THE DATA SCIENCE LAB
Data Wrangling with Hadoop

COM 490 – Spring 2022

Week 6

THIS CLASS WILL BE RECORDED

EPFL

Agenda Spring 2022

M1

- 23.02
  - General introduction to Data Science
  - Python 3.x, Numpy, Scipy, Pandas, Matplotlib, Scikit-learn
- 02.03
  - Collaborative Data Science
  - Jupyter notebooks, Docker, Git(lab), interactive visualization

M2

- 09.03
  - General introduction to big data, best practices and guidelines
  - HDFS, Hive, Loading and querying data, assessed project 1
- 16.03
  - Data wrangling with Hadoop
  - Parquet, ORC, HBase
- 23.03
  - Introduction to the Spark Runtime architecture
  - Python on Spark, Basic RDD manipulation; **assessed project 1 due**

M3

- 30.03
  - Spark data Frames
  - Basic data analytics with Spark Data Frames
- 06.04
  - Advanced Spark, optimizations and partitioning
  - Data partitioning; **assessed project 2 is due**

M4

- 13.04
  - Introduction to data stream processing
  - Stream processing with Apache Kafka
- 20.04
  - Spring break 15.04 – 25.04
- 27.04
  - Advanced data stream processing
  - Spark stream, data stream windows, Kafka-Spark; **assessed project 3 is due**
- 04.05
  - Analytics on data at rest and data in motion
  - **Final project is announced**

Rev

- 11.05
  - Putting it all together - building scalable applications with real data
  - **Assessed project 4 is due**

Fin

- 18.05
  - Final project: tips and hints
- 25.05
  - Final project: tips and hints
- 01.06
  - **Final project is due Sunday 29.05** : code and a short video presentation
  - Oral sessions on 01.06

*Details on Moodle

Week 5 Recap

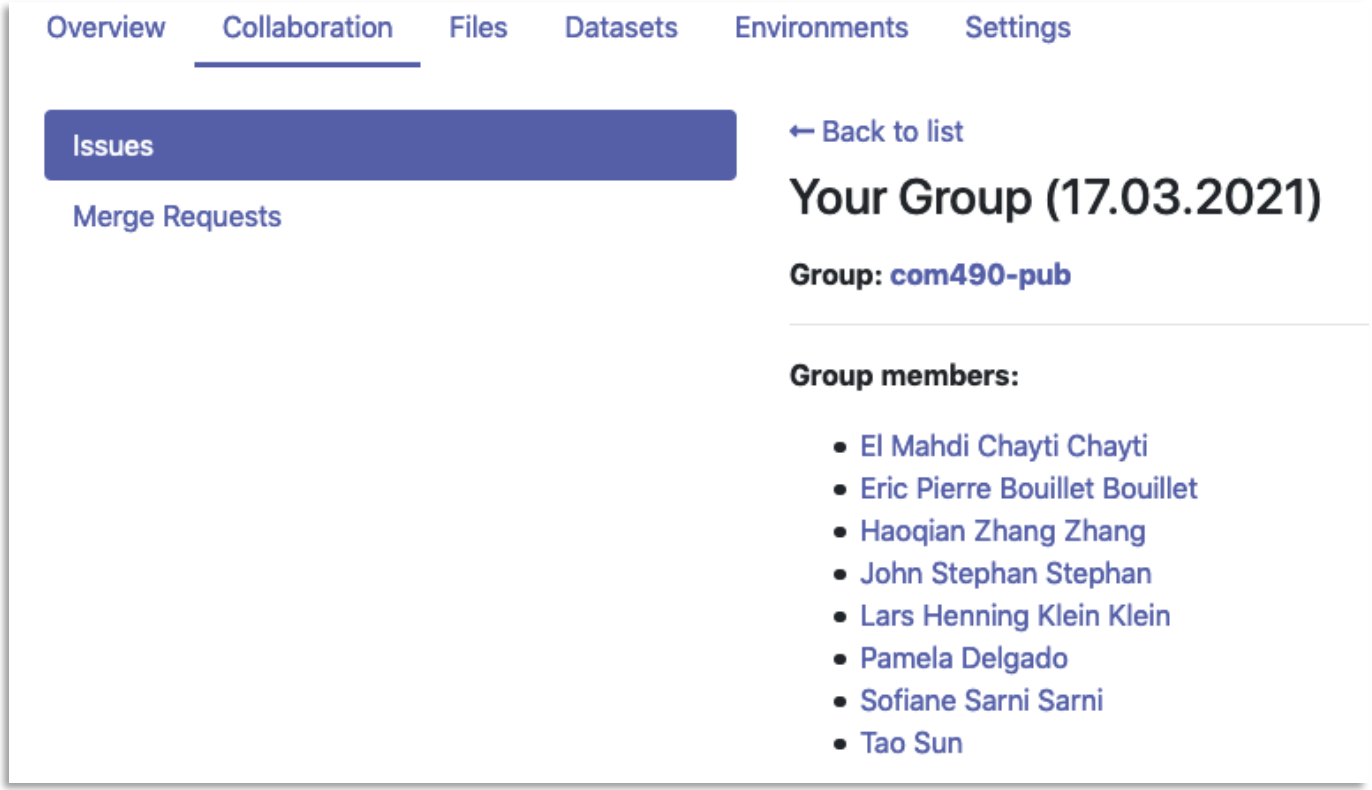
- What is Spark?
- RDDs
  - Immutable
  - Resilient
  - Lineage
- Operations on RDDs
  - Transformations
  - Actions
  - Lazy execution
- Exercises to get started using Guttenberg corpus

Week 5 – Questions?

Week 6 – Homework 2 : Checkpoint

Have you seen this in your homework project ?

If not, contact us !



Today's Agenda

- Introduction to Data Frames
- DataFrame demo
- DataFrames and PySpark under the hood
- Exercises week 6
  - Guttenberg corpus part II

Introduction to Spark DataFrames

EPFL

RDD Revisited

- Resilient (lineage)
- Distributed (partitioned)
- Unstructured (mostly), rows or key-row pairs
- Type safe
  - Scala's compiler optimization
- Use of lambda functions
- Fine grained control – tell spark how to transform a data
  - low level – more responsibility to the programmer:
    - Decide transformations and actions
    - Which part of the data
    - In what order

Spark DataFrames

- Distributed Collections of Data
  - Organized into rows of named columns
  - Very much like relational database Tables
  - Optimized for relational-type of queries on tables (logical plan optimization)

Structured

Col 1	Col 2	Col 3
1	a	10:00
2	b	11:00
3	c	12:00
4	d	13:00
5	e	14:00

Origin of Data Frames

Spark Data Frame API Inspired by R and Python's Pandas

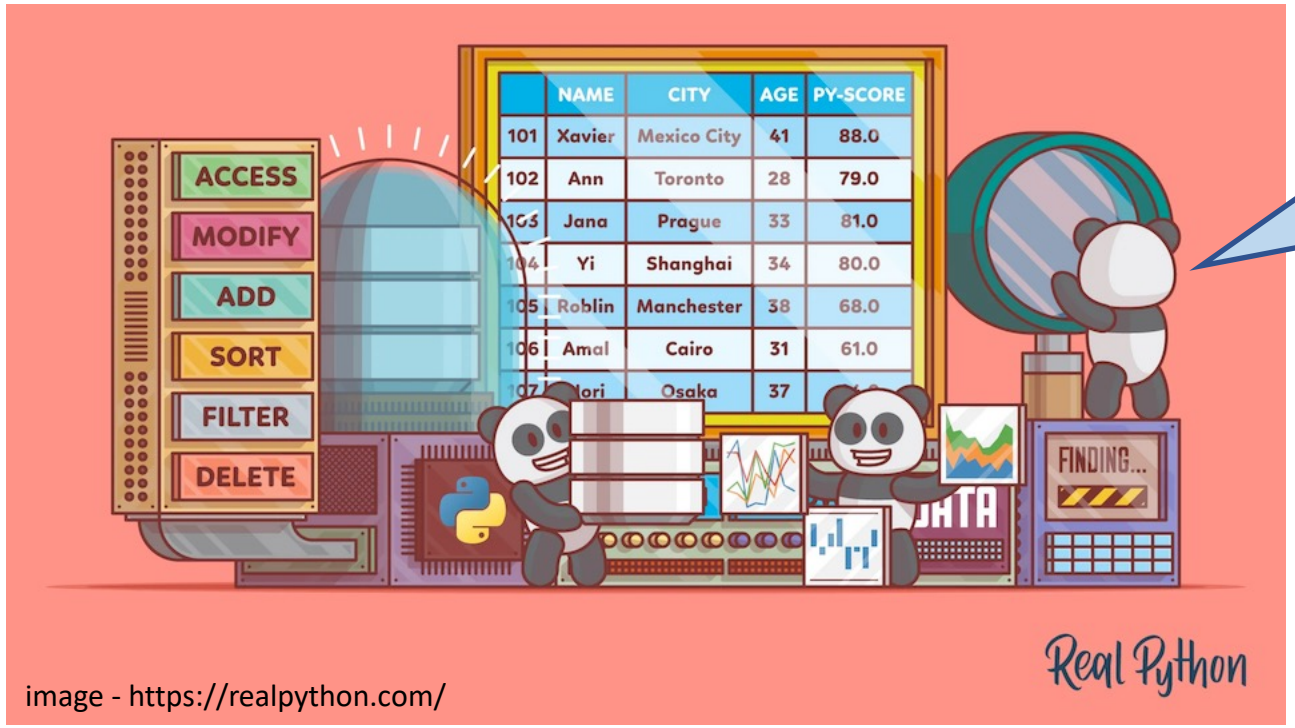


image - https://realpython.com/

Want to join the Big Data party!

What are Spark Data Frames

- Inspired by R and Python Pandas

SOURCE

- Local File Systems
- Distributed File Systems (HDFS)
- Cloud Storage (S3)
- External data bases
- Spark RDD

DATA FORMAT – out of the box

- TEXT
- JSON
- CSV
- Parquet
- ORC
- Hive Table

+ **Other with plugins** (Avro, ElasticSearch, Cassandra, ...)

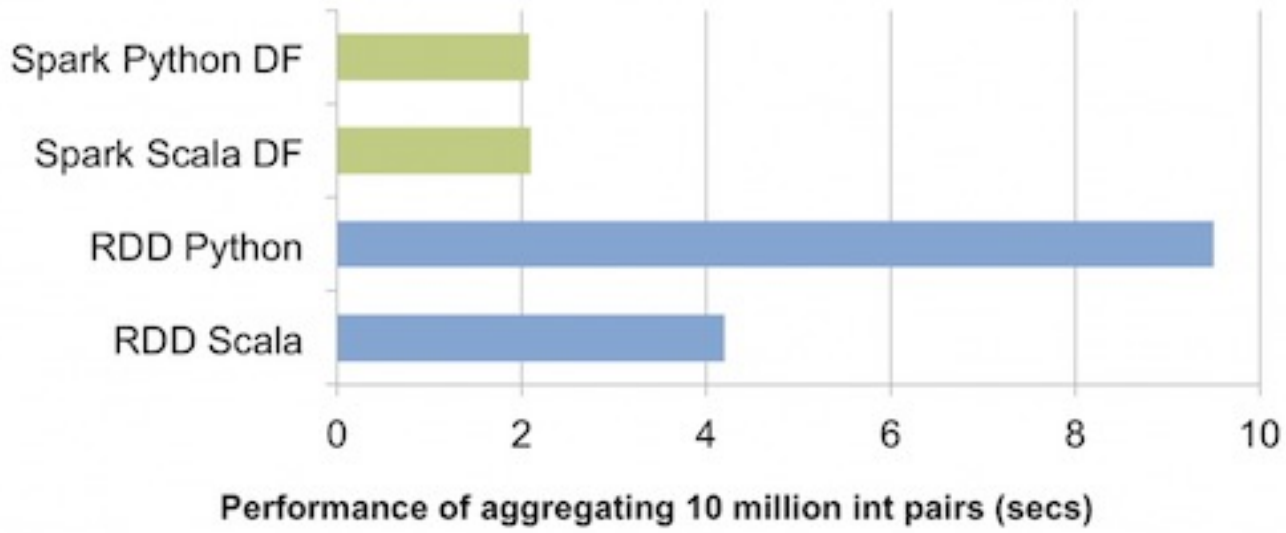
- Parallelism & query optimizer, unlike R and Python

Why Spark Data Frames – RDD vs DataFrame

Resilient Distributed Dataset (RDD)	Data Frame
Structured & unstructured data	Structured data (table, named columns)
Schema must be declared manually	Auto discovery of file schema
Lambda functions (map, reduce)	Declarative, almost as SQL queries
Lower level language	Higher level language
No built-in other than generic compiler optimization. Must be done manually	Execution optimization
Type safety at compile time	Type safety at run-time (e.g. trying to access a non-existing column)

Spark DataFrame performance

- In DataFrame data is optimally managed off-heap (off JVM)
  - No need for Java/Scala (de)serialization when accessing object, unlike RDD
  - Avoid garbage collection, unlike RDD
- Aggregation (group by) is harder and not as efficient with RDD. In comparison exploration analysis is quick and easier on large DataFrame



databrkick blog 2015
(*) to be taken with a grain of salt

Which one should I use? RDD vs DataFrame

- Use RDD for operations that requires low level functionalities and control on unstructured data
- Use DataFrame for high level (SQL like) operations on structured data

DataFrame under the hood

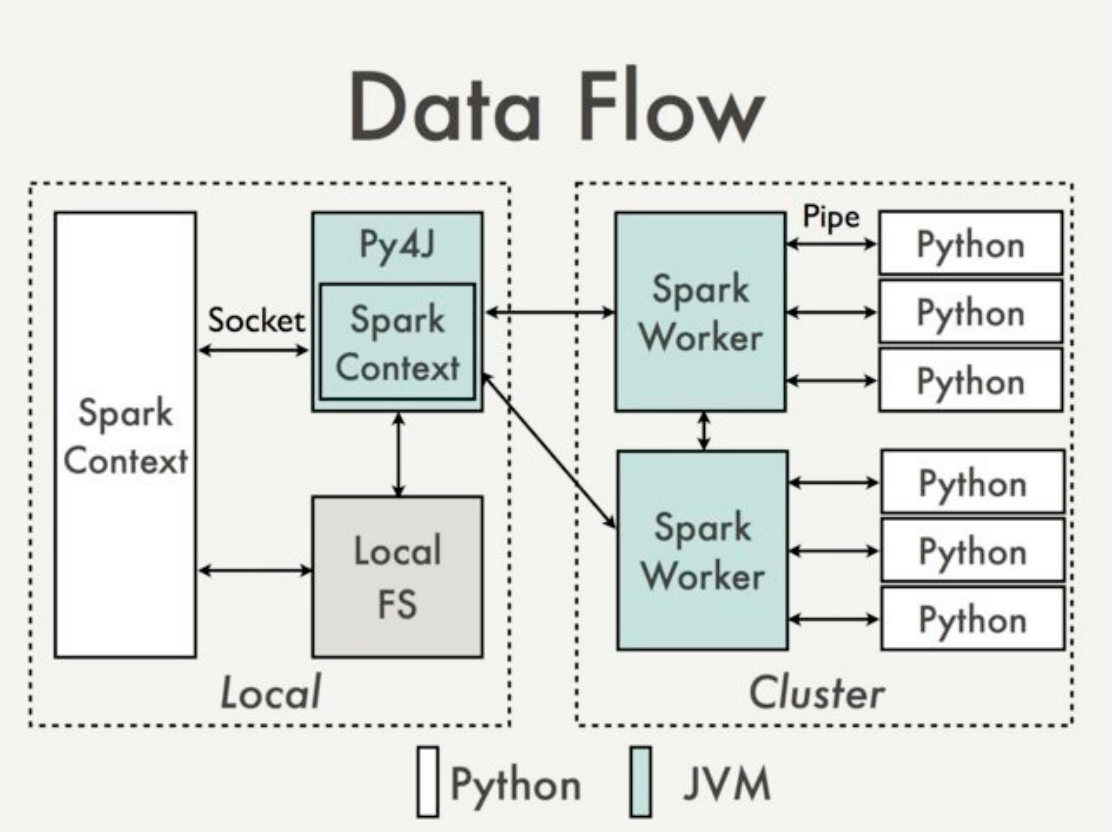
PySpark

- PySpark - Python front end API for Spark

- Interface RDDs with Python
- Py4J – python library to dynamically access JVM objects
- Compatible with
  - PySparkSQL – SQL query library for DataFrame
  - MLib – Machine learning library
  - GraphFrames – Graph processing based on DataFrames (Graphx is on RDDs)

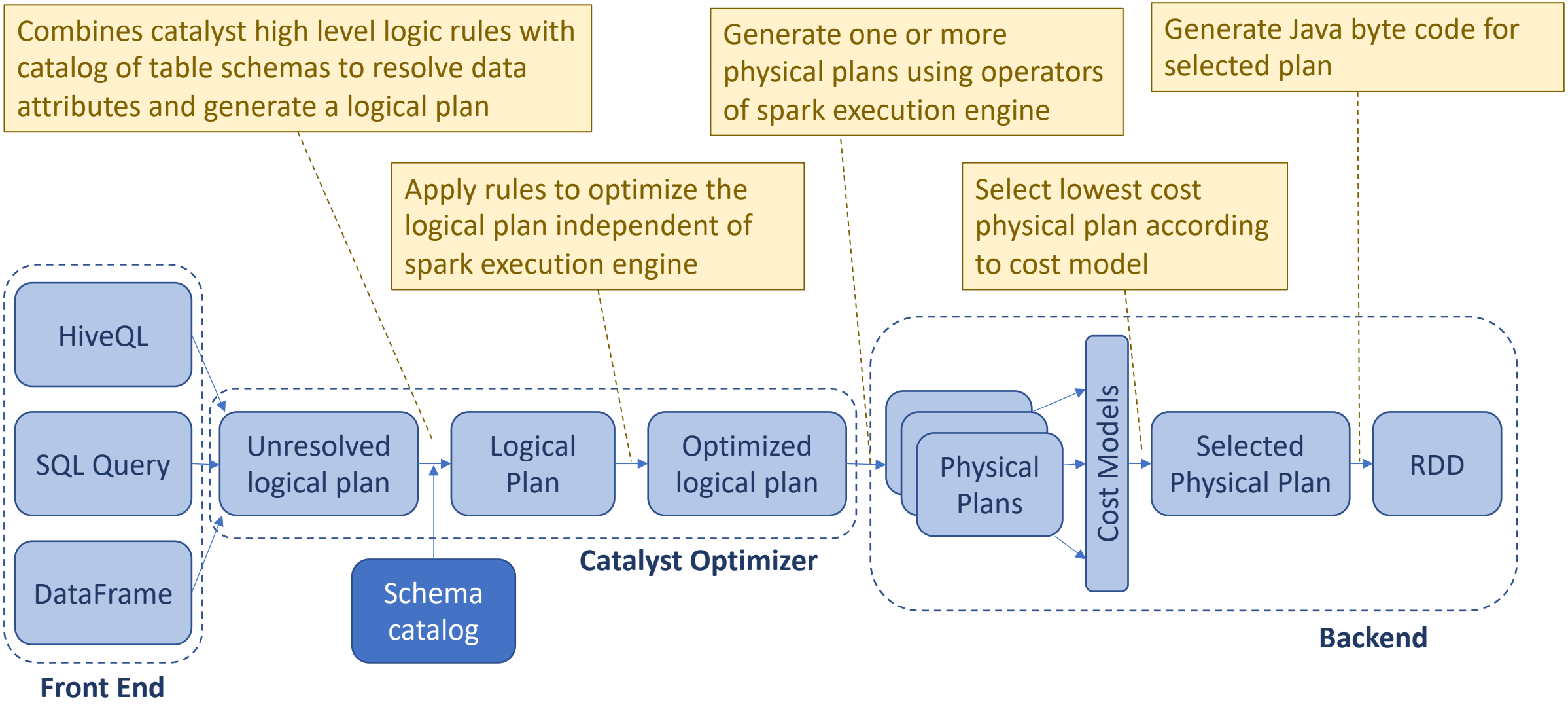
PySpark

- Spark workers pull data from source into JVM
- Data is actually processed into python subprocesses
- (de)serialization and streaming at every step



PySpark internal

Catalyst Optimizer



A parting word on Spark DataSets

- DataSet = extensions of DataFrame with convenience of RDD.
  - Strong type safety
  - RDD with Spark SQL optimized execution engine
  - Operate on serialized data (no deserialization overhead)
- Available only on Scala and Java
  - Since 1.6 DataFrame on Scala and Java are alias for DataSet[row]

Useful References

- Spark docs http://spark.apache.org/docs/latest
- DataFrames and code generation https://medium.com/virtuslab/spark-sql-under-the-hood-part-i-26077f85ebf0
- Python Spark DataFrames starter documentation https://spark.apache.org/docs/latest/api/python/getting_started/quickstart_df.html
- Spark MLlib guide https://spark.apache.org/docs/latest/ml-guide.html

Start your engines

https://dslab2022-renku.epfl.ch/projects/com490/lab-course

EPFL