Analyse Exploratoire et Prédictive des Données Massives

- avec Apache Spark Core, Apache Spark SQL et Apache Spark ML -

Khaled Jouini

j.khaled@gmail.com

ISITCom, University of Sousse

Ce document peut comporter des erreurs, à utiliser donc en connaissance de cause!

Plan du cours - Partie 1

Plan du cours - Partie 2

Lectures intéressantes (1/2)

- *Big Data Analytics with Spark: A Practitioner's Guide to Using Spark for Large-Scale Data Processing, Machine Learning, and Graph Analytics, and High-Velocity Data Stream Processing*.
  Mohammed Guller
  Apress, 9781484209646. 2016.

- *Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing.*.
  Zaharia, M., Chowdhury, M., Franklin, M. J., Shenker, S., & Stoica, I.
  Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI '12), 15-28. 2012.

- *Spark SQL: Relational Data Processing in Spark.*.
  Armbrust, M., Xin, R. S., Lian, C., Huai, Y., Liu, D., Bradley, J. K., & Zaharia, M.
  Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, 1383-1394. 2015.

- *MLlib: Machine Learning in Apache Spark.*.
  Meng, X., Bradley, J., Yuvaz, B., Sparks, E., Venkataraman, S., Liu, D. & Zaharia, M.
  Journal of Machine Learning Research, 17(34), 1-7. 2016.

Lectures intéressantes (2/2)

- *Learning Spark Lightning-Fast Data Analytics*.
  Damji J.S., Wenig B., Wentling T., Das T. & Lee D.
  eBook, O'Reilly, 978-1-098-15195-9. July 2020.
  Disponible sur https://pages.databricks.com/rs/094-YMS-629/images/LearningSpark2.0.pdf.

- *Documentation officielle d'Apache Spark*

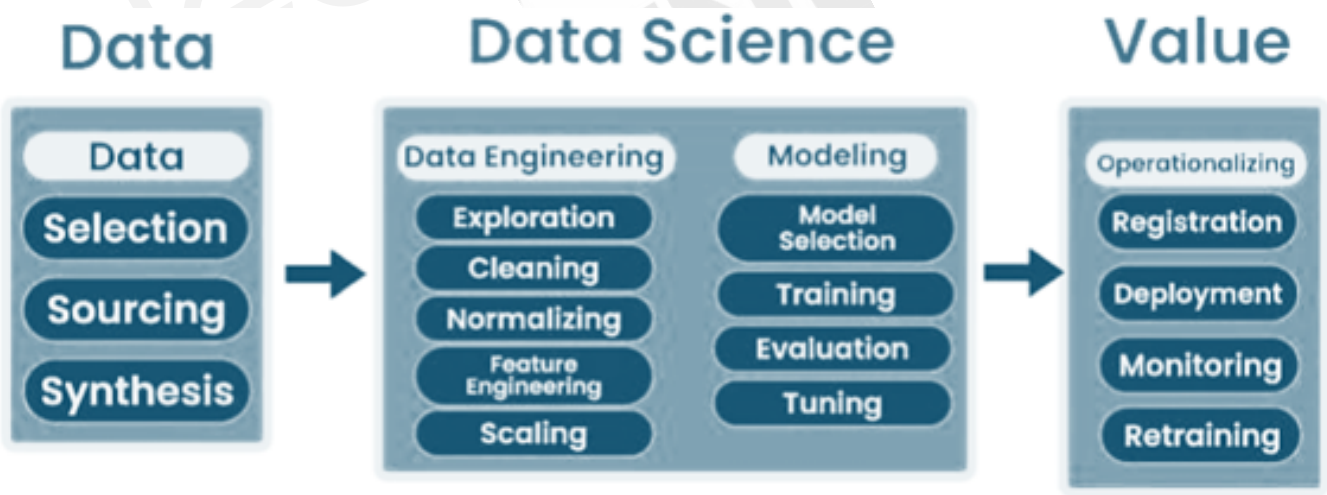
1. Spark, quésaco?

Apache Spark : **moteur d'exécution** (pas de stockage de données!) open source pour le **traitement distribué**, pouvant être utilisé pour une variété de tâches, notamment l'**ingénierie des données** et la **science des données**.

Support pour SQL, ETL/ELT, Machine Learning/Deep Learning, Graph, etc..

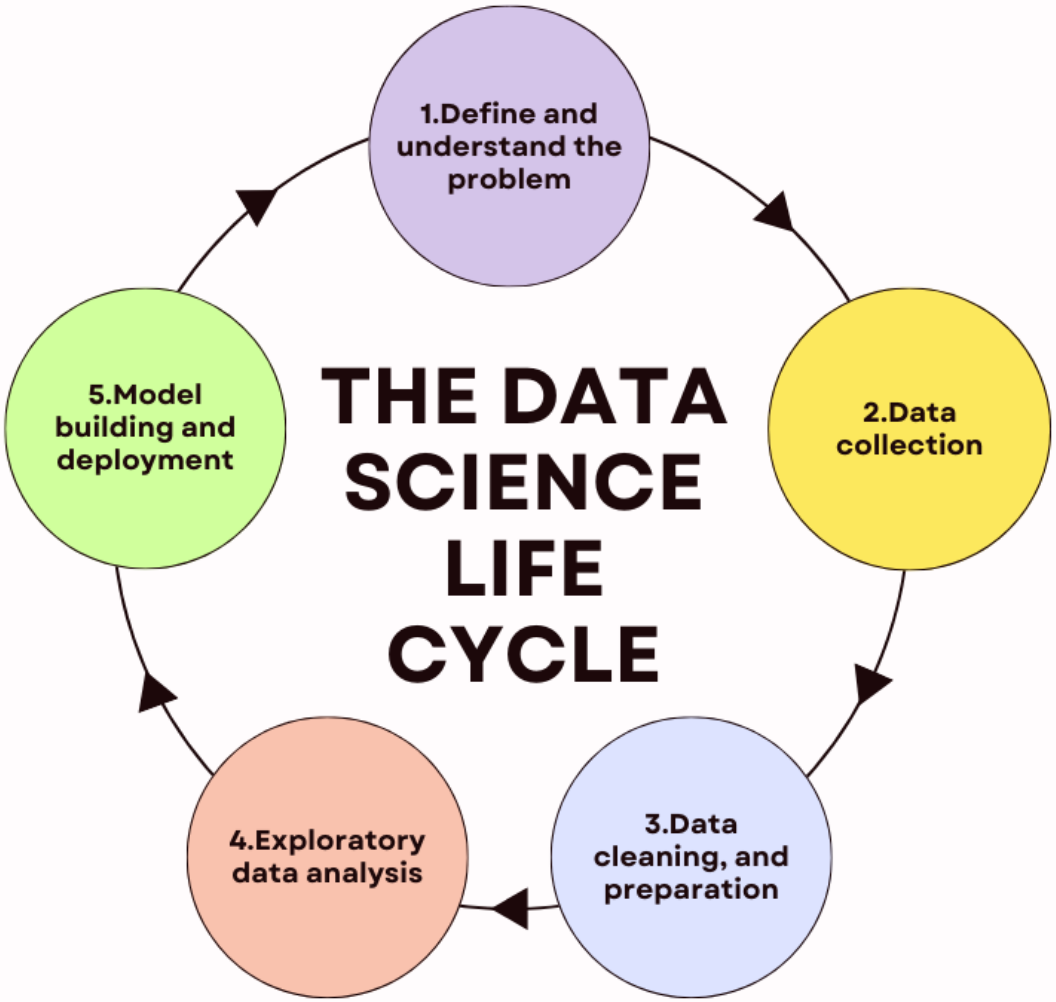
Standard de-facto pour le Big Data (Science)

Entreprises utilisant Spark : Netflix pour la recommandation, Uber pour le dispatching, etc.



(Source : Databricks)

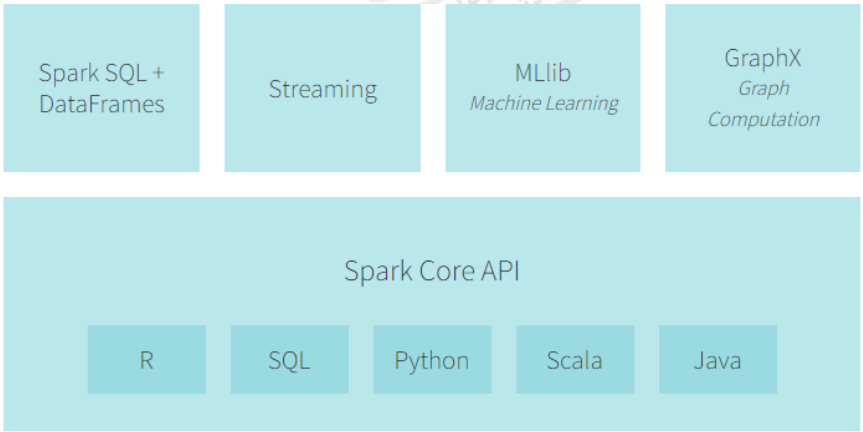
1. Spark, quésaco?



1. Spark, quésaco?

Unified Analytics Engine for Big Data :

- **Spark Core** : Le noyau de Spark, fournit les fonctionnalités de base pour le traitement de données distribuées.
- **SparkSQL** : Pour le traitement des données structurées et semi-structurées. Permet entre autres d'utiliser SQL sur les collections de données.
- **Spark Streaming** : traitement de flux de données en temps réel
- **Spark ML** : algorithme de machine learning
- **Spark Graphx** : sharding et interrogation de données en graphe.



1. Spark vs Hadoop

Spark : Permet le traitement des données en mémoire, ce qui le rend jusqu'à 100 fois plus rapide que Hadoop pour certaines applications.

Hadoop (MapReduce) : Utilise un modèle basé sur des étapes successives de lecture-écriture sur disque, ce qui entraîne des latences plus élevées pour les traitements itératifs comme les algorithmes de machine learning.

- **Traitement itératif** : Spark surpasse Hadoop dans des scénarios où des algorithmes doivent passer plusieurs fois sur les données (ex. machine learning).

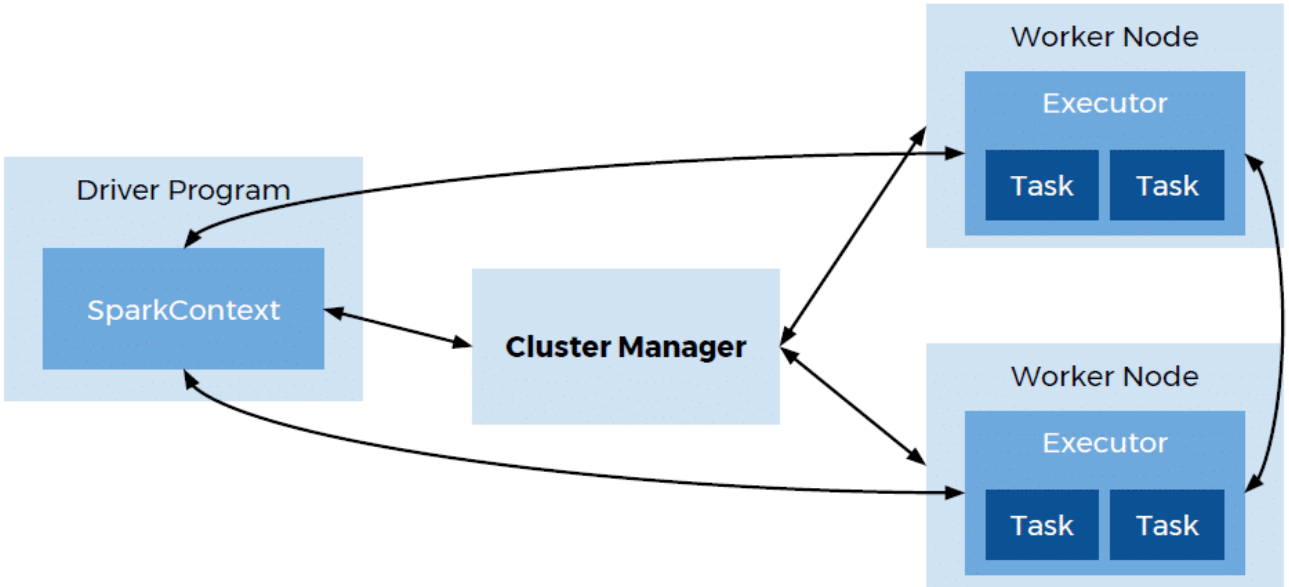
- **Applications en streaming** : Spark Streaming permet le traitement en temps réel, contrairement à Hadoop MapReduce qui se concentre sur le traitement par lots.

1. Architecture de Spark

Driver : Le Driver est le nœud principal dans Spark, il contrôle l'exécution des jobs et distribue les tâches aux nœuds de calcul (Executors).

Executors : Ce sont les nœuds qui exécutent les tâches distribuées et renvoient les résultats au Driver.

Cluster Manager : Spark peut fonctionner avec différents gestionnaires de cluster (**YARN**, Mesos, Kubernetes) pour orchestrer les ressources de calcul.



Chapitre 2 - Fondements de Spark

2.1. RDD (Resilient Distributed Dataset)

Spark : idée essentielle, on compose des **chaînes de traitement** avec des **opérateurs de haut niveau** et les **données sont constamment gardés en mémoire vive**.

RDD : Techniquement, une **collection** d'éléments ou d'**objets** Scala, **immuables** (non modifiables), **partitionnée** entre les executors du cluster

Les **opérations** sur un RDD prennent la forme d'une **séquence**.

RDD Transformations

```
words = sc.textFile("hdfs://large/file/")
          .map(_.toLowerCase)
          .flatMap(_.split(" "))
alpha = words.filter(_.matches("[a-z]+"))
nums  = words.filter(_.matches("[0-9]+"))
```

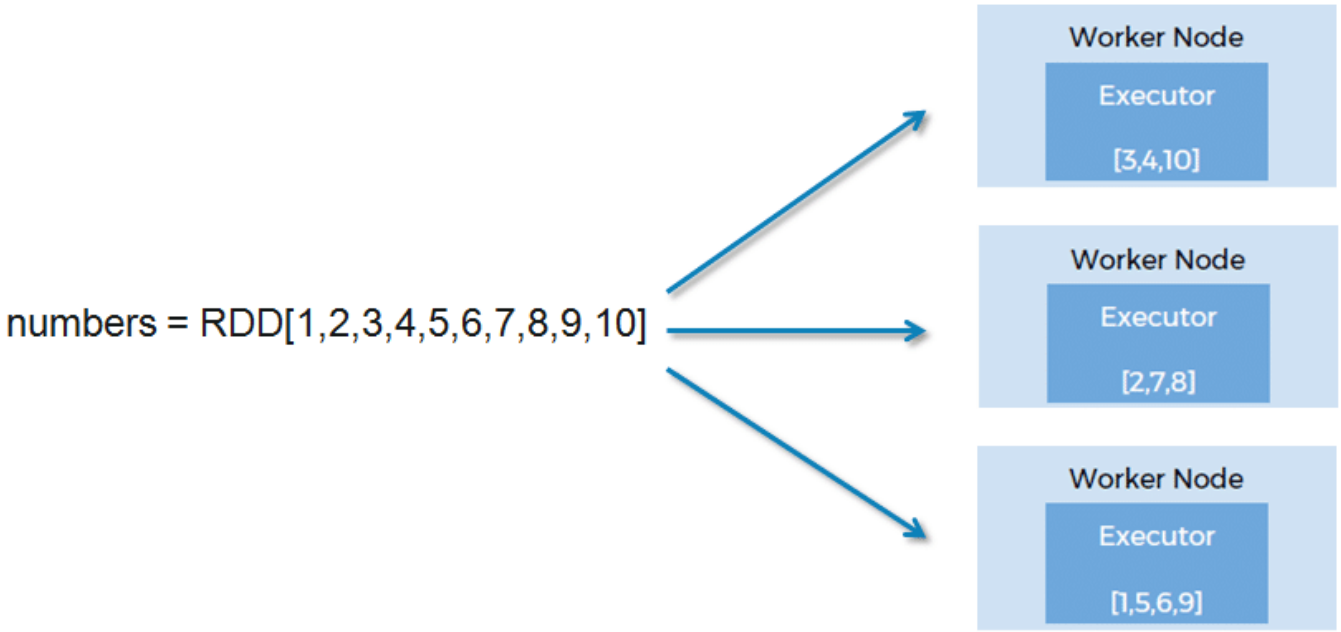
```
alpha.count()
```

Action (run job on the cluster)

2.1. RDD (Resilient Distributed Dataset)

Resilient Distributed Dataset (RDD)

- **Immuable** : Une fois créé, un RDD ne peut pas être modifié. Chaque transformation crée un nouveau RDD.
- **Distribué** : Les données sont automatiquement partitionnées et distribuées sur plusieurs nœuds d'un cluster.
- **Tolérant aux pannes** : En cas de panne d'un nœud, le RDD peut être recréé à partir de son **lineage**.



Chapitre 2 - Fondements de Spark

2.2. Transformations vs Actions

- Les **opérations** appliquées à un RDD sont soit des **transformations** soit des **actions**

- **Transformation** : ne modifie pas un RDD, mais dérive un nouvel RDD enrichi de nouvelles informations correspondant à cette transformation.

  - *e.g.* `val aLines = distFile.filter(l => l.contains("a"))`
    créé un nouvel RDD ne contenant que les lignes contenant la lettre "a".

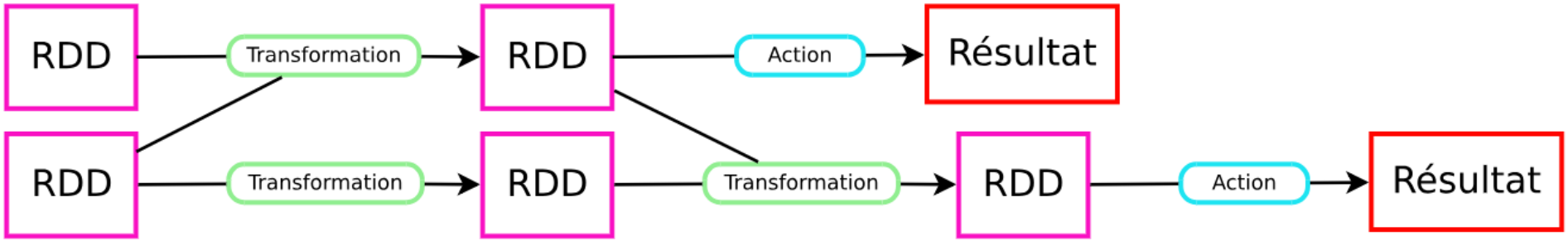
  - *e.g.* `val lineLengths =distFile.map(l => l.length)`
    créé un nouvel RDD contenant la longueur des lignes.

- **Action** : retourne une valeur après avoir effectué un certain calcul

  - *e.g.* `int totalLength = lineLengths.reduce(_+_)`
    retrourne la somme des éléments du RDD lineLength

`reduce` peut s'écrire `.reduce((a,b)=>a+b)`. "_" est caractère générique dans Scala. Ici il fait référence aux éléments du tableau.

2.2. Transformations vs Actions



Opérations sur un RDD

- Les transformations qui donnent en sortie un autre RDD.
- Les actions qui donnent en sortie... autre chose qu'un RDD.
- **Data Lineage** : Les transformations et les actions réalisées sur les RDD permettent de construire un graphe acyclique orienté (DAG : "directed acyclic graph")

2.2. Transformations

Opérateur	Description
`map`	Prend un document en entrée et produit un document en sortie
`filter`	Filtre les documents de la collection
`flatMap`	Prend un document en entrée, produit un ou plusieurs document(s) en sortie
`groupByKey`	Regroupement de documents par une valeur de clé commune
`reduceByKey`	Réduction d'une paire *(k, [v])* par une agrégation du tableau *[v]*
`crossProduct`	Produit cartésien de deux collections
`join`	Jointure de deux collections
`union`	Union de deux collections
`cogroup`	Cf. la description de l'opérateur dans la section sur Pig
`sort`	Tri d'une collection

2.2. Transformations

- Transformations : peuvent produire
  - un **RDD à partir d'un autre RDD** : map, filter, sort, etc.
  - un **RDD à partir de deux RDD** : union, join, crossJoin, etc.
  - **0 ou plusieurs RDD à partir d'un RDD** : flatMap

```
texteLicence.flatMap(line => line.split(" ")).groupByKey(identity).count().show(10)
```

2.2. Actions

Action	Meaning
reduce(func)	Aggregate the elements of the dataset using a function func (which takes two arguments and returns one). The function should be commutative and associative so that it can be computed correctly in parallel.
collect()	Return all the elements of the dataset as an array at the driver program. This is usually useful after a filter or other operation that returns a sufficiently small subset of the data.
count()	Return the number of elements in the dataset.
first()	Return the first element of the dataset (similar to take(1)).
take(n)	Return an array with the first n elements of the dataset.
takeSample(withRep, num, [seed])	Return an array with a random sample of num elements of the dataset, with or without replacement, optionally pre-specifying a random number generator seed.
takeOrdered(n, [ordering])	Return the first n elements of the RDD using either their natural order or a custom comparator.
saveAsTextFile(path)	Write the elements of the dataset as a text file (or set of text files) in a given directory in the local filesystem, HDFS or any other Hadoop-supported file system. Spark will call toString on each element to convert it to a line of text in the file.
saveAsSequenceFile(path)	Write the elements of the dataset as a Hadoop SequenceFile in a given path in the local filesystem, HDFS or any other Hadoop-supported file system. This is available on RDDs of key-value pairs that implement Hadoop's Writable interface. In Scala, it is also available on types that are implicitly convertible to Writable (Spark includes conversions for basic types like Int, Double, String, etc).
saveAsObjectFile(path) (Java and Scala)	Write the elements of the dataset in a simple format using Java serialization, which can then be loaded usingSparkContext.objectFile().
countByKey()	Only available on RDDs of type (K, V). Returns a hashmap of (K, Int) pairs with the count of each key.
foreach(func)	Run a function func on each element of the dataset. This is usually done for side effects such as updating an accumulator or interacting with external storage systems.

Shuffle : Certaines actions comme join, reduceByKey, etc. déclenchent un shuffle, i.e. la redistribution des données entre les exécutors. Ces actions sont couteuses et complexes à exécuter

Chapitre 2 - Fondements de Spark

2.3. Évaluation paresseuse

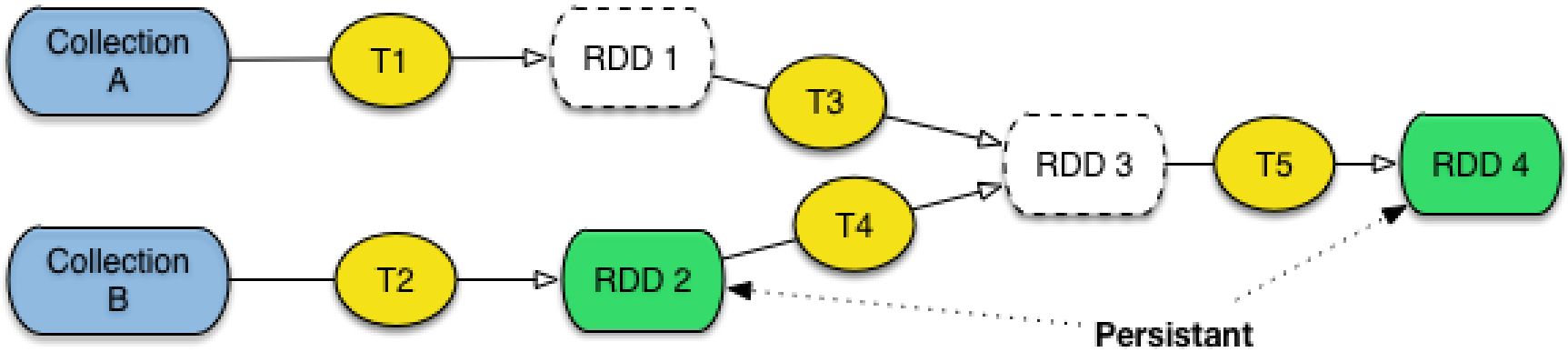
- Transformations : opérations **lazy** ou à évaluation **paresseuse**, elles ne vont lancer aucun calcul sur un Cluster (même principe que les vues dans le relationnel).
- Les transformations ne sont exécutées que lors du lancement d'une action
- Ceci permet à spark de libérer la mémoire vive et de ne pas avoir à retourner au driver les résultats intermédiaires d'une chaîne d'opérations d'être donc plus performant
- **collect()**
  - Récupère tous les éléments d'un RDD distribué et les renvoie vers le nœud principal (driver).
  - Doit être utilisée avec prudence, car elle peut entraîner un transfert massif de données du cluster vers le nœud principal. Cela peut causer des problèmes de performance voire des échecs si la taille des données est très importante.

2.3. Évaluation paresseuse

- Si un RDD est ré-utilisé dans différents traitements et que nous ne voulons pas pas le re-caluculer à chaque fois, il est posible de le persister (matérialiser)

- `distFiles.persist()`

- Le résultat est gardé en mémoire vive (mais peut également être écrit sur disque)



2.3. Évaluation paresseuse

- `persist()` et `cache()`
  - Rendent un RDD "persistant" : les éléments sont collectés en mémoire pour éviter de ré-appliquer la chaîne de traitement qui les a créés.
  - Différents niveau de persistance sont possible. La méthode cache() est un raccourci de `persist(StorageLevel.MEMORY_ONLY)` (stockage en mémoire vive sous la forme d'objets Java désérialisés)

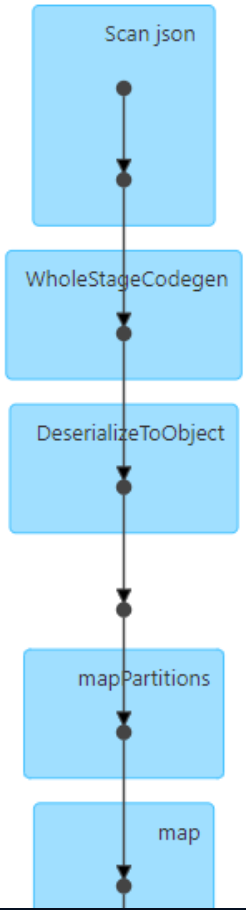
Storage Level	Meaning
MEMORY_ONLY	Store RDD as deserialized Java objects in the JVM. If the RDD does not fit in memory, some partitions will not be cached and will be recomputed on the fly each time they're needed. This is the default level.
MEMORY_AND_DISK	Store RDD as deserialized Java objects in the JVM. If the RDD does not fit in memory, store the partitions that don't fit on disk, and read them from there when they're needed.
MEMORY_ONLY_SER (Java and Scala)	Store RDD as *serialized* Java objects (one byte array per partition). This is generally more space-efficient than deserialized objects, especially when using a fast serializer, but more CPU-intensive to read.
MEMORY_AND_DISK_SER (Java and Scala)	Similar to MEMORY_ONLY_SER, but spill partitions that don't fit in memory to disk instead of recomputing them on the fly each time they're needed.
DISK_ONLY	Store the RDD partitions only on disk.
MEMORY_ONLY_2, MEMORY_AND_DISK_2, etc.	Same as the levels above, but replicate each partition on two cluster nodes.
OFF_HEAP (experimental)	Similar to MEMORY_ONLY_SER, but store the data in off-heap memory. This requires off-heap memory to be enabled.

Chapitre 2 - Fondements de Spark

2.4. Data Lineage et Reprise sur panne

Reprise sur panne

- Spark conserve l'historique des opérations qui ont permis de constituer un RDD (**Data Lineage**)
- En cas de panne sur un nœud, le système détecte les RDD manquants et relance leur lineage.



Il est possible de créer des **points de restauration** (checkpoints) de l'état d'un RDD (`RDD.checkpoint()`) pour une reprise sur panne plus rapide.

Chapitre 2 - Fondements de Spark

Quiz : Sciences des données et Spark

Questions :

1. En quoi consiste le rôle du **Driver** et des **Executors** dans l'architecture de Spark ?
2. Expliquez le concept de **Resilient Distributed Dataset (RDD)**. Pourquoi est-il qualifié de "résilient" ?
3. Quelle est la différence entre une transformation et une action dans Spark ? Donnez un exemple pour chaque type d'opération.
4. Qu'est-ce qu'un **DAG (Directed Acyclic Graph)** et quel rôle joue-t-il dans Spark ?
5. Pourquoi Spark est-il souvent décrit comme étant "jusqu'à 100 fois plus rapide que Hadoop" dans certains scénarios ?
6. Que se passe-t-il lors d'un **shuffle** dans Spark et pourquoi le shuffle est-il une opération coûteuse ?
7. Comment Spark gère-t-il la tolérance aux pannes dans le cas d'un échec d'un nœud pendant le traitement ?
8. Expliquez le principe de l'évaluation paresseuse ?

Chapitre 3 - Analyse exploratoire des données avec Saprk SQL

3.1. Spark SQL

- **Spark SQL** : module de traitement de données au-dessus de Spark permettant d'exécuter des requêtes SQL sur des données structurées.

- Principales utilisations et fonctionnalités de Spark SQL :

  1. **Traitement de données structurées** : traitement des données structurées à l'aide de DataFrame et DataSet, qui sont des abstractions des RDD, similaires aux tables dans une base de données relationnelle.

  2. **Intégration avec des sources de données diverses** : Intégration avec une variété de sources de données, notamment les fichiers CSV, JSON, Parquet, JDBC, Hive, HDFS, etc. Il permet également de traiter des données en streaming à l'aide de Spark Structured Streaming.

  3. **Interrogation des données à l'aide de SQL** : possibilité d'exécuter des requêtes SQL standards pour analyser et interroger les données. Spark SQL convertit ces requêtes SQL en un plan d'exécution distribué et les exécute sur le cluster Spark.

- En résumé, Spark SQL, permet l'utilisation du langage SQL pour simplifier, la **transformation** et l'**analyse exploratoire des données** (EDA).

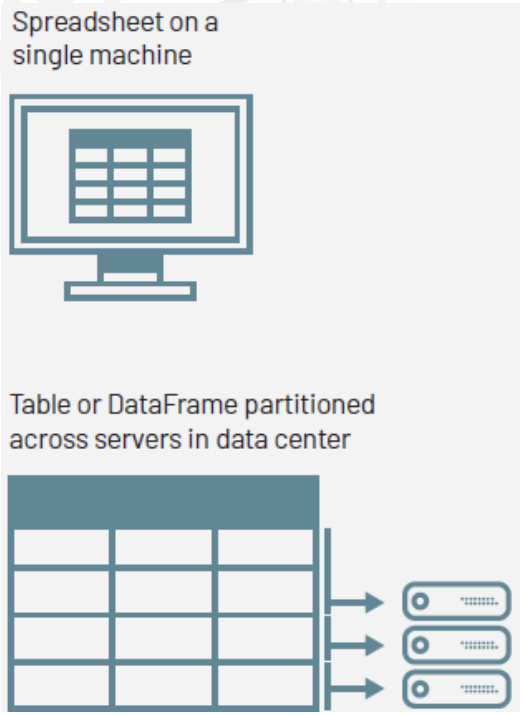
Chapitre 3 - Analyse exploratoire des données avec Saprk SQL

3.2. API DataFrame

- DataFrame API (introduit dans Spark 1.3), renommé en DataSet(Row) depuis la version 2.2
  - **Possibilité d'interroger les données avec des opérateurs SQL-Like ou bien avec une chaîne de caractère SQL**
  - Introduction de la notion de **schéma** pour améliorer les performances de la distribution/sérialisation/désérialisation et en évitant l'envoi/stockage de la structure des classes (gain de performances par rapport à la sérialisation/désérialisation native de Java).
  - Spark essaye d'inférer le schéma s'il n'est pas fourni
  - L'API DataFrame utilise également la sérialisation off-heap (gains par rapport au GC Java) et effectue certaines transformations directement sur le format binaire.
- Conceptuellement un Dataframe se présente sous une forme tabulaire similaire à une table relationnelle

3.2. API DataFrame

- **Data Drame** : Abstractionde haut niveau d'un RDD. Représente un tableau de données avec des lignes et des colonnes (spreadsheet).
- Techniquement : collection distribuée d'objets typés. Distribution sur un cluster de machines (Big Data).
- Le concept de Data Frame, n'est pas propre à SparK. On retrouve le même concept dans Python (pandas) et R. Il est facile de convertir un Data Frame Spark en un Data frame Python ou R et inversement.



3.2. Untyped API (a.k.a. DataFrame) - Création d'un DataFrame

- Création d'un DataFrame à partir d'un RDD
  `createDataFrame(data, schema, samplingRatio, verifySchema)`

- **data** : RDD, liste de Rows, des NamedTuples ou pandas.DataFrame.

- **schema** : une liste de noms de colonnes. Les types des colonnes peuvent être inférés.

- Exemple 1 (**avec spécification du schéma**) :

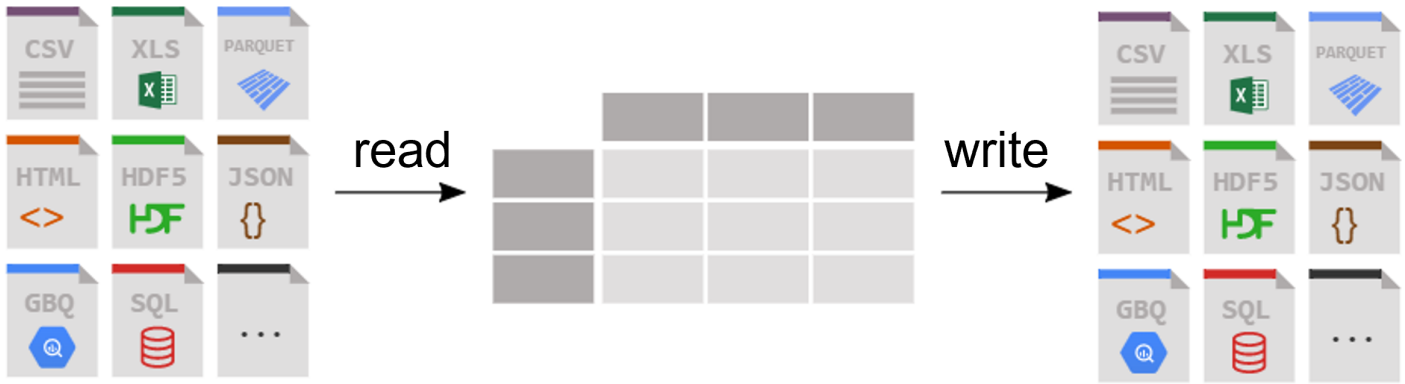
```
sentenceData = spark.createDataFrame([
            (1, "ISITCom Spark SQL"),
            (1, "ISITCom Spark Spark ML"),
            (0, "ISITCom MapReduce"),
            (0, "ISITCom Hadoop")],
          ["label", "sentence"])
```

3.2. Untyped API (a.k.a. DataFrame) - Création d'un DataFrame

- Exemple 2 (avec structType):

```
schema = StructType([StructField("name", StringType()),
                     StructField("age", IntegerType())])
peopleDF = spark.createDataFrame(people, schema)
```

3.2. Création d'un dataframe à partir d'une source de données



3.2. Création d'un dataframe à partir d'une source de données

- Les opérateurs `spark.read` et `spark.write` permettent de lire et d'écrire depuis et vers différentes sources de données.

  `spark.read.format("format") / spark.write.format("format") :`
  format : *json, orc, csv, parquet, avro, jdbc*, etc.

- Autres formes : `spark.read.json(".."), spark.write.json("..")`
  `spark.read.load("..,format="json"),`
  `spark.write.save("..,format="json")`

- Le format de lecture écriture par défaut utilisé par l'API Dataframe est parquet

```
df = spark.read.load("...users.parquet")
df.select("_id", "name").write.save("idsNames.parquet")
```

3.2. Untyped API (a.k.a. DataFrame) - Opérateurs SQL-Like

Exemple

```
dfArtists = spark.read.json("/FileStore/tables/artistsSpark.json")

dfArtistesFiltres = dfArtists.where("birth_date > 1940")

dfArtistesFiltres2 = dfArtists.filter("birth_date > 1940")

dfMovies = spark.read.option("multiline","true").json("/FileStore/tables/
    moviesSpark-2.json")

dfDrama = dfMovies.select("_id", "title", "year").where("genre= 'drama' ").where("
    summary" IS NOT NULL)

joined_df = dfArtists.join(dfMovies, dfArtists._id == dfMovies.director._id)
```

3.2. Untyped API (a.k.a. DataFrame) - Chaîne de caractères SQL

- **SQL sous forme de chaîne de caractères.** La **fonction sql** de sparkSession permet la spécification d'un traitement sous la forme d'une requête SQL. Le résultat est un DataFrame

- Exemple

```
1 dfMovies.createOrReplaceTempView("movies")
2
3 spark.sql("SELECT genre, count(*) AS nb_films FROM movies GROUP BY genre")
4
5 dfArtists.createOrReplaceTempView("artists")
6
7 # on peut  galement  faire une jointure !
8 joined_df = spark.sql("select * from movies, artists where movies.director.
      _id = artists._id")
```

Chapitre 4 - 4. L'API DataSet

4.1. API DataFrame, les pour et les contres

Dans l'**API DataFrame** un traitement est exprimé sous la forme d'un **plan de requête relationnel**.

Ce plan est optimisé et exécuté par l'optimiseur de requêtes **Spark Catalyst**. Les plans s'expriment sous forme de chaînes de caractères SQL ou en utilisant les expressions lambda.

SQL style :

```
df.where("age > 21")
```

Prog Fonctionnelle style :

```
df.filter(df["age"].gt(21))
```

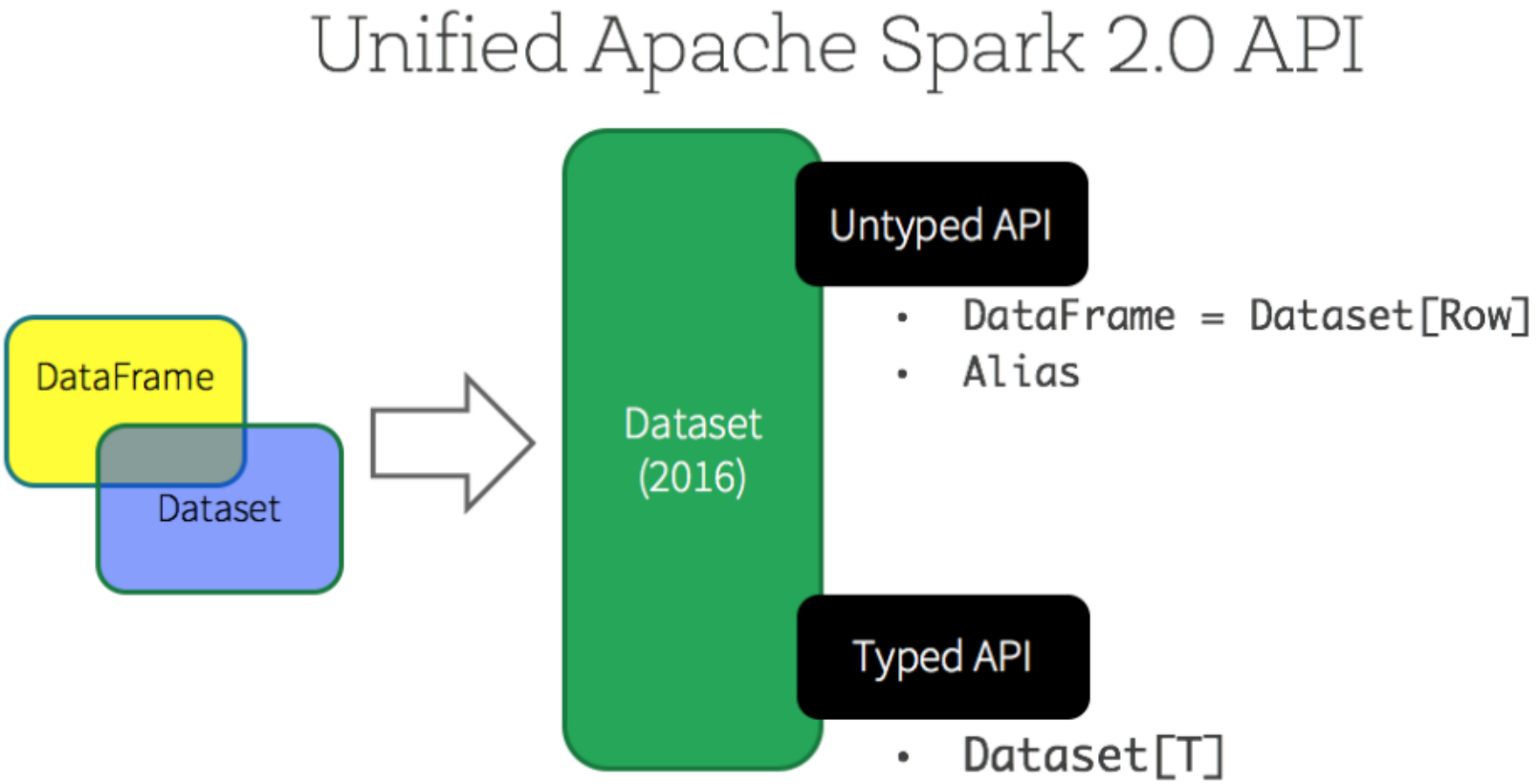
Avantage : grande **facilité** pour exprimer les traitements.

Inconvénient : comme les noms des attributs sont données sous la forme de chaînes de caractères, il n'est pas possible de **vérifier les erreurs à la compilation**

Chapitre 4 - 4. L'API DataSet

4.2. API Dataset

L'API DataSet a été introduite dans la version 1.6. Depuis la version 2.0 Les API DataFrame et Dataset ont été unifiées pour simplifier la tâche aux développeurs



4.2. API Dataset

L'API DataSet représente désormais l'API principale de Spark. On y distingue deux sortes d'APIs strongly-typed API et une untyped API.

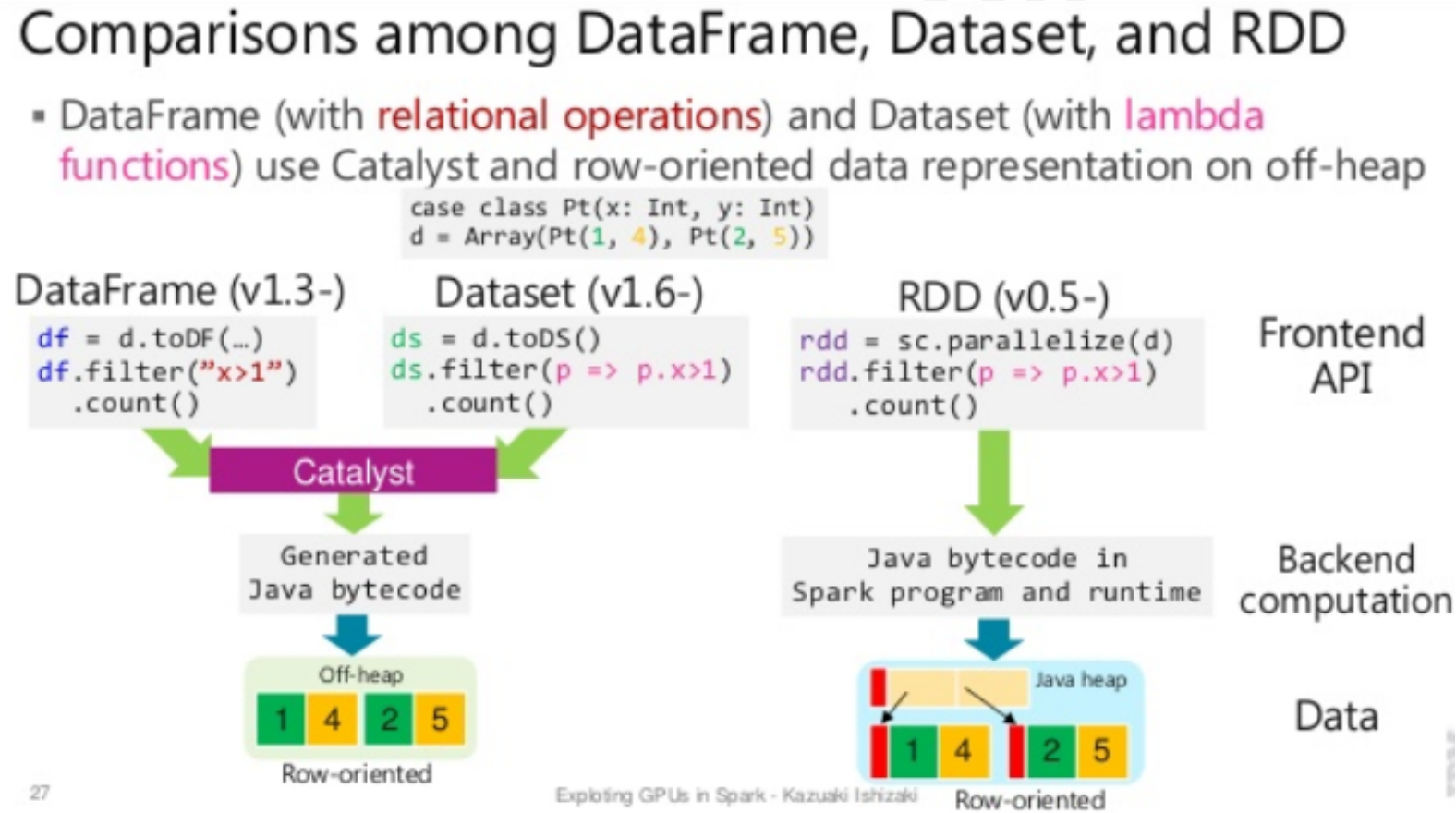


Figure reprise de la documentation officielle d'Apache Spark

4.2. Catalyst Optimizer : Optimisation des Requêtes dans Spark

Catalyst est le moteur d'optimisation de requêtes intégré à Spark, introduit pour améliorer les performances des requêtes DataFrame et Dataset.

S'appuie sur une série de règles d'optimisation qui transforment et améliorent le plan logique de la requête.

Étapes d'optimisation :

- **Analyse logique** : Vérification de la syntaxe de la requête, résolution des colonnes et des types de données.
- **Optimisation logique** : Simplification des opérations redondantes (élimination des filtres inutiles, fusion, etc.).
- **Planification physique** : Génération d'un plan d'exécution optimal tenant compte des coûts et des ressources disponibles.

Catalyst optimise automatiquement certaines transformations comme le **predicate pushdown** (application des filtres plus tôt) pour minimiser le volume de données à traiter.

Différence entre DataFrame et Dataset

DataFrame :

- Une collection de données organisée en colonnes nommées, similaire à une table dans une base de données relationnelle.
- Non typé : les types de colonnes ne sont pas vérifiés à la compilation, ce qui peut entraîner des erreurs à l'exécution.
- Compatible avec plusieurs langages : Python, Scala, Java, R.

Dataset :

- Une collection typée d'objets JVM, introduisant le typage fort pour plus de sécurité.
- Permet de détecter les erreurs à la compilation grâce au typage fort.
- Utilisable principalement avec Scala et Java (pas avec Python et R).

Quand utiliser Dataset, DataFrame, ou RDD ?

Utiliser un Dataset :

- Lorsque vous avez besoin d'un typage fort pour garantir la sécurité du code.
- Pour des transformations complexes sur des objets JVM natifs.

Utiliser un DataFrame :

- Pour manipuler des données tabulaires avec un langage SQL-like.
- Lorsque vous avez besoin de meilleures performances pour des opérations sur des colonnes.

Utiliser un RDD :

- Pour des traitements bas niveau, où vous avez besoin d'un contrôle total sur les partitions et les transformations.
- Pour les traitements non-optimisables par Catalyst (ex : calculs itératifs complexes).

Chapitre 4 - 4. L'API DataSet

Quiz

1. Qu'est-ce que **Spark SQL** et quels types de données permet-il de traiter ?
2. Quelles sont les principales fonctionnalités de Spark SQL qui en font un outil adapté à l'analyse des données structurées ?
3. Expliquez comment Spark SQL optimise l'exécution des requêtes SQL sur des données distribuées.
4. Quelle est la différence entre l'API DataFrame et l'API Dataset dans Spark SQL ?
5. Quels sont les avantages du stockage orienté colonne ?
6. En quoi Spark SQL facilite-t-il l'analyse exploratoire des données (EDA) ?
7. Quelles sont les principales différences entre le traitement SQL dans une base de données relationnelle classique et dans Spark SQL ?

Chapitre 5 - Analyse prédictive avec Spark ML

5.1. AI vs. ML vs. DL

Artificial Intelligence : donner à la machine l'aptitude d'"imiter" le comportement et/ou le raisonnement humain.

Machine Learning (supervisé) : la machine apprend à partir d'exemples déjà connus.

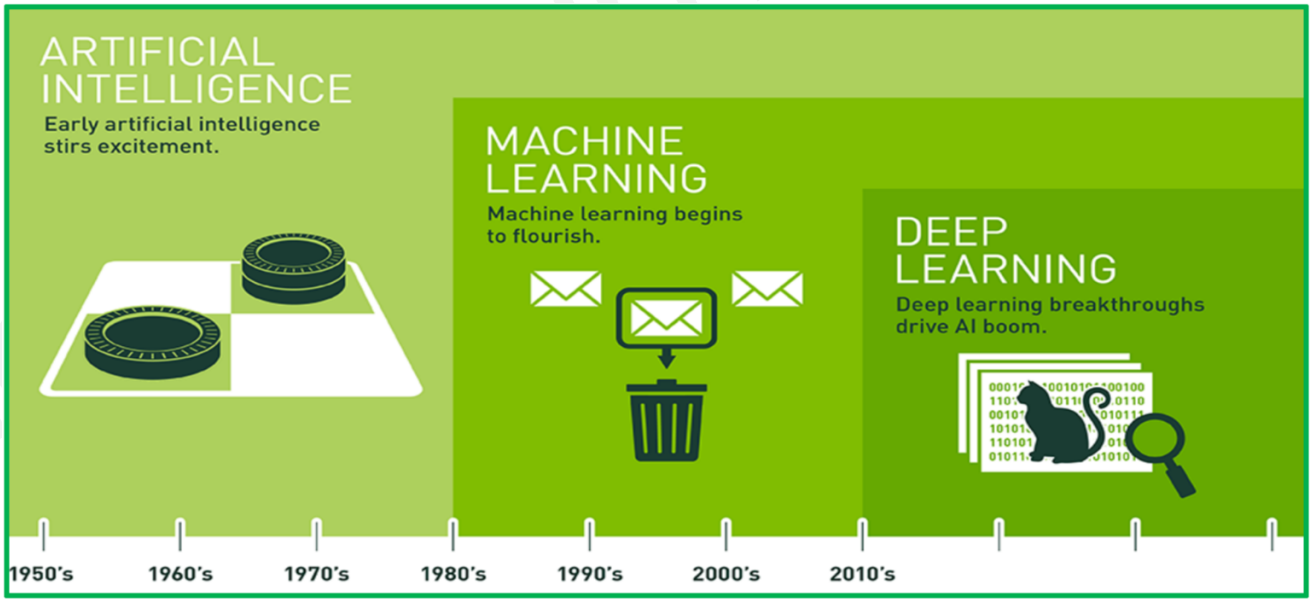


Figure: *Figure rerprise de (IBM Skills Academy, "Course 2 - Introduction to DataScience", BigData Engineer, 2019)

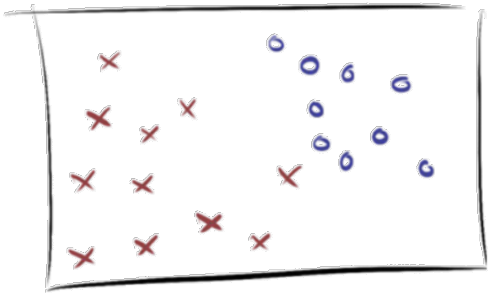
IA et ML : pas vraiment nouveaux, mais **révolutionnés par le Big Data** et les **plateformes Big Data** qui permettent l'analyse de gros ensembles de données.

5.1. Classification vs. Régression

2 types de modèles en ML supervisé : modèles de classification et modèles de régression.

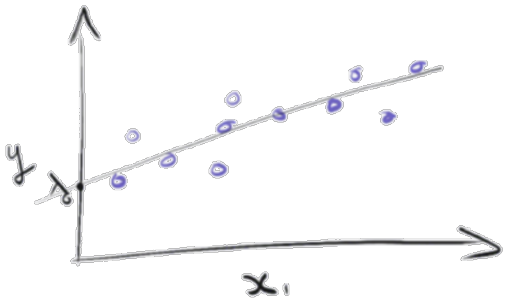
Classification : apprendre à prédire un label de classe (sortie discrète)

E.g. : Test COVID, Fake News Detection, Spam, etc.



Régression : apprendre à prédire une valeur quantitative (sortie continue).

(*E.g.*) : Nombre de lits d'urgence, Argus, etc.



Formellement : "découvrir" la relation (*i.e.* **le modèle**) entre une variable d'entrée *x* et une variable de sortie *y* qui peut être soit discrète soit continue.

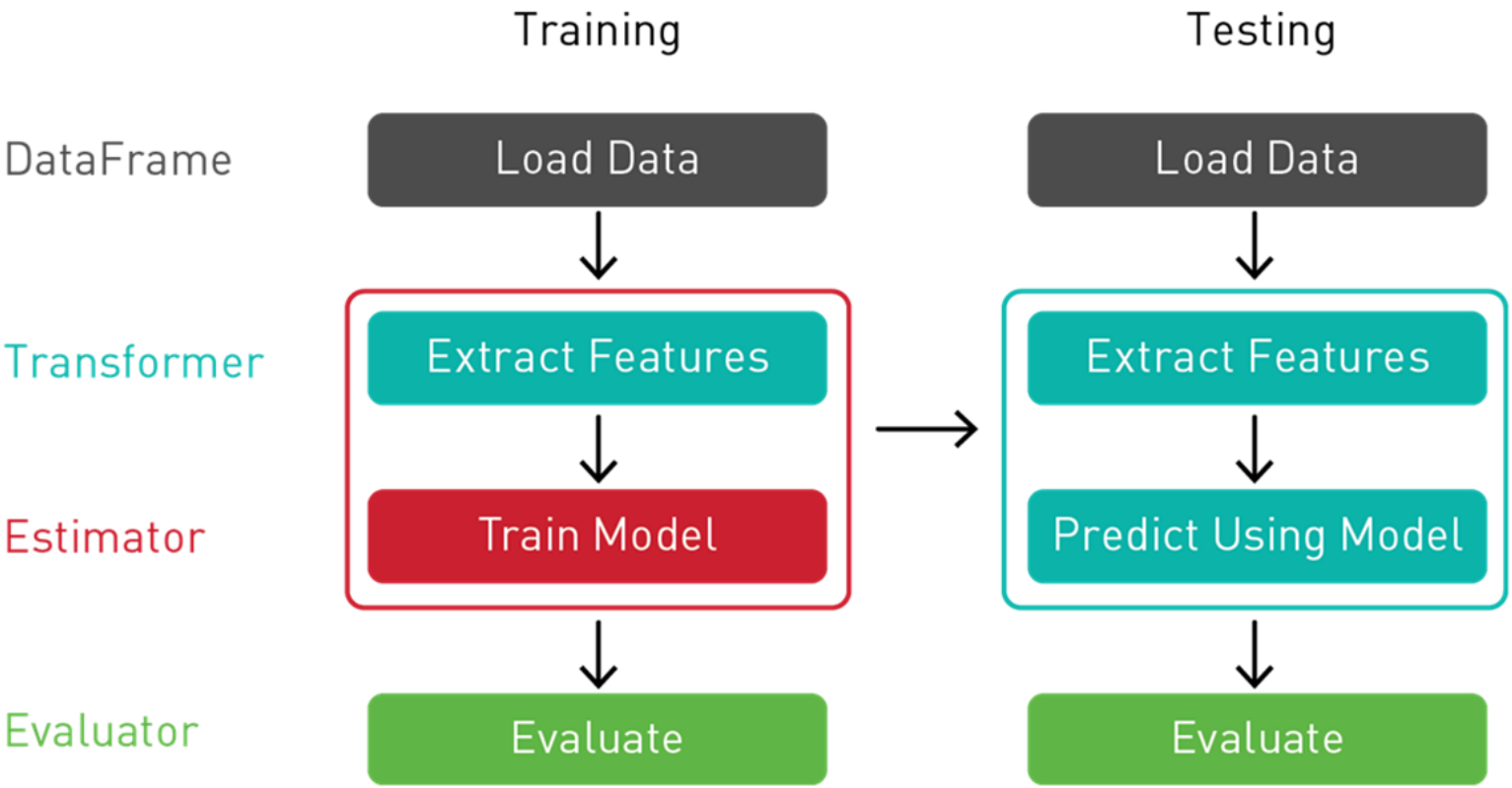
5.1. Apprentissage vs. Inférence

Phase d'**apprentissage** : des données d'apprentissage (étiquetées) sont présentées au modèle pour lui permettre d'apprendre progressivement les relations entre x et y.

Phase d'**inférence** : appliquer le modèle appris, sur des instances non annotées pour prédire les étiquettes qui devraient leur être associées.

Chapitre 5 - Analyse prédictive avec Spark ML

5.2. Principaux Concepts



5.2. Principaux Concepts

Transformer (Transformateur)

1. Prend en entrée un DataFrame et renvoie un nouveau DataFrame avec une ou plusieurs colonnes ajoutées ou transformées.
2. Utilisés pour appliquer des transformations telles que la normalisation, la création de nouvelles caractéristiques, ou la prédiction des valeurs à partir d'un modèle ajusté.
3. Accompagné d'une méthode **transform()** prenant un DataFrame en entrée et renvoie un nouveau DataFrame avec les transformations appliquées.

Estimator (Estimateur)

1. Apprentissage des paramètres du modèle à partir des données fournies.
2. *e.g.* algorithme de régression linéaire, de classification ou de clustering.
3. Dispose d'une méthode **fit()** prenant en entrée les données d'apprentissage et renvoyant un modèle.

5.2. Principaux Concepts

Evaluator (Évaluateur)

1. Dispose d'une méthode **evaluate()** permettant d'évaluer les performance prédictive d'un modèle.
2. Mesure à quel point les prédictions du modèle correspondent aux valeurs réelles.
3. Utilisé pour comparer différents modèles ou pour ajuster les hyperparamètres afin d'optimiser les performances du modèle.

Pipeline

1. Permet de chaîner de manière séquentielle des estimateurs et des transformateurs pour former un flux de travail d'apprentissage automatique structuré.
2. Chaque étape étant soit un estimateur, soit un transformateur. Les étapes sont exécutées dans l'ordre spécifié.
3. Facilite la répétition d'un même processus sur différentes données.

Chapitre 5 - Analyse prédictive avec Spark ML

5.3. Exemple ave c Spark ML - Régression Linéaire

Notebook Databricks - Régression

```
data = [
    (1, 200000, "Petrol", "Toyota", 15000),
    (2, 200000, "Diesel", "Volkswagen", 7000),
    (3, 150000, "Petrol", "Toyota", 18000),
    (4, 40000, "Diesel", "Peugeot", 35000),
    (5, 50000, "Petrol", "Kia", 33000),
    (6, 60000, "Diesel", "Volkswagen", 35000),
    (7, 170000, "Petrol", "Toyota", 16000),
    (8, 80000, "Diesel", "Peugeot", 25000),
    (9, 90000, "Petrol", "Kia", 17000),
    (10, 10000, "Diesel", "Volkswagen", 35000),
    (11, 1000, "Petrol", "Toyota", 50000),
    (12, 120000, "Petrol", "Peugeot", 23000),
    (13, 130000, "Petrol", "kia", 21000),
    (14, 140000, "Diesel", "Volkswagen", 17000),
    (15, 150000, "Petrol", "Toyota", 20000),
    (16, 60000, "Diesel", "Peugeot", 28000)
]

schema = ["id","mileage","fuel_type","car_name","price_euro"]

df = spark.createDataFrame(data, schema)

display(df)
```

5.3. Exemples avec Spark ML - Regression Linéaire

```
from pyspark.ml.feature import StringIndexer, VectorAssembler
from pyspark.ml.regression import LinearRegression
from pyspark.ml.evaluation import RegressionEvaluator

# Convertir les variables cat gorielles en repr sentations num riques
indexeur = StringIndexer(inputCol="fuel_type", outputCol="fuel_type_index")
df = indexeur.fit(df).transform(df)

indexeur = StringIndexer(inputCol="car_name", outputCol="car_name_index")
df = indexeur.fit(df).transform(df)

# Pr parer le vecteur de caract ristiques
assembleur = VectorAssembler(inputCols=["mileage", "fuel_type_index", "
    car_name_index"], outputCol="features")
df = assembleur.transform(df)

# Diviser les donn es en apprentissage et  test
train, test = df.randomSplit([0.8, 0.2], seed=7)

# D finir le mod le de r gression lin aire
lr = LinearRegression(featuresCol='features', labelCol='price_euro')

# Entra ner le mod le
modele_lr = lr.fit(train)
```

5.3. Exemples avec Spark ML - Regression Linéaire

```
# Faire des prédictions
predictions = modele_lr.transform(test)

display(predictions)

# Evaluer le modèle
evaluateur = RegressionEvaluator(labelCol="price_euro", predictionCol="prediction"
    , metricName="mae")
mae = evaluateur.evaluate(predictions)

print("Erreur absolue moyenne (MAE) sur les données de test :", mae)
```

5.3. Exemples avec Spark - Association Rules Learning

Frequent itemsets mining, exemple typique : trouver les groupes d'articles fréquents (*frequent itemsets*) dans les paniers des clients (*transactions*)

Association rules : à partir des frequent itemsets, dériver des règles d'association, de la forme $A \Rightarrow B$ (si A alors B). Exemple : si $\{a, b, c, d\}$ est un itemset fréquent, on peut dériver une règle de la forme $\{a, b\} \Rightarrow \{c, d\}$.

Toutes les règles ne sont pas "intéressantes". une règle inétéressante doit avoir un minimum de confiance et de support

Support : fréquence de la règle ou d'un itemset.
$Support(X) = \frac{|t \in T \; t.q. \; X \subseteq t|}{|T|}$

Confiance d'une règle $X \Rightarrow Y$: probabilité de X sachant Y
$Confiance(X \Rightarrow Y) = \frac{Support(X \cup Y)}{Support(X)}$

Spark utilise l'algorithme FP-Growth pour trouver les itemsets fréquents et les règles d'association

5.3. Exemples avec Spark ML - Association Rules Learning

Notebook Databricks - Frequent Itemsets

```
df = spark.createDataFrame([
    (0, [1, 2, 5]),
    (1, [1, 2, 3, 5]),
    (2, [1, 2]),
    (3, [2, 4, 5])
], ["id", "items"])
fpGrowth = FPGrowth(itemsCol="items", minSupport=0.5, minConfidence=0.6)
model = fpGrowth.fit(df)
display(model.associationRules.select("antecedent","consequent","confidence","
    support"))
```

	antecedent	consequent	confidence	support
1	[5]	[1]	0.6666666666666666	0.5
2	[5]	[2]	1	0.75
3	[5, 1]	[2]	1	0.5
4	[5, 2]	[1]	0.6666666666666666	0.5
5	[2]	[1]	0.75	0.75
6	[2]	[5]	0.75	0.75

```
df = model.transform(df)
display(df.where(size(df['prediction']) > 0))
```

	id	items	prediction
1	2	[1, 2]	[5]
2	3	[2, 4, 5]	[1]

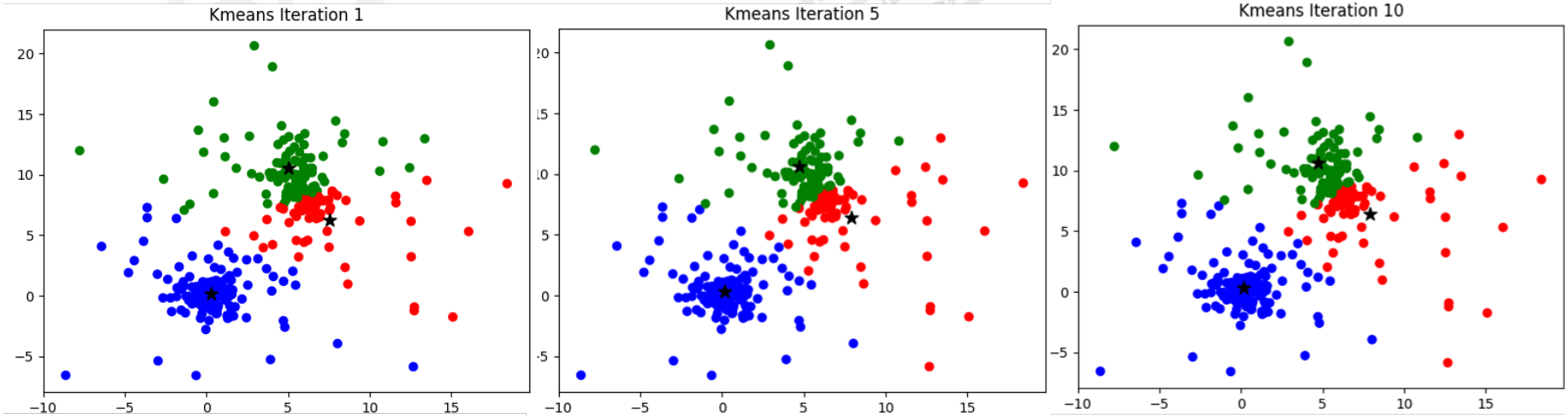
5.3. Exemples avec Spark ML - Clustering avec k-means

Objectif : regrouper les données en K clusters (groupes)

Choisir k points représentant chacun la position moyenne d'un groupe (au hasard par exemple)

Répéter jusqu'à ce qu'il y ait convergence (plus de modification des moyennes entre 2 itérations) :

- assigner chaque donnée au groupe le plus proche
- mettre à jour le centre (la moyenne) de chaque cluster



5.3. Exemples avec Spark ML - Clustering avec k-means

```scala
import org.apache.spark.ml.clustering.KMeans
import org.apache.spark.ml.evaluation.ClusteringEvaluator

// Loads data.
val dataset = spark.read.format("libsvm").load("data/mllib/sample_kmeans_data.txt")

// Trains a k-means model.
val kmeans = new KMeans().setK(2).setSeed(1L)
val model = kmeans.fit(dataset)

// Make predictions
val predictions = model.transform(dataset)

// Evaluate clustering by computing Silhouette score
val evaluator = new ClusteringEvaluator()

val silhouette = evaluator.evaluate(predictions)
println(s"Silhouette with squared euclidean distance = $silhouette")

// Shows the result.
println("Cluster Centers: ")
model.clusterCenters.foreach(println)
```

5.3. Exemples avec Spark ML - Classification et Pipeline

Notebook Databricks - Classification

```
from pyspark.ml import Pipeline
from pyspark.ml.classification import LogisticRegression
from pyspark.ml.feature import HashingTF, Tokenizer

training = spark.createDataFrame([
    (0, "apache spark", 1.0),
    (1, "apache", 0.0),
    (2, "spark SQL", 1.0),
    (3, "hadoop mapreduce", 0.0),
    (4, "spark ML", 1.0),
    (5, "hadoop hdfs", 0.0),
    (6, "spark streaming", 1.0),
    (7, "yarn  mapreduce", 0.0),
    (8, "SQL", 1.0),
    (9, "hadoop yarn", 0.0),
    (10, "streaming SQL", 1.0),
], ["id", "text", "label"])

tok = Tokenizer(inputCol="text", outputCol="words")

hTF = HashingTF(inputCol=tok.getOutputCol(), outputCol="features", numFeatures=5)

lr = LogisticRegression(maxIter=10)

pipeline = Pipeline(stages=[tok, hTF, lr])
```

5.3. Exemples avec Spark ML - Classification et Pipeline

Notebook Databricks - Classification

```
model = pipeline.fit(training)

test = spark.createDataFrame([
    [4, "SQL"],
    [5, "hdfs"],
    [6, "mapreduce spark"],
    [7, "spark hadoop"]
], ["id", "text"])

predictions = model.transform(test)

display(predictions.select("text","prediction", "probability"))
```

Chapitre 5 - Analyse prédictive avec Spark ML

5.4. Grid Search & Cross-Validation

Cross-Validation (Validation Croisée)

- Technique d'évaluation des performances d'un modèle et d'estimation de sa capacité à se généraliser à de nouveaux ensembles de données.

- Dans la validation croisée k-fold, l'ensemble de données est divisé en k sous-ensembles (ou plis) de taille égale. Le modèle est ensuite entraîné k fois, chaque fois en utilisant k-1 plis comme ensemble d'apprentissage et le pli restant comme ensemble de validation.

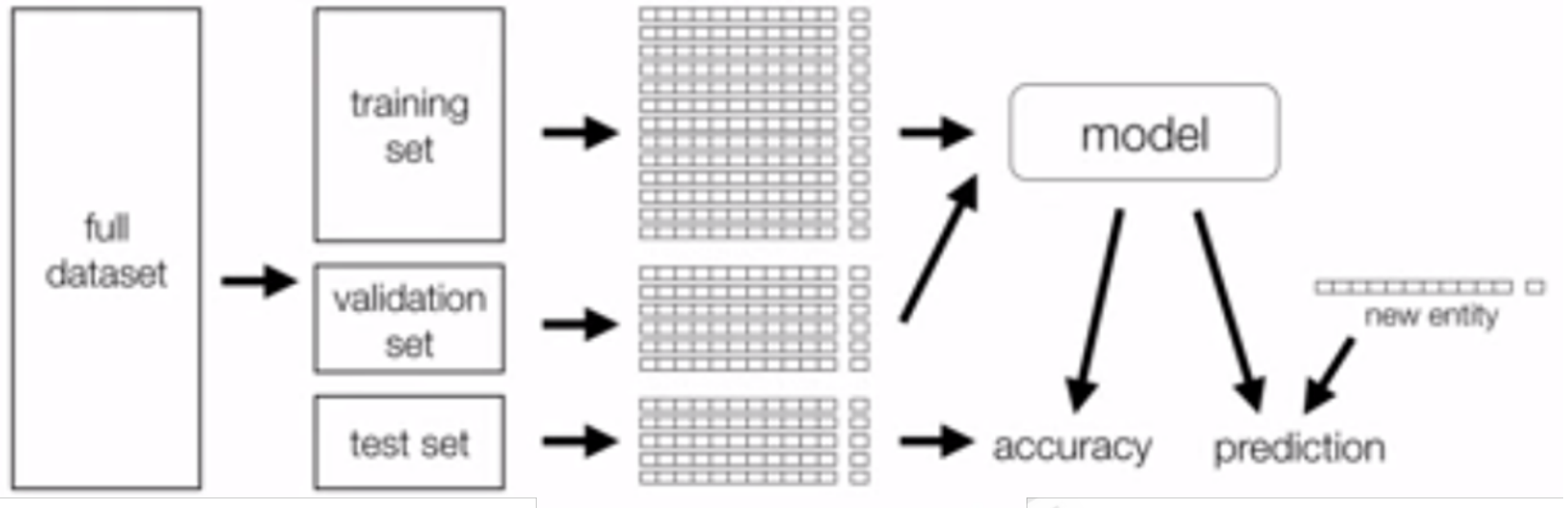
- Permet d'utiliser efficacement toutes les données disponibles pour l'apprentissage et l'évaluation, ce qui donne une estimation plus fiable de la performance du modèle.

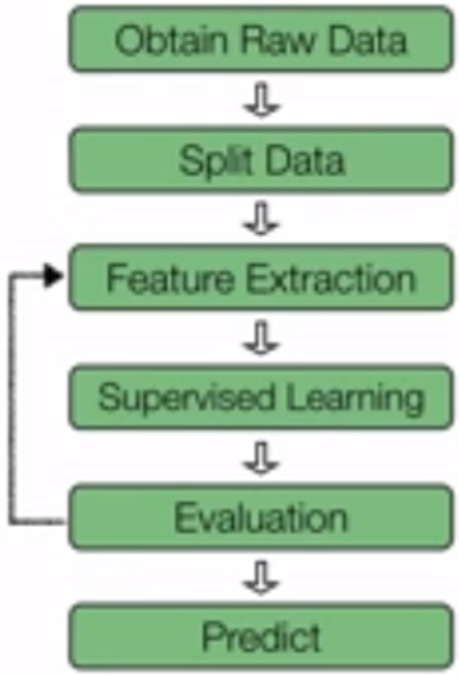
5.4. Grid Search & Cross-Validation

paramGrid (*Grille de Paramètres*)

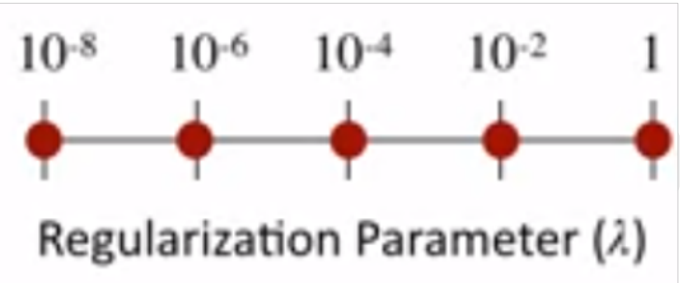
- Spécifie les combinaisons possibles d'hyperparamètres à tester lors de l'apprentissage d'un modèle.
- Lors de l'apprentissage avec une validation croisée, plusieurs combinaisons d'hyperparamètres peuvent être testées pour trouver les valeurs qui donnent les meilleures performances sur l'ensemble de validation.
- Par exemple, pour un modèle de régression logistique, une grille de paramètres peut inclure différentes valeurs de régularisation (alpha) et de paramètres de convergence (tolérance).

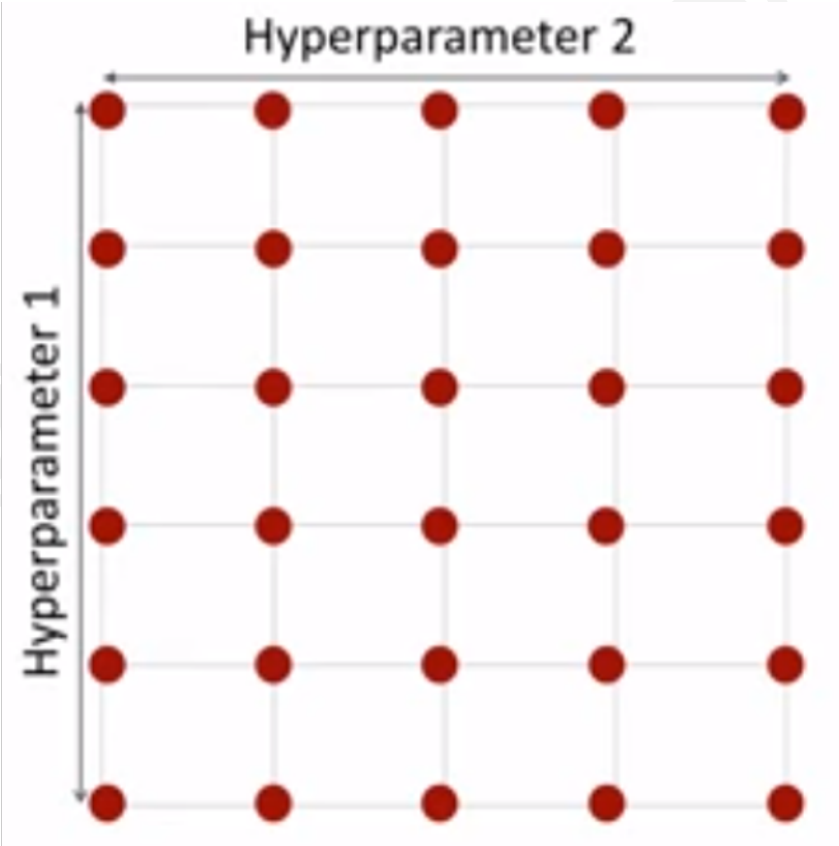
5.4. Grid Search & Cross-Validation





5.4. Grid Search & Cross-Validation





5.4. Grid Search & Cross-Validation

Notebook Databricks - Classification

```
from pyspark.ml.evaluation import BinaryClassificationEvaluator
from pyspark.ml.tuning import CrossValidator, ParamGridBuilder

paramGrid = ParamGridBuilder().addGrid(hashingTF.numFeatures, [10, 20, 100, 1000])
     .addGrid(lr.regParam, [0.5, 0.1, 0.01]).build()

crossval = CrossValidator(estimator=pipeline,
                          estimatorParamMaps=paramGrid,
                          evaluator=BinaryClassificationEvaluator(),
                          numFolds=3)

cvModel = crossval.fit(training)

model = cvModel.bestModel

best_hashTF = model.stages[1]

best_lr = model.stages[2]

print(best_hashTF.getNumFeatures())

print(best_lr._java_obj.getRegParam())

print(best_lr._java_obj.getMaxIter())
```

5.4. Spark Pipeline

Exemple

- Nous disposons d'un enesemble d'emails, nous savons que certains sont des spams et que d'autres ne le sont pas
- Nous voulons construire un modèle de classification spam/non spams
- Pipeline : extraction des mots, calcul du TF-IDF, classification avec la régression logistique
- Nous voulons en plus choisir les valeurs optimales pour nos algorithmes => le paramGrid permet de tester différentes combinaisons de valeur pour le pipeline
- Cross Validation : les données sont divisées en partitions (fold). Pour chaque partition, nous allons réserver une partie des données pour l'apprentissage et la partie restante au tests de l'efficacité du modèle.

https://databricks-prod-cloudfront.cloud.databricks.com/public/4027ec902e239c93eaaa8714f173bcfc/5374664368047379/3115681329431923/1220604254746701/latest.html

Chapitre 5 - Analyse prédictive avec Spark ML

Quiz : Analyse Prédictive avec Spark ML

1. Expliquez les phases d'**apprentissage** et d'**inférence** dans un modèle de machine learning supervisé.
2. Que signifie le terme **transformer** dans le contexte de Spark ML ? Comment est-il utilisé ?
3. Décrivez le rôle d'un **estimateur** et donnez un exemple d'algorithme qui pourrait être utilisé comme estimateur dans Spark ML.
4. Décrivez le rôle d'un **transformer** et donnez un exemple d'algorithme qui pourrait être utilisé comme transformer dans Spark ML.
5. Un modèle prédictif est-il un estimator ou un transformer.
6. Qu'est-ce qu'un **pipeline** et pourquoi est-il utile dans un flux de travail d'apprentissage automatique ?
7. Expliquez le concept de **validation croisée** (cross-validation) et pourquoi il est important dans l'évaluation des modèles de machine learning.
8. Que permet d'accomplir la méthode **evaluate()** ?
9. En quoi consiste une **grid search** et comment est-elle utilisée pour optimiser un modèle de machine learning ?

Chapitre 6 - 6. Datascience - Exemple complet

Databricks Notebooks

- Databricks est une plateforme cloud collaborative qui intègre Apache Spark.
- Cofondé en 2013 par Matei Zaharia, l'inventeur d'Apache Spark, ainsi que d'autres chercheurs du laboratoire AMPLab de l'Université de Californie à Berkeley.
- Les Notebooks Dtabricks permettent de :
  - Écrire et exécuter du code Spark.
  - Collaborer avec d'autres utilisateurs grâce aux commentaires et annotations.
  - Visualiser des résultats sous forme de graphiques interactifs.
- Intègre aussi **Langage Markdown dans Databricks** pour structurer et documenter le code.

Exemple d'usage de Markdown dans un notebook Databricks :

```
%md
# Analyse des donn es COVID-19
Ce notebook effectue une analyse exploratoire des donn es COVID-19

## Calcul de la distribution des cas
La distribution des cas est calcul e en divisant les nouveaux cas par
 intervalle de 100 :
\[
\text{Plage des cas} = \left\lfloor \frac{\text{new\_cases}}{100} \right\
 rfloor \times 100
\]
Cela regroupe les cas dans des plages de 0-99, 100-199, 200-299, etc.

```

Chapitre 6 - 6. Datascience - Exemple complet

Transformations et Data Munging

Utilisation d'une UDF pour l'extraction de dates

```
#Definition de la fonction pour convertir la chaine en date
def string_to_date(date_string):
    return datetime.strptime(date_string, '%d/%m/%Y')

date_udf = udf(string_to_date, DateType())
dfCovid = dfCovid.withColumn('date', date_udf(dfCovid['date']))
```

Transtypage, renommage des attributs avec SQL et suppression des nulls

```
%sql
SELECT *, CAST(new_cases_smoothed_per_million AS FLOAT) AS new_cases
    FROM dfCovid_table
    WHERE new_cases_smoothed_per_million AS FLOAT IS NOT NULL
```

Jointure pour ajouter une colonne avec le nombre de cas avant 7 jours

```
dfCovid.createOrReplaceTempView("covid_table")
dfCovid.createOrReplaceTempView("j7")

dfCovid = spark.sql("select covid_table.date, covid_table.iso_code, j7.new_cases
    as j7_cases, covid_table.new_cases as cases from covid, j7  where covid_table
    .iso_code = j7.iso_code  AND datediff(covid_table.date, j7.date)=7 order by
    cases.date")
```

Garder en mémoire (important pour les performances!)

```
dfCovid.cache()
```

Chapitre 6 - 6. Datascience - Exemple complet

EDA avec Databricks et Spark SQL

Résumé des statistiques descriptives avec SparkSQL

Utilisez SparkSQL pour obtenir des statistiques telles que :

- Moyenne, écart-type, minimum et maximum du nombre de cas.

Exemple en SparkSQL :

```
%sql
SELECT
    MIN(new_cases) AS min_cases,
    MAX(new_cases) AS max_cases,
    AVG(new_cases) AS avg_cases,
    STDDEV(new_cases) AS stddev_cases
FROM covid_table
WHERE new_cases IS NOT NULL;
```

Exemple avec PySpark :

```
from pyspark.sql.functions import mean, stddev, min, max
dfCovid.select(mean('new_cases'), stddev('new_cases'),
               min('new_cases'), max('new_cases')).show()
```

EDA avec Databricks et Spark SQL

Identifier les valeurs manquantes dans les données

Exemple en SparkSQL :

```
1    SELECT
2        SUM(CASE WHEN new_cases IS NULL THEN 1 ELSE 0 END) AS missing_new_cases,
3        SUM(CASE WHEN population IS NULL THEN 1 ELSE 0 END) AS missing_population
4    FROM covid_table;
```

EDA avec Databricks et Spark SQL

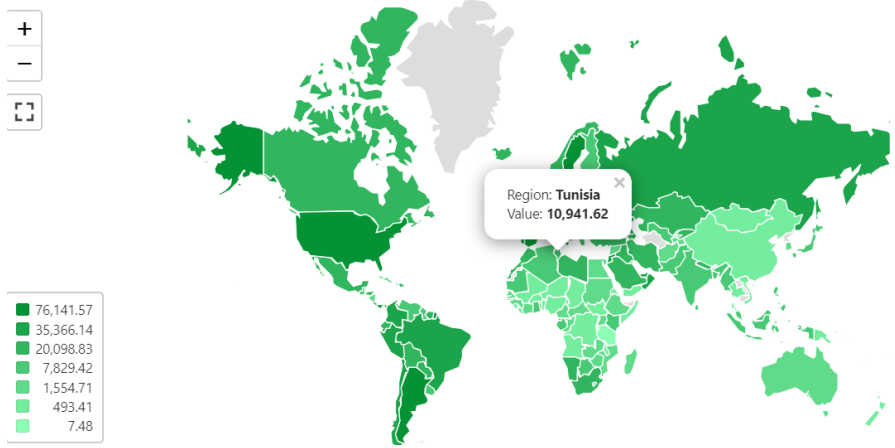
Exploration de la répartition des cas COVID-19 par pays

Pour explorer la répartition des cas par pays, vous pouvez regrouper les données par pays et compter le nombre total de cas.

Exemple en SparkSQL

```
%sql
SELECT
    iso_code,
    SUM(new_cases) AS total_cases
FROM covid_table
GROUP BY iso_code
ORDER BY total_cases DESC;
```

Graphique interactif de la répartition géographique des cas COVID-19 avec Databricks



EDA avec Databricks et Spark SQL

Explorer les corrélations entre les variables (nouveaux cas, population)

La corrélation permet d'explorer la relation entre deux variables continues. Dans cet exemple, nous explorons la corrélation entre les nouveaux cas de COVID et la population.

Exemple en SparkSQL

```
%sql
SELECT corr(new_cases, population) AS correlation_cases_population
FROM covid_table
WHERE new_cases IS NOT NULL AND population IS NOT NULL;
```

Analyse temporelle de l'évolution des cas COVID-19

Vous pouvez explorer comment le nombre de nouveaux cas évolue au fil du temps en agrégeant les cas par date.

Exemple en SparkSQL

```
SELECT
    date,
    SUM(new_cases) AS total_new_cases
FROM covid_table
GROUP BY date
ORDER BY date;
```

Distribution des nouveaux cas

Explorer la distribution des nouveaux cas avec des buckets (intervalles)

Utilisez des buckets pour explorer la distribution des nouveaux cas de COVID par intervalle (par exemple, des intervalles de 100 cas).

Exemple en SparkSQL :

```
    SELECT
        FLOOR(new_cases/100)*100 AS case_range,
        COUNT(*) AS frequency
    FROM covid_table
    GROUP BY case_range
    ORDER BY case_range;

```

Widgets Interactifs dans Databricks

Introduction aux Widgets

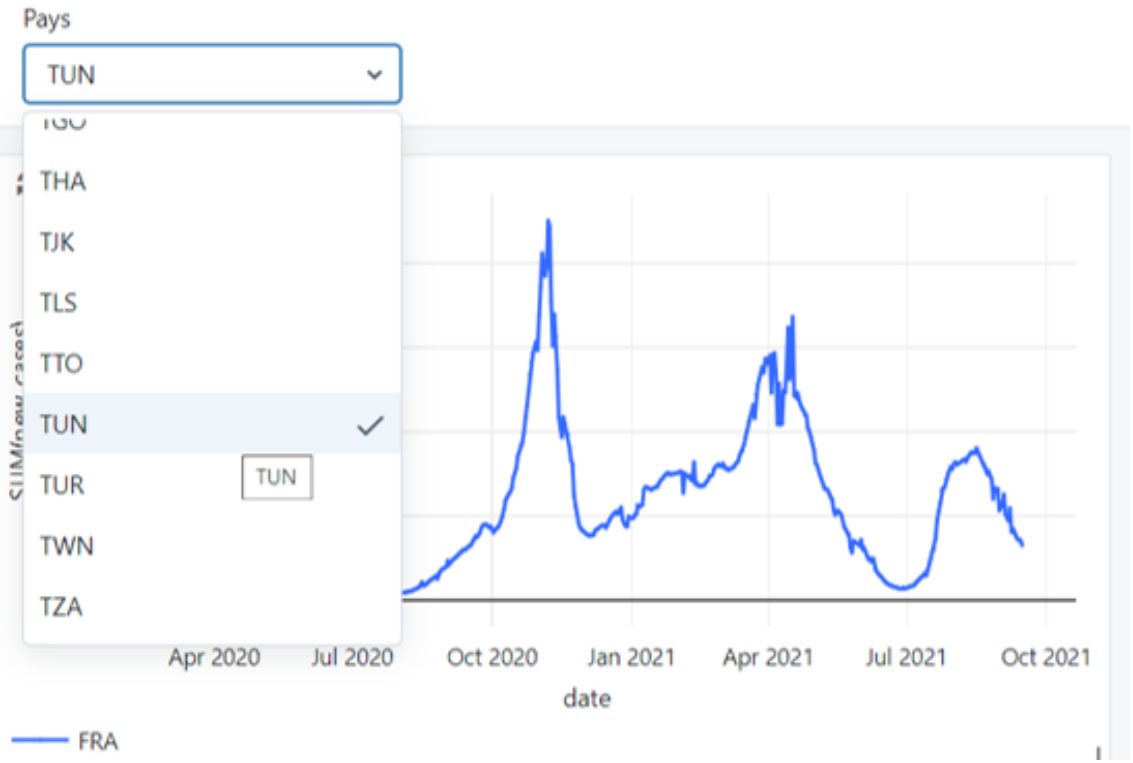
- Les widgets permettent de rendre un notebook interactif.
- Ils ajoutent des contrôles dynamiques, tels que des champs de texte, des listes déroulantes, et des boutons.

Exemple de création de Widget : Sélectionner un pays

```
1    iso = dfCovid.select("iso_code").distinct().collect()
2    # Parcours du dataframe iso
3    for row in iso:
4        iso_list.append(row.iso_code)
5
6    dbutils.widgets.dropdown("Pays", "TUN", iso_list, "S lectionnez un pays")
7
8    selected_country = dbutils.widgets.get("iso_code")
9    df_filtered = dfCovis.where(dfCovid['iso_code'] == selected_country)
```

Widgets Interactifs dans Databricks

Graphique interactif avec Widget pour afficher l'évolution des cas par pays



Tous les graphiques générés peuvent être ajoutés à un dashboard Databricks

Chapitre 6 - 6. Datascience - Exemple complet

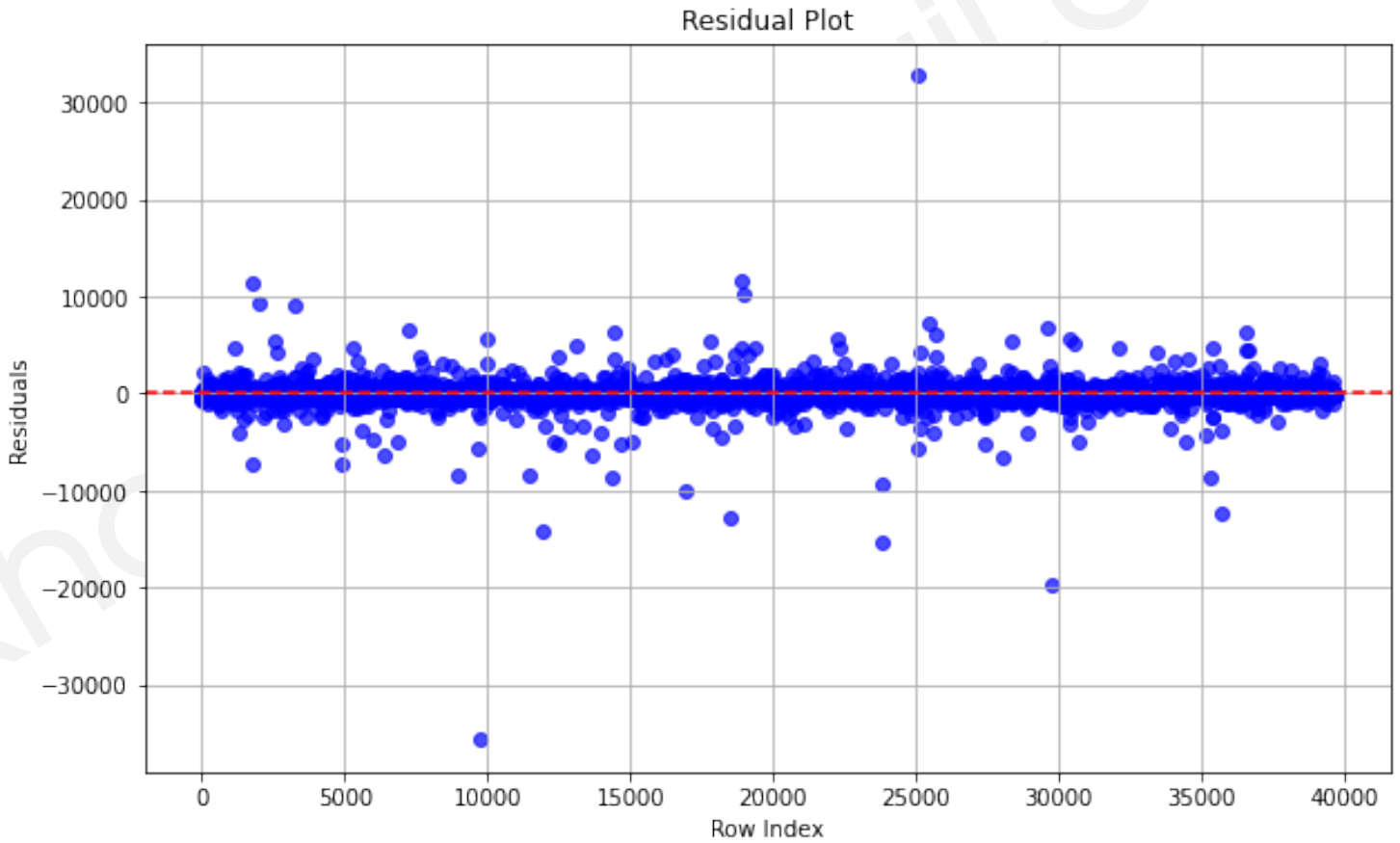
Modèles Prédictifs avec Databricks et Spark ML

Databricks permet également d'exploiter des bibliothèques externes pour visualiser les résultats. Une fois les résultats visualisés, vous pouvez intégrer le graphique généré directement dans votre dashboard.

Exemple : Visualisation des résidus d'un modèle prédictif avec Matplotlib

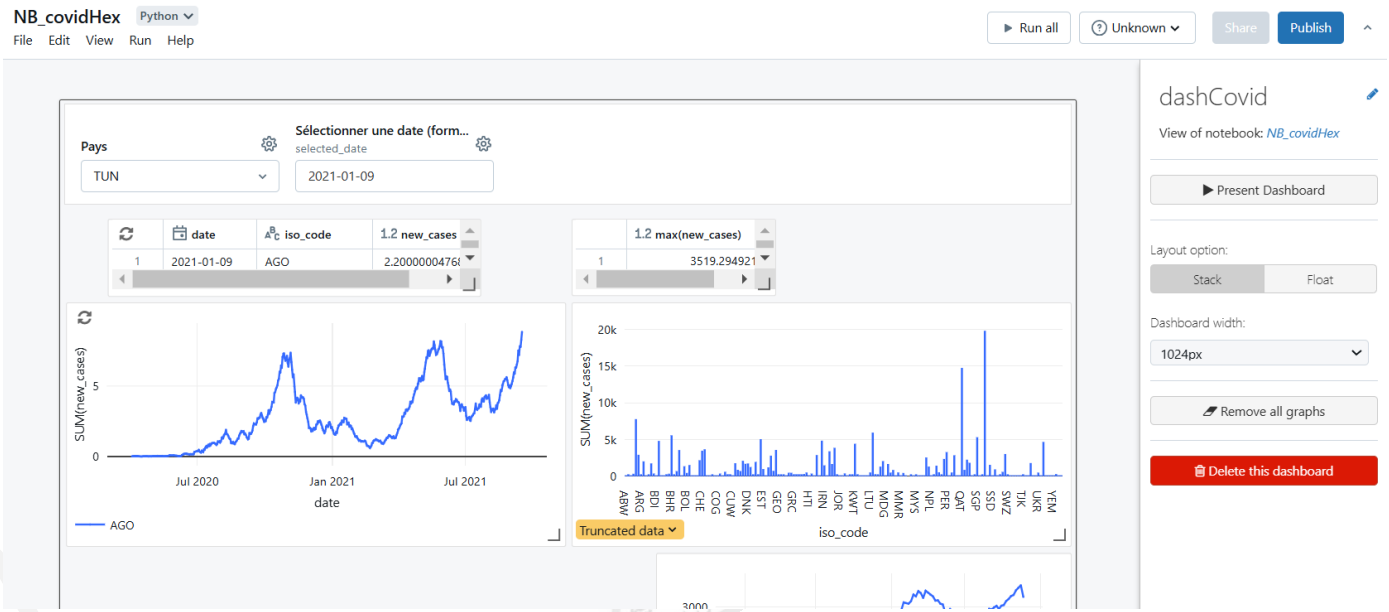
```
    predictions_pd = predictions.toPandas()

    predicted_values = np.array(predictions_pd["prediction"])
    true_values = np.array(predictions_pd["cases"])

    residuals = true_values - predicted_values

    indices = np.array(range(len(predictions_pd)))

    plt.figure(figsize=(10, 5))
    plt.scatter(indices, residuals, color='blue')
    plt.axhline(y=0, color='r', linestyle='--')
    plt.title('Graphique des r sidus')
    plt.xlabel('Indice de ligne')
    plt.ylabel('R sidu (Valeurs r elles - Pr dites)')
    plt.show()

```

Visualisation avec Matplotlib



Étapes supplémentaires

1. Dashoard interactif avec les différents graphiques et widgets



2. Déploiement du modèle prédictif sous la forme d'un web service REST (avec MLFlow ou FLask)

3. Analyse en **temps réel** sur les **flux de données**, **Lake house** et **architecture en médaillon**

   $\Rightarrow$ **Rendez-vous au prochain cours!**