



INSTITUTO POLITÉCNICO NACIONAL  
ESCUELA SUPERIOR DE CÓMPUTO  
ACADEMIA INGENIERÍA DE SOFTWARE



**Unidad de Aprendizaje:** Big Data

**Profesor:** Ing. Tania Rodriguez Sarabia

**Práctica no. 1:** ETL con pyspark

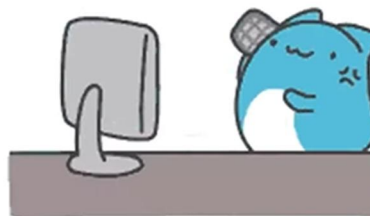
**ALUMNO:**

**GRUPO:**

**FECHA DE ENTREGA:**

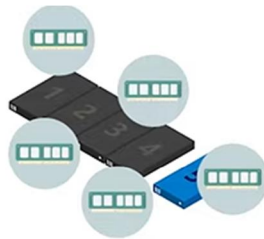
Cuando queremos hacer un análisis de datos sea hallar algún estadístico, hacer una visualización, un filtrado o entrenar a un modelo de machine learning, usualmente cargaremos nuestros datos desde un archivo almacenado en nuestra computadora hacia un DataFrame de pandas, aquí gracias a la magia de pandas vamos a poder hacer el análisis que queramos, esta rapidez de pandas se debe a que los dataframes se cargan en la memoria RAM de sus computadoras una sola vez, luego de esta carga inicial, cualquier análisis u operación ejecutada sobre el DataFrame se hace con los datos ya cargados en la memoria RAM, lo interesante es que las memorias RAM son muy rápidas al momento de operar sobre datos que ahí se encuentran. Pero ¿qué pasa cuando mis datos no entran en la memoria RAM? Esto podría suceder muy fácilmente ya que las computadoras de hoy pueden tener entre 8 y 16 GB de memoria RAM, en este punto podríamos obtener un error si queremos cargar todo en un dataframe de pandas

## Memoria RAM



Ahora nos tocará implementar una solución en donde nuestro código donde nuestro código lea y opere sobre nuestros datos por partes, estas múltiples lecturas al disco duro matarán la eficiencia y rapidez en nuestro proceso pues a comparación de la memoria RAM leer datos del disco duro es de 10 a 100 veces mas lento dependiendo de la calidad del disco.

Sin embargo, si tenemos mas de una maquina a nuestra disposición, por cada máquina tendríamos una memoria RAM adicional que podríamos utilizar para que este procesamiento siga siendo rápido y por sobre todo que soporte grandes cantidades de datos, con esta premisa nacen algunos problemas ¿cómo dividimos los datos en las memorias RAM de varias máquinas? ¿Cómo controlamos que dato están en cual máquina? ¿Cómo unimos los datos cuando queremos hacer algún análisis como una agrupación?



## ¿Dividir los Datos?

## ¿Control de los Datos?

## ¿Unión de los datos?

Con estos problemas nace Apache Spark, que es un framework para procesar grandes cantidades de datos cuya idea principal es distribuir los datos en las memorias RAM de varias máquinas y que cada máquina se encargue del procesamiento de la parte de los datos que esta en ella. A este conjunto de computadoras se les denomina clusters y a cada computadora en el cluster se le llama nodo, por cada cluster existe un nodo maestro o master node y varios nodos trabajadores o workers. El master node se encarga de recibir las tareas que tienen que ejecutarse y dividir esas tareas entre los nodos trabajadores los cuales van a ejecutar estas tareas en paralelo sobre su parte de los datos utilizando el paradigma de programación MapReduce, que es un divide y conquista para grandes cantidades de datos. Las tareas pueden ser Cargar los datos, Filtrar los datos, hacer alguna Agregación o incluso entrenar un modelo de machine learning.

## MapReduce

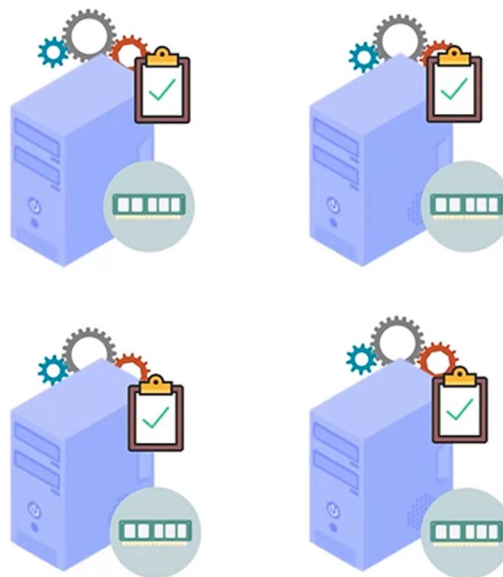
Cargar Datos

Filtrar Datos

Agregación



Nodo Maestro  
(Master)



Nodos Trabajadores (Workers)

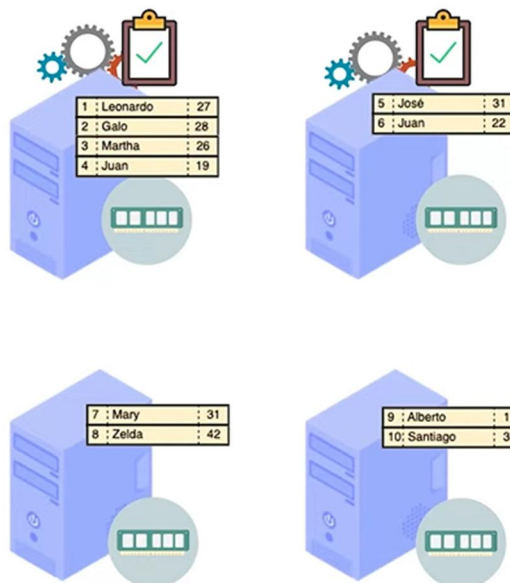
C  
L  
U  
S  
T  
E  
R

Cuando los datos se cargan estos se distribuyen idealmente de manera uniforme a lo largo de las memorias RAM de los nodos workers y el nodo maestro conoce hasta cierto punto cómo se encuentran hechas estas divisiones de los datos de modo que el nodo maestro puede saber a que nodos trabajadores asignar ciertas tareas. Por ejemplo, si el usuario nos pide mostrar los 5 primeros registros de un dataset en este ejemplo solo el nodo trabajador 1 y 2 van a recibir tareas, evitando el trabajo innecesario de los otros nodos, los resultados de cada tarea son retornados por cada worker directamente a la aplicación, donde existe un ente que se encarga de unirlos y presentarlos al usuario.

 `.head(5)`



Master



Workers

1	: Leonardo	: 27
2	: Galo	: 28
3	: Martha	: 26
4	: Juan	: 19

5	: José	: 31
6	: Juan	: 22

7	: Mary	: 31
8	: Zelda	: 42

9	: Alberto	: 11
10	: Santiago	: 31

1	: Leonardo	: 27
2	: Galo	: 28
3	: Martha	: 26
4	: Juan	: 19
5	: José	: 31

Cabe aclarar que Spark es un framework, esto quiere decir que lo podemos utilizar en muchos lenguajes de programación incluyendo Python con la librería de pyspark, siempre y cuando tengamos un cluster de computadoras con spark instalado y configurado, lo cual puede llegar a ser muy complejo. Pero por suerte, existen algunos proveedores de servicios en la nube como AWS que nos dan herramientas que con un par de clicks, ya tenemos configurado un cluster de máquinas para ejecutar nuestro código spark sobre nuestros datos.



Vamos a hacer un proceso de ETL en Spark, primero que nada y por simplicidad vamos a abrir el Google colab, que es un entorno que permite ejecutar código Python en un navegador sin necesidad de configuraciones avanzadas. Primero instalamos todo lo necesario para trabajar con pyspark.

A screenshot of the Google Colab interface. The top bar shows the Colab logo and the file name "ETLbasica.ipynb". Below the menu bar, there are two tabs: "+ Código" and "+ Texto". The main area shows a code cell with the command "pip install pyspark". The output of the cell is displayed, showing that the requirement is already satisfied for pyspark (3.3.0) and py4j (0.10.9.5).

```
pip install pyspark
```

Looking in indexes: <https://pypi.org/simple>, <https://us-python.pkg.dev/colab-wheels/public/simple/>  
Requirement already satisfied: pyspark in /usr/local/lib/python3.7/dist-packages (3.3.0)  
Requirement already satisfied: py4j==0.10.9.5 in /usr/local/lib/python3.7/dist-packages (from pyspark) (0.10.9.5)

Una vez que tengamos todo el ambiente preparado, importamos todas las herramientas que vamos a usar. También vamos a crear una sesión de spark y también vamos a inicializar nuestro contexto de sql.

A screenshot of the Google Colab interface showing the next step in the code. The code cell contains imports for SparkSession, SQLContext, IntegerType, StringType, StructType, StructField, col, and lit. It then creates a SparkSession and an SQLContext.

```
import pyspark

from pyspark.sql import SparkSession, SQLContext
from pyspark.sql.types import IntegerType, StringType, StructType, StructField
from pyspark.sql.functions import col, lit

spark = SparkSession.builder.appName("etlbasica").getOrCreate()
sqlContext = SQLContext(spark)
```

Ahora le vamos a indicar la ruta del archivo que se vamos a trabajar, y también vamos a crear una variable con los datos de un csv llamado `players_21.csv`, en este caso es csv, pero pyspark puede igualmente consumir otros tipos de datos como parquet, json, etc. Luego, vamos a mostrar el csv con `display()`, esto mostrará el DataFrame tal cual está constituido

```
[3] path = "/content/drive/MyDrive/CharlaNUCBA/"

players_21 = spark.read.csv(path+"players_21.csv")

[5] display(players_21)

DataFrame[_c0: string, _c1: string, _c2: string, _c3: string, _c4: string, _c5: string, _c6: string, _c7: string, _c8:
string, _c11: string, _c12: string, _c13: string, _c14: string, _c15: string, _c16: string, _c17: string, _c18: string,
_c21: string, _c22: string, _c23: string, _c24: string, _c25: string, _c26: string, _c27: string, _c28: string, _c29:
string, _c32: string, _c33: string, _c34: string, _c35: string, _c36: string, _c37: string, _c38: string, _c39: string,
_c42: string, _c43: string, _c44: string, _c45: string, _c46: string, _c47: string, _c48: string, _c49: string, _c50:
string, _c53: string, _c54: string, _c55: string, _c56: string, _c57: string, _c58: string, _c59: string, _c60: string,
_c63: string, _c64: string, _c65: string, _c66: string, _c67: string, _c68: string, _c69: string, _c70: string, _c71:
string, _c74: string, _c75: string, _c76: string, _c77: string, _c78: string, _c79: string, _c80: string, _c81: string,
_c84: string, _c85: string, _c86: string, _c87: string, _c88: string, _c89: string, _c90: string, _c91: string, _c92:
string, _c95: string, _c96: string, _c97: string, _c98: string, _c99: string, _c100: string, _c101: string, _c102: string,
string, _c105: string]
```

También hay otras formas de mostrar el archivo csv como por ejemplo con el comando `show()`, que lo que hará es mostrar de manera ordenada las columnas.

```
players_21.show()

+-----+-----+-----+-----+-----+-----+-----+-----+
|_c0|_c1|_c2|_c3|_c4|_c5|_c6|_c7|_c8|
+-----+-----+-----+-----+-----+-----+-----+-----+
|soffifa_id|player_url|short_name|long_name|age|dob|height_cm|weight_kg|nationality|
+-----+-----+-----+-----+-----+-----+-----+-----+
|158023|https://soffifa.co...|L. Messi|Lionel Andrés Mes...|33|1987-06-24|170|72|Argentina|
|208001|https://soffifa.co...|Cristiano Ronaldo|Cristiano Ronaldo...|35|1985-02-05|187|83|Portugal|
|200389|https://soffifa.co...|J. Oblak|Jan Oblak|27|1993-01-07|188|87|Slovenia|
|188545|https://soffifa.co...|R. Lewandowski|Robert Lewandowski|31|1988-08-21|184|80|Poland|
|190871|https://soffifa.co...|Neymar Jr|Neymar da Silva S...|28|1992-02-05|175|68|Brazil|Pa
|192985|https://soffifa.co...|K. De Bruyne|Kevin De Bruyne|29|1991-06-28|181|70|Belgium|
|231747|https://soffifa.co...|K. Mbappé|Kylian Mbappé Lottin|21|1998-12-20|178|73|France|Pa
|192448|https://soffifa.co...|M. ter Stegen|Marc-André ter St...|28|1992-04-30|187|85|Germany|
|203376|https://soffifa.co...|V. van Dijk|Virgil van Dijk|28|1991-07-08|193|92|Netherlands|
|212831|https://soffifa.co...|Alisson|Alisson Ramsés Be...|27|1992-10-02|191|91|Brazil|
|208722|https://soffifa.co...|S. Mané|Sadio Mané|28|1992-04-10|175|69|Senegal|
|209331|https://soffifa.co...|M. Salah|Mohamed Salah Ghalay|28|1992-06-15|175|71|Egypt|
|192119|https://soffifa.co...|T. Courtois|Thibaut Courtois|28|1992-05-11|199|96|Belgium|
|153079|https://soffifa.co...|S. Agüero|Sergio Leonel Agü...|32|1988-06-02|173|70|Argentina|
|155862|https://soffifa.co...|Sergio Ramos|Sergio Ramos García|34|1986-03-30|184|82|Spain|
|165153|https://soffifa.co...|K. Benzema|Karim Benzema|32|1987-12-19|185|81|France|
|167495|https://soffifa.co...|M. Neuer|Manuel Neuer|34|1986-03-27|193|92|Germany|
```

Ya hemos hecho la carga de los datos, pero nos damos cuenta de que hay un montón de datos que no nos sirven, como URL, además los encabezados de las columnas (`_c0`, `_c2`, etc.) tampoco nos sirven, también el id por ejemplo no lo vamos a necesitar.

Primero vamos a sobrescribir mi variable `players_21`, indicando solo las columnas que deseamos mantener con `select`.

```
[6] only showing top 20 rows

players_21 = players_21.select(_c3, _c10, _c33, _c34, _c35, _c36, _c37, _c38)
```

Luego definimos una función para quitar el encabezado (`_c0`, `_c1`, `_c2`, etc.) y queremos el nombre que yo quiero, osea el de la segunda fila (`soffifa_id`, `player_url`, `short_name`, `long_name`, etc.)

```
def dropFirstRow(index, iterator):
    return iter([List(iterator)[1:]])
```

Luego creamos un RDD a partir del dataframe.

```
[19] return iter(cast(iterator)[1:])  
  
rdd = players_21.rdd
```

Luego hacemos un collect para ver qué información trae el rdd.

```
[21] rdd.collect()  
  
Row(c3='Thilo Kehrer', c16='CB, RB', c33='72', c34='44', c35='65', c36='67', c37='78', c38='78'),  
Row(c3='황희찬 黄喜灿', c16='ST', c33='93', c34='75', c35='71', c36='79', c37='28', c38='69'),  
Row(c3='Rubén Duarte Sánchez', c16='LB', c33='67', c34='36', c35='65', c36='67', c37='77', c38='69'),  
Row(c3='Gautier Larssonneur', c16='GK', c33=None, c34=None, c35=None, c36=None, c37=None, c38=None),  
Row(c3='Franck Yannick Kessié', c16='CM', c33='71', c34='72', c35='71', c36='77', c37='77', c38='87'),  
Row(c3='Marc Roca Junqué', c16='CM, CM', c33='59', c34='62', c35='77', c36='72', c37='70', c38='68'),  
Row(c3='Jamaal Lascelles', c16='CB', c33='58', c34='28', c35='51', c36='57', c37='78', c38='80'),  
Row(c3='Thomas Foket', c16='RB', c33='75', c34='57', c35='70', c36='72', c37='73', c38='76'),  
Row(c3='Neal Maupay', c16='ST', c33='77', c34='77', c35='67', c36='73', c37='41', c38='71'),  
Row(c3='Alex Iwobi', c16='LM', c33='78', c34='66', c35='73', c36='79', c37='28', c38='69'),  
Row(c3='Kenny Tete', c16='RB', c33='74', c34='47', c35='65', c36='66', c37='77', c38='76'),  
Row(c3='Daley Sinkgraven', c16='LB, CM', c33='79', c34='60', c35='78', c36='80', c37='73', c38='64'),  
Row(c3='Alexander Sarloth', c16='ST', c33='75', c34='77', c35='66', c36='78', c37='36', c38='83'),
```

Ahora vamos a convertir nuestro rdd, nuevamente a un dataframe, así que primero creamos el esquema. Para ello definimos el StructField de cada una de las columnas, su tipo y si aceptan valores nulos o no.

```
schema = StructType([  
    StructField("Name",StringType(),False),  
    StructField("Positions",StringType(),True),  
    StructField("PAC",StringType(),True),  
    StructField("SHO",StringType(),True),  
    StructField("PAS",StringType(),True),  
    StructField("DRI",StringType(),True),  
    StructField("DEF",StringType(),True),  
    StructField("PHY",StringType(),True),  
)
```

Ahora le decimos al rdd que va a hacer un dataframe utilizando el esquema que hicimos anteriormente con el esquema que hicimos anteriormente.

```
dataframe = sqlContext.createDataFrame(rdd,schema)  
dataframe.printSchema()  
  
root  
|-- Name: string (nullable = false)  
|-- Positions: string (nullable = true)  
|-- PAC: string (nullable = true)  
|-- SHO: string (nullable = true)  
|-- PAS: string (nullable = true)  
|-- DRI: string (nullable = true)  
|-- DEF: string (nullable = true)  
|-- PHY: string (nullable = true)
```

Luego volvemos a mostrar nuestro dataframe, como va quedando hasta ahora



```
dataframe.show()
```

	Name	Positions	PAC	SHO	PAS	DRI	DEF	PHY
	Lionel Andrés Me...	RW, ST, CF	85	92	91	95	38	65
	Cristiano Ronaldo...	ST, LW	89	93	81	89	35	77
	Jan Oblak	GK	null	null	null	null	null	null
	Robert Lewandowski	ST	78	91	78	85	43	82
	Neymar da Silva S...	LW, CAM	91	85	86	94	36	59
	Kevin De Bruyne	CAM, CM	76	86	93	88	64	78
	Kylian Mbappé Lottin	ST, LW, RW	96	86	78	91	39	76
	Marc-André ter St...	GK	null	null	null	null	null	null
	Virgil van Dijk	CB	76	60	71	71	91	86
	Alisson Ramsés Be...	GK	null	null	null	null	null	null
	Sadio Mané	LW	94	85	80	90	44	76
	Mohamed Salah Ghal...	RW	93	86	81	90	45	75
	Thibaut Courtois	GK	null	null	null	null	null	null
	Sergio Leonel Agü...	ST	78	90	77	88	33	73
	Sergio Ramos García	CB	71	70	76	73	88	85
	Karim Benzema	CF, ST	74	85	81	86	40	76
	Manuel Neuer	GK	null	null	null	null	null	null
	Carlos Henrique V...	CDM	65	73	76	72	86	91
	Ederson Santana d...	GK	null	null	null	null	null	null
	Raheem Shaquille ...	LW, RW	93	81	79	90	45	67

Si notamos en los datos hay muchos valores nulos que deseamos quitar, para ello utilizamos la función dropna

```
dataframe = dataframe.dropna()
```

También vamos a castear o convertir el tipo de columnas a entero ya que están como strings.

```
dataframe = dataframe.withColumn("PAC", col("PAC").cast(IntegerType()))
dataframe = dataframe.withColumn("SHO", col("SHO").cast(IntegerType()))
dataframe = dataframe.withColumn("PAS", col("PAS").cast(IntegerType()))
dataframe = dataframe.withColumn("DRI", col("DRI").cast(IntegerType()))
dataframe = dataframe.withColumn("DEF", col("DEF").cast(IntegerType()))
dataframe = dataframe.withColumn("PHY", col("PHY").cast(IntegerType()))
```

Ahora vamos a imprimir el esquema y vemos que ya están como enteros

```
dataframe.printSchema()

root
|-- Name: string (nullable = false)
|-- Positions: string (nullable = true)
|-- PAC: integer (nullable = true)
|-- SHO: integer (nullable = true)
|-- PAS: integer (nullable = true)
|-- DRI: integer (nullable = true)
|-- DEF: integer (nullable = true)
|-- PHY: integer (nullable = true)
```

Luego vamos a hacer otra columna llamada medio (mean) y hacemos una suma de cada una de estas columnas y lo dividimos entre el número de columnas.

```
[30] dataframe = dataframe.withColumn('Mean', ((col('PAC') + col('SHO') + col('PAS') + col('DRI') + col('DEF') + col('PHY')) / lit(6)))
```

Ahora volvemos a mostrar todo el dataframe.

```
dataframe.show()
```

	Name	Positions	PAC	SHO	PAS	DRI	DEF	PHY	Mean
	Lionel Andrés Mes...	RW, ST, CF	85	92	91	95	38	65	77.66666666666667
	Cristiano Ronaldo...	ST, LW	89	93	81	89	35	77	77.33333333333333
	Jan Oblak	GK	null	null	null	null	null	null	null
	Robert Lewandowski	ST	78	91	78	85	43	82	76.16666666666667
	Neymar da Silva S...	LW, CAM	91	85	86	94	36	59	75.16666666666667
	Kevin De Bruyne	CAM, CM	76	86	93	88	64	78	80.83333333333333
	Kylian Mbappé Lottin	ST, LW, RW	96	86	78	91	39	76	77.66666666666667
	Marc-André ter St...	GK	null	null	null	null	null	null	null
	Virgil van Dijk	CB	76	60	71	71	91	86	75.83333333333333
	Alisson Ramsés Be...	GK	null	null	null	null	null	null	null
	Sadio Mané	LW	94	85	80	90	44	76	78.16666666666667
	Mohamed Salah Ghal...	RW	93	86	81	90	45	75	78.33333333333333
	Thibaut Courtois	GK	null	null	null	null	null	null	null
	Sergio Leonel Agü...	ST	78	90	77	88	33	73	73.16666666666667
	Sergio Ramos García	CB	71	70	76	73	88	85	77.16666666666667
	Karim Benzema	CF, ST	74	85	81	86	40	76	73.16666666666667

Creamos un output un archivo csv llamado output1.csv



```
dataframe.repartition(1).write.csv('output1.csv', sep=',')  
[35] spark.read.csv('output1.csv')  
DataFrame[_c0: string, _c1: string, _c2: string, _c3: string, _c4: string, _c5: string, _c6: string, _c7:  
string, _c8: string]
```

Hemos hecho la parte de carga y transformación de los datos, el resultado es un archivo csv que puede usarse para cargarse en la nube, para análisis de datos o lo que se requiera, todo esta en csv ya tratado.

## Por hacer

Ahora vamos a aprender a cargar, explorar y limpiar los datos usando PySpark, el archivo datos\_succios.csv contiene errores como: Valores nulos o faltantes, Columnas con nombres incorrectos, Tipos de datos mezclados, Duplicados, Errores tipográficos. Tu tarea será:

1. Cargar los datos en un DataFrame de Spark
2. Renombrar columnas con errores tipográficos
3. Eliminar duplicados
4. Reemplazar valores incorrectos
5. Convertir tipos de datos
6. Normalizar formatos de fecha
7. Cargar los datos limpios en "datos\_limpios.csv"