Atelier N° 03: Exploration et Analyse de Données de Transport avec PySpark

→ Objectifs

- Créer et manipuler un environnement Spark dans Google Colab.
- Charger, nettoyer et enrichir un dataset volumineux.
- Réaliser des analyses avec RDD, DataFrame et Spark SQL.
- Visualiser les résultats avec matplotlib/pandas.

L'entreprise **TransData Maroc** souhaite mieux comprendre la performance de ses services de transport (bus, trains, taxis, etc.). Chaque jour, elle génère des fichiers CSV contenant les trajets effectués à travers les régions du pays.

Partie 1 : Préparation de l'environnement

1. Exécution sur Google Colab

Pour faciliter l'exécution du code PySpark sans installation locale, l'environnement **Google Colab** peut être utilisé. Il suffit d'ouvrir un notebook Colab et d'exécuter les commandes suivantes pour installer et configurer **PySpark** :

```
!apt-get update -qq && apt-get install -y openjdk-11-jdk
version = "3.5.6"
hadoop_version = "hadoop3"
url = f"https://downloads.apache.org/spark/spark-{version}/spark-{version}-bin-{hadoop_version}.tgz"
!wget {url} -O spark-{version}-bin-{hadoop_version}.tgz
!ls -lh spark-{version}-bin-{hadoop_version}.tgz
!tar -xvzf spark-{version}-bin-{hadoop_version}.tgz
!pip install -q findspark
import os, findspark
os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-11-openjdk-amd64"
os.environ["SPARK_HOME"] = f"/content/spark-{version}-bin-{hadoop_version}"
findspark.init()
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName("Atelier_PySpark").getOrCreate()
spark
```

Une fois ces cellules exécutées, il devient possible de créer une **SparkSession** et d'exécuter des programmes PySpark directement dans Colab

2. Exécution locale avec Docker

Une autre option consiste à utiliser **Docker**, qui permet de créer un environnement isolé avec **JupyterLab** et **Apache Spark** déjà installés via l'image officielle : [jupyter/all-spark-notebook]

L'image peut être téléchargée directement depuis **Docker Hub** ou à l'aide de la commande suivante dans ***PowerShell*** :

```
docker pull jupyter/all-spark-notebook
```

→ Lancement du conteneur

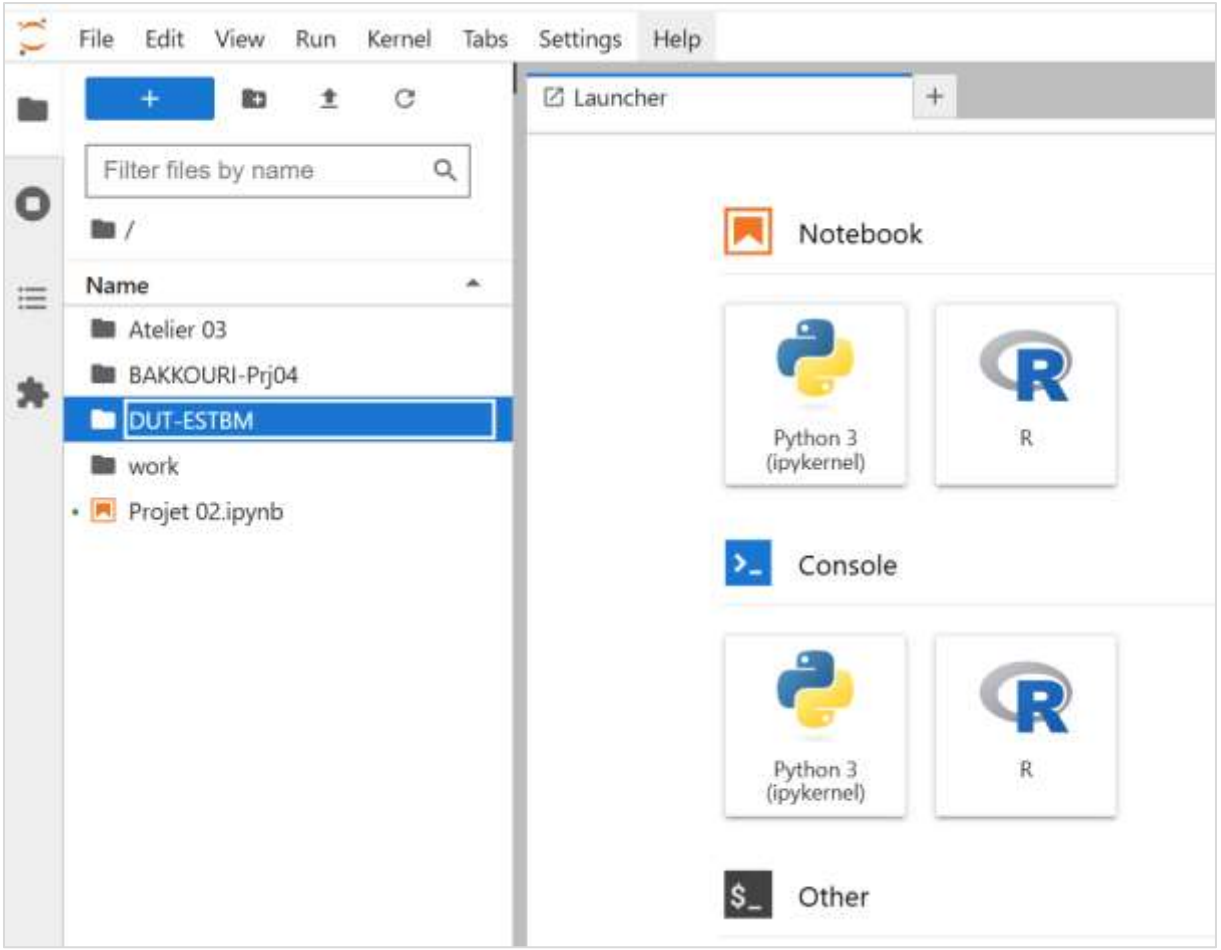
Pour démarrer un serveur ***JupyterLab*** avec Spark intégré

```
docker run -it --rm -p 8888:8888 jupyter/all-spark-notebook
```

Le terminal affichera un lien du type : ***http://127.0.0.1:8888/lab?token=xxxxxx*** . Copier ce lien et le coller dans le navigateur pour accéder à l'interface **JupyterLab**.

→ Création d'un notebook PySpark

Depuis l'interface, créer un **nouveau notebook Python 3**, puis exécuter le code suivant pour vérifier le bon fonctionnement de PySpark



Partie 2 : Chargement et exploration

Nous souhaitons générer un jeu de données synthétique représentant des trajets.

→ Chaque trajet contient plusieurs informations : la ***région de départ*** et la ***région d'arrivée***, la ***distance*** en ***kilomètres***, la ***durée*** en minutes, le ***nombre de passagers***, le ***revenu*** généré et le ***moyen de transport*** utilisé (***Bus***, ***Train***, ***Taxi***, ***Tram***).

→ Au total, nous avons généré **1000 trajets** aléatoires, que nous avons ensuite stockés dans un fichier CSV (***`trajets.csv`***) afin de pouvoir les réutiliser facilement. Ce fichier a ensuite été chargé dans un DataFrame PySpark, ce qui nous permet d'explorer les données, d'afficher leur structure et de visualiser quelques exemples. Cette étape illustre la création d'un dataset artificiel.

```
import pandas as pd
from random import randint, choice, uniform

regions = ["Casablanca", "Rabat", "Marrakech", "Fès", "Agadir"]
moyens = ["Bus", "Train", "Taxi", "Tram"]

data = [{
    "trajet_id": i,
    "region_depart": choice(regions),
    "region_arrivee": choice(regions),
    "distance_km": round(uniform(5, 600), 1),
    "duree_min": randint(10, 600),
    "nb_passagers": randint(1, 50),
    "revenu": round(uniform(50, 5000), 2),
    "moyen_transport": choice(moyens)
} for i in range(1000)]

df = pd.DataFrame(data)
df.to_csv("trajets.csv", index=False)
df_spark = spark.read.csv("trajets.csv", header=True, inferSchema=True)
df_spark.printSchema()
df_spark.show(5)
```

Partie 3 : Nettoyage et enrichissement

Dans cette partie, nous nettoyons et enrichissons le dataset PySpark.

- On filtre les trajets pour ne garder que ceux dont la distance est positive.
- On ajoute de nouvelles colonnes calculées **:**
  - ***`revenu_par_km`*** : revenu par kilomètre.
  - ***`revenu_par_passager`*** : revenu par passager.
- On crée une colonne catégorielle ***`categorie_distance`*** qui classe les trajets en « Courte », « Moyenne » ou « Longue » selon leur distance.
- Enfin, on affiche un aperçu des 5 premières lignes du DataFrame enrichi.

Cette étape permet d'améliorer la qualité des données et de préparer le dataset pour des analyses plus pertinentes.

```
from pyspark.sql.functions import col, round as sround, when

df_clean = df_spark.filter(col("distance_km") > 0)
df_clean = df_clean.withColumn("revenu_par_km", sround(col("revenu")/col("distance_km"), 2))
df_clean = df_clean.withColumn("revenu_par_passager", sround(col("revenu")/col("nb_passagers"), 2))
df_clean = df_clean.withColumn(
    "categorie_distance",
    when(col("distance_km") <= 100, "Courte")
    .when(col("distance_km") <= 400, "Moyenne")
    .otherwise("Longue")
)
df_clean.show(5)
```

Partie 4 : Analyse via RDD

Dans cette partie, nous utilisons les RDD pour analyser les données :

- Conversion du DataFrame en RDD.
- Calcul du revenu total par région de départ**.**
- Affichage des 5 régions les plus rentables.

```
rdd = df_clean.rdd
revenu_par_region = rdd.map(lambda x: (x["region_depart"], x["revenu"])) \
                       .reduceByKey(lambda a,b: a+b) \
                       .sortBy(lambda x: x[1], ascending=False)
revenu_par_region.take(5)
```

Partie 5 : Analyse avancée (SQL & DataFrame)

Dans cette partie, nous voulons :

- Calculer le **revenu total par région de départ**.
- Calculer le **revenu moyen, distance moyenne et nombre de trajets par moyen de transport**.
- Obtenir le **revenu moyen et le nombre de trajets par catégorie de distance** avec SQL.

```python
from pyspark.sql.functions import avg, sum as ssum, count

# revenu total par région
revenu_region = df_clean.groupBy("region_depart").agg(ssum("revenu").alias("revenu_total"))
revenu_region.show()

# revenu moyen et distance moyenne par transport
stats_transport = df_clean.groupBy("moyen_transport").agg(
    avg("revenu").alias("revenu_moyen"),
    avg("distance_km").alias("distance_moyenne"),
    count("*").alias("nb_trajets")
)
stats_transport.show()
df_clean.createOrReplaceTempView("trajets")
spark.sql("""
SELECT categorie_distance, AVG(revenu) AS revenu_moyen, COUNT(*) AS nb_trajets
FROM trajets GROUP BY categorie_distance
""").show()
```

Partie 6 : Visualisation et interprétation

Dans cette partie, nous voulons :

- Visualiser ***le revenu total par région de départ*** avec un graphique en barres.
- Comparer plusieurs indicateurs moyens par moyen de transport (***revenu moyen, distance moyenne, nombre de trajets***) avec un graphique en barres.
- **Interpréter les résultats** pour identifier les tendances et insights à partir des données.

```python
#1. Conversion vers Pandas
pdf = revenu_region.toPandas()
#2. Graphique en barres
import matplotlib.pyplot as plt

plt.figure(figsize=(8,4))
plt.bar(pdf["region_depart"], pdf["revenu_total"])
plt.title("Revenu total par région de départ")
plt.xlabel("Région")
plt.ylabel("Revenu total (DH)")
plt.show()
#3. Comparaison multi-indicateurs
pdf_stats = stats_transport.toPandas()
pdf_stats.plot(x="moyen_transport", kind="bar", figsize=(8,4))
plt.title("Comparaison des indicateurs moyens par transport")
plt.grid(True)
plt.show()
```

Partie 7 : Exercice pratique

Dans cette partie, nous allons approfondir l'analyse du dataset ***`trajets.csv`*** en combinant PySpark SQL et l'API DataFrame. L'objectif est de manipuler, interroger et visualiser les données pour en extraire des informations pertinentes.

Questions d'analyse

1. Calculer le revenu moyen par région de départ.
2. Calculer le nombre total de trajets par région d'arrivée.
3. Trouver le revenu total et la distance moyenne par moyen de transport.
4. Identifier la région de départ ayant le plus grand nombre de trajets.
5. Afficher les 3 trajets ayant la plus grande distance.
6. Calculer le revenu moyen par catégorie de distance (Courte, Moyenne, Longue).
7. Déterminer la moyenne et la variance du revenu pour chaque région.
8. Comparer le revenu moyen entre les différents moyens de transport.
9. Calculer le nombre et le pourcentage de trajets par moyen de transport.
10. Identifier la région générant le revenu total le plus élevé.

Visualisation

11. Créer un graphique en barres représentant le revenu total par région de départ.
12. Créer un second graphique en barres comparant le revenu moyen par moyen de transport.