# Machine Learning avec Spark (Spark ML)

## Objectifs de la formation

1. **Maîtriser les fondamentaux de Spark MLlib**
   → Comprendre le fonctionnement de Spark dans un environnement distribué et l'intégration de MLlib pour le Machine Learning à grande échelle.

2. **Savoir collecter et préparer les données efficacement**
   → Être capable de récupérer, nettoyer et transformer des données issues de différentes sources à l'aide de **Azure Synapse**, **Databricks** ou **Google Colab** pour un traitement optimisé.

3. **Modéliser et entraîner des algorithmes de Machine Learning**
   → Concevoir et entraîner des modèles de régression, classification ou clustering dans **Spark MLlib**, en exploitant les capacités de calcul de **Databricks** ou Colab.

4. **Évaluer et optimiser les performances des modèles**
   → Utiliser les outils d'évaluation intégrés (métriques, validation croisée, tuning d'hyperparamètres) pour améliorer la qualité des modèles dans un contexte **big data**.

5. **Industrialiser et déployer des applications de prédiction**
   → Mettre en production un pipeline ML complet, du chargement des données à la prédiction, dans des environnements professionnels comme **Azure Synapse Analytics** et **Databricks**.

1. Machine Learning sous Spark, capable de traiter de grands volumes de données et de générer des prédictions à grande échelle.

# Programme

## 1 - L'écosystème SPARK et l'apprentissage Big Data

- Enjeux machine learning et Big Data
- L'écosystème Apache Spark
- Les différentes briques de base
- Focus SPARK SQL
- Dataframes et Datasets
- Lab : Mise en oeuvre de l'écosystème SPARK pour l'apprentissage machine Big Data

## 2 - Le chargement de données d'entraînement massives

- Chargement générique de données
- Chargement de fichiers de formats spécifiques
- Interrogation de bases HIVE
- Interrogation de bases externes
- Lab : Chargement de données de sources diverses sur un cluster SPARK

## 3 - L'exploration de données d'entraînement massives

- Réalisation de statistiques de base avec SPARK
- Exploitation des librairies graphiques statistiques dans un cadre Big Data
- Lab : Exploration de données d'entraînement sur un cas concret

## 4 - Le "Pipelining"

- Le concept de Pipeline Spark
- Les composants d'un Pipeline
- Le fonctionnement d'un Pipeline
- La gestion des paramètres Persistance et chargement de Pipelines
- Lab : Création d'un premier pipeline d'apprentissage machine avec SPARK

## 5 - Le prétraitement et l'ingénierie des variables prédictives

- Extraction de variables prédictives

- Transformation de variables
- Sélection de variables prédictives Hachage de variables
- Lab : Prétraitement et ingénierie des variables prédictives sur un cas concret

## 6 - La création de modèles d'apprentissage Big Data

- Classification de données massives
- Régression de données massives
- Clustering de données massives
- Systèmes de recommandation Big Data
- Lab : Réalisation de modèles d'apprentissage sur des cas concrets Big Data

## 7 - L'optimisation du réglage des modèles d'apprentissage

- Réglage des hyper-paramètres des modèles
- Validation croisée
- Séparation des données (entraînement, validation)
- Lab : Optimisation du réglage de modèles d'apprentissage sur cas concrets

## 8 - Déploiement de modèles d'apprentissage Big Data

- Création d'application prédictive en batch
- Création d'application prédictive en streaming
- Mise en oeuvre concrète sur un cluster Big Data
- Bonnes pratiques de déploiement
- Lab : Création d'applications en batch et en streaming sur cas concrets

# *Machine Learning avec Apache Spark*

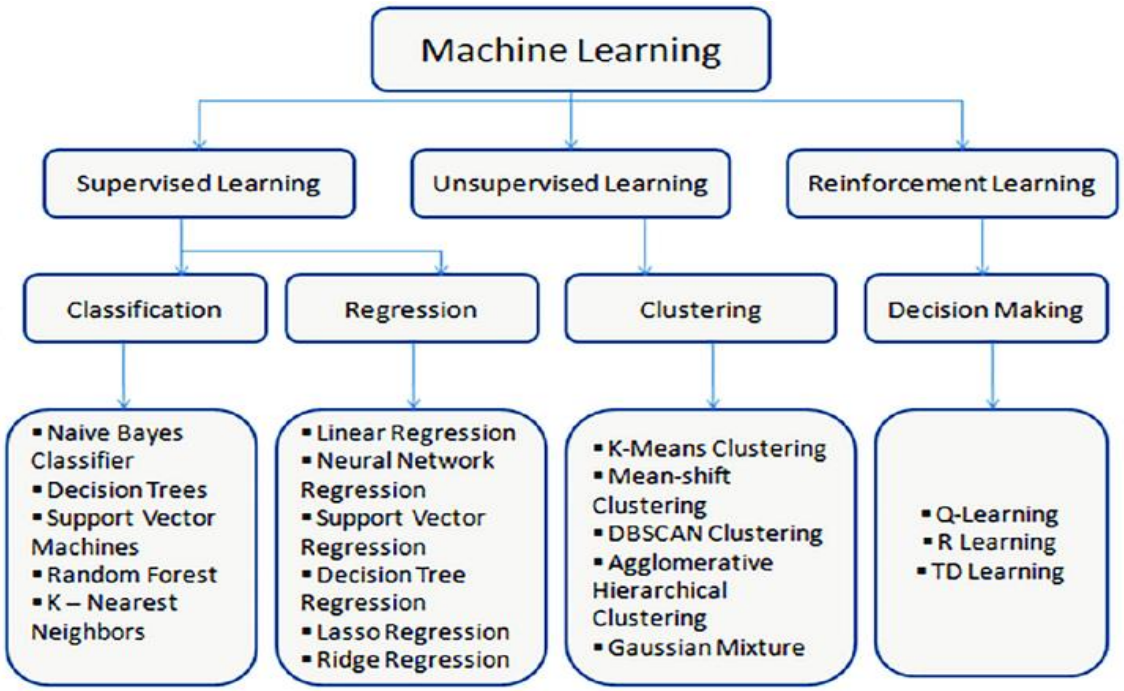
---

# Module 1 : Machine Learning

## 1- Introduction générale au Machine Learning

## 1.1. Qu'est-ce que le Machine Learning ?

Le *Machine Learning* (ou apprentissage automatique) est une branche de l'intelligence artificielle qui permet à un système informatique **d'apprendre à partir de données** plutôt que d'être explicitement programmé pour chaque tâche.




## 1.2. Pourquoi utiliser le Machine Learning ?

- **Automatiser la prise de décision** (ex : détection de fraude bancaire).
- **Faire des prédictions** (ex : prévision de ventes, météo).
- **Découvrir des schémas cachés** (ex : segmentation de clients).
- **Améliorer des processus existants** grâce à l'analyse de grandes quantités de données.

## 1.3. Composants d'un système de Machine Learning

Un modèle de machine learning se construit autour de :

1. **Données d'entrée (features)** — les caractéristiques observables.
2. **Sortie attendue (label)** — la variable cible à prédire.
3. **Algorithme d'apprentissage** — la méthode utilisée pour créer le modèle.
4. **Modèle entraîné** — la fonction finale qui permet de prédire de nouvelles valeurs.

Exemple concret :

On veut prédire le **prix d'une maison** à partir de ses **caractéristiques** (surface, nombre de pièces, localisation).
Les features = [surface, nombrede pieces, localisation]
Le label = prix
L'algorithme apprend une fonction :
**prix = f(surface, nb_pieces, quartier)**

---

# 2- Types d'apprentissage

## 2.1. Apprentissage supervisé

C'est la forme la plus courante de Machine Learning.
Les données sont **étiquetées**, c'est-à-dire que pour chaque observation, on connaît la valeur attendue (le label).

- **Objectif :**

Apprendre une fonction qui relie les entrées aux sorties connues afin de **prédire** la sortie d'une nouvelle donnée.

- **Exemples :**

- **Régression linéaire** : prédiction de valeurs continues

  Exemple : prédire le **prix d'un bien immobilier**.

- **Classification** : prédiction de catégories

  Exemple : détecter si un email est **spam / non spam**.

**⚙ Méthodes courantes :**

- Régression linéaire, régression logistique
- Arbres de décision, Random Forest
- Support Vector Machines (SVM)
- Réseaux de neurones (NN)

---

## 2.2. Apprentissage non supervisé

Dans ce cas, **aucune étiquette n'est disponible**.
Le but est de découvrir **des structures cachées** ou des **regroupements naturels** dans les données.

- **Objectif :**

Explorer les données pour en extraire des motifs, des segments ou des relations entre variables.

- **Exemples :**

- **Clustering (regroupement)** : segmenter des clients selon leurs comportements.
  → Exemple : regrouper les clients selon leurs habitudes d'achat (fidèles, occasionnels, premium).
- **Réduction de dimension (PCA)** : simplifier un grand nombre de variables tout en conservant l'essentiel de l'information.

**⚙ Algorithmes principaux :**

- K-Means, DBSCAN
- PCA (Principal Component Analysis)

---

## 2.3. Apprentissage semi-supervisé

Combine **données étiquetées** et **non étiquetées**.
Très utile lorsque l'étiquetage manuel est coûteux.

Exemple : on a 10 000 images, mais seulement 1 000 sont annotées (chien/chat).
Le modèle utilise les 1 000 images étiquetées pour apprendre, et les 9 000 autres pour affiner sa compréhension.

## 2.4. Apprentissage par renforcement

Le modèle apprend **par essai-erreur**, en recevant des **récompenses** ou **pénalités** selon ses actions.

Exemple : un robot apprend à marcher, un algorithme apprend à jouer aux échecs.
Spark MLlib n'implémente pas directement cette catégorie, mais elle est utile pour comprendre les logiques adaptatives.

---

# 3- Concepts clés du Machine Learning

## 3.1. Dataset, features et labels

- **Dataset** : ensemble de données utilisé pour l'apprentissage.
- **Features** : variables explicatives (entrées).
- **Label** : variable cible (sortie à prédire).

Exemple :

| Surface | Nb_pieces | Quartier | Prix |
|---|---|---|---|
| 70 | 3 | Centre | 250000 |
| 90 | 4 | Banlieue | 220000 |
| 120 | 5 | Centre | 360000 |

→ Features = [Surface, Nb_pieces, Quartier]
→ Label = Prix

---

## 3.2. Entraînement / Test Split

Avant d'entraîner un modèle, on divise le dataset :

- **Train set (70–80%)** : sert à apprendre.
- **Test set (20–30%)** : sert à évaluer la performance.

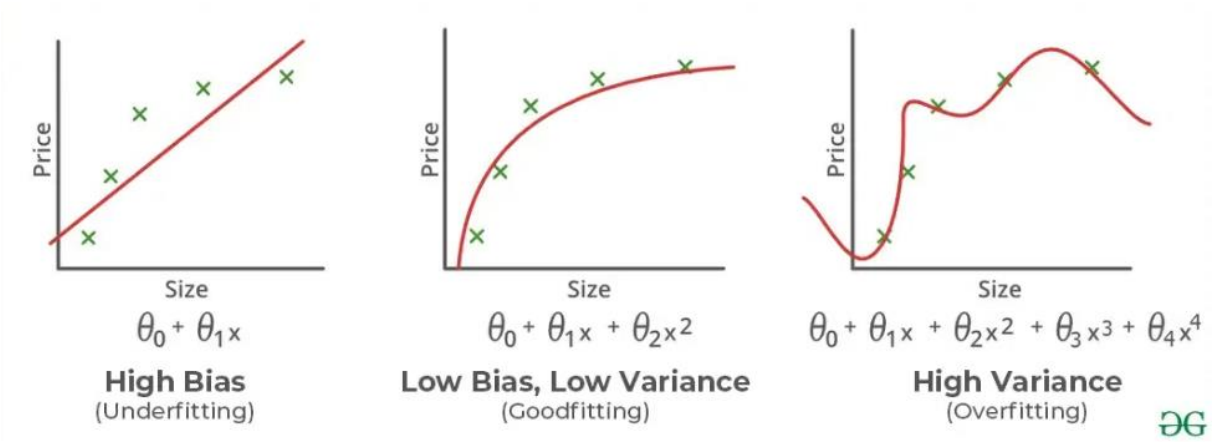
```
# Exemple PySpark
train, test = data.randomSplit([0.8, 0.2], seed=42)
```

---

## 3.3. Overfitting et Underfitting

| Terme | Description | Symptôme |
|---|---|---|
| **Overfitting** | Le modèle apprend trop les détails du jeu d'entraînement (il "mémorise") | Très bon sur train, mauvais sur test |
| **Underfitting** | Le modèle est trop simple, ne capture pas la tendance | Mauvais sur train et test |

Objectif : trouver le juste équilibre avec une **complexité adaptée** et une **bonne généralisation**.




## Comment traiter le surapprentissage (overfitting) et le sous-apprentissage (underfitting)

### Techniques pour réduire le sous-apprentissage

- Augmenter la complexité du modèle.
- Augmenter le nombre de caractéristiques (features) en effectuant de l'ingénierie des variables (feature engineering).
- Supprimer le bruit dans les données.
- Augmenter le nombre d'époques (epochs) ou prolonger la durée de l'entraînement afin d'obtenir de meilleurs résultats.

### Techniques pour réduire le surapprentissage

- Améliorer la qualité des données d'entraînement afin de se concentrer sur les motifs pertinents et réduire le risque d'ajuster le bruit ou les caractéristiques non pertinentes.

- Augmenter la quantité de données d'entraînement pour améliorer la capacité du modèle à généraliser sur de nouvelles données et réduire le risque de surapprentissage.
- Réduire la complexité du modèle.
- Utiliser l'**arrêt anticipé (early stopping)** pendant la phase d'entraînement (surveiller la perte au cours de l'entraînement et arrêter dès qu'elle commence à augmenter).
- Appliquer la **régularisation Ridge** et la **régularisation Lasso**.
- Utiliser la technique de **dropout** dans les réseaux de neurones pour lutter contre le surapprentissage.

---

# 4 - Évaluation et optimisation

## 4.1. Métriques d'évaluation

Les indicateurs dépendent du type de problème :

◆ **Classification :**

- **Accuracy** : proportion de prédictions correctes
- **Precision** : proportion de vrais positifs parmi les positifs prédits
- **Recall** : proportion de vrais positifs détectés parmi les vrais
- **F1-score** : moyenne harmonique de precision et recall
- **AUC (Area Under Curve)** : capacité du modèle à distinguer les classes

◆ **Régression :**

- **MSE (Mean Squared Error)** : erreur quadratique moyenne
- **RMSE (Root MSE)** : racine de l'erreur quadratique
- **MAE (Mean Absolute Error)** : erreur absolue moyenne
- **$R^2$ (coefficient de détermination)** : pourcentage de variance expliquée

---

## 4.2. Validation croisée (Cross-validation)

Technique pour tester la robustesse d'un modèle :

- Le dataset est découpé en **k sous-ensembles** (folds).
- Le modèle est entraîné sur k-1 folds et testé sur le fold restant.
- On répète k fois, puis on calcule la moyenne des scores.

→ Permet de limiter les biais liés à un seul découpage train/test.

---

### 4.3. Optimisation d'hyperparamètres

Chaque modèle possède des **paramètres de configuration** (ex : profondeur d'un arbre, taux d'apprentissage).
L'objectif est de trouver la meilleure combinaison via :

- **Grid Search**
- **Random Search** (échantillonnage aléatoire)

Exemple en Spark MLlib :

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

paramGrid = (ParamGridBuilder()
               .addGrid(model.maxDepth, [3, 5, 7])
               .addGrid(model.numTrees, [20, 50])
               .build())
```

---

# Module 2 : Apache Spark

## 1 — Introduction à Apache Spark

### 1.1. Qu'est-ce que Spark ?

**Apache Spark** est un moteur de traitement de données **rapide, distribué et tolérant aux pannes**, conçu pour analyser des volumes massifs de données.
Contrairement aux traitements classiques, Spark permet de **paralléliser les calculs** sur plusieurs machines d'un cluster.

**Points clés :**

- Traitement en mémoire (plus rapide que Hadoop MapReduce).

- Compatible avec de nombreux formats : CSV, JSON, Parquet, ORC, etc.
- Fournit des APIs pour **Python (PySpark), Scala, Java, R**.
- Inclut des modules spécialisés :
    - **SparkSQL** → SQL et DataFrames
    - **MLlib** → machine learning distribué
    - **Spark Streaming** → traitement de flux en temps réel

Exemple : calculer la moyenne de ventes sur 1 million de lignes en quelques secondes sur un cluster Spark, alors qu'un script classique Python serait beaucoup plus lent.

---

### 1.2. Pourquoi Spark pour le Machine Learning ?

- Les modèles ML nécessitent souvent de manipuler de **grandes quantités de données**.
- Spark permet de traiter des datasets qui **ne tiennent pas dans la mémoire d'un seul ordinateur**.
- MLlib propose des **implémentations distribuées** de nombreux algorithmes ML classiques.

---

# 2 — Architecture de Spark

## 2.1. Composants principaux

**◆ Driver**

- Coordonne l'exécution des tâches sur le cluster.

**◆ Executors**

- Exécutent les tâches sur les **nœuds du cluster**.
- Stockent les données en mémoire pour accélérer les calculs.

**◆ Cluster Manager**

- Alloue les ressources du cluster aux jobs Spark.

---

## 2.2. RDD (Resilient Distributed Dataset)

**Définition :**
RDD est la **structure de données fondamentale de Spark**, distribuée sur le cluster et **immuable**.

- Résilient : tolérance aux pannes
- Distribué : les données sont **partitionnées sur plusieurs nœuds**.
- Dataset : contient des objets ou des lignes de données.

**Exemple PySpark :**

```
from pyspark import SparkContext

sc = SparkContext()
rdd = sc.parallelize([1, 2, 3, 4, 5])
rdd_squared = rdd.map(lambda x: x**2)
print(rdd_squared.collect())
# Résultat : [1, 4, 9, 16, 25]
```

---

## 2.3. DataFrame

**Définition :**
Un **DataFrame** est une abstraction haut niveau au-dessus des RDD, similaire à un **tableau Pandas**, mais distribué.

**Exemple :**

```
from pyspark.sql import SparkSession

spark = SparkSession.builder.appName("ExempleDF").getOrCreate()
data = [("Alice", 25), ("Bob", 30), ("Charlie", 28)]
df = spark.createDataFrame(data, ["Nom", "Age"])
df.show()
```

**Résultat :**

```
+-------+---+
|   Nom |Age|
+-------+---+
|  Alice| 25|
|    Bob| 30|
|Charlie| 28|
+-------+---+
```

---

# 3 — SparkSQL

## 3.1. Définition

**SparkSQL** permet de **manipuler des DataFrames via du SQL** ou des API Python/Scala.

- Compatible avec SQL standard
- Optimisé pour le traitement distribué

## 3.2. Exemples

### ◆ Requêtes SQL sur DataFrame

```
df.createOrReplaceTempView("personnes")
result = spark.sql("SELECT Nom, Age FROM personnes WHERE Age > 26")
result.show()
```

**Résultat :**

```
+-------+---+
|   Nom |Age|
+-------+---+
|    Bob| 30|
|Charlie| 28|
+-------+---+
```

---

# 4 — Programmation avec PySpark

## 4.1. Lecture et manipulation de données

**Lecture de fichiers :**

```
df_csv = spark.read.csv("data.csv", header=True, inferSchema=True)
df_parquet = spark.read.parquet("data.parquet")
```

**Sélection et filtrage :**

```
df.select("Nom", "Age").filter(df.Age > 25).show()
```

**Agrégation :**

```
df.groupBy("Age").count().show()
```

---

## 4.2. Transformations et actions

- **Transformations** : créent un nouvel RDD/DataFrame
  Ex : `map()`, `filter()`, `select()`, `groupBy()`
- **Actions** : déclenchent le calcul et retournent un résultat
  Ex : `collect()`, `count()`, `show()`

```
# Transformation
rdd2 = rdd.map(lambda x: x*2)

# Action
print(rdd2.collect())  # [2,4,6,8,10]
```

---

### 4.3. Gestion du schéma

- Définir explicitement les colonnes et types : `StructType`, `StringType`, `IntegerType`

```
from pyspark.sql.types import StructType, StructField, StringType,
IntegerType

schema = StructType([
    StructField("Nom", StringType(), True),
    StructField("Age", IntegerType(), True)
])

df = spark.createDataFrame(data, schema)
df.printSchema()
```

---

# 5 — Gestion des partitions et parallélisation

## 5.1. Partitionnement

- Les données sont **divisées en partitions** sur les différents nœuds du cluster.
- Chaque partition est traitée en parallèle par un **executor**.

**Exemple :**

```
rdd = sc.parallelize(range(10), 3)  # 3 partitions
print(rdd.getNumPartitions())  # 3
```

- Ajuster le nombre de partitions peut améliorer les performances selon la taille des données.

---

## 5.2. Parallélisation et performances

- Spark exécute les transformations sur chaque partition **en parallèle**, ce qui accélère les calculs.

- **Cache** : permet de stocker un RDD/DataFrame en mémoire pour réutilisation multiple.

# Module 3:

# SPARK MLlib :PIPELINE ET WORKFLOW DE MACHINE LEARNING )

---

# 1 — Introduction à Spark MLlib

## 1.1. Qu'est-ce que Spark MLlib ?

**Spark MLlib** est la librairie de Machine Learning distribuée intégrée à Apache Spark.
Elle permet de créer des modèles ML qui fonctionnent sur **des datasets massifs**, avec des algorithmes optimisés pour **l'exécution parallèle sur un cluster**.

**Avantages principaux :**

- Exécution **distribuée** sur plusieurs nœuds
- Intégration avec **DataFrames et SparkSQL**
- Pipeline complet : préparation → modèle → évaluation → industrialisation
- Algorithmes supervisés et non supervisés : régression, classification, clustering

---

## 1.2. Pourquoi utiliser un pipeline ML ?

Un **pipeline ML** permet de standardiser et **reproduire** un workflow de machine learning complet :

- Préparation des données
- Entraînement d'un modèle
- Évaluation de la performance

Avantage : le pipeline peut être **industrialisé**, sauvegardé et appliqué à de nouvelles données sans réécrire de code.

---

# 2 — Composants d'un pipeline MLlib

## 2.1. Transformers

**Définition :** objets qui **transforment les données** en nouvelles colonnes.
Exemples :

- `Tokenizer` → découpe du texte en mots
- `StringIndexer` → convertit des catégories en indices numériques
- `OneHotEncoder` → encode les catégories en vecteurs binaires
- `StandardScaler` → standardise les features

Tous les Transformers créent une **nouvelle colonne** dans le DataFrame.

---

## 2.2. Estimators

**Définition :** objets qui **apprennent un modèle** à partir des données.
Exemples :

- `LogisticRegression` → classification binaire
- `LinearRegression` → prédiction continue
- `DecisionTreeClassifier` → arbre de décision
- `KMeans` → clustering

Les Estimators génèrent un **modèle entraîné (Model)** qui peut être appliqué à de nouvelles données.

---

## 2.3. Evaluators

**Définition :** objets qui **mesurent la performance d'un modèle**.

- `RegressionEvaluator` → métriques RMSE, MSE, $R^2$ pour la régression

- `BinaryClassificationEvaluator` → métriques AUC, ROC, accuracy pour la classification
- `MulticlassClassificationEvaluator` → accuracy, f1-score pour multi-classes

Les Evaluators sont utilisés **après l'entraînement** pour valider la qualité du modèle.

---

# 3 — Préparation des données

## 3.1. Étapes clés de préparation

Avant d'entraîner un modèle, les données doivent être **préparées et converties en vecteur de features** :

1. **StringIndexer** : transforme les colonnes catégorielles en indices numériques
2. **OneHotEncoder** : encode les catégories en vecteurs binaires
3. **VectorAssembler** : combine toutes les features en une seule colonne `features`
4. **StandardScaler** : normalise ou standardise les valeurs numériques
5. **Gestion des valeurs manquantes** : `fillna()` ou suppression de lignes

---

## 3.2. Exemple concret

Supposons un dataset client :

| Sexe | Age | Salaire | Label |
|---|---|---|---|
| Homme | 25 | 3000 | 0 |
| Femme | 30 | 4500 | 1 |

**Préparation :**

```
from pyspark.ml.feature import StringIndexer, VectorAssembler,
StandardScaler

# Indexer la colonne Sexe
indexer = StringIndexer(inputCol="Sexe", outputCol="SexeIndex")

# Assembler toutes les features en une colonne
assembler = VectorAssembler(inputCols=["SexeIndex","Age","Salaire"],
outputCol="features")

# Standardiser
scaler = StandardScaler(inputCol="features", outputCol="scaledFeatures")
```

Le pipeline pourra ensuite utiliser `scaledFeatures` comme entrée pour l'estimateur.

---

# 4 — Construction du pipeline

## 4.1. Définition d'un pipeline

Un pipeline ML est une **suite ordonnée d'étapes** :

```
[Transformer1, Transformer2, ..., Estimator]
```

- Chaque étape transforme le DataFrame ou entraîne un modèle
- Le pipeline s'exécute en un seul appel `fit()` sur le dataset d'entraînement

---

## 4.2. Exemple complet

```
from pyspark.ml import Pipeline
from pyspark.ml.classification import LogisticRegression

# Étapes
pipeline = Pipeline(stages=[indexer, assembler, scaler,
LogisticRegression(featuresCol="scaledFeatures", labelCol="Label")])

# Entraîner le pipeline
model = pipeline.fit(train_df)

# Prédictions
predictions = model.transform(test_df)
predictions.show()
```

Avantage : **chaque transformation et modèle est intégré dans un objet unique**, ce qui simplifie l'industrialisation.

---

# 5 — Division du dataset et évaluation

## 5.1. Train/Test split

Avant d'évaluer un modèle, il faut diviser le dataset :

```
train_df, test_df = df.randomSplit([0.8,0.2], seed=42)
```

- 80% pour l'entraînement
- 20% pour le test
- `seed` fixe la répartition pour la reproductibilité

---

### 5.2. Évaluation avec Evaluator

Exemple pour classification binaire :

```
from pyspark.ml.evaluation import BinaryClassificationEvaluator

evaluator = BinaryClassificationEvaluator(labelCol="Label",
metricName="areaUnderROC")
auc = evaluator.evaluate(predictions)
print("AUC :", auc)
```

- Pour régression : `RegressionEvaluator` avec métriques RMSE, MAE, $R^2$
- Pour multi-classes : `MulticlassClassificationEvaluator`

---

# 6 — Recherche d'hyperparamètres

### 6.1. ParamGridBuilder et CrossValidator

Pour optimiser les hyperparamètres :

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

# Définir grille de paramètres
paramGrid = (ParamGridBuilder()
             .addGrid(lr.regParam, [0.01, 0.1])
             .addGrid(lr.elasticNetParam, [0.0, 0.5, 1.0])
             .build())

# Cross-validation
cv = CrossValidator(estimator=pipeline,
                    estimatorParamMaps=paramGrid,
                    evaluator=BinaryClassificationEvaluator(),
                    numFolds=5)

# Entraîner
cvModel = cv.fit(train_df)
```

Permet de tester plusieurs combinaisons et sélectionner le **meilleur modèle** automatiquement.

---

### 6.2. Avantages

- Meilleur réglage des paramètres
- Réduction du risque d'overfitting
- Méthode intégrée pour workflow complet

---

# 7 — Bonnes pratiques

1. **Préparer correctement les données** : indexer les colonnes catégorielles, gérer les valeurs manquantes
2. **Standardiser / normaliser** les features numériques
3. **Utiliser un pipeline** pour automatiser le workflow
4. **Diviser dataset train/test** pour évaluer correctement le modèle
5. **Évaluer le modèle** avec des métriques adaptées
6. **Tester les hyperparamètres** avec CrossValidator
7. **Sauvegarder le pipeline entraîné** pour réutilisation :

---

# 8 — Résumé et workflow complet

### Workflow Spark MLlib :

1. **Chargement des données** → DataFrame Spark
2. **Préparation** → Transformers : StringIndexer, OneHotEncoder, VectorAssembler, StandardScaler
3. **Division train/test**
4. **Pipeline** → Transformers + Estimator
5. **Entraînement** → pipeline.fit(train_df)
6. **Prédictions** → pipeline.transform(test_df)
7. **Évaluation** → BinaryClassificationEvaluator / RegressionEvaluator
8. **Optimisation hyperparamètres** → ParamGridBuilder + CrossValidator
9. **Industrialisation** → sauvegarde et réutilisation du pipeline

Un pipeline Spark MLlib permet de **répliquer, scaler et industri**