Royaume du Maroc
Université Sultan Moulay Slimane
Ecole Supérieure de Technologie – Béni Mellal
Département Informatique et Techniques de Gestion

Spécialité : DUT en Sécurité Informatique et Réseaux

Compte rendu de TD N°04

Sous le thème

Analyse des consommations d'énergie

Réalisé par :

Hajar Rachid

Année universitaire 2025_2026

Objectif du TD

L'objectif de ce travail est d'analyser les consommations électriques enregistrées par les capteurs de l'entreprise EcoWatt. L'analyse a pour but d'identifier :

- Les zones les plus énergivores,
- Les périodes de surconsommation,
- Les tendances selon les types de bâtiments.

Pour ce faire, nous avons utilisé PySpark afin de charger, nettoyer et analyser les données.

I. Préparation et exploration

Exercice 1 : Chargement et découverte des données

Une session Spark appelée EcoWattSpark a été créée. Comme le fichier CSV n'était pas disponible, un petit DataFrame a été construit avec des données d'exemple contenant les colonnes suivantes : id_mesure, date_heure, ville, type_batiment, conso_kwh, temperature_ext, cout_kwh, nb_personnes.

Ensuite :

- Le schéma du DataFrame a été affiché,
- Le **nombre total d'enregistrements** a été calculé,
- Et les 5 premières lignes ont été affichées pour vérifier le contenu.

Exercice 1 : Chargement et découverte des données

```
[2]
✓ 3s
    from pyspark.sql import SparkSession

    # 1. Création de la session Spark
    spark = SparkSession.builder.appName("EcoWattSpark").getOrCreate()

    # 2. Création d'un petit DataFrame d'exemple (car on n'a pas le fichier CSV)
    data = [
        (1, "2025-01-01 10:00:00", "Beni Mellal", "Résidentiel", 500.0, 25.0, 1.2, 5),
        (2, "2025-02-10 14:00:00", "Casablanca", "Industriel", 8000.0, 30.0, 1.5, 100),
        (3, "2025-03-05 09:00:00", "Rabat", "Bureau", 2000.0, 20.0, 1.0, 20),
        (4, "2025-03-15 18:00:00", "Fès", "École", 700.0, 15.0, 1.3, 50),
        (5, "2025-04-20 22:00:00", "Marrakech", "Résidentiel", 1200.0, 35.0, 1.1, 6)
    ]

    colonnes = ["id_mesure","date_heure","ville","type_batiment","conso_kwh","temperature_ext","cout_kwh","nb_personnes"]
    df = spark.createDataFrame(data, colonnes)

    # 3. Afficher le schéma, le nombre total d'enregistrements et les 5 premières lignes
    df.printSchema()
    print("Nombre total d'enregistrements :", df.count())
    df.show()
```

```
root
 |-- id_mesure: long (nullable = true)
 |-- date_heure: string (nullable = true)
 |-- ville: string (nullable = true)
 |-- type_batiment: string (nullable = true)
 |-- conso_kwh: double (nullable = true)
 |-- temperature_ext: double (nullable = true)
 |-- cout_kwh: double (nullable = true)
 |-- nb_personnes: long (nullable = true)

Nombre total d'enregistrements : 5
+---------+-------------------+-----------+-------------+---------+---------------+--------+------------+
|id_mesure|         date_heure|      ville|type_batiment|conso_kwh|temperature_ext|cout_kwh|nb_personnes|
+---------+-------------------+-----------+-------------+---------+---------------+--------+------------+
|        1|2025-01-01 10:00:00|Beni Mellal|  Résidentiel|    500.0|           25.0|     1.2|           5|
|        2|2025-02-10 14:00:00| Casablanca|   Industriel|   8000.0|           30.0|     1.5|         100|
|        3|2025-03-05 09:00:00|      Rabat|       Bureau|   2000.0|           20.0|     1.0|          20|
|        4|2025-03-15 18:00:00|        Fès|       École|    700.0|           15.0|     1.3|          50|
|        5|2025-04-20 22:00:00|  Marrakech|  Résidentiel|   1200.0|           35.0|     1.1|           6|
+---------+-------------------+-----------+-------------+---------+---------------+--------+------------+
```

Exercice 2 : Nettoyage et enrichissement

Suppression des valeurs nulles avec dropna().

Élimination des valeurs aberrantes :

- conso_kwh inférieure à 0 ou supérieure à 20000,
- temperature_ext inférieure à -10°C ou supérieure à 50°C.

Création de deux nouvelles colonnes :

- cout_total = conso_kwh * cout_kwh
- conso_par_personne = conso_kwh / nb_personnes

Affichage du nombre de lignes avant et après le nettoyage pour vérifier la perte éventuelle de données.

Exercice 2 : Nettoyage et enrichissement

```
from pyspark.sql.functions import col

# 1. Supprimer les valeurs nulles
df_clean = df.dropna()

# 2. Supprimer les valeurs aberrantes
df_clean = df_clean.filter(
    (col("conso_kwh") >= 0) & (col("conso_kwh") <= 20000) &
    (col("temperature_ext") >= -10) & (col("temperature_ext") <= 50)
)

# 3. Créer les nouvelles colonnes
df_clean = df_clean.withColumn("cout_total", col("conso_kwh") * col("cout_kwh"))
df_clean = df_clean.withColumn("conso_par_personne", col("conso_kwh") / col("nb_personnes"))

# 4. Afficher le nombre de lignes avant / après nettoyage
print("Avant nettoyage :", df.count())
print("Après nettoyage :", df_clean.count())

df_clean.show()
```

```
Avant nettoyage : 5
Après nettoyage : 5
+---------+-------------------+-----------+-------------+---------+---------------+--------+------------+----------+------------------+
|id_mesure|        date_heure|      ville|type_batiment|conso_kwh|temperature_ext|cout_kwh|nb_personnes|cout_total|conso_par_personne|
+---------+-------------------+-----------+-------------+---------+---------------+--------+------------+----------+------------------+
|        1|2025-01-01 10:00:00|Beni Mellal|  Résidentiel|    500.0|           25.0|     1.2|           5|     600.0|             100.0|
|        2|2025-02-10 14:00:00| Casablanca|   Industriel|   8000.0|           30.0|     1.5|         100|   12000.0|              80.0|
|        3|2025-03-05 09:00:00|      Rabat|       Bureau|   2000.0|           20.0|     1.0|          20|    2000.0|             100.0|
|        4|2025-03-15 18:00:00|       Fès|       École|    700.0|           15.0|     1.3|          50|     910.0|              14.0|
|        5|2025-04-20 22:00:00|  Marrakech|  Résidentiel|   1200.0|           35.0|     1.1|           6|    1320.0|             200.0|
+---------+-------------------+-----------+-------------+---------+---------------+--------+------------+----------+------------------+
```

Exercice 3 : Analyses descriptives

Consommation totale par ville → regrouper les données avec groupBy("ville").sum("conso_kwh").

Consommation moyenne par type de bâtiment – calculée avec avg("conso_kwh").

Classement des villes de la plus énergivore à la moins énergivore grâce à orderBy(desc).

Ces résultats permettent d'identifier quelles villes consomment le plus d'énergie.

Exercice 4 : Analyse temporelle

De nouvelles colonnes ont été créées à partir de la date :

- date = to_date(date_heure)
- heure = hour(date_heure)
- jour = date_format(..., 'EEEE')
- mois = month(date_heure)

Puis, on a calculé :

- La **consommation totale** par mois,
- La **consommation moyenne par jour de la semaine,**
- Et **la consommation horaire moyenne**.

Cette étape a permis d'identifier **les périodes de forte consommation**, par exemple les soirées ou certains mois de l'année.

Exercice 4 : Analyse temporelle

```
from pyspark.sql.functions import to_date, hour, date_format, month

# 1. Créer les colonnes date, heure, jour, mois
df_time = df_clean.withColumn("date", to_date(col("date_heure"))) \
                  .withColumn("heure", hour(col("date_heure"))) \
                  .withColumn("jour", date_format(to_date(col("date_heure")), "EEEE")) \
                  .withColumn("mois", month(col("date_heure")))

# 2. Calculs demandés
# a) Consommation totale par mois
df_time.groupBy("mois").sum("conso_kwh").show()

# b) Consommation moyenne par jour de la semaine
df_time.groupBy("jour").avg("conso_kwh").show()

# c) Consommation horaire moyenne
df_time.groupBy("heure").avg("conso_kwh").show()
```

```
+----+--------------+
|mois|sum(conso_kwh)|
+----+--------------+
|   1|         500.0|
|   2|        8000.0|
|   3|        2700.0|
|   4|        1200.0|
+----+--------------+

+---------+---------------+
|     jour|avg(conso_kwh)|
+---------+---------------+
|Wednesday|         1250.0|
|   Monday|         8000.0|
| Saturday|          700.0|
|   Sunday|         1200.0|
+---------+---------------+

+-----+---------------+
|heure|avg(conso_kwh)|
+-----+---------------+
|   10|          500.0|
|   14|         8000.0|
|   22|         1200.0|
|    9|         2000.0|
|   18|          700.0|
+-----+---------------+
```

II. Analyses avancées et Spark SQL

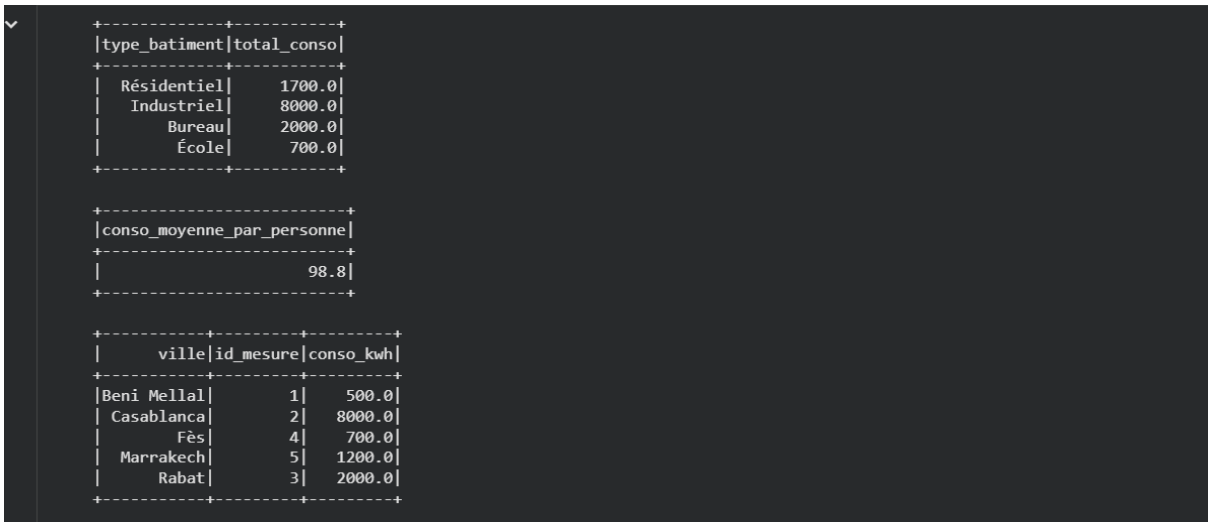
Exercice 5 : Comparaison DataFrame / SQL

Une vue temporaire `conso` a été créée pour exécuter des requêtes SQL. Les analyses ont été refaites en SQL afin de comparer les résultats :

- Consommation totale par type de bâtiment
- Consommation moyenne par personne

- Top 2 bâtiments les plus énergivores par ville (en utilisant la fonction ROW_NUMBER() OVER (...))

Les résultats obtenus en SQL étaient identiques à ceux trouvés avec les fonctions DataFrame.



Exercice 6 : Agrégations conditionnelles

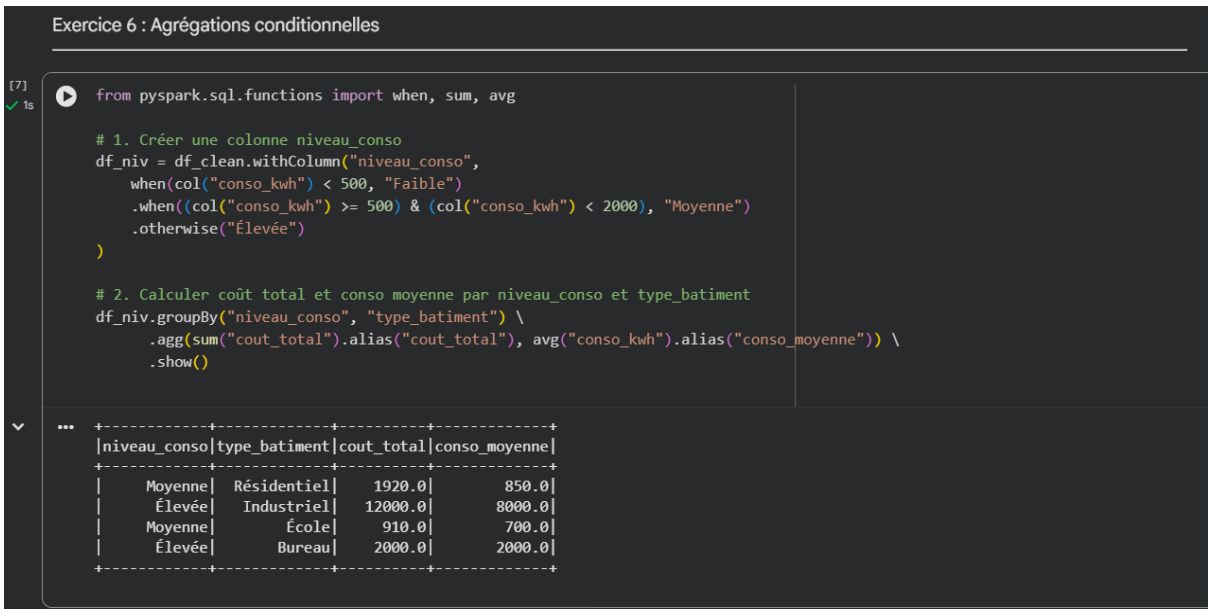
Une nouvelle colonne niveau_conso a été ajoutée pour classer les mesures selon leur niveau de consommation :

- Faible : conso_kwh < 500
- Moyenne : 500 ≤ conso_kwh < 2000
- Élevée : conso_kwh ≥ 2000

Ensuite, on a calculé :

- Le coût total (sum(cout_total))
- Et la consommation moyenne (avg(conso_kwh))
- par niveau de consommation et type de bâtiment.

Cette analyse montre les catégories de bâtiments où la consommation est la plus importante.



Conclusion

Ce TD a permis de manipuler un dataset énergétique en utilisant PySpark, de la phase de préparation jusqu'à l'analyse descriptive et temporelle.

Les étapes ont montré :

- Comment nettoyer et enrichir les données,
- Comment utiliser les fonctions d'agrégation et SQL dans Spark,
- Et comment obtenir des indicateurs utiles pour comprendre les tendances énergétiques.

Le travail réalisé montre la puissance de Spark pour le traitement et l'analyse de grandes quantités de données d'énergie.

Outils utilisés

- PySpark (SparkSession, DataFrame API, SQL)
- Python 3
- Google Colab