By default, the latest version of the API and the latest supported Spark version is chosen. To specify your own: `%use spark(spark=3.3.0, scala=2.13, v=1.2.0)`

You can also define `displayLimit` and `displayTruncate` to control the display of the result.

Finally, any other property you pass, like `spark.master=local[4]`, will be passed on to Spark.

In [1]:

```
%use spark
```

```
received properties: Properties: {spark=3.3.1, scala=2.13, v=1.2.3, displayLimit=20, displayTruncate=30, spark.app.name=Jupyter,
spark.master=local[*], spark.sql.codegen.wholeStage=false, fs.hdfs.impl=org.apache.hadoop.hdfs.DistributedFileSystem, fs.file.im
pl=org.apache.hadoop.fs.LocalFileSystem}, providing Spark with: {spark.app.name=Jupyter, spark.master=local[*], spark.sql.codege
n.wholeStage=false, fs.hdfs.impl=org.apache.hadoop.hdfs.DistributedFileSystem, fs.file.impl=org.apache.hadoop.fs.LocalFileSyste
m}
23/09/01 19:54:57 INFO SparkContext: Running Spark version 3.3.1
23/09/01 19:54:57 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes wh
ere applicable
23/09/01 19:54:57 INFO ResourceUtils: ==============================================================
23/09/01 19:54:57 INFO ResourceUtils: No custom resources configured for spark.driver.
23/09/01 19:54:57 INFO ResourceUtils: ==============================================================
23/09/01 19:54:57 INFO SparkContext: Submitted application: Jupyter
23/09/01 19:54:57 INFO ResourceProfile: Default ResourceProfile created, executor resources: Map(cores -> name: cores, amount:
1, script: , vendor: , memory -> name: memory, amount: 1024, script: , vendor: , offHeap -> name: offHeap, amount: 0, script: ,
vendor: ), task resources: Map(cpus -> name: cpus, amount: 1.0)
23/09/01 19:54:57 INFO ResourceProfile: Limiting resource is cpu
23/09/01 19:54:57 INFO ResourceProfileManager: Added ResourceProfile id: 0
23/09/01 19:54:57 INFO SecurityManager: Changing view acls to: User
23/09/01 19:54:57 INFO SecurityManager: Changing modify acls to: User
23/09/01 19:54:57 INFO SecurityManager: Changing view acls groups to:
23/09/01 19:54:57 INFO SecurityManager: Changing modify acls groups to:
23/09/01 19:54:57 INFO SecurityManager: SecurityManager: authentication disabled; ui acls disabled; users  with view permission
s: Set(User); groups with view permissions: Set(); users  with modify permissions: Set(User); groups with modify permissions: Se
t()
23/09/01 19:54:57 INFO Utils: Successfully started service 'sparkDriver' on port 54951.
23/09/01 19:54:57 INFO SparkEnv: Registering MapOutputTracker
23/09/01 19:54:57 INFO SparkEnv: Registering BlockManagerMaster
23/09/01 19:54:57 INFO BlockManagerMasterEndpoint: Using org.apache.spark.storage.DefaultTopologyMapper for getting topology inf
ormation
23/09/01 19:54:57 INFO BlockManagerMasterEndpoint: BlockManagerMasterEndpoint up
23/09/01 19:54:57 INFO SparkEnv: Registering BlockManagerMasterHeartbeat
23/09/01 19:54:57 INFO DiskBlockManager: Created local directory at C:\Users\User\AppData\Local\Temp\blockmgr-3c23d5d0-3b5b-436e
-a785-d2a4fdb690a4
23/09/01 19:54:57 INFO MemoryStore: MemoryStore started with capacity 9.4 GiB
23/09/01 19:54:57 INFO SparkEnv: Registering OutputCommitCoordinator
23/09/01 19:54:57 INFO Utils: Successfully started service 'SparkUI' on port 4040.
23/09/01 19:54:58 INFO Executor: Starting executor ID driver on host JULIUS-VON-MAYER
23/09/01 19:54:58 INFO Executor: Starting executor with user classpath (userClassPathFirst = false): ''
23/09/01 19:54:58 INFO Executor: Using REPL class URI: spark://JULIUS-VON-MAYER:54951/classes
23/09/01 19:54:58 INFO Utils: Successfully started service 'org.apache.spark.network.netty.NettyBlockTransferService' on port 55
002.
23/09/01 19:54:58 INFO NettyBlockTransferService: Server created on JULIUS-VON-MAYER:55002
23/09/01 19:54:58 INFO BlockManager: Using org.apache.spark.storage.RandomBlockReplicationPolicy for block replication policy
23/09/01 19:54:58 INFO BlockManagerMaster: Registering BlockManager BlockManagerId(driver, JULIUS-VON-MAYER, 55002, None)
```

```
23/09/01 19:54:58 INFO BlockManagerMasterEndpoint: Registering block manager JULIUS-VON-MAYER:55002 with 9.4 GiB RAM, BlockManag
erId(driver, JULIUS-VON-MAYER, 55002, None)
23/09/01 19:54:58 INFO BlockManagerMaster: Registered BlockManager BlockManagerId(driver, JULIUS-VON-MAYER, 55002, None)
23/09/01 19:54:58 INFO BlockManager: Initialized BlockManager: BlockManagerId(driver, JULIUS-VON-MAYER, 55002, None)
Spark session (Spark: 3.3.1, Scala: 2.13, v: 1.2.3)  has been started and is running. No `withSpark { }` necessary, you can acce
ss `spark` and `sc` directly. To use Spark streaming, use `%use spark-streaming` instead.
```

establecemos los Datos de trabajo

Se define dos clases de enumeración: EyeColor, Gender. También se define una clase de datos: Person

In [2]:

```
enum class ColorOjos {
    BLUE, BROWN, GREEN
}

enum class Genero {
    MALE, FEMALE, OTHER
}
```

In [3]:

```
data class Persona(
    val color_ojos: ColorOjos,
    val name: String,
    val gender: Genero,
    val length: Double,
    val age: Int,
)
```

Se define una variable "ds" que es un conjunto de datos de tipo Dataset. El conjunto de datos contiene tres objetos "Persona", cada uno con diferentes propiedades, como color_ojos, name, gender, length y age.

In [4]:

```
val ds: Dataset<Persona> = dsOf(
    Persona(
        color_ojos = ColorOjos.BLUE,
        name = "Alice",
        gender = Genero.FEMALE,
        length = 1.70,
        age = 25,
    ),
    Persona(
        color_ojos = ColorOjos.BLUE,
        name = "Bob",
        gender = Genero.MALE,
        length = 1.67,
        age = 25,
    ),
    Persona(
        color_ojos = ColorOjos.BROWN,
        name = "Charlie",
        gender = Genero.OTHER,
        length = 1.80,
        age = 17,
    ),
)
```

In [5]:

```
// Se imprime "ds". La salida muestra los tres objetos Person.

ds
```

Out[5]:

color_ojos	name	gender	length	age
BLUE	Alice	FEMALE	1.7	25
BLUE	Bob	MALE	1.67	25
BROWN	Charlie	OTHER	1.8	17

Operaciones

Los efectos de operaciones como el filtrado también se pueden ver inmediatamente, así como la clasificación, selección de columnas, etc

In [6]:

```
ds.filter { it.age > 20 }
```

Out[6]:

color_ojos	name	gender	length	age
BLUE	Alice	FEMALE	1.7	25
BLUE	Bob	MALE	1.67	25

In [7]:

```
ds.sort(col(Persona::age), col(Persona::length))
```

Out[7]:

color_ojos	name	gender	length	age
BROWN	Charlie	OTHER	1.8	17
BLUE	Bob	MALE	1.67	25
BLUE	Alice	FEMALE	1.7	25

In [8]:

```
val res: Dataset<Tuple2<Int, Double>> = ds.select(col(Persona::age), col(Persona::length))
res
```

Out[8]:

age	length
25	1.7
25	1.67
17	1.8

In [13]:

```
"Promedio de [length]: " +
    ds
        .map { it.length }
        .reduceK { a, b -> a + b } / ds.count()
```

Out[13]:

```
Promedio de [length]: 1.7233333333333334
```

In [14]:

```
"Promedio de [age]: " +
    ds
        .map { it.age }
        .reduceK { a, b -> a + b } / ds.count()
```

Out[14]:

```
Promedio de [age]: 22
```

También podemos crear RDD usando `sc: JavaSparkContext` que se representan de manera similar a los conjuntos de datos. Puede ver que todas las funciones auxiliares de Tuple también están disponibles de inmediato.

In [15]:

```
val rdd: JavaRDD<Tuple2<Int, String>> = rddOf(
    1 X "aaa",
    t(2, "bbb"),
    tupleOf(3, "cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc"),
)

rdd
```

Out[15]:

Values
[1, aaa]
[2, bbb]

Finalmente, también podemos configurar diplayLimit y displayTruncate sobre la marcha usando sparkProperties.

In [16]:

```
sparkProperties {
    displayLimit = 2
    displayTruncate = -1
}

rdd
```

Out[16]:

Values
[1, aaa]
[2, bbb]

In [ ]: