

Arquitecturas Cloud y Big Data



GOBIERNO
DE ESPAÑA

VICEPRESIDENCIA
PRIMERA DEL GOBIERNO
MINISTERIO
DE ASUNTOS ECONÓMICOS
Y TRANSFORMACIÓN DIGITAL

SECRETARÍA DE ESTADO
DE DIGITALIZACIÓN
E INTELIGENCIA ARTIFICIAL

red.es

Centro de
Referencia Nacional
en Comercio Electrónico
y Marketing

CRN
Digital





UNIÓN EUROPEA

Barrabés

The Valley

"El FSE invierte en tu futuro"
Fondo Social Europeo

Índice

1. **RDDs: Acciones**
2. **Ejercicio práctico: archivo de texto**
3. **Ejercicio práctico: archivo con valores numéricos**

4. **RDDs pares clave-valor (K,V). Transformaciones**

5. **Ejercicio: Transformaciones pares clave-valor (K,V)**

1. RDDs: Acciones



RDDs: Acciones

- Devuelven un valor (relativo al RDD)
- Desencadena la ejecución de toda la secuencia de transformaciones necesarias para obtener dicho valor.
- Ejecuta la "receta"


```
rdd2 = rdd1.flatMap(...).filter(...)  
  
print(rdd.count())
```

RDDs: Acciones más comunes

Acción	Descripción
<code>count()</code>	Devuelve el número de elementos del RDD
<code>reduce(func)</code>	Agrega los elementos del RDD usando <i>func</i>
<code>take(n)</code>	Devuelve una lista con los primeros n elementos del RDD
<code>collect()</code>	Devuelve una lista con todos los elementos del RDD
<code>takeOrdered(n[,key=func])</code>	Devuelve n elementos en orden ascendente. Opcionalmente se puede especificar la clave de ordenación

Acción “count”

- Devuelve el número de elementos del RDD



```
numeros = sc.parallelize([1,2,3,4,5,6,7,8,9,10])  
  
pares = numeros.filter(lambda elemento: elemento%2==0)  
  
pares.count()
```

RDD: numeros

```
[1, 2, 3,  
4, 5, 6,  
7, 8, 9,  
10]
```



RDD: pares

```
[2, 4,  
6, 8,  
10]
```

Acción “reduce”

- Agrega todos los elementos del RDD **por pares** hasta obtener un único valor (expresión **lambda**)

```
numeros = sc.parallelize([1,2,3,4,5])  
  
print(numeros.reduce(lambda elem1,elem2: elem1+elem2))
```

- Resultado: 15
- La función que se pasa a “**reduce**” debe:
 - Recibir dos argumentos y devolver uno **de tipo compatible**
 - Operación debe ser conmutativa y asociativa, de forma que se pueda calcular bien en paralelo (workers)

Acción “reduce”: otro ejemplo

```
palabras = sc.parallelize(['HOLA', 'Que', 'TAL', 'Bien'])  
  
pal_minus = palabras.map(lambda elemento: elemento.lower())  
  
print(pal_minus.reduce(lambda elem1,elem2: elem1+ "-" + elem2))
```

- Resultado: “hola-que-tal-bien”
- ¿La función de reducción cumple las condiciones?
 - No del todo. Aquí ha salido bien pero no es conmutativa
 - **Lo veremos mejor más adelante con “reduceByKey”**

Acción “reduce”: otra función de reducción

```
palabras2 = sc.parallelize(['Como', 'te', 'encuentras', 'hoy'])
```

- Función de reducción: palabra más larga (un poco más elaborada, **no nos valdría lambda, tenemos que crearla**):

```
def cadena_larga(elem1, elem2):  
  
    if len(elem1) >= len(elem2):  
        return elem1  
    else:  
        return elem2
```

```
palabras2.reduce(cadena_larga)
```

- **Resultado:** ‘encuentras’



Sólo el nombre
de la función

Acción “collect”

- Devuelve una lista con todos los elementos del RDD
- **¡¡OJO!!:** por debajo recupera todas las particiones del RDD de los nodos workers en los que se encuentre repartidos y lo lleva al nodo DRIVER == > POSIBLES PROBLEMAS DE MEMORIA
- Sólo se debe utilizar si estamos seguros del tamaño de lo que se va a recuperar

```
numeros = sc.parallelize([5,3,2,1,4])  
  
print(numeros.collect())
```

- Resultado: [5, 3, 2, 1, 4]

Acción “take”

- Devuelve una lista con los primeros "n" elementos del RDD
- Útil en aquellos casos que por tamaño el uso de “collect()” para recuperar contenido penaliza

```
numeros = sc.parallelize([5,3,2,1,4])  
  
print(numeros.take(3))
```

- Resultado: [5,3,2]

Acción “takeOrdered”

- Devuelve una lista con los primeros "n" elementos del RDD en orden ascendente

```
numeros = sc.parallelize([3,2,1,4,5])  
  
print(numeros.takeOrdered(3))
```

- Resultado: [1,2,3]

Acción “takeOrdered”: cambiar criterio ordenación

- Podemos alterar el criterio de ordenación ascendente
- Para ello debemos pasar una **función** como **parámetro** (función o “clave” de ordenación interna)
- Dicha función se utilizará a nivel interno, **devolviéndose los elementos originales correspondientes** (no sus “claves” internas), para ordenar según el criterio que queramos (ejplo descendente)

```
numeros = sc.parallelize([3,2,1,4,5])  
  
print(numeros.takeOrdered(3, lambda elem: -elem))
```

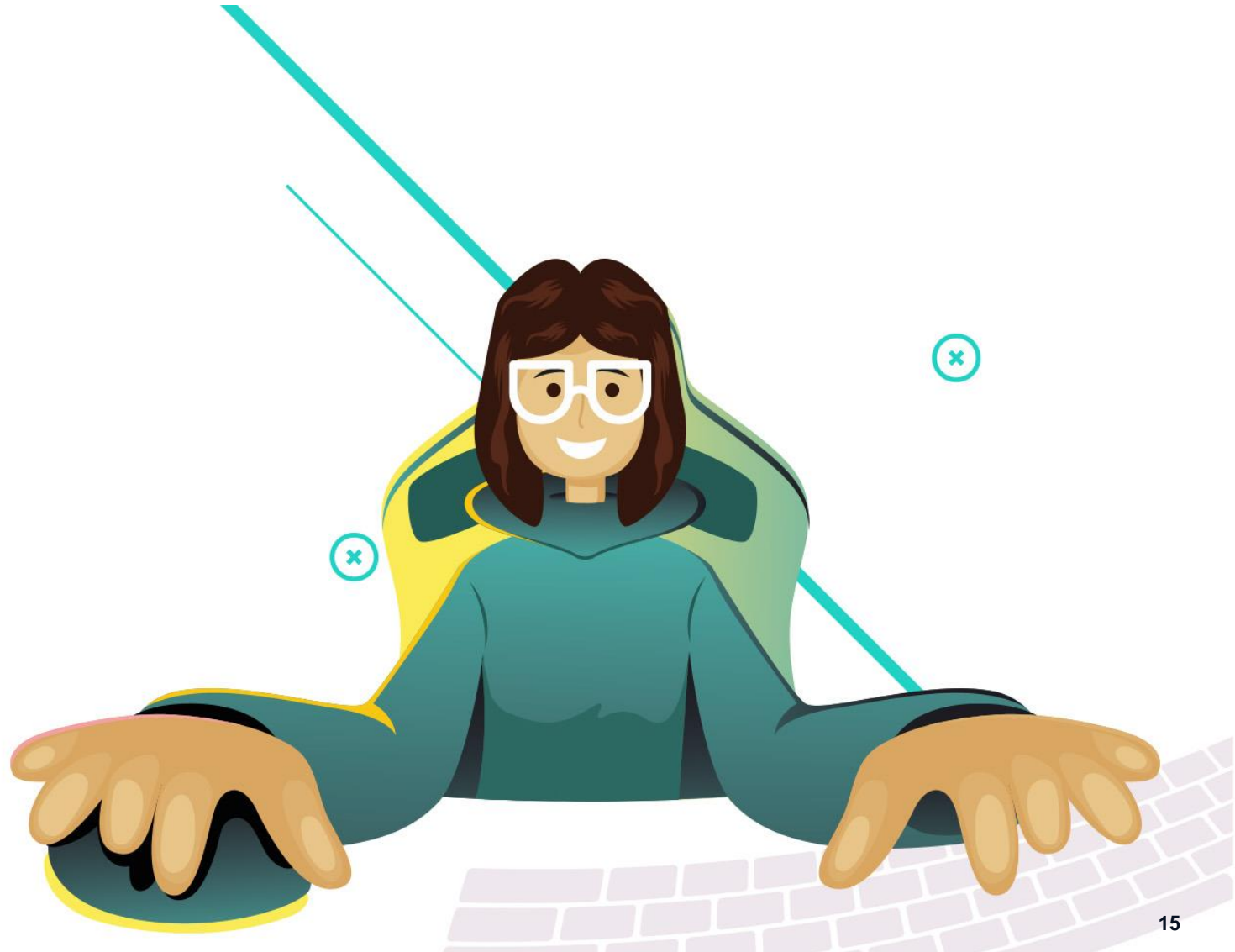
- Resultado: [5,4,3]
- ¿Cómo ordenarías para que primero aparezcan los pares ordenados y luego los impares?

2. EJERCICIO

PRÁCTICO: archivo de texto

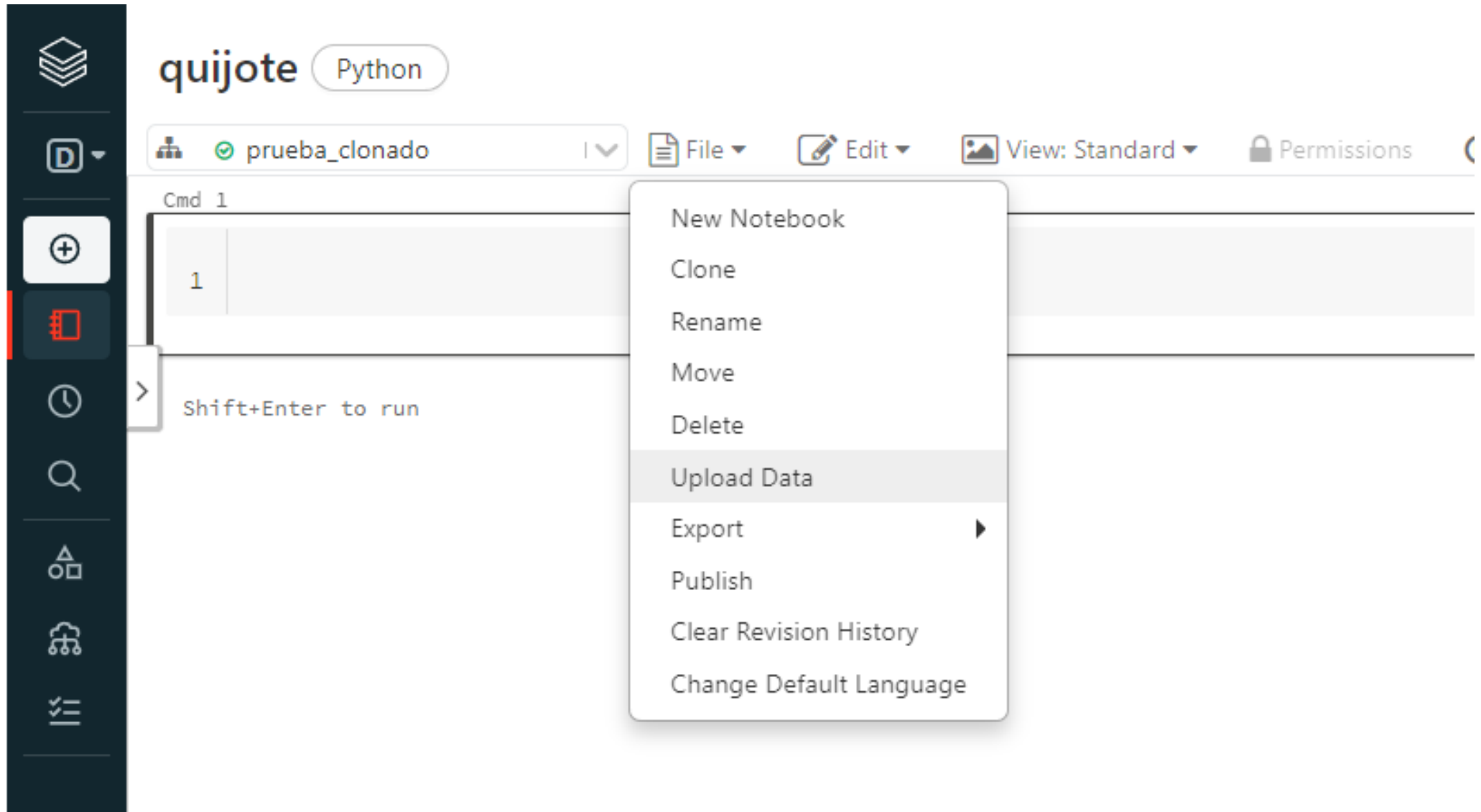


EJERCICIO 1: contar palabras de un fichero



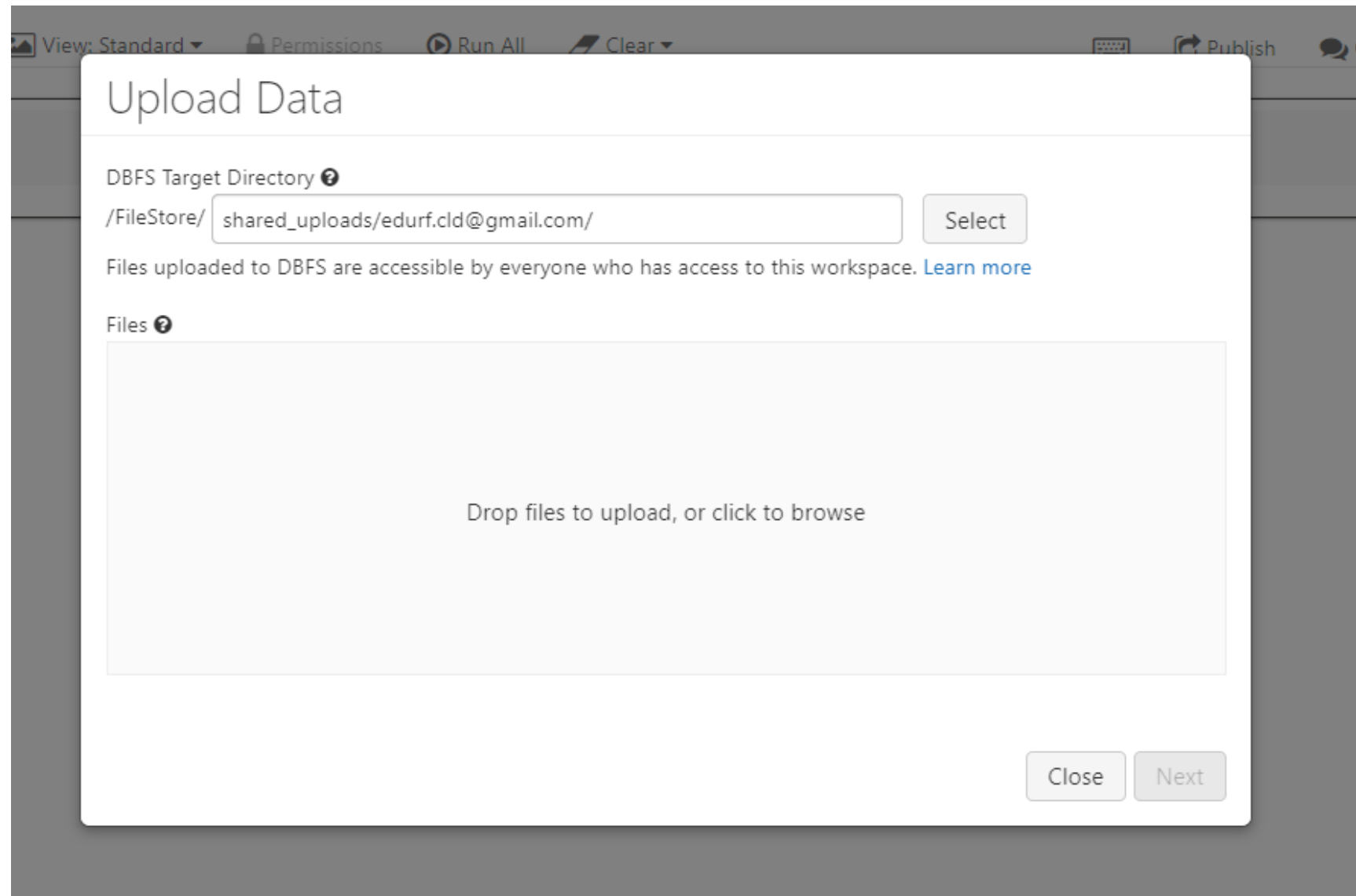
Cómo “subimos” un archivo de texto

- En el notebook nos vamos en el menú de arriba a “File”, desplegamos y picamos en “Upload Data”



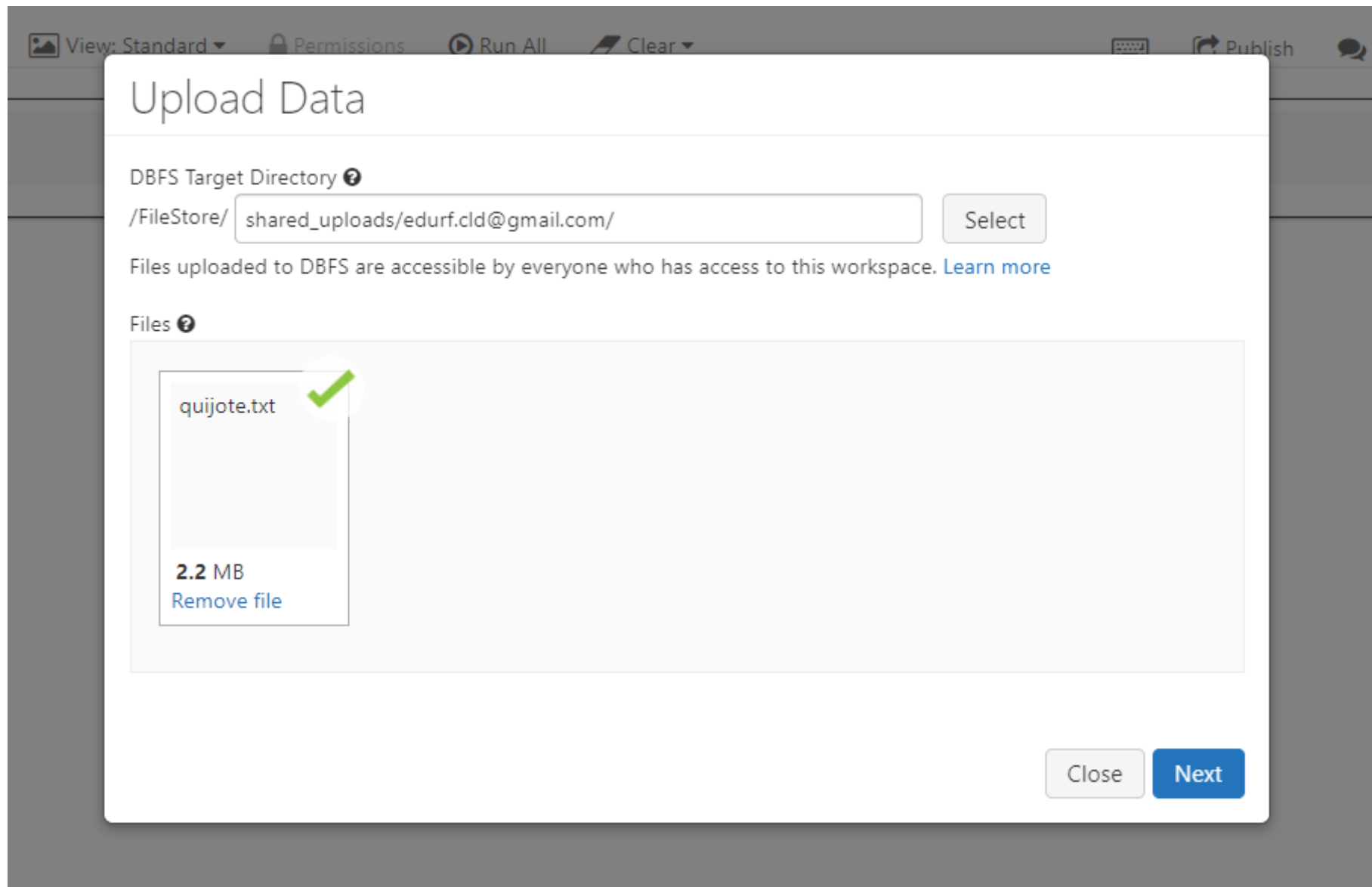
Cómo “subimos” un archivo de texto

- En la nueva ventana podemos clicar para buscar el archivo, o directamente arrastrar al recuadro de fondo gris el archivo.



Cómo “subimos” un archivo de texto

- Una vez subido (tarda según el tamaño del archivo, conexión) le damos al botón “**Next**”:



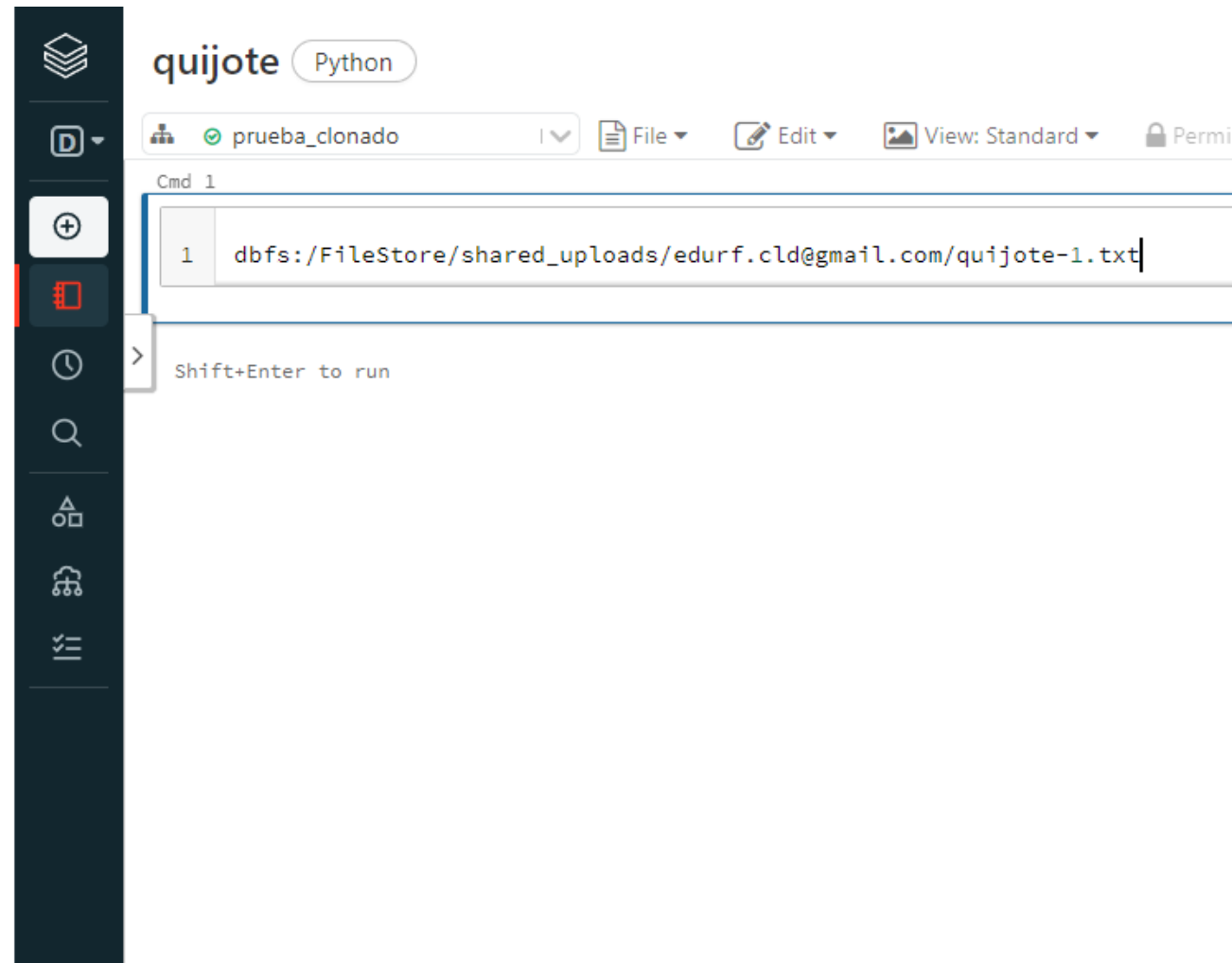
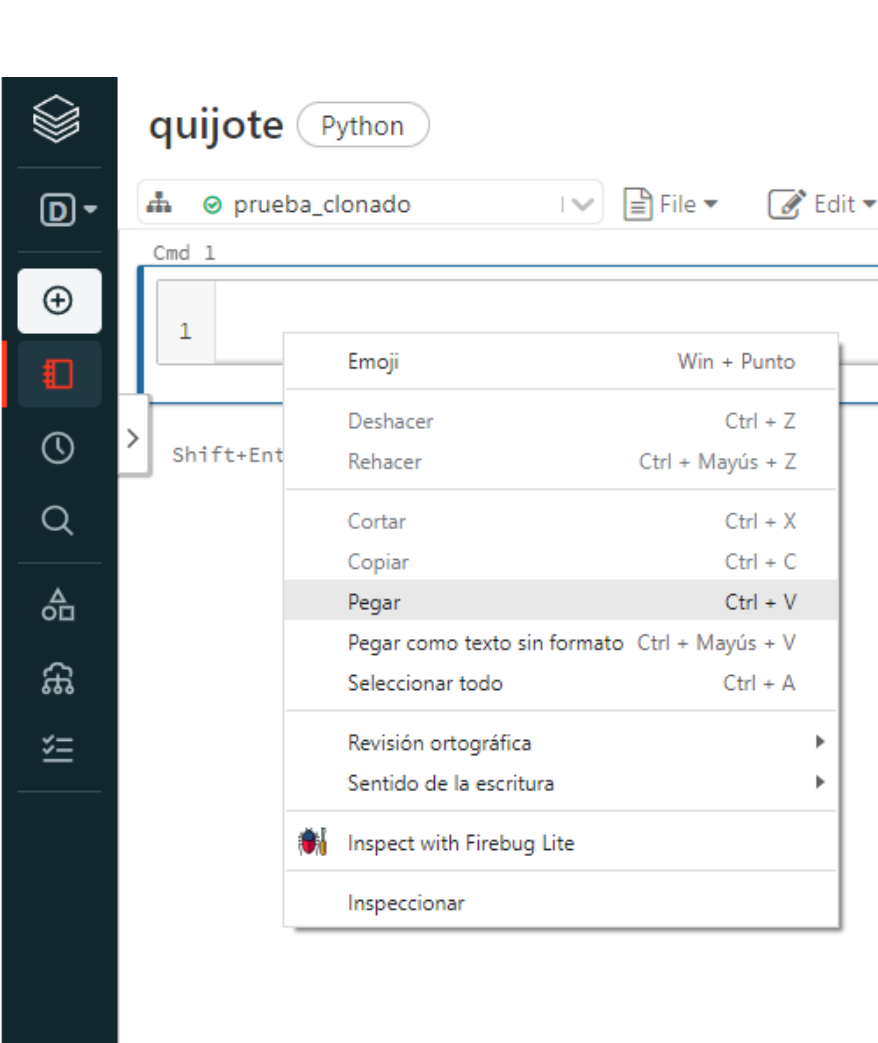
Cómo “subimos” un archivo de texto

- En esta nueva ventana solo clickamos en “**Copy**” en la esquina inferior derecha (resto opciones por defecto las dejamos) y finalmente a “**Done**”:



Cómo “subimos” un archivo de texto

- Volvemos al notebook y en una de las celdas copiamos el contenido. Ya tenemos la ruta del archivo.

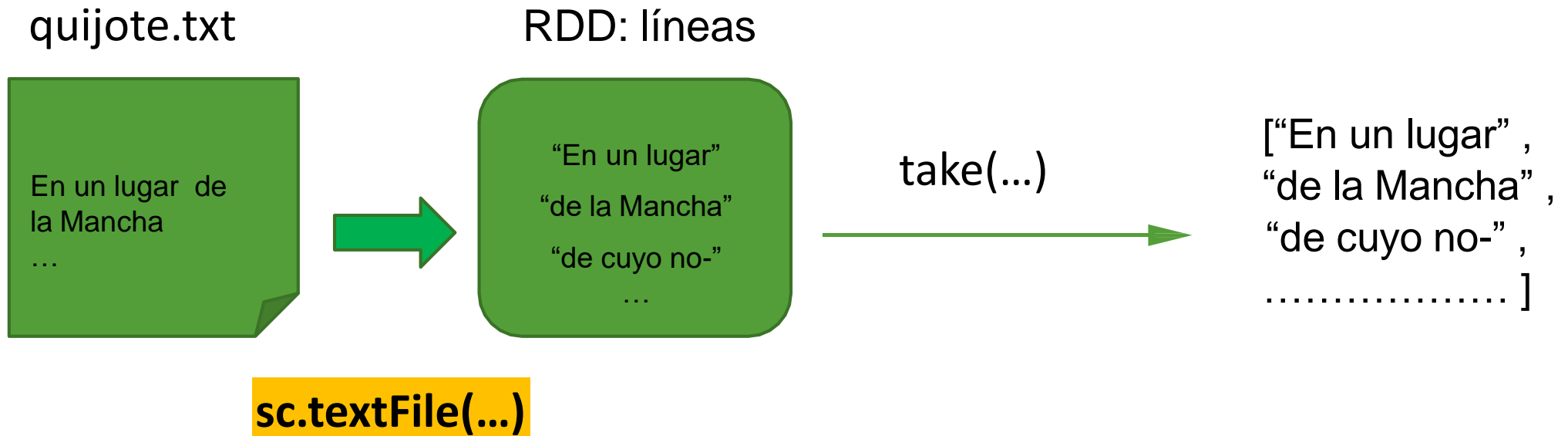


Ejercicio 1: Fase leer archivo de texto

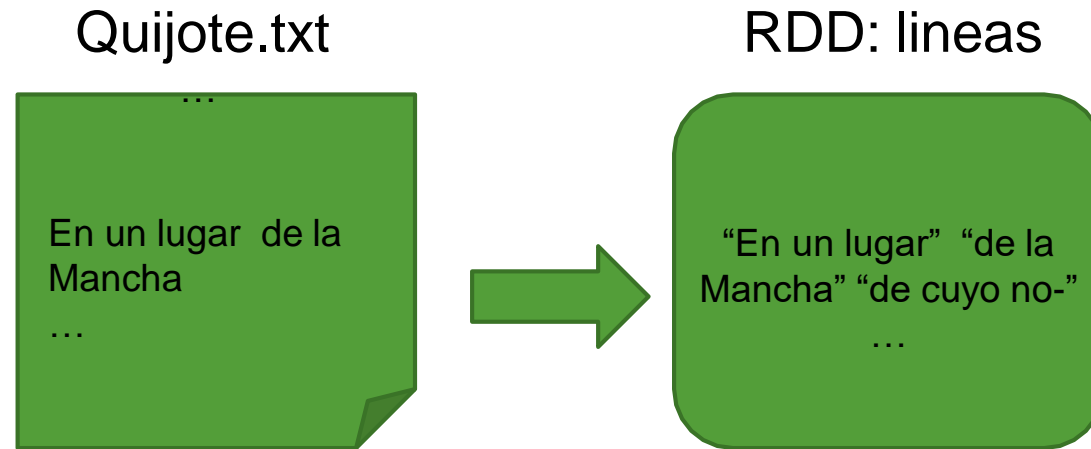
```
file = 'dbfs:/FileStore/shared_uploads/edurf.cld@gmail.com/quijote.txt'

lineas = sc.textFile(file)

lineas.take(5)
```



EJERCICIO 1: Contar palabras del fichero



- Posibles elementos a utilizar (no hay solución única):
 - `cadena.split('carácter')`: crea lista de subcadenas separadas por 'carácter' (si no se especifica, por defecto el carácter es un espacio)
 - `len(lista)`: devuelve nº de elementos de la lista
 - Acción `count()`

EJERCICIO 1: Contar palabras de un fichero

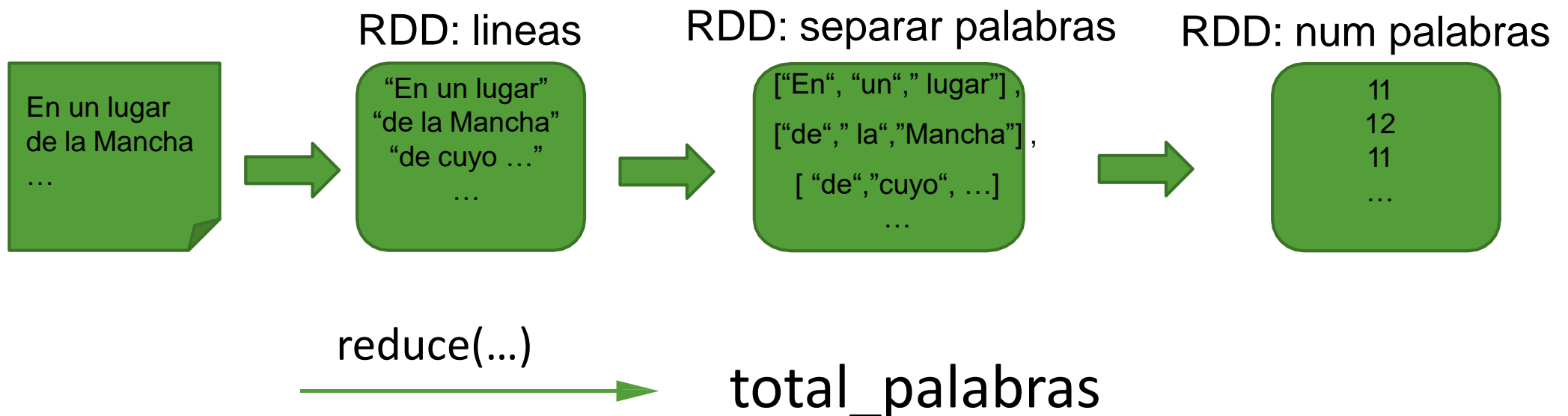
```
file = 'dbfs:/FileStore/shared_uploads/edurf.cld@gmail.com/quijote.txt'

lineas = sc.textFile(file)

separar_palabras = lineas.map(lambda elto: elto.split())

num_palabras = separar_palabras.map(lambda elemento: len(elemento))

total_palabras = num_palabras.reduce(lambda e1,e2: e1+e2)
```



3. EJERCICIO

PRÁCTICO: archivo con valores numéricos



EJERCICIO 2: Calcular la media de las medidas de sensores de humedad de un terreno (plantación)



Uso de sensores de humedad del suelo para efficientizar el riego

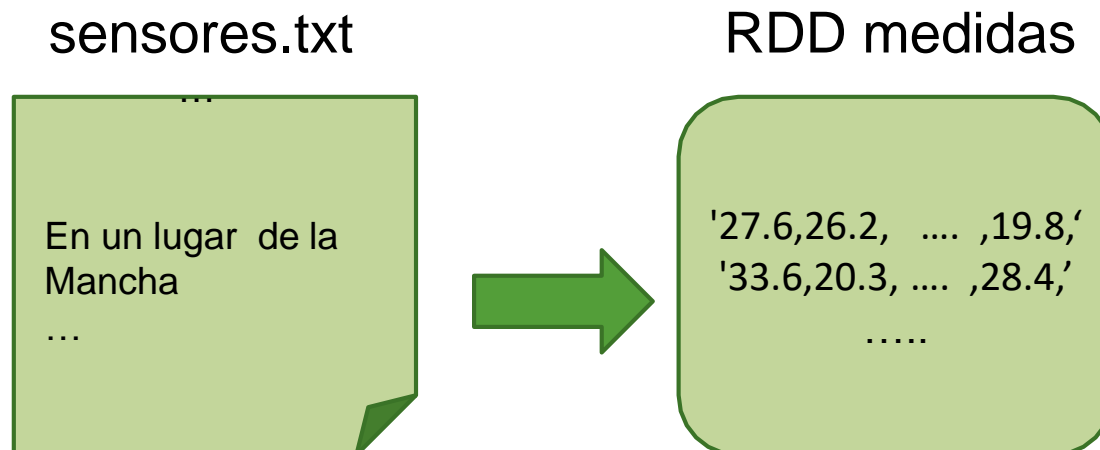
Juan M. Enciso, Dana Porter and Xavier Périès*

El monitorear el contenido de agua en el suelo es esencial para ayudar a los agricultores a optimizar la producción, conservar agua, reducir los impactos ambientales y ahorrar dinero. El monitorear la humedad del suelo le puede ayudar a tomar mejores decisiones en la programación del riego, tales como el determinar la cantidad de agua a aplicar y cuándo aplicarla. También le puede ayudar a igualar los requerimientos de agua del cultivo con la cantidad aplicada con el riego; y así evitar pérdidas de agua excesivas por percolación profunda o por escurrimientos o bien evitar aplicar una cantidad insuficiente. El exceso de irrigación puede incrementar el consumo de energía y los costos de agua, aumentar el movimiento de fertilizantes por debajo de la zona radicular, producir erosión y transporte de suelo y partículas de químicos a los canales de drenaje. El riego insuficiente puede reducir la producción de las cosechas.



EJERCICIO 2: Calcular la media de las medidas de sensores de humedad de un terreno (plantación)

- Desde un nuevo Notebook que crearemos (recomendable) vamos a subir el archivo “SENSORES.txt”, igual que hicimos en el ejercicio del Quijote (menu superior File -> Upload Data ...)
- Asignamos la ruta del archivo a una variable (file) y creamos el RDD con `sc.textFile(file)`
- ¡¡OJO!! % humedad → sensibilidad sensor: valor testigo -99,9



EJERCICIO 2: Calcular la media de las medidas de sensores de humedad de un terreno (plantación)

- Cada elemento del RDD “medidas” contiene una cadena de valores separados por coma (recordar leer archivo de texto, cada elemento RDD es una línea)
- Podemos separar las medidas de forma similar a como separamos las palabras en el Quijote. Obtenemos un RDD “separar_medidas” cuyos elementos son listas (corchetes) de los valores como cadena, separadas por comas.

```
separar_medidas = medidas.flatMap(lambda elto: elto.split(','))
```

```
separar_medidas.take(25)
```

RDD: medidas

'27.6,26.2, ,19.8,'
'33.6,20.3, ,28.4,'
.....



RDD:
separar_medidas

'27.6' , '26.2', , '19.8',
'33.6' , '20.3', , '28.4',
.....

EJERCICIO 2: Calcular la media de las medidas de sensores de humedad de un terreno (plantación)

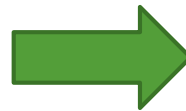
- Ahora lo que necesitamos son los valores en formato numérico (no como cadenas, texto), para poder sumarlos (reduce(...)):

```
medidas_num = separar_medidas.map(float)

medidas_num.take(15)
```

RDD separar_medidas

'27.6' , '26.2' , , '19.8',
'33.6' , '20.3' , , '28.4',
.....



RDD: medidas_num

27.6, 26.2, , 19.8,
33.6, 20.3, , 28.4,
.....

EJERCICIO 2: Calcular la media de las medidas de sensores de humedad de un terreno (plantación)

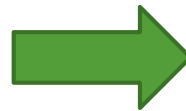
- Filtramos para eliminar los valores no válidos (valor testigo -99,9). Recordemos que hablamos de porcentaje de humedad:

```
final_medidas = medidas_num.filter(lambda elemento: 0<elemento<100)

final_medidas.take(15)
```

RDD: medidas_num

27.6,26.2, ,19.8,
33.6,-99.9, ,28.4,
.....



RDD: final_medidas

27.6,26.2, ,19.8,
33.6,.... , ,28.4,
.....

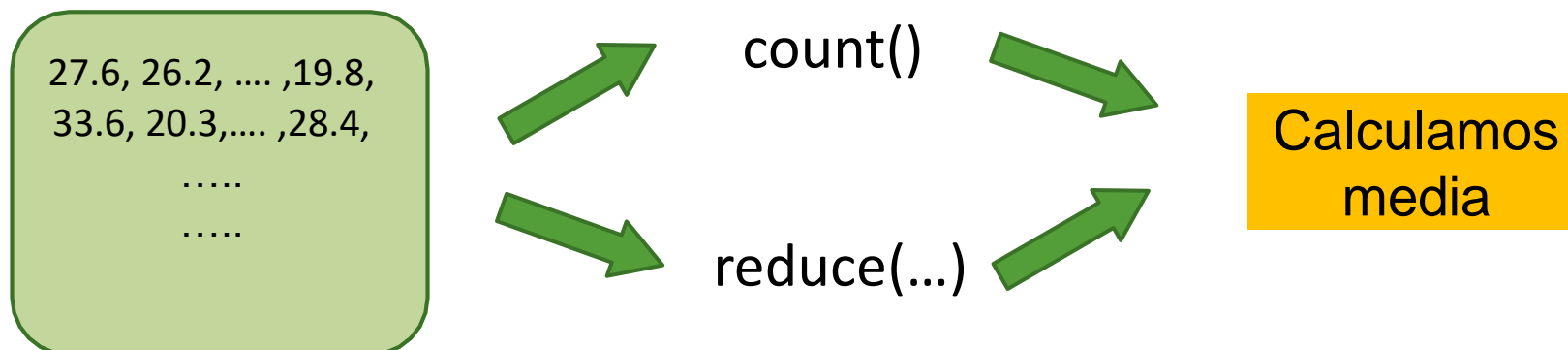
EJERCICIO 2: Calcular la media de las medidas de sensores de humedad de un terreno (plantación)

- Ahora ya tenemos un RDD cuyos elementos son valores numéricos, las medidas. Para calcular la media solo tenemos que contar cuantos valores hay, sumarlos (¿os acordais cómo lo hacíamos en un RDD?) y hacer una división

```
numero_medidas = final_medidas.count()
suma_total_medidas = final_medidas.reduce(lambda e1,e2: e1 + e2)

print('La media es: ', round(suma_total_medidas/numero_medidas, 1))
```

RDD: final_medidas



4. RDDs pares clave-valor (K, V): Transformaciones



Registros de pares clave-valor (K, V)

- El par clave-valor es un concepto bien establecido en muchos lenguajes de programación (diccionario Python)

prefijos = { 'Madrid': 91, 'BCN': 93, 'Bilbao': 944, ... }

- En cada par clave-valor, la clave se representa mediante un tipo inmutable, a menudo una cadena arbitraria, como un nombre de archivo, URI ...
- Es un método simple de almacenar datos, y se sabe que escala bien.
- BBDD clave-valor: Redis, Amazon DynamoDB, Riak...

RDDs de pares clave-valor (K, V)

- Son RDD donde cada elemento de la colección es una tupla de dos elementos, encerrados entre paréntesis y separados por una coma.
 - El primer elemento se interpreta como la clave
 - El segundo como el valor
- Se pueden construir directamente, o a partir de otras transformaciones:

```
palabras = sc.parallelize(['HOLA', 'Que', 'TAL', 'Bien'])  
  
pal_long = palabras.map(lambda elem: (elem, len(elem)))
```

- Resultado: `[('HOLA', 4), ('Que', 3), ('TAL', 3), ('Bien', 4)]`

Transformaciones clave-valor (K, V)

Transformación	Descripción
mapValues(f)	Realiza operaciones sobre los valores de los pares (K,V)
reduceByKey(f)	Al llamarlo sobre un RDD de pares clave-valor (K, V), devuelve otro de pares (K, V) donde los valores de cada clave se han agregado usando la función dada.
groupByKey(f)	Al llamarlo sobre un RDD de pares clave-valor (K, V), devuelve otro de pares (K, seq[V]) donde los valores de cada clave se han convertido a una secuencia.
sortByKey()	Ordena un RDD de pares clave-valor (K, V) por clave.
join(rdd)	Hace un join de dos rdd de pares (K, V1) y (K,V2) y devuelve otro RDD con claves (K, (V1, V2))

Transformación: “mapValues()”

- Realiza una operación (aplica la función argumento) sobre los valores de los pares.
- El resultado sigue como pares (K,V), esto en un RDD nuevo

```
frutas = sc.parallelize([('P', 'peras,piñas'), ('M',  
'manzanas,melones,melocotones'), ('N','naranjas,nectarinas'), ('L',  
'limones')])  
  
frutas_MAY = frutas.mapValues(lambda elemento: elemento.upper())
```

- Resultado:
[('P', 'PERAS,PIÑAS'), ('M', 'MANZANAS,MELONES,MELOCOTONES'), ('N',
'NARANJAS,NECTARINAS'), ('L', 'LIMONES')]

Transformación: “reduceByKey()”

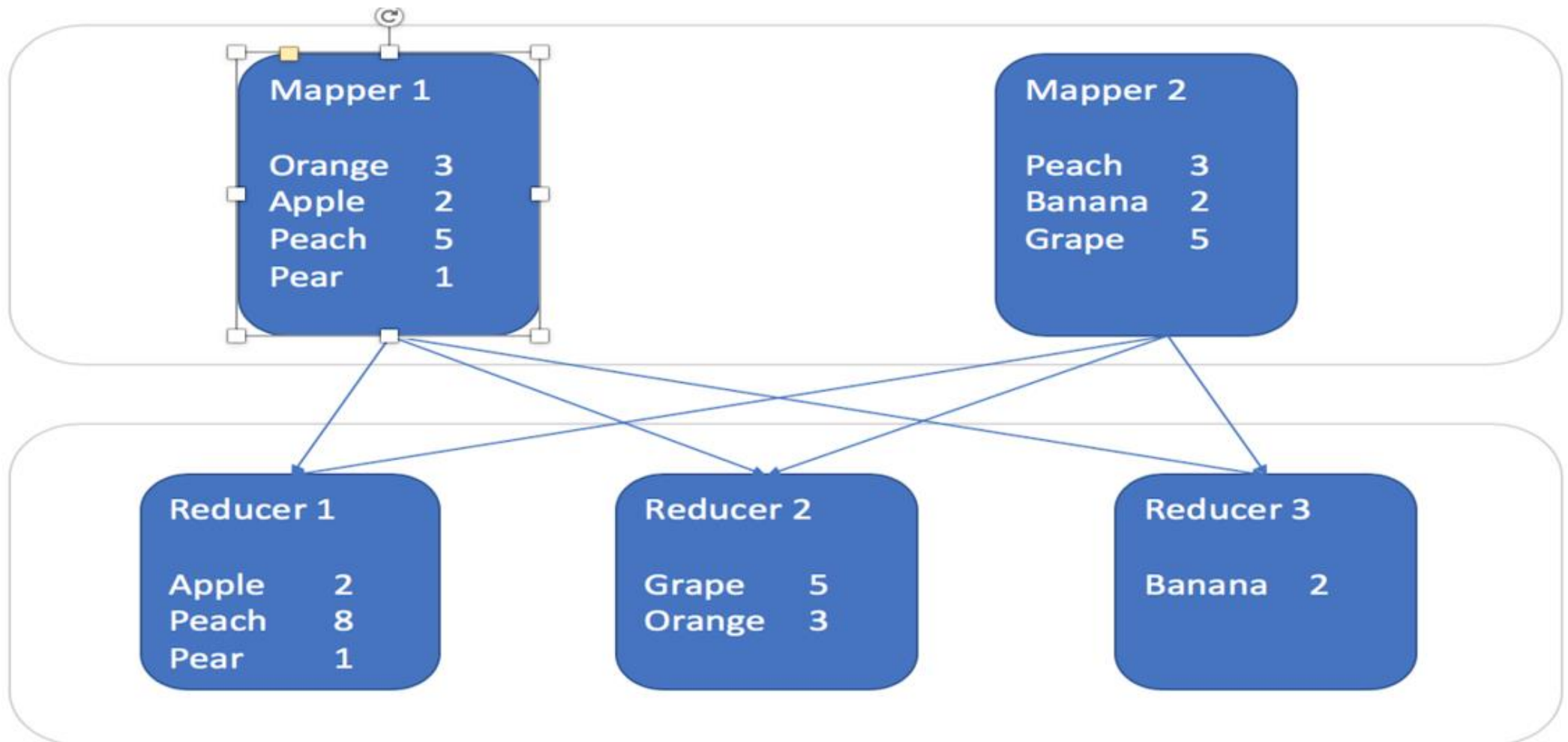
- Agrega todos los elementos del RDD hasta obtener un único valor por clave
- El resultado sigue como pares (K,V), esto en un RDD

```
r = sc.parallelize([('A', 1), ('C', 4), ('A', 1), ('B', 1), ('B', 4)])  
rr = r.reduceByKey(lambda v1,v2:v1+v2)  
print(rr.collect())
```

- Resultado: [('A', 2), ('C', 4), ('B', 5)]
- La función que se pasa a reduceByKey debe (como para reduce):
 - Recibir dos argumentos y devolver uno **de tipo compatible**
 - Ser conmutativa y asociativa de forma que se pueda calcular bien el paralelo
 - A la función se le van a pasar dos valores de elementos con la misma clave

ReduceByKey(): ejecución en el clúster

- Agrupar en **un nodo** los elementos de una misma clave, que provendrán de **distintos nodos**.



Cuestiones sobre “reduceByKey()”

- ¿De qué tamaño es el RDDs de salida?
 - Igual o menor que el RDD original
 - Exactamente igual al número de claves distintas en el RDDs original

```
r = sc.parallelize([('A', 1), ('C', 4), ('A', 1), ('B', 1), ('B', 4)])  
rr1 = r.reduceByKey(lambda v1,v2:v1+v2)  
print(rr1.collect())
```

- Resultado: [('B', 5), ('C', 4), ('A', 2)]
- ¿Qué pasaría si ponemos lambda v1,v2: str(v1+v2)?

Transformación: “groupByKey()”

- Agrupa todos los elementos del RDD para obtener un único valor por clave con valor igual a la secuencia de valores
- Es necesario utilizar mapValues(list) para ver el resultado (se suele usar como paso intermedio en un proceso)

```
r = sc.parallelize([('A', 1), ('C', 2), ('A', 3), ('B', 4), ('B', 5)])  
  
print(r.groupByKey().mapValues(list).collect())  
print(r.groupByKey().mapValues(len).collect())
```

- Resultado:
[('B', [4, 5]), ('C', [2]), ('A', [1, 3])]
[('B', 2), ('C', 1), ('A', 2)]
- Se usa en procesos intermedios. Si lo que queremos una vez agrupados es agregar los datos (resultado final) ➔ “**reduceByKey()**”
- Como referencia (suele aparecer si consultas publicaciones o webs)

Transformación: “sortByKey()”

- Ordena **por clave** un RDD de pares (K,V)
- Si le pasas False ordena de forma inversa

```
rdd = sc.parallelize([('A',1), ('B',6), ('C',3), ('A',4), ('A',5), ('B',2)])  
  
res = rdd.sortByKey(False)  
  
print(res.collect())
```

- Resultado: [('C', 3), ('B', 6), ('B', 2), ('A', 1), ('A', 4), ('A', 5)]

Transformación: “join()”

- Realiza una operación de “unión” de dos RDD (K,V) y (K,W) por clave para dar un RDD (K,(V,W))
- Característica especial: devuelve en caso de correspondencia (clave en ambos RDDs) el producto cartesiano → todos con todos de distinto RDD

```
rdd1 = sc.parallelize([('A',1), ('B',2)])  
rdd2 = sc.parallelize([('A',4), ('B',5), ('A',7)])  
  
rddjoin = rdd1.join(rdd2)
```

- Resultado: [('B', (2, 5)), ('A', (1, 4)), ('A', (1, 7))]

Consideraciones sobre “join()”

- Semejanza con “join()” visto en SQL.
- Variantes **leftOuterJoin()**, **rightOuterJoin()**, **fullOuterJoin()**

```
# DATOS DE PARTIDA
```

```
rdd1 = sc.parallelize([('A',1), ('B',2), ('C',3)])
```

```
rdd2 = sc.parallelize([('A',4), ('A',5), ('B',6), ('D',7)])
```

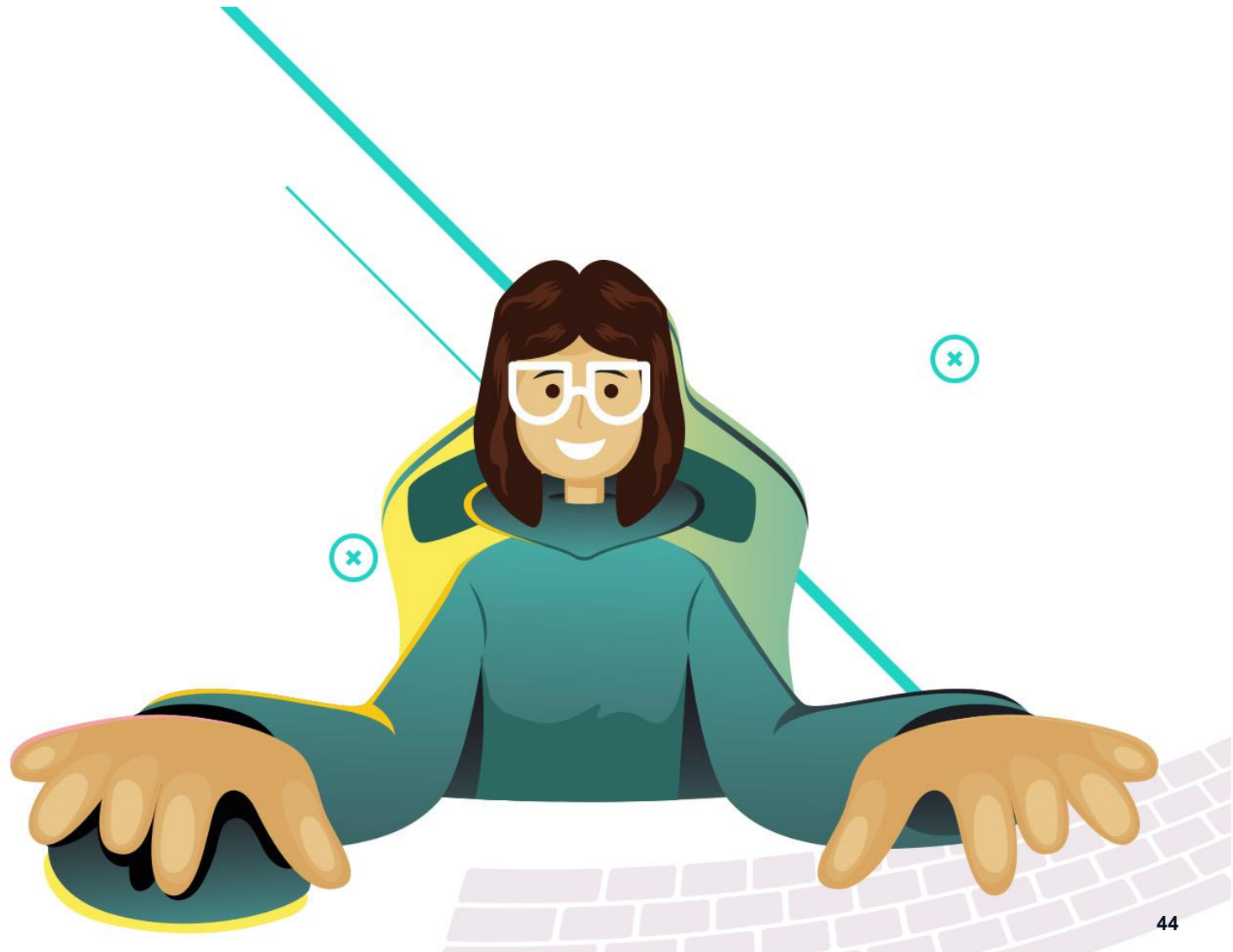
- Modifica join por leftOuterJoin, rightOuterJoin y fullOuterJoin
¿Qué claves aparecerán en el resultado de cada uno?

5. Ejercicio: Transformaciones clave-valor (K, V)



EJERCICIO 3: RDDs de pares clave-valor (K, V)

EJERCICIO 3: Agrupar ventas del periodo actual. Comparar con ventas totales de un intervalo del periodo anterior (¿se cumple tendencia?)



EJERCICIO 3: Agrupar ventas por marca en un MES. Comparar con ventas totales TRIMESTRE año anterior

- Datos entrada: Compras marca 'Adidas', 'Nike', 'Puma'.

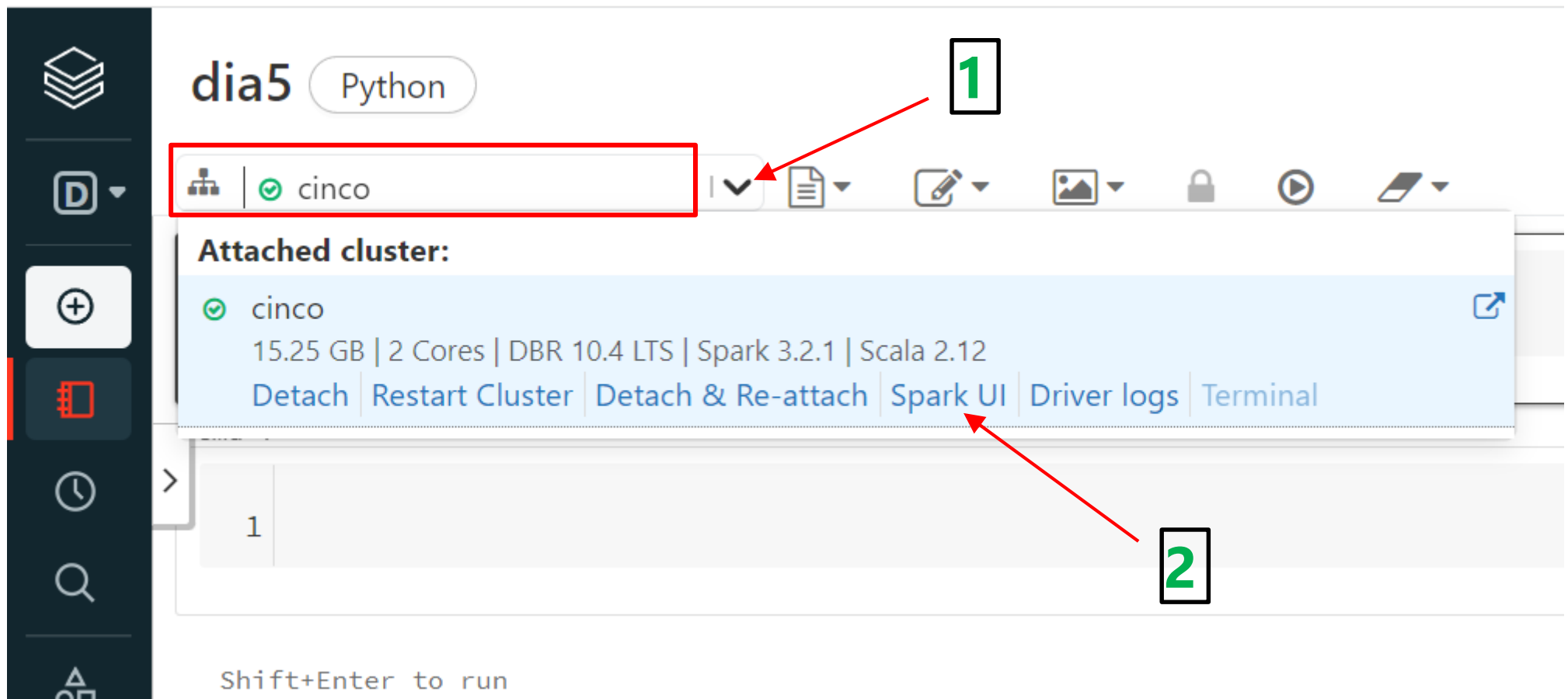
```
rdd1 = sc.parallelize([('Nike', 51805), ('Puma', 42329), ('Adidas', 63542),  
('Puma', 27923), ('Nike', 75335), ('Puma', 45102), ('Adidas', 49583),  
('Puma', 37869), ('Adidas', 54201), ('Puma', 31582), ('Nike', 62747)])  
  
rdd2 = sc.parallelize([('Nike', 224589), ('Adidas', 219123), ('Puma', 166524)])
```

- Caso: Nueva marca en el mercado ('Novedad')

```
rdd1 = sc.parallelize([('Nike', 51805), ('Puma', 42329), ('Novedad', 18536),  
('Adidas', 63542), ('Puma', 27923), ('Nike', 75335), ('Puma', 45102),  
('Adidas', 49583), ('Puma', 37869), ('Novedad', 27196), ('Adidas', 54201),  
('Puma', 31582), ('Nike', 62747), ('Novedad', 25409)])  
  
rdd2 = sc.parallelize([('Nike', 224589), ('Adidas', 219123), ('Puma', 166524)])
```

Acceso a la consola de Spark en DBCE

1. En un notebook abierto, nos vamos a la esquina superior izquierda al recuadro con el nombre del clúster asignado, clickamos flecha desplegable a la derecha
2. Entre las opciones en azul de la parte inferior, clickamos en “Spark UI”



Acceso a la consola de Spark en DBCE

- Se nos abre una nueva pestaña. En el menú de la parte superior podemos ver y acceder a la información entre otros sobre Jobs (info general, DAG), Stages (duración, resumen Task), Environment (configuración) y Executors

The screenshot shows a web browser with two tabs: 'Databricks Community Edition' and 'Databricks Shell - Spark Jobs'. The address bar shows the URL: `community.cloud.databricks.com/sparkui/0418-002648-w1f1s4xq/driver-6189623682931611512/jobs/?o=210304517805659`. A red box highlights the top navigation menu with the following items: Jobs, Stages, Storage, Environment, Executors, SQL, JDBC/ODBC Server, and Structured Streaming. Below the menu, the 'Spark Jobs (?)' section displays the following information:

- User:** root
- Total Uptime:** 2.7 h
- Scheduling Mode:** FAIR
- Completed Jobs:** 6
- Failed Jobs:** 1

Below this information, there are two expandable sections: 'Event Timeline' and 'Completed Jobs (6)'. At the bottom left, there is a 'Page: 1' indicator.



red.es



UNIÓN EUROPEA

"El FSE invierte en tu futuro"

Fondo Social Europeo

