

API de DataFrames

- Funcionalidades similares a SQL
- Los DF funcionan como los RDDs: son inmutables y lazy
- No se pueden cambiar los datos de un DF directamente => hay que transformarlo en otro
- Son lazy porque no devuelven resultados, devuelven otro DF
- La API puede usarse para seleccionar, filtrar, agrupar, hacer join,

select

```
df = spark.read.json("gente.json")
```

```
df.show()
```

```
+---+---+
| age|  name|
+---+---+
|null|Michael|
| 30|   Andy|
| 19|  Justin|
+---+---+
```

```
df.select('age').show()
```

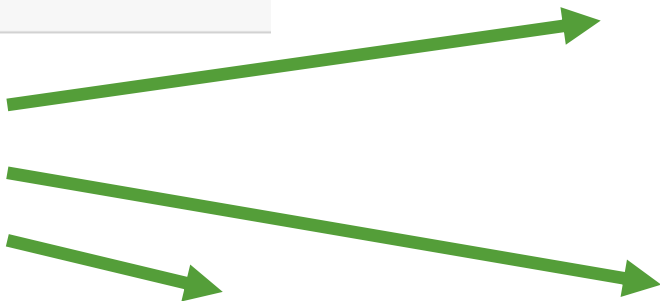
```
+---+
| age|
+---+
|null|
| 30|
| 30|
| 19|
+---+
```

```
df.select(df.age).show()
```

```
+---+
| age|
+---+
|null|
| 30|
| 19|
+---+
```

```
df.select(df["name"]).show()
```

```
+-----+
|  name|
+-----+
|Michael|
|   Andy|
|  Justin|
+-----+
```



drop

- Devuelve un DF donde no está la columna(s) especificada(s)

```
df.drop(df.age).show()
```

```
+-----+  
|  name |  
+-----+  
|Michael|  
|   Andy|  
|  Justin|  
+-----+
```

Filtrar datos: **where** y **filter**

```
df.filter(df.age > 20).show()
```

```
+---+-----+
|age|name|
+---+-----+
| 30|Andy|
+---+-----+
```

```
df.filter("age > 20").show()
```

```
+---+-----+
|age|name|
+---+-----+
| 30|Andy|
+---+-----+
```

```
df.where(df.age > 20).show()
```

```
+---+-----+
|age|name|
+---+-----+
| 30|Andy|
+---+-----+
```

```
df.where("age > 20").show()
```

```
+---+-----+
|age|name|
+---+-----+
| 30|Andy|
+---+-----+
```

Seleccionar las primeras n filas: **limit**

```
df.limit(1).show()
```

```
+----+-----+  
| age|   name|  
+----+-----+  
|null|Michael|  
+----+-----+
```

Añadir y renombrar columnas: `withColumnRenamed` y `withColumn`

```
df.withColumnRenamed('age', 'edad').show()
```

```
+----+-----+
|edad|  name|
+----+-----+
|null|Michael|
|  30|   Andy|
|  19|  Justin|
+----+-----+
```

```
df.withColumn('age2', df.age + 2).show()
```

```
+----+-----+----+
| age|  name|age2|
+----+-----+----+
|null|Michael|null|
|  30|   Andy|  32|
|  19|  Justin|  21|
+----+-----+----+
```

Ordenar: **orderBy**

```
df.sort(df.age.desc()).show()
```

age	name
30	Andy
19	Justin
null	Michael

```
df.sort(df.age).show()
```

age	name
null	Michael
19	Justin
30	Andy

**Añadimos nueva fila
en fichero json**

```
df.orderBy(["age", "name"], ascending=[0, 1]).show()
```

age	name
30	Andy
30	Mary
19	Justin
null	Michael

Funciones SQL para hacer cálculos sobre los datos

- **Funciones escalares**

- Devuelven un valor para cada fila

- **Funciones agregadas**

- Devuelven un valor para un grupo de filas

- **UDFs (User-defined functions)**

- Permiten extender la funcionalidad de Spark SQL

- **Funciones de ventanas**

- Devuelven varios valores para un grupo de filas

Funciones escalares

<code>abs, hypot, log, cbrt, ...</code>	Cálculos matemáticos
<code>length, trim, substring, ...</code>	Operaciones en cadenas
<code>year, date_add, ...</code>	Operaciones con fechas

```
from pyspark.sql.functions import *
```

```
df.select(cbrt('age')).show()
```

```
+-----+
|      CBRT(age) |
+-----+
|           null |
| 3.1072325059538586 |
| 3.1072325059538586 |
|  2.668401648721945 |
+-----+
```

Funciones agregadas

- min, max , count, avg (mean) y sum
- Se usan con groupBy o en el método select o withColumn

```
df.groupBy().avg('age').show()
```

```
+-----+  
|          avg(age) |  
+-----+  
|26.333333333333332|  
+-----+
```

- groupBy
 - agrupa por columnas
 - devuelve DataFrame

```
df.groupBy(['name', df.age]).count().show()
```

```
+-----+-----+-----+  
|   name|   age|count|  
+-----+-----+-----+  
|   Andy|   30|    1|  
|Michael| null|    1|  
|  Justin|   19|    1|  
+-----+-----+-----+
```

Funciones agregadas (II)

- La función `agg` permite aplicar funciones agregadas a todo el DF

```
df.agg({"age": "max"}).show()
```

```
+-----+  
|max(age)|  
+-----+  
|      30|  
+-----+
```

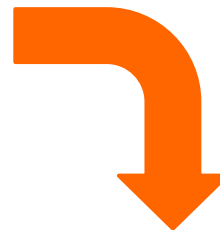
```
df.agg(min(df.age)).show()
```

```
+-----+  
|min(age)|  
+-----+  
|      19|  
+-----+
```

rollup y cube

- Dos funciones más para agrupamiento y agregación
- **groupBy** – calcula valores agregados para todas las combinaciones de valores en las columnas especificadas

coche	name	sexo
Renault	María	F
Seat	Juan	M
Seat	Pepa	F
Renault	Luis	M
Renault	Diego	M
Seat	Elisa	F



```
df.groupBy(df.coche,df.sexo).count().show()
```

coche	sexo	count
Seat	M	1
Renault	F	1
Renault	M	2
Seat	F	2

`rollup (a, b, c) => (a, b, c), (a, b), (a), ()`

coche	name	sexo
Renault	María	F
Seat	Juan	M
Seat	Pepa	F
Renault	Luis	M
Renault	Diego	M
Seat	Elisa	F



```
df.rollup(df.coche,df.sexo).count().show()
```

coche	sexo	count
Seat	null	3
Seat	F	2
null	null	6
Seat	M	1
Renault	null	3
Renault	F	1
Renault	M	2

$\text{cube}(a, b, c) \Rightarrow (a, b, c), (a, b), (b, c), (a, c), (a), (b), (c), ()$

coche	name	sexo
Renault	María	F
Seat	Juan	M
Seat	Pepa	F
Renault	Luis	M
Renault	Diego	M
Seat	Elisa	F



```
df.cube(df.coche,df.sexo).count().show()
```

coche	sexo	count
Seat	null	3
Seat	F	2
null	F	3
null	null	6
Seat	M	1
null	M	3
Renault	null	3
Renault	F	1
Renault	M	2

join

```
df2 = spark.read.json("gente_sex.json")
```

```
df2.show()
```

name	sex
Michael	M
Andy	M
Mary	F
Justin	M

inner, outer, left_outer, right_outer, leftsemi



por defecto

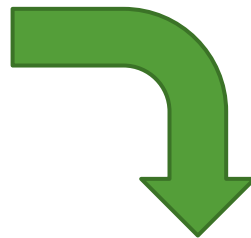
```
df.join(df2, df.name == df2.name, 'inner').drop(df.name).show()
```

age	name	sex
null	Michael	M
30	Andy	M
30	Mary	F
19	Justin	M

User-Defined Functions (UDFs)

- Permiten extender las funcionalidades de Spark SQL

book_id	temas
1	<policial>;<terror>
2	<aventuras>



temas	countTemas(temas)
<policial>;<terror>	2
<aventuras>	1

UDFs (II)

1. Importar módulo udf

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

2. Definir la función

```
def countTemas (s): return s.count("<")
```

3. Registrar la función como UDF

```
countTemas_udf = udf(countTemas)
```

4. Llamarla

```
df.select(df.temas,countTemas_udf(df.temas)).show()
```

Funciones de ventana

- A diferencia de las agregadas, no agrupan en una fila de salida por grupo
- Permiten definir un grupo móvil de filas (llamados marcos) relacionadas con la fila actual de alguna forma y que se usan para los cálculos sobre dicha fila.
- Ej: medias móviles o sumas acumulativas
 - \Rightarrow requieren subconsultas o joins complejos para calcularlas
- Cada fila tiene un marco único asociado

Primer ejemplo

productRevenue

product	category	revenue
Thin	Cell phone	6000
Normal	Tablet	1500
Mini	Tablet	5500
Ultra thin	Cell phone	5000
Very thin	Cell phone	6000
Big	Tablet	2500
Bendable	Cell phone	3000
Foldable	Cell phone	3000
Pro	Tablet	4500
Pro2	Tablet	6500

➤ Queremos contestar dos preguntas

1. ¿Cuáles son el primer y segundo producto con mas ingresos por categoría?
2. ¿Cuál es la diferencia entre los ingresos de cada producto y los ingresos del producto con mas ingresos en la misma categoría?

Primera pregunta

productRevenue

product	category	revenue
Thin	Cell phone	6000
Normal	Tablet	1500
Mini	Tablet	5500
Ultra thin	Cell phone	5000
Very thin	Cell phone	6000
Big	Tablet	2500
Bendable	Cell phone	3000
Foldable	Cell phone	3000
Pro	Tablet	4500
Pro2	Tablet	6500



product	category	revenue
Pro2	Tablet	6500
Mini	Tablet	5500
Thin	Cell Phone	6000
Very thin	Cell Phone	6000
Ultra thin	Cell Phone	5500

En SQL

```
SELECT
  product,
  category,
  revenue
FROM (
  SELECT
    product,
    category,
    revenue,
    dense_rank() OVER (PARTITION BY category ORDER BY revenue DES
FROM productRevenue) tmp
WHERE
  rank <= 2
```

category	product	revenue	rank
Cell phone	Thin	6000	1
Cell phone	Very thin	6000	1
Cell phone	Ultra thin	5000	2
Cell phone	Bendable	3000	3
Cell phone	Foldable	3000	3
Tablet	Pro2	6500	1
Tablet	Mini	5500	2
Tablet	Pro	4500	3
Tablet	Big	2500	4
Tablet	Normal	1500	5

En Spark SQL

- Paso 1. Definir una especificación de ventana
 - Define qué filas se incluyen en el marco asociado a una fila dada
 - Hay que especificar tres cosas
 - El particionamiento => filas incluidas en el marco
 - El ordenamiento => cómo se ordenan las filas en una partición
 - El marco => qué filas se incluyen en el marco asociado a una fila, basándose en su posición relativa con respecto a dicha fila. Ej: las 3 anteriores

```
windowSpec.rowsBetween(start, end)  
windowSpec.rangeBetween(start, end)
```

```
from pyspark.sql import Window
```

```
overCategory=Window.partitionBy(df.category).orderBy(df.revenue.desc())
```

En Spark SQL (II)

➤ Paso 2. Usar una función de ventana

	DataFrame API
Ranking functions	rank
	denseRank
	percentRank
	ntile
	rowNumber
Analytic functions	cumeDist
	firstValue
	lastValue
	lag
	lead

Table 5.1 Ranking and analytic functions that can be used as window functions

Function name	Description
<code>first (column)</code>	Returns the value in the first row in the frame.
<code>last (column)</code>	Returns the value in the last row in the frame.
<code>lag (column, offset, [default])</code>	Returns the value in the row that is offset rows behind the row in the frame. Use default if such a row doesn't exist.
<code>lead (column, offset, [default])</code>	Returns the value in the row that is offset rows before the row in the frame. Use default if such a row doesn't exist.
<code>ntile (n)</code>	Divides the frame into n parts and returns the part index of the row. If the number of rows in the frame isn't divisible by n and division gives a number between x and x+1, the resulting parts will contain x or x+1 rows, with parts containing x+1 rows coming first.
<code>cumeDist</code>	Calculates the fraction of rows in the frame whose value is less than or equal to the value in the row being processed.
<code>rank</code>	Returns the rank of the row in the frame (first, second, and so on). Rank is calculated by the value.
<code>denseRank</code>	Returns the rank of the row in the frame (first, second, and so on), but puts the rows with equal values in the same rank.
<code>percentRank</code>	Returns the rank of the row divided by the number of rows in the frame.
<code>rowNumber</code>	Returns the sequential number of the row in the frame.

➤ También funciones agregadas

En Spark SQL (III)

- Hay que indicar que la función se va a usar como función de ventana con el método `over`

```
ranked = df.withColumn("rank", dense_rank().over(overCategory))
```

```
ranked.show()
```

category	product	revenue	rank
Cell phone	Thin	6000	1
Cell phone	Very thin	6000	1
Cell phone	Ultra thin	5000	2
Cell phone	Bendable	3000	3
Cell phone	Foldable	3000	3
Tablet	Pro2	6500	1
Tablet	Mini	5500	2
Tablet	Pro	4500	3
Tablet	Big	2500	4
Tablet	Normal	1500	5

```
ranked.where(ranked.rank <= 2).show()
```

category	product	revenue	rank
Cell phone	Thin	6000	1
Cell phone	Very thin	6000	1
Cell phone	Ultra thin	5000	2
Tablet	Pro2	6500	1
Tablet	Mini	5500	2

Segunda pregunta

productRevenue

product	category	revenue
Thin	Cell phone	6000
Normal	Tablet	1500
Mini	Tablet	5500
Ultra thin	Cell phone	5000
Very thin	Cell phone	6000
Big	Tablet	2500
Bendable	Cell phone	3000
Foldable	Cell phone	3000
Pro	Tablet	4500
Pro2	Tablet	6500



product	category	revenue	revenue_difference
Pro2	Tablet	6500	0
Mini	Tablet	5500	1000
Pro	Tablet	4500	2000
Big	Tablet	2500	4000
Normal	Tablet	1500	5000
Thin	Cell Phone	6000	0
Very thin	Cell Phone	6000	0
Ultra thin	Cell Phone	5500	500
Foldable	Cell Phone	3000	3000
Bendable	Cell Phone	3000	3000

Segunda pregunta (II)

```
df.withColumn('reveDiff',max(df.revenue).over(overCategory) - df.revenue).show()
```

category	product	revenue	reveDiff
Cell phone	Thin	6000	0
Cell phone	Very thin	6000	0
Cell phone	Ultra thin	5000	1000
Cell phone	Bendable	3000	3000
Cell phone	Foldable	3000	3000
Tablet	Pro2	6500	0
Tablet	Mini	5500	1000
Tablet	Pro	4500	2000
Tablet	Big	2500	4000
Tablet	Normal	1500	5000

Missing values

- Los datos pueden contener campos vacíos o strings equivalentes “N/A”, null, ...
- Tres opciones para limpiarlos
 1. Eliminar filas que contienen null

```
df.show()
```

age	name
null	Michael
30	Andy
30	Mary
19	Justin



```
df.na.drop().show()
```

age	name
30	Andy
30	Mary
19	Justin

Drop("all") elimina filas que tienen NULL en todas las columnas



```
df.na.drop("all").show()
```

age	name
null	Michael
30	Andy
30	Mary
19	Justin

Missing values (II)

2. Rellenar valor null con constantes



```
df.na.fill(0).show()
```

age	name
0	Michael
30	Andy
30	Mary
19	Justin

3. Sustituir ciertos valores por otros



```
df.na.replace('Justin', 'Justine').show()
```

age	name
null	Michael
30	Andy
30	Mary
19	Justine

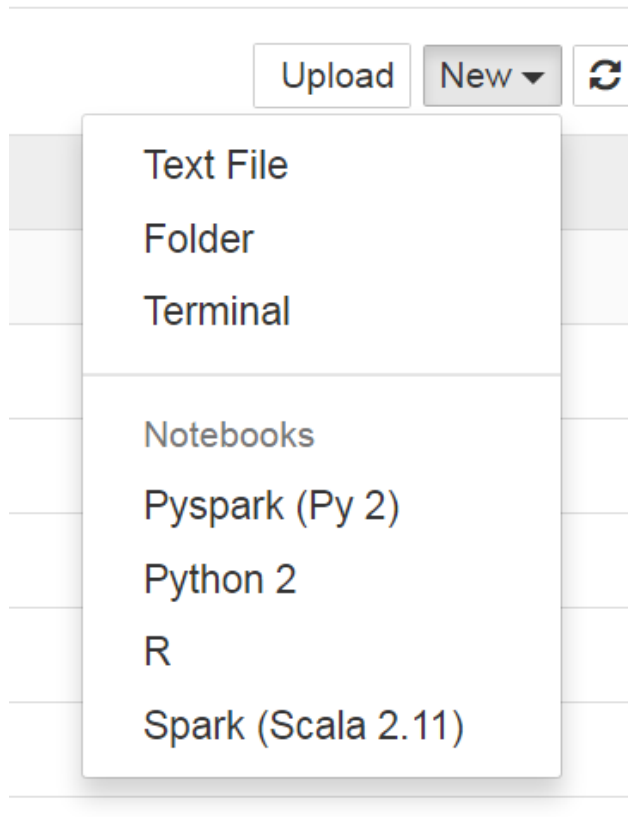
no específico para null values

La shell

- Arrancar una ventana de terminal



- `[vmuser@vm-ipnb-spark ~]$ spark-sql`



Bibliografía

- **Spark Cookbook.** Rishi Yadav. Packt Publishing. 2015
- Spark in action. Peter Zecevic, Marko Bonaci. Manning Publications. 2017.
- <https://jaceklaskowski.gitbooks.io/mastering-apache-spark/content/spark-sql.html>
- <https://databricks.com/blog/2015/07/15/introducing-window-functions-in-spark-sql.html>
- <https://docs.databricks.com/spark/latest/spark-sql/index.html>
- <https://www.qubole.com/resources/pyspark-cheatsheet/>