# Série TP 2: Spark

Ces exercices incluent la gestion de données météorologiques en streaming, l'analyse des anomalies, ainsi que e

## I. Préparation de l'Environnement PySpark

### 1. Installer PySpark

Google Colab ne vient pas avec PySpark préinstallé, alors nous allons l'installer. Dans une cellule Colab, exécutez :

```
!apt-get install openjdk-8-jdk-headless -qq > /dev/null
!wget -q https://archive.apache.org/dist/spark/spark-3.1.1/spark-3.1.1-bin-hadoop2.7.tgz
!tar xf spark-3.1.1-bin-hadoop2.7.tgz
!pip install -q findspark
```

### 2. Configurer les Variables d'Environnement

Configurez les variables d'environnement pour que Colab puisse utiliser Spark. Exécutez ce code :

```
import os
os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-8-openjdk-amd64"
os.environ["SPARK_HOME"] = "/content/spark-3.1.1-bin-hadoop2.7"
```

### 3. Initialiser PySpark

Lancez PySpark en important et initialisant findspark :

```
import findspark
findspark.init()
from pyspark.sql import SparkSession
spark = SparkSession.builder \
   .appName("SparkTPs") \
   .config("spark.sql.streaming.schemaInference", True) \
   .getOrCreate()
print("Spark est prêt !")
```

## II. Travaux pratiques :

### Exercice 1 : Annalyse de données météo en streaming

#### Objectifs

1. Générer des fichiers JSON représentant des données météo (température, humidité, pression) en continu.
2. Traiter les fichiers en streaming avec Spark.
3. Détecter des anomalies comme des températures extrêmes.
4. Regrouper les données météo par ville et calculer des statistiques en streaming (température moyenne, humidité moyenne).
5. Utiliser des fenêtres temporelles pour détecter les variations de température par ville.

#### Simulation de données :

Pour simuler une source de données, générer des fichiers JSON représentant des données météo (température, humidité, pression) en continu :

```
import json
import random
import time
```

```
# Chemin où les données seront écrites
output_dir = "/content/streaming_data/"
os.makedirs(output_dir, exist_ok=True)

# Fonction pour générer des données météo
def generate_weather_data(output_dir, num_files=10, delay=2):
    cities = ["Paris", "Lyon", "Marseille", "Toulouse", "Nice"]
    for i in range(num_files):
        data = {
            "timestamp": time.time(),
            "city": random.choice(cities),
            "temperature": round(random.uniform(-10, 40), 2),
            "humidity": random.randint(20, 100),
            "pressure": random.randint(950, 1050)
        }
        filename = os.path.join(output_dir, f"weather_data_{i}.json")
        with open(filename, "w") as f:
            json.dump(data, f)
        print(f"Fichier généré : {filename}")
        time.sleep(delay)

# Générer les fichiers de données
generate_weather_data(output_dir)
```

## Exercice 2 : Analyse de données Booking d'hotels avec PySpark et LlaMa

Les en utilisant Apache Spark sur Colab pour travailler sur des données de streaming et un système de recommandation basé sur LLaMA (un modèle de langage de Meta).

- Configurer Spark Streaming pour lire les fichiers générés en continu.
- Traiter et nettoyer les données (par exemple, filtrer ou agréger les informations).
- Extraire des features utiles à partir des données de réservation (ex. utilisateur, hôtel, notes, etc.).
- Diviser les données en ensembles d'entraînement et de test.
- Utiliser PySpark MLlib pour générer des recommandations de base à l'aide d'algorithmes de factorisation matricielle.
- Ensuite, intégrer le modèle LLaMA pour améliorer les recommandations en utilisant du NLP pour analyser les préférences des utilisateurs ou les descriptions d'hôtels.
- Tester le système en fournissant des entrées utilisateur simulées.
- Afficher les recommandations et les mesures d'évaluation (ex. RMSE, précision).

### Simulation de données :

```
import time
import json
import random
import os

output_folder = "/content/streaming_data"
os.makedirs(output_folder, exist_ok=True)

hotels = ["Hotel_A", "Hotel_B", "Hotel_C", "Hotel_D"]
users = ["User_1", "User_2", "User_3", "User_4"]
ratings = [1, 2, 3, 4, 5]

for i in range(10):  # Générer 10 fichiers
    data = [
```

```
    {
        "user": random.choice(users),
        "hotel": random.choice(hotels),
        "rating": random.choice(ratings),
        "timestamp": time.time()
    }
    for _ in range(100)
]
file_path = os.path.join(output_folder, f"data_{i}.json")
with open(file_path, "w") as f:
    json.dump(data, f)
print(f"Généré: {file_path}")
time.sleep(2)  # Attendre 2 secondes avant de générer un nouveau fichier
```