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 (SIR)

COMPTE RENDU DE

Mini Projet :

Gestion et analyse des données académiques avec MongoDB et PySpark

Réalisé par :

Hajar Rachid

Année universitaire 2025_2026

lien : Mini-Projet-Big-Data.ipynb

Introduction

Dans le contexte de la transformation numérique, la gestion et l'analyse des données massives occupent une place centrale, notamment dans le domaine éducatif. Les établissements d'enseignement génèrent quotidiennement un volume important de données académiques telles que les informations sur les étudiants, leurs filières et leurs notes.

Ce mini-projet s'inscrit dans le cadre du module Big Data & Bases de Données NoSQL et vise à illustrer l'utilisation d'une base de données NoSQL orientée documents (MongoDB via Mongita) ainsi que l'exploitation de l'outil PySpark pour l'analyse de données. Le projet permet de comprendre l'intérêt des technologies NoSQL et Big Data pour le traitement efficace des données académiques.

Objectifs du mini-projet

Les principaux objectifs de ce mini-projet sont :

- Comprendre le fonctionnement des bases de données NoSQL orientées documents.
- Apprendre à insérer, stocker et interroger des données JSON avec Mongita.
- Manipuler des requêtes d'exploration des données (filtrage, recherche, mise à jour, suppression).
- Exploiter PySpark pour réaliser des analyses analytiques simples sur un jeu de données.
- Utiliser SparkSQL pour interroger les données de manière déclarative.
- Produire un compte rendu structuré et professionnel du travail réalisé.

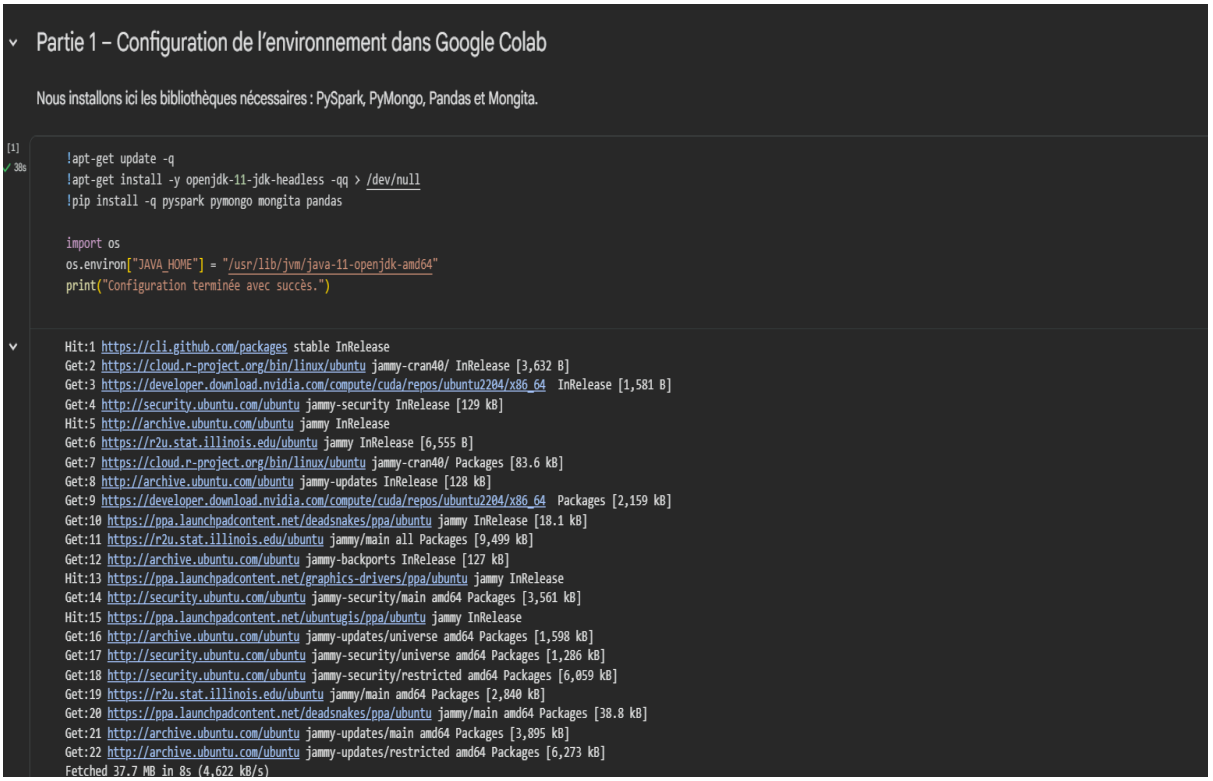
Étapes principales du mini-projet :

1. Configuration de l'environnement

L'environnement de travail a été configuré dans **Google Colab**. Les bibliothèques nécessaires ont été installées, notamment :

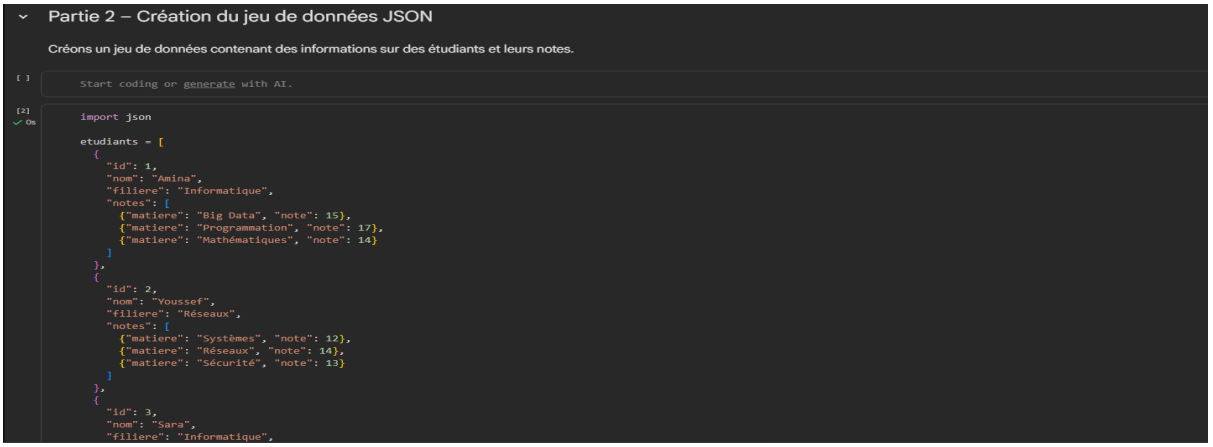
- PySpark pour le traitement Big Data,
- Mongita (compatible avec PyMongo) pour la base NoSQL locale,
- Pandas pour la manipulation et l'exportation des données.

La configuration de Java a également été vérifiée afin d'assurer le bon fonctionnement de PySpark.



2. Création du jeu de données JSON

Dans cette partie, un jeu de données au format JSON a été conçu afin de représenter des informations académiques relatives aux étudiants, telles que l'identifiant, le nom, la filière et les notes par matière. Ce format semi-structuré met en évidence la flexibilité des bases de données NoSQL orientées documents.



Partie 2 – Création du jeu de données JSON

Créons un jeu de données contenant des informations sur des étudiants et leurs notes.

```
Start coding or generate with AI.
```

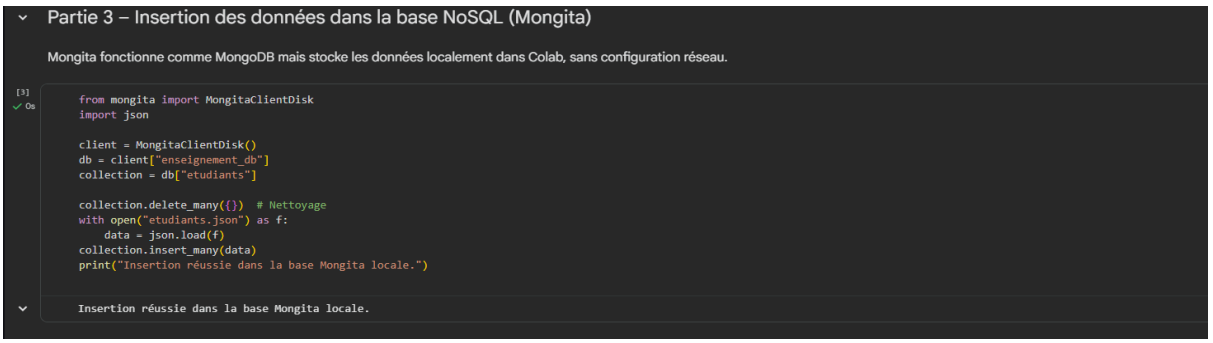
```
import json

etudiants = [
  {
    "id": 1,
    "nom": "Amina",
    "filiere": "Informatique",
    "notes": [
      {"matiere": "Big Data", "note": 15},
      {"matiere": "Programmation", "note": 17},
      {"matiere": "Mathématiques", "note": 14}
    ]
  },
  {
    "id": 2,
    "nom": "Youssef",
    "filiere": "Réseaux",
    "notes": [
      {"matiere": "Systèmes", "note": 12},
      {"matiere": "Réseaux", "note": 14},
      {"matiere": "Sécurité", "note": 13}
    ]
  },
  {
    "id": 3,
    "nom": "Sara",
    "filiere": "Informatique",
```

```
  {
    "id": 3,
    "nom": "Sara",
    "filiere": "Informatique",
    "notes": [
      {"matiere": "Big Data", "note": 18},
      {"matiere": "IA", "note": 17},
      {"matiere": "Programmation", "note": 16}
    ]
  }
]

with open("etudiants.json","w") as f:
    json.dump(etudiants,f,indent=2)

print("Fichier JSON créé avec succès.")
```

```
Fichier JSON créé avec succès.
```

3. Insertion des données dans la base NoSQL

L'objectif de cette partie est d'insérer le jeu de données JSON dans une base de données NoSQL locale en utilisant Mongita, une solution compatible avec MongoDB. Cette étape permet de comprendre le principe du stockage des documents dans une base orientée documents et de simuler le fonctionnement d'une base NoSQL sans configuration réseau complexe.



Partie 3 – Insertion des données dans la base NoSQL (Mongita)

Mongita fonctionne comme MongoDB mais stocke les données localement dans Colab, sans configuration réseau.

```
from mongita import MongitaClientDisk
import json

client = MongitaClientDisk()
db = client["enseignement_db"]
collection = db["etudiants"]

collection.delete_many({})  # Nettoyage
with open("etudiants.json") as f:
    data = json.load(f)
collection.insert_many(data)
print("Insertion réussie dans la base Mongita locale.")
```

```
Insertion réussie dans la base Mongita locale.
```

4. Requêtes d'exploration des données

Cette partie est consacrée à l'exploration et à la manipulation des données stockées dans la base NoSQL. Différentes requêtes ont été réalisées afin d'afficher, filtrer et rechercher les informations académiques. Ces opérations permettent de démontrer la souplesse des bases NoSQL dans l'accès et la manipulation des données.

Partie 4 – Requêtes d'exploration des données

```
# Afficher tous les étudiants
print("Liste complète des étudiants :")
for doc in collection.find():
    print(doc)

# Afficher seulement le nom et la filière (projection manuelle en Python)
print("\nNoms et filières des étudiants :")
for doc in collection.find():
    print({"nom": doc.get("nom"), "filiere": doc.get("filiere")})

# Rechercher les étudiants d'une filière donnée
print("\nÉtudiants en Informatique :")
for doc in collection.find({"filiere": "Informatique"}):
    print(doc)

# Afficher les étudiants ayant une note > 15 en Programmation
print("\nÉtudiants ayant une note > 15 en Programmation :")
for doc in collection.find():
    for note in doc.get("notes", []):
        if note["matiere"] == "Programmation" and note["note"] > 15:
            print(f"{doc['nom']} : {note['note']}")

# Compter le nombre total d'étudiants
print("\nNombre total d'étudiants :", collection.count_documents({}))
```

```
Liste complète des étudiants :
{'id': 1, 'nom': 'Amina', 'filiere': 'Informatique', 'notes': [{'matiere': 'Big Data', 'note': 15}, {'matiere': 'Programmation', 'note': 17}, {'matiere': 'Mathématiques', 'note': 14}], '_id': ObjectId('693033dee09b4b3f2ecbcc
{'id': 2, 'nom': 'Youssef', 'filiere': 'Réseaux', 'notes': [{'matiere': 'Systèmes', 'note': 12}, {'matiere': 'Réseaux', 'note': 14}, {'matiere': 'Sécurité', 'note': 13}], '_id': ObjectId('693033dee09b4b3f2ecbcc61')}
{'id': 3, 'nom': 'Sara', 'filiere': 'Informatique', 'notes': [{'matiere': 'Big Data', 'note': 18}, {'matiere': 'IA', 'note': 17}, {'matiere': 'Programmation', 'note': 16}], '_id': ObjectId('693033dee09b4b3f2ecbcc62')}

Noms et filières des étudiants :
{'nom': 'Amina', 'filiere': 'Informatique'}
{'nom': 'Youssef', 'filiere': 'Réseaux'}
{'nom': 'Sara', 'filiere': 'Informatique'}

Étudiants en Informatique :
{'id': 1, 'nom': 'Amina', 'filiere': 'Informatique', 'notes': [{'matiere': 'Big Data', 'note': 15}, {'matiere': 'Programmation', 'note': 17}, {'matiere': 'Mathématiques', 'note': 14}], '_id': ObjectId('693033dee09b4b3f2ecbcc
{'id': 3, 'nom': 'Sara', 'filiere': 'Informatique', 'notes': [{'matiere': 'Big Data', 'note': 18}, {'matiere': 'IA', 'note': 17}, {'matiere': 'Programmation', 'note': 16}], '_id': ObjectId('693033dee09b4b3f2ecbcc62')}

Étudiants ayant une note > 15 en Programmation :
Amina : 17
Sara : 16

Nombre total d'étudiants : 3
```

5 . Exportation des données vers PySpark

Dans cette partie, les données stockées dans la base NoSQL ont été exportées vers un fichier JSON afin de pouvoir être exploitées par PySpark. Cette étape assure la liaison entre la base de données NoSQL et l'outil Big Data, permettant ainsi de passer d'un stockage des données à une phase d'analyse.

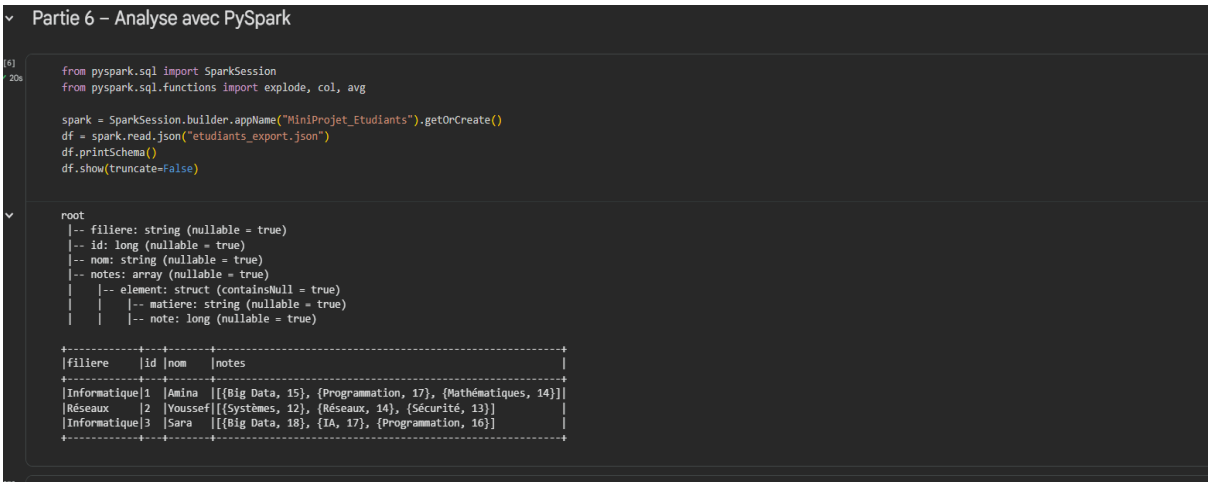
Partie 5 – Exportation du dataset vers PySpark

```
import pandas as pd

# Extraire tous les documents (sans projection)
docs = list(collection.find())

# Supprimer le champ _id qui pose problème (ajout automatique par Mongita)
for doc in docs:
    if "_id" in doc:
        del doc["_id"]

# Exporter vers JSON
pd.DataFrame(docs).to_json("etudiants_export.json", orient="records")
print("Export vers etudiants_export.json terminé.")
```

```
Export vers etudiants_export.json terminé.
```

6 . Analyse des données avec PySpark

L'objectif de cette partie est d'analyser les données académiques à l'aide de PySpark. Après le chargement du fichier JSON dans un DataFrame, plusieurs opérations analytiques ont été réalisées, telles que l'explosion des tableaux de notes, le calcul des moyennes par filière et l'évaluation des performances globales des étudiants.



7 .Manipulation, analyse et requêtes SQL

Cette partie constitue l'étape finale du mini-projet. Elle regroupe des opérations avancées de manipulation des données avec Mongita, des analyses détaillées avec PySpark ainsi que l'utilisation de SparkSQL pour interroger les données de manière déclarative. Cette partie permet de consolider l'ensemble des compétences acquises tout au long du projet et de démontrer l'intérêt combiné du NoSQL et du Big Data dans l'analyse des données académiques.

Partie 7 – Travail à faire

--- A.1 Ajouter 2 nouveaux étudiants et réinsérer ---

```
import json
from mongita import MongitaClientDisk

# connexion Mongita (même DB / collection que toi)
client = MongitaClientDisk()
db = client["enseignement_db"]
collection = db["etudiants"]

# Nouveaux étudiants à ajouter
nouveaux = [
  {
    "id": 4,
    "nom": "Khalid",
    "filiere": "Informatique",
    "notes": [
      {"matiere": "Big Data", "note": 14},
      {"matiere": "Programmation", "note": 15},
      {"matiere": "Mathématiques", "note": 13}
    ]
  },
  {
    "id": 5,
    "nom": "Nadia",
    "filiere": "Réseaux",
    "notes": [
      {"matiere": "Systèmes", "note": 16},
      {"matiere": "Réseaux", "note": 17},
      {"matiere": "Sécurité", "note": 15}
    ]
  }
]

# Charger le fichier existant et ajouter les nouveaux (ou reconstruire le fichier)
with open("etudiants.json", "r") as f:
    data = json.load(f)

# Ajouter les nouveaux (évite les doublons d'id)
ids_existants = {d["id"] for d in data}
for nd in nouveaux:
    if nd["id"] not in ids_existants:
        data.append(nd)

# Écrire le fichier mis à jour
with open("etudiants.json", "w") as f:
    json.dump(data, f, indent=2, ensure_ascii=False)

# Réinsertion dans Mongita : on nettoie et on insert tout
collection.delete_many({})
collection.insert_many(data)
print("Insertion des documents (avec 2 nouveaux) effectuée. Total inséré :", collection.count_documents({}))
```

```
Insertion des documents (avec 2 nouveaux) effectuée. Total inséré : 5
```

--- A.2 Afficher tous les étudiants insérés ---

```
print("Tous les étudiants :")
for doc in collection.find():
    print(doc)
```

```
Tous les étudiants :
{'id': 1, 'nom': 'Amina', 'filiere': 'Informatique', 'notes': [{'matiere': 'Big Data', 'note': 15}, {'matiere': 'Programmation', 'note': 17}, {'matiere': 'Mathématiques', 'note': 14}], '_id': ObjectId('693035a7e09b4b3f2ecb
{'id': 2, 'nom': 'Youssef', 'filiere': 'Réseaux', 'notes': [{'matiere': 'Systèmes', 'note': 12}, {'matiere': 'Réseaux', 'note': 14}, {'matiere': 'Sécurité', 'note': 13}], '_id': ObjectId('693035a7e09b4b3f2ecbcc64')}
{'id': 3, 'nom': 'Sara', 'filiere': 'Informatique', 'notes': [{'matiere': 'Big Data', 'note': 18}, {'matiere': 'IA', 'note': 17}, {'matiere': 'Programmation', 'note': 16}], '_id': ObjectId('693035a7e09b4b3f2ecbcc65')}
{'id': 4, 'nom': 'Khalid', 'filiere': 'Informatique', 'notes': [{'matiere': 'Big Data', 'note': 14}, {'matiere': 'Programmation', 'note': 15}, {'matiere': 'Mathématiques', 'note': 13}], '_id': ObjectId('693035a7e09b4b3f2ec
{'id': 5, 'nom': 'Nadia', 'filiere': 'Réseaux', 'notes': [{'matiere': 'Systèmes', 'note': 16}, {'matiere': 'Réseaux', 'note': 17}, {'matiere': 'Sécurité', 'note': 15}], '_id': ObjectId('693035a7e09b4b3f2ecbcc67')}
```

--- A.3 Afficher seulement noms et filières (filtrage en Python) ---

```
print("\nNoms et filières :")
for doc in collection.find():
    print({"nom": doc.get("nom"), "filiere": doc.get("filiere")})
```

```
Noms et filières :
{'nom': 'Amina', 'filiere': 'Informatique'}
{'nom': 'Youssef', 'filiere': 'Réseaux'}
{'nom': 'Sara', 'filiere': 'Informatique'}
{'nom': 'Khalid', 'filiere': 'Informatique'}
{'nom': 'Nadia', 'filiere': 'Réseaux'}
```

--- A.4 Rechercher étudiants de la filière "Réseaux" ---

```
print("\nÉtudiants en Réseaux :")
for doc in collection.find({"filiere": "Réseaux"}):
    print(doc)
```

```
Étudiants en Réseaux :
{'id': 2, 'nom': 'Youssef', 'filiere': 'Réseaux', 'notes': [{'matiere': 'Systèmes', 'note': 12}, {'matiere': 'Réseaux', 'note': 14}, {'matiere': 'Sécurité', 'note': 13}], '_id': ObjectId('693035a7e09b4b3f2ecbcc64')}
{'id': 5, 'nom': 'Nadia', 'filiere': 'Réseaux', 'notes': [{'matiere': 'Systèmes', 'note': 16}, {'matiere': 'Réseaux', 'note': 17}, {'matiere': 'Sécurité', 'note': 15}], '_id': ObjectId('693035a7e09b4b3f2ecbcc67')}
```

--- A.5 Étudiants dont la note en "Programmation" > 15 ---

```
print("\nÉtudiants avec note > 15 en Programmation :")
for doc in collection.find():
    for note in doc.get("notes", []):
        if note["matiere"] == "Programmation" and note["note"] > 15:
            print(f"{doc['nom']} : {note['note']}")
```

```
Étudiants avec note > 15 en Programmation :
Amina : 17
Sara : 16
```

--- A.6 Modifier la note de "Big Data" d'un étudiant (+1 point) et afficher ---

```
# Exemple : augmenter Big Data de Sara (id 3)
doc = collection.find_one({"id": 3})
if doc:
    # modifier dans la structure Python
    for note in doc["notes"]:
        if note["matiere"] == "Big Data":
            note["note"] = note["note"] + 1  # +1 point
    # Mettre à jour le document complet (remplacer)
    collection.replace_one({"id": doc["id"]}, doc)
    print(f"Nouvelle note Big Data pour {doc['nom']} :",
          next(n["note"] for n in doc["notes"] if n["matiere"]=="Big Data"))
else:
    print("Étudiant id 3 introuvable.")
```

--- A.7 Supprimer un étudiant de test ajouté par erreur ---

```
# Exemple : supprimer l'étudiant de test ayant id = 999 si existe
res = collection.delete_one({"id": 999})
print("Suppression (id 999) - documents supprimés :", res.deleted_count)
```

```
Suppression (id 999) - documents supprimés : 0
```

--- A.8 Compter le nombre total d'étudiants ---

```
total = collection.count_documents({})
print("Nombre total d'étudiants :", total)
```

```
Nombre total d'étudiants : 5
```

--- A.9 Compter le nombre d'étudiants par filière (dictionnaire Python) ---

```
from collections import defaultdict
compte = defaultdict(int)
for doc in collection.find():
    compte[doc["filiere"]] += 1

print("Nombre d'étudiants par filière :", dict(compte))
```

```
Nombre d'étudiants par filière : {'Informatique': 3, 'Réseaux': 2}
```

--- A.10 Trouver l'étudiant ayant la meilleure moyenne générale (calcul en Python) --

-

```
def moyenne(doc):
    notes = [n["note"] for n in doc.get("notes", [])]
    return sum(notes)/len(notes) if notes else 0

meilleure = None
best_avg = -1
for doc in collection.find():
    avg = moyenne(doc)
    if avg > best_avg:
        best_avg = avg
        meilleure = doc

if meilleure:
    print("Étudiant avec la meilleure moyenne :", meilleure["nom"], "->", round(best_avg, 2))
else:
    print("Aucun étudiant trouvé.")
```

```
Étudiant avec la meilleure moyenne : Sara -> 17.33
```

B. Analyse et transformation (PySpark)

```
# Réexporter vers etudiants_export.json (supprimer _id si présent)
docs = list(collection.find())
for d in docs:
    if "_id" in d:
        del d["_id"]

import pandas as pd
pd.DataFrame(docs).to_json("etudiants_export.json", orient="records", force_ascii=False)
print("Export vers etudiants_export.json terminé. Documents exportés :", len(docs))
```

```
Export vers etudiants_export.json terminé. Documents exportés : 5
```

--- B.1 Charger et afficher schéma + types ---

[20] ✓ 1s

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import explode, col, avg, round as spark_round

spark = SparkSession.builder.appName("MiniProjet_Etudiants_Partie7").getOrCreate()
df = spark.read.json("etudiants_export.json")

print("Schéma du DataFrame :")
df.printSchema()
df.show(truncate=False)
```

```
Schéma du DataFrame :
root
 |-- filiere: string (nullable = true)
 |-- id: long (nullable = true)
 |-- nom: string (nullable = true)
 |-- notes: array (nullable = true)
 |    |-- element: struct (containsNull = true)
 |    |    |-- matiere: string (nullable = true)
 |    |    |-- note: long (nullable = true)

+------------+---+-------+-----------------------------------------------------------+
|filiere     |id |nom    |notes                                                      |
+------------+---+-------+-----------------------------------------------------------+
|Informatique|1  |Amina  |[{Big Data, 15}, {Programmation, 17}, {Mathématiques, 14}]|
|Réseaux     |2  |Youssef|[{Systèmes, 12}, {Réseaux, 14}, {Sécurité, 13}]           |
|Informatique|3  |Sara   |[{Big Data, 19}, {IA, 17}, {Programmation, 16}]            |
|Informatique|4  |Khalid |[{Big Data, 14}, {Programmation, 15}, {Mathématiques, 13}]|
|Réseaux     |5  |Nadia  |[{Systèmes, 16}, {Réseaux, 17}, {Sécurité, 15}]           |
+------------+---+-------+-----------------------------------------------------------+
```

--- B.2 / B.3 Créer df_explode correctement (matiere, note) ---

[22] ✓ 1s

```
# --- B.2 / B.3 Créer df_explode correctement (matiere, note) ---
from pyspark.sql.functions import explode

# explode renvoie une struct; on l'extrait ensuite en colonnes matiere/note
df_explode_struct = df.select("filiere", explode(col("notes")).alias("note_struct"))
df_explode = df_explode_struct.select(
    "filiere",
    col("note_struct.matiere").alias("matiere"),
    col("note_struct.note").alias("note")
)

print("Aperçu df_explode :")
df_explode.show(truncate=False)
```

```
Aperçu df_explode :
+------------+-------------+----+
|filiere     |matiere      |note|
+------------+-------------+----+
|Informatique|Big Data     |15  |
|Informatique|Programmation|17  |
|Informatique|Mathématiques|14  |
|Réseaux     |Systèmes     |12  |
|Réseaux     |Réseaux      |14  |
|Réseaux     |Sécurité     |13  |
|Informatique|Big Data     |19  |
|Informatique|IA           |17  |
|Informatique|Programmation|16  |
|Informatique|Big Data     |14  |
|Informatique|Programmation|15  |
|Informatique|Mathématiques|13  |
|Réseaux     |Systèmes     |16  |
|Réseaux     |Réseaux      |17  |
|Réseaux     |Sécurité     |15  |
+------------+-------------+----+
```

--- B.4 Calculer la moyenne des notes par filière ---

[23] ✓ 2s

```
moy_par_filiere = df_explode.groupBy("filiere").agg(spark_round(avg("note"), 2).alias("moyenne_filiere"))
moy_par_filiere.show()
```

```
+------------+---------------+
|     filiere|moyenne_filiere|
+------------+---------------+
|Informatique|          15.56|
|     Réseaux|           14.5|
+------------+---------------+
```

--- B.5 Calculer la moyenne des notes par matière ---

[24] ✓ 1s

```
moy_par_matiere = df_explode.groupBy("matiere").agg(spark_round(avg("note"),2).alias("moyenne_matiere"))
moy_par_matiere.orderBy(col("moyenne_matiere").desc()).show(truncate=False)
```

```
+-------------+---------------+
|matiere      |moyenne_matiere|
+-------------+---------------+
|IA           |17.0           |
|Big Data     |16.0           |
|Programmation|16.0           |
|Réseaux      |15.5           |
|Systèmes     |14.0           |
|Sécurité     |14.0           |
|Mathématiques|13.5           |
+-------------+---------------+
```

--- B.6 Identifier la filière ayant la meilleure moyenne ---

```
[25] ✓ 0s
best_filiere = moy_par_filiere.orderBy(col("moyenne_filiere").desc()).limit(1)
print("Filière avec la meilleure moyenne :")
best_filiere.show()
```

```
Filière avec la meilleure moyenne :
+-----------+---------------+
|     filiere|moyenne_filiere|
+-----------+---------------+
|Informatique|          15.56|
+-----------+---------------+
```

--- B.7 Identifier la matière la mieux réussie ---

```
[26] ✓ 1s
best_matiere = moy_par_matiere.orderBy(col("moyenne_matiere").desc()).limit(1)
print("Matière la mieux réussie :")
best_matiere.show()
```

```
Matière la mieux réussie :
+-------+---------------+
|matiere|moyenne_matiere|
+-------+---------------+
|     IA|           17.0|
+-------+---------------+
```

--- B.8 Moyenne générale (toutes filières) ---

```
[27] ✓ 0s
moyenne_generale = df_explode.agg(spark_round(avg("note"),2).alias("moyenne_generale"))
moyenne_generale.show()
```

```
+---------------+
|moyenne_generale|
+---------------+
|          15.13|
+---------------+
```

--- B.9 Ajouter une colonne "mention" selon la moyenne individuelle ---

```
[30] ✓ 3s
# ===== B.9 Ajouter une colonne "mention" (CODE UNIQUE PRÊT À COLLER) =====

from pyspark.sql.functions import explode, col, avg as spark_avg, round as spark_round, when

# 1) Explode
df_explode = df.select(
    "id",
    "nom",
    "filiere",
    explode(col("notes")).alias("note_struct")
).select(
    "id",
    "nom",
    "filiere",
    col("note_struct.matiere").alias("matiere"),
)

# 2) Calcul de la moyenne par étudiant
moy_par_etudiant = df_explode.groupBy(
    "id", "nom"
).agg(
    spark_round(spark_avg("note"), 2).alias("moyenne")
)

# 3) Ajout de la colonne "mention"
resultat_final = moy_par_etudiant.withColumn(
    "mention",
    when(col("moyenne") > 16, "Excellent")
    .when((col("moyenne") >= 14) & (col("moyenne") <= 16), "Bien")
    .when((col("moyenne") >= 12) & (col("moyenne") < 14), "Assez-bien")
    .otherwise("Passable")
)

# 4) Affichage final
resultat_final.orderBy(col("moyenne").desc()).show(truncate=False)
```

```
+---+-------+-------+----------+
|id |nom    |moyenne|mention   |
+---+-------+-------+----------+
|3  |Sara   |17.33  |Excellent |
|5  |Nadia  |16.0   |Bien      |
|1  |Amina  |15.33  |Bien      |
|4  |Khalid |14.0   |Bien      |
|2  |Youssef|13.0   |Assez-bien|
+---+-------+-------+----------+
```

C. Requêtes SQL avec SparkSQL

--- C.1 Créer table temporaire ---

[31] ✓ 0s

```
# Utiliser df (ou df_explode selon la requête)
df.createOrReplaceTempView("etudiants_sql")
df_explode.createOrReplaceTempView("etudiants_notes_sql")
```

--- C.21 Requête SQL : afficher les étudiants de la filière "Informatique" ---

[32] ✓ 0s

```
q1 = spark.sql("""
SELECT id, nom, filiere
FROM etudiants_sql
WHERE filiere = 'Informatique'
ORDER BY id
""")
q1.show()
```

```
+---+------+------------+
| id|   nom|     filiere|
+---+------+------------+
|  1| Amina|Informatique|
|  3|  Sara|Informatique|
|  4|Khalid|Informatique|
+---+------+------------+
```

--- C.22 Requête SQL : moyenne des notes par matière ---

[33] ✓ 1s

```
q2 = spark.sql("""
SELECT matiere, ROUND(AVG(note),2) as moyenne_matiere
FROM etudiants_notes_sql
GROUP BY matiere
ORDER BY moyenne_matiere DESC
""")
q2.show(truncate=False)
```

```
+-------------+---------------+
|matiere      |moyenne_matiere|
+-------------+---------------+
|IA           |17.0           |
|Big Data     |16.0           |
|Programmation|16.0           |
|Réseaux      |15.5           |
|Systèmes     |14.0           |
|Sécurité     |14.0           |
|Mathématiques|13.5           |
+-------------+---------------+
```

--- C.23 Requête SQL : filière ayant la meilleure moyenne ---

[34] ✓ 0s

```
q3 = spark.sql("""
SELECT filiere, ROUND(AVG(note),2) as moyenne_filiere
FROM etudiants_notes_sql
GROUP BY filiere
ORDER BY moyenne_filiere DESC
LIMIT 1
""")
q3.show()
```

```
+------------+---------------+
|     filiere|moyenne_filiere|
+------------+---------------+
|Informatique|          15.56|
+------------+---------------+
```

--- C.24 Requête SQL : compter le nombre total d'étudiants dans chaque filière ---

[35] ✓ 1s

```
q4 = spark.sql("""
SELECT filiere, COUNT(*) as nombre_etudiants
FROM etudiants_sql
GROUP BY filiere
ORDER BY nombre_etudiants DESC
""")
q4.show()
```

```
+------------+---------------+
|     filiere|nombre_etudiants|
+------------+---------------+
|Informatique|              3|
|     Réseaux|              2|
+------------+---------------+
```

Conclusion

Ce mini-projet a permis de mettre en pratique les notions fondamentales des bases de données NoSQL et du Big Data dans un contexte concret lié au domaine éducatif. L'utilisation de MongoDB (via Mongita) a montré la flexibilité et la simplicité des bases orientées documents pour le stockage des données semi-structurées.

Par ailleurs, l'exploitation de PySpark et SparkSQL a mis en évidence la puissance des outils Big Data pour effectuer des analyses rapides et efficaces sur des volumes importants de données. Ce projet a également renforcé notre compréhension des opérations de transformation, d'agrégation et d'analyse des données.

conclusion, les technologies NoSQL et Big Data représentent aujourd'hui des solutions indispensables pour la gestion et l'analyse des données académiques à grande échelle, offrant performance, flexibilité et scalabilité.