Cloud & Big Data
Theory and Practice

MSc. AIB — 2025-2026

aivancity

Paris Île-de-France · Nice Côte d'Azur

Chapitre 1: Cloud Computing in Practice : PySpark

Travail à rendre

Afin de structurer le travail et d'évaluer les compétences de manière progressive, tous les étudiants doivent réaliser l'Exercice commun, portant sur la création, la manipulation et la transformation de DataFrames en PySpark.

Ensuite, vous devez choisir un seul exercice spécialisé parmi les trois proposés :

- Exercice avancé PySpark (structure et transformations) sur le dataset netflix
- Exercice d'analyse par SQL du dataset movies
- Exercice apprentissage automatique (Kickstarter).

Ce second exercice sera choisi librement en fonction de vos intérêts, de votre aisance technique ou de votre projet professionnel.

L'ensemble du travail devra être déposé sur Blackboard sous la forme d'un fichier .py ou .ipynb. Le code rendu doit être propre, commenté lorsque nécessaire, lisible et structuré. Vous êtes vivement encouragés à créer autant de fonctions que nécessaire pour organiser votre code de manière claire et modulaire. Le code doit être rendu pour le 9 janvier au plus tard.

Directives générales :

- Respectez la structure des fonctions demandées dans l'énoncé.
- Toute dépendance externe inutile doit être évitée ; utilisez uniquement les bibliothèques autorisées en cours.
- Votre code doit pouvoir être exécuté en local sans modification.
- Les résultats intermédiaires ou finaux doivent être clairement affichés.
- Veillez à respecter les bonnes pratiques vues en cours (documentation, nommage, structuration du code).
- Le rendu final doit permettre de comprendre facilement votre raisonnement, vos choix d'implémentation et la logique de votre pipeline de traitement ou d'apprentissage.

Part 1.1

Exercice commun : PySpark Data Structure

Nous pouvons créer nos propres DataFrame (version PySpark) avec la méthode :

```
from pyspark.sql import SparkSession

spark = SparkSession.builder \
    .appName("PySparkHomework") \
    .master("local[*]") \
    .getOrCreate()

    ...

    df = spark.createDataFrame(data,["id", "value1", "value2"]
```

Cette syntaxe sous entend que `data` est une liste de tuple de la forme (id, value, value).

Question 1.1: Exercices

a) Implémentez une fonction pour générer une liste aléatoire de triplet. Le premier élément doit être un int allant de 0 à n sans doublons. Le second élément est un entier variant de 0 à 50 et le dernier un entier entre 0 et 5.

b) Implémentez la fonction `init_df(nb_rows:int) -> DataFrame` permettant de générer un DataFrame aléatoire de taille nb_rows et ayant trois colonnes : id, value1, value2.

c) Implémentez la fonction `transformation(df:DataFrame) -> DataFrame` qui
- normalise `value1` entre 0 et 1 ;
- ajoute une colonne `ratio = value1 / value2` ;
- filtre les colonnes dont le ratio est inférieur au ratio median.
- retourne le résultat

d) Implémentez la fonction `merge(df1, df2)` prenant 2 DataFrame un DataFrame représentant la fusion des deux :
- Les lignes de même IDs voient leurs valeurs ajoutées (pensez à mettre le ratio à jour)
- Les lignes sans équivalent d'ID sont ajoutées telles quelles.

Part 1.2

Exercice Spécialisé : Analyse PySpark du dataset `netflix`

On vous fournit un extrait anonymisé d'un dataset Netflix contenant les colonnes suivantes :

- `show_id` : identifiant unique du contenu,
- `type` : "Movie" ou "TV Show",
- `title` : nom du contenu,
- `director` : nom du ou des réalisateurs,
- `cast` : liste des acteurs séparés par ",",
- `country` : pays de production,
- `date_added` : date d'ajout au catalogue,
- `release_year` : année de sortie,
- `rating` : classification (PG, R, ...),
- `duration` : durée en minutes (pour les films) ou "X Seasons" (pour les séries),
- `listed_in` : catégories séparées par ",",
- `description` : courte description.

Vous utiliserez la méthode :

```
df = spark.read.option("header", "true") \\
\phantom{df}    .option("inferSchema", "true") \\
\phantom{df}    .csv("netflix\_sample.csv")
```

Question 1.1: Exercices

a) Implémentez une fonction `prepare(df:DataFrame) -> DataFrame` qui :
- sépare correctement la colonne `duration` en deux colonnes : `duration_min` (entier) et `duration_season` (entier) ;
- convertit `date_added` au format `Date` ;
- normalise les catégories dans `listed_in` en une liste (array de strings).

b) Implémentez une fonction `top_countries(df)` qui retourne pour chaque type de contenu (Movie / TV Show) les **3 pays** qui produisent le plus de titres. Vous devez utiliser les fonctions de ranking (Window + `dense_rank`).

c) Implémentez une fonction `actor_stats(df)` qui construit un DataFrame contenant pour chaque acteur :
- le nombre total de contenus dans lesquels il apparaît ;
- le nombre de films ;
- le nombre de séries ;
- la diversité de genres (`listed_in`) dans lesquels il apparaît.

Vous devez utiliser `explode` et des agrégations complexes.

d) Une "collaboration" est définie comme un couple d'acteurs apparaissant ensemble dans au moins un titre. Implémentez `actor_pairs(df)` retournant le top 10 des couples d'acteurs collaborant le plus fréquemment. On attend :

$$(actor\_1, actor\_2, nb\_collaborations)$$

Indice : utiliser une combinaison `explode` + `crossJoin` filtrée.

e) **Analyse temporelle avancée.** Implémentez une fonction `time_analysis(df)` qui retourne :
- le nombre de nouveaux contenus ajoutés par mois ;
- la moyenne annuelle des nouvelles productions (basée sur `release_year`) ;
- la croissance (ou décroissance) des ajouts Netflix par un modèle linéaire simple via `pyspark.ml.regression.LinearRe`

Vous devez produire un DataFrame final contenant :

$$(\text{year}, \text{nb\_added}, \text{regression\_trend})$$

f) Implémentez une fonction `cluster_contents(df)` qui réalise un clustering K-Means basé sur :
- la durée en minutes (0 si série),
- l'année de sortie,
- la taille du casting.

Les étapes attendues :
- création d'un vecteur de features (`VectorAssembler`),
- normalisation (`StandardScaler`),
- clustering avec `KMeans(k=4)`,
- retour d'un DataFrame annoté avec sa classe de cluster.

Part 1.3

Exercice Spécialisé : Modélisation prédictive avec le dataset Kickstarter

On vous fournit un DataFrame PySpark issu du fichier `ks-projects-201612.csv`, contenant entre autres les colonnes :

- `ID` : identifiant du projet
- `name` : nom du projet
- `category`, `main_category`
- `currency`
- `deadline`, `launched`
- `goal`, `pledged`, `backers`
- `state` : *successful*, *failed*, etc.

L'objectif de cet exercice est de construire un pipeline d'apprentissage supervisé permettant de prédire si un projet sera *successful* ou *failed*.

Question 1.1: Exercices

a) **Nettoyage du dataset.** Implémentez une fonction `clean_df(df:DataFrame) -> DataFrame` qui :
   - filtre les lignes où `state` est soit `"successful"`, soit `"failed"` (**Fonctions :** `filter`, `isin`) ;
   - supprime les lignes où `goal` ≤ 0 (**Fonctions :** `filter`).

b) **Ajout de variables temporelles.** Implémentez la fonction `add_time_features(df) -> DataFrame` qui :
   - convertit `deadline` et `launched` en colonnes `timestamp` (**Fonction :** `to_timestamp`) ;
   - crée une colonne `duration_days` représentant la durée de la campagne (**Fonction :** `datediff`).

c) **Encodage des variables catégorielles.** Implémentez la fonction `encode(df) -> DataFrame, stages` qui encode :
   - `main_category` avec `StringIndexer` ;
   - `currency` avec `StringIndexer` ;
   - assemble les colonnes numériques dans un vecteur `features` (**Fonction :** `VectorAssembler`).

   La fonction doit retourner les stages à utiliser dans le pipeline.

d) **Séparation du dataset.** Implémentez `split(df) -> (train, test)` qui divise le dataset final en deux parties : 80% / 20%. (**Fonction :** `randomSplit`)

e) **Entraînement du modèle.** Implémentez `train_model(train, stages)` qui :
   - ajoute un `LogisticRegression` aux stages du pipeline ;
   - entraîne un `Pipeline` complet sur les données d'entraînement (**Fonctions :** `Pipeline`, `fit`).

f) **Évaluation du modèle.** Implémentez `evaluate(model, test)` qui :
   - applique le modèle au test set (`transform`) ;
   - calcule l'accuracy du modèle grâce à `MulticlassClassificationEvaluator`.

g) **Analyse des coefficients.** Implémentez la fonction `inspect(model)` qui récupère :
   - les coefficients du modèle (**via** `model.stages[-1].coefficients`) ;
   - l'intercept (`model.stages[-1].intercept`).

   L'objectif est d'interpréter quelles variables ont le plus d'impact sur la probabilité de succès.

Part 1.4

Exercices Spécialisé : Movies et SQL

On vous fournit un DataFrame PySpark issu du fichier `movies.csv`, enregistré sous forme de table SQL temporaire via :

```
df = spark.read.option("header","true").option("inferSchema","true").csv("movies.csv")
df.createOrReplaceTempView("movies")
```

Les colonnes disponibles sont :

- `movieId` : identifiant unique
- `title` : titre du film (souvent avec l'année entre parenthèses, ex. *The Matrix (1999)*)
- `genres` : liste de genres séparés par "|", par exemple *Adventure|Sci-Fi|Action*

Répondez exclusivement en SQL (Spark SQL).

Question 1.1: Exercices

a) **Extraction de l'année de sortie.** Écrire une requête créant une colonne `year` extraite du titre (nombre entre parenthèses à la fin du titre). Vous devez retourner les colonnes : `movieId`, `title`, `year`. *Indication :* utilisez `regexp_extract`.

b) **Comptage des genres les plus populaires.** Chaque film peut appartenir à plusieurs genres séparés par "|". Écrire une requête SQL affichant les 10 genres les plus fréquents, avec les colonnes : `genre`, `nb_films`. *Indication :* utilisez `split` et `explode`.

c) **Films multi-genres.** Écrire une requête SQL retournant tous les films appartenant à au moins **3 genres**, avec les colonnes : `movieId`, `title`, `nb_genres`. *Indication :* utilisez `size(split(...))`.

d) **Années les plus productives.** À partir de l'année extraite dans la question 1, écrire une requête qui retourne les **5 années** ayant produit le plus de films, triées par nombre décroissant. Vous devez utiliser une CTE (Common Table Expression).

e) **Genres dominants par décennie.** Construire une requête SQL qui :

- calcule une colonne `decade` = `year - (year % 10)`
- explode les genres
- retourne, pour chaque décennie, le genre le plus représenté

Vous devez utiliser une fonction de fenêtre (`row_number`).

f) **Détection des titres dupliqués.** Certains films partagent exactement le même titre mais ont des `movieId` différents. Écrire une requête SQL retournant la liste des titres présents au moins en double, avec les colonnes : `title`, `nb_versions`, `ids_list`. *Indication :* utilisez `collect_list` dans la clause `GROUP BY`.