Program: **MSc of Data Science**
Module: **Big Data Tools and Techniques**

Week 4 – Part 2

Spark DataFrames

2025

Learning Outcomes

1. To learn what is a DataFrame

2. To learn DataFrame Operations in PySpark

3. To convert a DataFrame to an RDD and vice versa

What is a DataFrame?

This is dataFrame!!

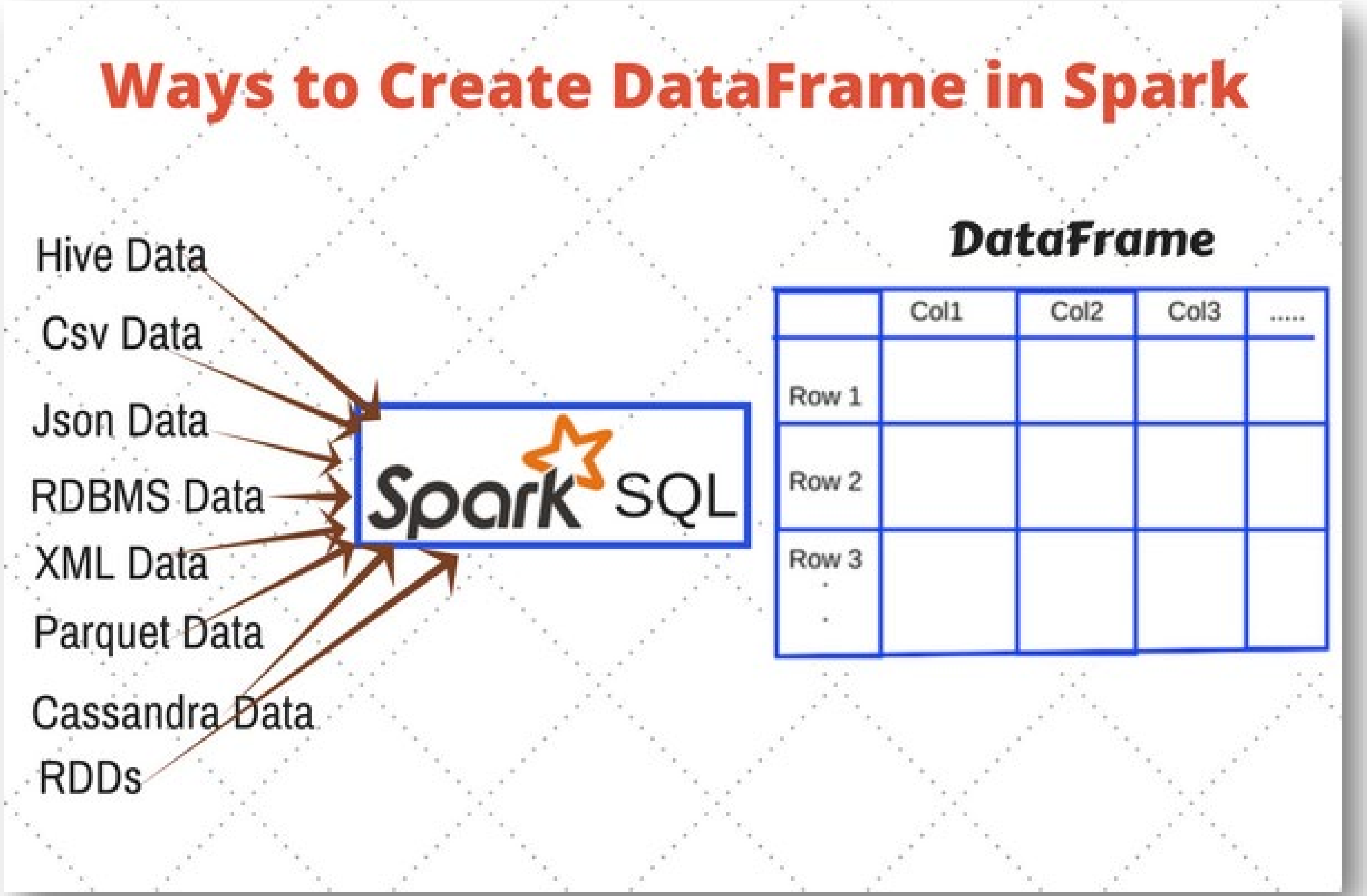
If you Put the Data in a nice Frame You will have a DataFrame

Columns

Rows

name	region	sales	expenses
William	East	50000	42000
Emma	North	52000	43000
Sofia	East	90000	50000
Markus	South	34000	44000
Edward	West	42000	38000
Thomas	West	72000	39000
Ethan	South	49000	42000
Olivia	West	55000	60000
Arun	West	67000	39000
Anika	East	65000	44000
Paulo	South	67000	45000

Spark SQL can convert various types of data into a DataFrame



SCHOOL OF SCIENCE, ENGINEERING & ENVIRONMENT

What is DataFrame?

DataFrame is a distributed collection of rows under named columns. In simple terms, it looks like an Excel sheet with Column headers, or you can think of it as the equivalent of a table in a relational database or a DataFrame in R or Python.

It has three main common characteristics with RDD:

- **Immutable in nature**: You will be able to create a DataFrame but you will not be able to change it. A DataFrame just like an RDD can be transformed
- **Lazy Evaluations:** a task is not executed until an action is performed.
- **Distributed:** DataFrames just like RDDs are distributed in nature

https://www.databricks.com/glossary/spark-api

What is DataFrame?

A DataFrame is a programming abstraction in the Spark SQL module. DataFrames resemble relational database tables or excel spreadsheets with headers: the data resides in rows and columns of different datatypes.

What is a Dataframe Schema

```
root
 |-- Direction: string (nullable = true)
 |-- Year: string (nullable = true)
 |-- Date: string (nullable = true)
 |-- Weekday: string (nullable = true)
 |-- Country: string (nullable = true)
 |-- Commodity: string (nullable = true)
 |-- Transport_Mode: string (nullable = true)
 |-- Measure: string (nullable = true)
 |-- Value: string (nullable = true)
 |-- Cumulative: string (nullable = true)
```

Schema defines the structure of the DataFrame

```
+---------+----+----------+--------+-------+---------+--------------+-------+---------+----------+
|Direction|Year|      Date| Weekday|Country|Commodity|Transport_Mode|Measure|    Value|Cumulative|
+---------+----+----------+--------+-------+---------+--------------+-------+---------+----------+
|  Exports|2015|01/01/2015|Thursday|    All|      All|           All|      $|104000000| 104000000|
|  Exports|2015|02/01/2015|  Friday|    All|      All|           All|      $| 96000000| 200000000|
|  Exports|2015|03/01/2015|Saturday|    All|      All|           All|      $| 61000000| 262000000|
|  Exports|2015|04/01/2015|  Sunday|    All|      All|           All|      $| 74000000| 336000000|
|  Exports|2015|05/01/2015|  Monday|    All|      All|           All|      $|105000000| 442000000|
+---------+----+----------+--------+-------+---------+--------------+-------+---------+----------+
only showing top 5 rows
```

DataFrames contain an ordered collection of Row objects

- Rows contain an ordered collection of values
- Row values can be basic types (such as integers, strings, and floats) or collections of those types (such as arrays and lists)
- A schema maps column names and types to the values in a row

By default, Spark **infers the schema** from the data, however, sometimes we may need to define our own schema (column names and data types), especially while working with unstructured and semi-structured data.

Creating a DataFrame

DataFrames can be created

- From an existing data source (Parquet file, JSON file, etc.)
- From an existing RDD
- By performing an operation or query on another DataFrame
- By programmatically defining a schema

Creating a DataFrame from a JSON file: example

The `users.json` file contains sample data

- Each line contains a single JSON record that can include a name, age, and postal code field

```
{"name":"Alice", "pcode":"94304"}
{"name":"Brayden", "age":30, "pcode":"94304"}
{"name":"Carla", "age":19, "pcode":"10036"}
{"name":"Diana", "age":46}
{"name":"Etienne", "pcode":"94104"}
```

Creating a DataFrame from a JSON file: example

```
usersDF = spark.read.json("users.json")
```

File: people.json

```
{"name":"Alice", "pcode":"94304"}
{"name":"Brayden", "age":30, "pcode":"94304"}
{"name":"Carla", "age":19, "pcode":"10036"}
{"name":"Diana", "age":46}
{"name":"Étienne", "pcode":"94104"}
```

age	name	pcode
null	Alice	94304
30	Brayden	94304
19	Carla	10036
46	Diana	null
null	Étienne	94104

Creating a DataFrame from a JSON file: example

```
usersDF = spark.read.json("users.json")
```

File: people.json

```
{"name":"Alice", "pcode":"94304"}
{"name":"Brayden", "age":30, "pcode":"94304"}
{"name":"Carla", "age":19, "pcode":"10036"}
{"name":"Diana", "age":46}
{"name":"Étienne", "pcode":"94104"}
```

age	name	pcode
null	Alice	94304
30	Brayden	94304
19	Carla	10036
46	Diana	null
null	Étienne	94104

By default, Spark **infers the schema** from the data. In this example we didn’t define a schema and Spark automatically extracted schema from the JSON file structure.

Schema of a DataFrame

- DataFrames always have an associated schema
- `DataFrameReader` can infer the schema from the data
- Use `printSchema` to show the DataFrame's schema

```
usersDF = spark.read.json("users.json")
usersDF.printSchema()
root
 |-- age:  long (nullable = true)
 |-- name:  string (nullable = true)
 |-- pcode:  string (nullable = true)
```

Sample rows from a DataFrame

The show method displays the first few rows in a tabular format

```
usersDF = spark.read.json("users.json")
usersDF.printSchema()
root
 |-- age:  long (nullable = true)
 |-- name:  string (nullable = true)
 |-- pcode:  string (nullable = true)

usersDF.show()
+----+-------+-----+
| age|   name|pcode|
+----+-------+-----+
|null|  Alice|94304|
|  30|Brayden|94304|
|  19|  Carla|10036|
|  46|  Diana| null|
|null|Etienne|94104|
+----+-------+-----+
```

Converting RDDs to DataFrames

You can create a DataFrame from an RDD

- Useful with unstructured or semi-structured data such as text
- Define a schema
- Transform the base RDD to an RDD of Row lists (Python)
- Use `sparkSession.createDataFrame`

You can also return the underlying RDD of a DataFrame

- Use the `DataFrame.rdd` attribute to return an RDD of Row objects

Example: Create a DataFrame from an RDD

Example data: semi-structured text data source (people.txt)

```
02134,Hopper,Grace,52
94020,Turing,Alan,32
94020,Lovelace,Ada,28
87501,Babbage,Charles,49
02134,Wirth,Niklaus,48
```

Defining an RDD from the people.txt

Defining the main RDD:

Cmd 1

```
myRDD1 = sc.textFile("FileStore/tables/people.txt")
```

Command took 0.10 seconds --

Transforming the main RDD to make each peace of information a separate element in the RDD:

Cmd 2

```
myRDD2 = myRDD1.map(lambda line : line.split (",")) \
            .map(lambda values: [values[0],values[1],values[2],int(values[3])])
```

Command took 0.11 seconds --

Checking the resulted RDD:

Cmd 3

```
myRDD2.take(2)
```

▸ (1) Spark Jobs

```
Out[27]: [['02134 ', ' Hopper ', 'Grace ', 52], ['94020 ', ' Turing ', 'Alan ', 32]]
```

Command took 0.37 seconds --

Defining an RDD from the people.txt

Defining an schema for the dataframe that we are going to generate

Dataframe contains 4 columns

Cmd 4

```
from pyspark.sql.types import *

mySchema = \
StructType([
StructField ("pcode", StringType()) ,
StructField ("lastName", StringType()) ,
StructField ("firstName", StringType()) ,
StructField ("age", IntegerType())])
```

Create and show the Dataframe

Dataframe with 4 columns and based on the defined schema

Cmd 9

```
1   myDF.printSchema()
```

```
root
 |-- pcode: string (nullable = true)
 |-- lastName: string (nullable = true)
 |-- firstName: string (nullable = true)
 |-- age: integer (nullable = true)
```

week4 Python

File Edit View Run Help

Cmd 5

```
1   myDF = spark.createDataFrame(myRDD , mySchema)
```

myDF: pyspark.sql.dataframe.DataFrame = [pcode: string, lastName: string ... 2 more fields]

Command took 0.17 seconds

Cmd 6

```
1   myDF.show()
```

(2) Spark Jobs

```
+------+----------+---------+---+
| pcode|  lastName|firstName|age|
+------+----------+---------+---+
|02134 |    Hopper |   Grace | 52|
|94020 |    Turing |    Alan | 32|
|94020 |  Lovelace |     Ada | 28|
|87501 |   Babbage | Charles | 49|
|02134 |     Wirth | Niklaus | 48|
+------+----------+---------+---+
```

Example: return a DataFrame's underlying RDD

You can reverse the process and see the ancestor of the dataframe!!

```
Cmd 10

myRDD2 = myDF.rdd

for row in myRDD2.take(2):
    print(row)

▸ (1) Spark Jobs
Row(pcode='02134 ', lastName=' Hopper ', firstName='Grace ', age=52)
Row(pcode='94020 ', lastName=' Turing ', firstName='Alan ', age=32)

Command took 0.77 seconds --
```

DataFrame operations

There are two main types of DataFrame operations

- Transformations create a new DataFrame based on existing one(s)
  - Transformations are executed in parallel by the application's executors
- Actions output data values from the DataFrame
  - Output is typically returned from the executors to the main Spark program (the driver) or saved to a file

DataFrame operations: Actions

Some common DataFrame actions include

- `count`: returns the number of rows
- `first`: returns the first row (synonym for `head()`)
- `take(n)`: returns the first `n` rows as an array (synonym for `head(n)`)
- `show(n)`: display the first `n` rows in tabular form (default is 20 rows)
- `collect`: returns all the rows in the DataFrame as an array
- `write`: save the data to a file or other data source

take( ) vs show( )

Cmd 11

```
usersDF = spark.read.json("/FileStore/tables/users.json")
```

▸ (1) Spark Jobs

▸ usersDF: pyspark.sql.dataframe.DataFrame = [age: long, name: string ... 1 more field]

take( )

Cmd 12

```
usersDF.take(3)
```

▸ (1) Spark Jobs

```
Out[42]: [Row(age=None, name='Alice', pcode='94304'),
 Row(age=30, name='Brayden', pcode='94304'),
 Row(age=19, name='Carla', pcode='10036')]
```

Python List containing Row objects

show( )

Cmd 13

```
usersDF.show(3)
```

▸ (1) Spark Jobs

```
+----+-------+-----+
| age|   name|pcode|
+----+-------+-----+
|null|  Alice|94304|
|  30|Brayden|94304|
|  19|  Carla|10036|
+----+-------+-----+
only showing top 3 rows
```

DataFrame operations: transformation

- Transformations create a new DataFrame based on an existing one
  - Transformations do not return any values or data to the driver
- The new DataFrame may have the same schema or a different one
  - Data remains distributed across the application's executors
- DataFrames are immutable
  - Data in a DataFrame is never modified
  - Use transformations to create a new DataFrame with the data you need

DataFrame operations: transformation

Common transformations include

- `select`: only the specified columns are included
- `where`: only rows where the specified expression is true are included (synonym for `filter`)
- `orderBy`: rows are sorted by the specified column(s) (synonym for `sort`)
- `join`: joins two DataFrames on the specified column(s)
- `limit(n)`: creates a new DataFrame with only the first `n` rows

Example: limit( ) transformation

Main dataframe

```
+----+-------+-----+
| age|   name|pcode|
+----+-------+-----+
|null|  Alice|94304|
|  30|Brayden|94304|
|  19|  Carla|10036|
|  46|  Diana| null|
|null|Etienne|94104|
+----+-------+-----+
```

Cmd 14

```
usersDF2 = usersDF.limit(2)

usersDF2.show()
```

First two rows of the main dataframe

▸ (1) Spark Jobs

▸ usersDF2: pyspark.sql.dataframe.DataFrame = [age: long, name: string ... 1 more field]

```
+----+-------+-----+
| age|   name|pcode|
+----+-------+-----+
|null|  Alice|94304|
|  30|Brayden|94304|
+----+-------+-----+
```

Example: select () transformation

Select some columns of the main dataframe and store in a new dataframe

Cmd 15

```
usersDF3 = usersDF.select("age")

usersDF3.show()
```

▸ (1) Spark Jobs

▸ usersDF3: pyspark.sql.dataframe.DataFrame = [age: long]

```
+----+
| age|
+----+
|null|
|  30|
|  19|
|  46|
|null|
+----+
```

Cmd 15

```
usersDF3 = usersDF.select("age","pcode")

usersDF3.show()
```

▸ (1) Spark Jobs

▸ usersDF3: pyspark.sql.dataframe.DataFrame = [age: long, pcode: string]

```
+----+-----+
| age|pcode|
+----+-----+
|null|94304|
|  30|94304|
|  19|10036|
|  46| null|
|null|94104|
+----+-----+
```

Example: where () transformation

Main dataframe

```
+----+-------+-----+
| age|   name|pcode|
+----+-------+-----+
|null|  Alice|94304|
|  30|Brayden|94304|
|  19|  Carla|10036|
|  46|  Diana| null|
|null|Etienne|94104|
+----+-------+-----+
```

Cmd 16

```
usersDF4 = usersDF.where("age is not null")

usersDF4.show()
```

▸ (1) Spark Jobs

▸ usersDF4: pyspark.sql.dataframe.DataFrame = [age: long,

```
+---+-------+-----+
|age|   name|pcode|
+---+-------+-----+
| 30|Brayden|94304|
| 19|  Carla|10036|
| 46|  Diana| null|
+---+-------+-----+
```

Transformed dataframes

Cmd 16

```
usersDF4 = usersDF.where("age > 21")

usersDF4.show()
```

▸ (1) Spark Jobs

▸ usersDF4: pyspark.sql.dataframe.DataFrame =

```
+---+-------+-----+
|age|   name|pcode|
+---+-------+-----+
| 30|Brayden|94304|
| 46|  Diana| null|
+---+-------+-----+
```

What is “Query”?

A sequence of transformations followed by an action is a query.

A query

Cmd 17

```
usersDF3 = usersDF.select("age","pcode")

usersDF4 = usersDF3.where("age > 20")

usersDF4.show()
```

(1) Spark Jobs

usersDF3: pyspark.sql.dataframe.DataFrame = [age: long, pcode: string]

usersDF4: pyspark.sql.dataframe.DataFrame = [age: long, pcode: string]

```
+---+-----+
|age|pcode|
+---+-----+
| 30|94304|
| 46| null|
+---+-----+
```

Chaining transformations

Same result as the previous slide

```
usersDF4 = usersDF.select("age","pcode").where("age > 20")

usersDF4.show()
```

▸ (1) Spark Jobs

▸ usersDF4: pyspark.sql.dataframe.DataFrame = [age: long, pcode: string]

```
+---+-----+
|age|pcode|
+---+-----+
| 30|94304|
| 46| null|
+---+-----+
```

Different ways to access columns

```
df = spark.createDataFrame(
    [("G1", None, 19 , 15), ("G2", 12, 31 , 12), ("G3", 8, 11 , 18), ("G4", 12, 6 , 41)],
    ['Group', 'week1' , 'week2' , 'week3'])
```

df: pyspark.sql.dataframe.DataFrame = [Group: string, week1: long ... 2 more fields]

Command took 0.22 seconds --

Cmd 21

```
df.show()
```

(3) Spark Jobs

```
+-----+-----+-----+-----+
|Group|week1|week2|week3|
+-----+-----+-----+-----+
|   G1| null|   19|   15|
|   G2|   12|   31|   12|
|   G3|    8|   11|   18|
|   G4|   12|    6|   41|
+-----+-----+-----+-----+
```

* df.select(df.week1)

df.select("week1")

df.select(df[1])

```
+-----+
|week1|
+-----+
| null|
|   12|
|    8|
|   12|
+-----+
```

* This operation doesn't work with column names containing space or special characters or starting with numbers (week   1 , 1week , #week1)

Column Expressions

- Using column references instead of simple strings allows you to create column expressions
- Column operations include
  - Arithmetic operators such as +, -, %, /, and *
  - Comparative and logical operators such as >, <, && and ||
    - The equality comparator is == in Python
  - String functions such as `contains`, `like`, and `substr`
  - Data testing functions such as `isNull`, `isNotNull`, and `NaN` (not a number)
  - Sorting functions such as asc and desc
    - Work only when used in `sort`/`orderBy`
- For the full list of operators and functions, see the API documentation for `Column`

SCHOOL OF SCIENCE, ENGINEERING & ENVIRONMENT

Example: Arithmetic operation on a column

Cmd 25

```
myDF.select(myDF.lastName , myDF.age+2).show()
```

▸ (2) Spark Jobs

```
+----------+---------+
|  lastName|(age + 2)|
+----------+---------+
|   Hopper |       54|
|   Turing |       34|
| Lovelace |       30|
|  Babbage |       51|
|    Wirth |       50|
+----------+---------+
```

Example: Arithmetic operation on a column

Cmd 23

```
1   myDF.select("lastName", myDF.age+2).show()
```

▸ (2) Spark Jobs

```
+----------+---------+
|  lastName|(age + 2)|
+----------+---------+
|   Hopper |       54|
|   Turing |       34|
| Lovelace |       30|
|  Babbage |       51|
|    Wirth |       50|
+----------+---------+
```

Example: String functions on a column

Cmd 24

```
1   myDF.where(myDF.firstName.startswith("A")).show()
```

▸ (2) Spark Jobs

```
+------+----------+---------+---+
| pcode|  lastName|firstName|age|
+------+----------+---------+---+
|94020 |   Turing |     Alan | 32|
|94020 | Lovelace |      Ada | 28|
+------+----------+---------+---+
```

Example: Sorting based on a column

Cmd 26

```
1  myDF.sort(myDF.age.desc()).show()
```

▸ (1) Spark Jobs

```
+------+----------+---------+---+
| pcode|  lastName|firstName|age|
+------+----------+---------+---+
|02134 |   Hopper |   Grace | 52|
|87501 |  Babbage | Charles | 49|
|02134 |    Wirth | Niklaus | 48|
|94020 |   Turing |    Alan | 32|
|94020 | Lovelace |     Ada | 28|
+------+----------+---------+---+
```