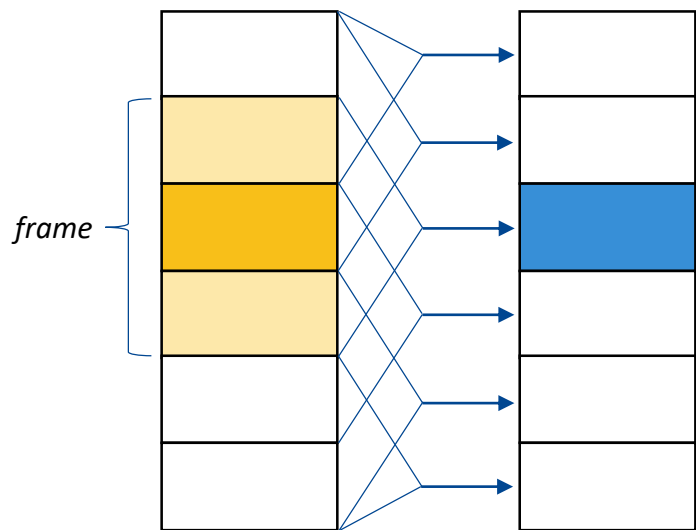
count(), sum(), max(), ecc.

Un valore per gruppo; input = gruppo



Window Functions

Una «**window function**» calcola un valore per ogni riga di un *DataFrame*, basandosi su un gruppo di righe chiamato «**frame**».



Window Functions

Le *Window Functions* possono essere usate per:

Calcolare una ***media mobile***

Calcolare una ***somma cumulativa***

Spark supporta tre tipi di *window function*:

- **Ranking** (*rank()*, *ntile()*, *rowNumber()*, ecc.)
- **Analitiche** (*cumeDist()*, *firstValue()*, *lag()*, ecc.)
- **Di aggregazione** (tutte le funzioni di aggregazione. Es. *sum()*, ecc.)

<https://databricks.com/blog/2015/07/15/introducing-window-functions-in-spark-sql.html>

Window Functions

In pratica:

```
import org.apache.spark.sql.expressions.Window

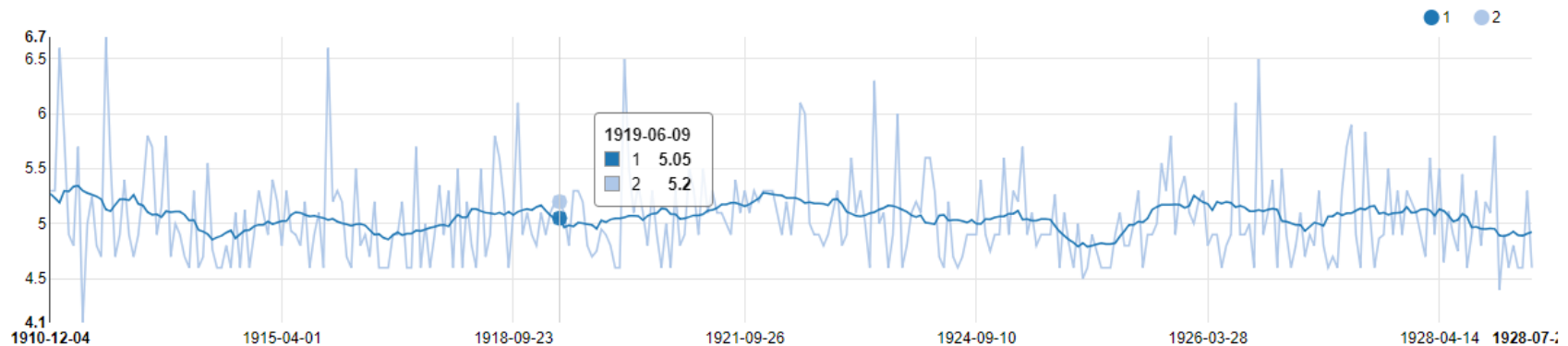
// definisco la finestra
val win = Window .partitionBy("city")
               .orderBy(desc("richter"))
               .rowsBetween(
                   Window.unboundedPreceding,
                   Window.currentRow)

// uso la window function
val rankedDf = df.select(
    $"city", $"date", $"richter",
    rank().over(win).as("rank"))
```

Window Functions

Ad esempio facciamo un filtro passabasso

```
import org.apache.spark.sql.expressions.Window
val df2 = df
  .where("richter>0")
  .withColumn("avg-richter",
    avg($"richter").over(Window
      .orderBy("date")
      .rowsBetween(-10,10)))
```



Window Functions

O elenchiamo i terremoti più forti per ogni città

```
import org.apache.spark.sql.expressions.Window
df.select($"city", $"date", $"richter",
          rank().over(Window
                        .partitionBy("city")
                        .orderBy(desc("richter"))
                        .rowsBetween(Window.unboundedPreceding, Window.currentRow))
          .as("rank"))
  .where("rank <= 2")
  .orderBy("city", "rank")
  .show()
```

city	date	richter	rank
adana	1945-03-20	5.7	1
adana	1952-10-22	5.4	2
adiyaman	1964-06-14	5.7	1
adiyaman	2008-09-03	5.1	2
afyonkarahisar	2002-02-03	5.6	1
afyonkarahisar	1924-11-20	5.6	1
agri	1941-09-10	5.6	1
agri	1936-05-01	5.4	2
aksaray	1924-12-13	4.9	1
aksaray	2011-06-13	3.9	2
amasya	1962-04-01	4.7	1
amasya	2003-09-27	4.2	2
amasya	2013-03-07	4.2	2
ankara	2007-12-20	5.7	1
ankara	2007-12-26	5.5	2
antalya	1969-01-14	6.0	1
antalya	1911-04-30	5.8	2
ardahan	1925-01-09	5.7	1
ardahan	2011-01-19	5.3	2
artvin	1934-11-02	4.7	1

only showing top 20 rows

```
import org.apache.spark.sql.expressions.Window
```

Took: 2.383s, at 2019-03-02 11:37

Rollups

Aggregazione multidimensionale

Se ad es. vogliamo contare

Quanti terremoti per città e intensità?

Quanti terremoti per città?

Quanti terremoti in tutto?

Possiamo farlo con un unico comando:

(ma attenzione ai null!)

```
df.na.drop().rollup($"city",$"richter").count.orderBy($"city", "richter").show
```

city	richter	count
null	null	1689
adana	null	14
adana	3.3	1
adana	3.5	1
adana	3.9	2
adana	4.1	1
adana	4.3	2
adana	4.5	1
adana	4.6	2
adana	4.8	1
adana	4.9	1
adana	5.4	1
adana	5.7	1
adiyaman	null	43
adiyaman	3.4	1
adiyaman	3.5	8
adiyaman	3.6	5
adiyaman	3.7	5
adiyaman	3.8	1
adiyaman	3.9	4

only showing top 20 rows

Took: 2.280s, at 2019-03-02 12:35

Cube

Aggregazione multidimensionale

Se ad es. vogliamo contare

Quanti terremoti per città e intensità?

Quanti terremoti per città?

Quanti terremoti per intensità?

Quanti terremoti in tutto?

Come l'esempio precedente, ma con `cube()` al posto di `rollup()`

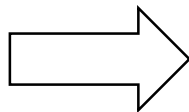
Pivot

I pivot permettono di trasformare righe in colonne e viceversa (da Spark 1.6)

Cos'è un PIVOT?

Raggruppa **A**, fai «perno» su **B** e somma **C**

A	B	C
G	X	1
G	Y	2
G	X	3
H	Y	4
H	Z	5



A	X	Y	Z
G	4	2	
H		4	5

```
df.groupBy("A").pivot("B").sum("C").show()
```

Funzioni di aggregazione definite dall'utente

UDAD (User-Defined Aggregation Functions)

Permettono di calcolare un valore a partire da un gruppo di record.

Bisogna creare una classe derivata da *UserDefinedAggregateFunction* e implementare:

- *inputSchema*
- *bufferSchema*
- *dataType*
- *deterministic*
- *initialize*
- *update*
- *merge*
- *evaluate*

Word count con i DataFrame

```
val text = spark.read.text("../data/divina_commedia.txt")
val cleanedText = text.withColumn("value", regexp_replace($"value", "[^a-zA-Z]", " "))
val words = cleanedText
  .select(explode(split($"value", "\\s+")) as "word")
  .filter("word != ''")
  .withColumn("word", lower($"word"))
words
  .filter(length($"word") > 8)
  .groupBy("word")
  .count()
  .orderBy(desc("count"))
```

text: org.apache.spark.sql.DataFrame = [value: string]

cleanedText: org.apache.spark.sql.DataFrame = [value: string]

words: org.apache.spark.sql.DataFrame = [word: string]

res9: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [word: string, count: bigint]

1 >>

Took: 6.169s, at 2019-03-05 09:19

word	count
"purgatorio"	35
"giustizia"	34
"principio"	26
"consiglio"	22
"maraviglia"	21
"cominciai"	20
"intelletto"	18
"qualunque"	17

