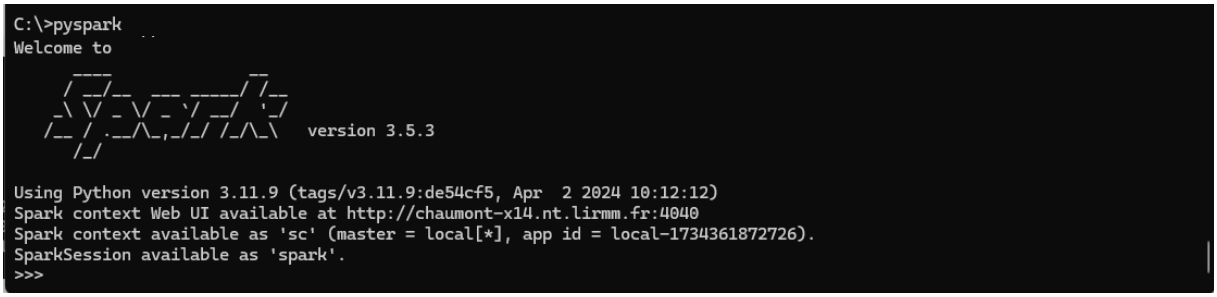
TD3/TP3 Introduction à Spark via l'interpréteur de commandes PySpark :

Big Data : enjeux, stockage et extraction
B.U.T. Science des données - S5 – 2025
Marc Chaumont Univ. Nîmes
utilisation de l'interpréteur de app.datacamp.com

L'interpréteur de commandes `pyspark` permet de communiquer directement avec un *cluster* Spark local. En pratique cela signifie que vous ouvrez une invite de commandes (sous Windows ou un terminal sous Linux) et que vous allez saisir du code Python permettant d'interagir avec un serveur de cluster Spark.

Note : Ci-dessous une image de l'interpréteur lancé (décembre 2024) sur ma machine :



> *Note : vous pourriez installer `pyspark` sur votre machine (en tapant la commande `pip install pyspark`) mais nous allons pour le moment passer par un émulateur en ligne.*

Ce TP vous montre comment lancer des calculs sur des tables. Il n'est plus nécessaire de programmer à la manière du modèle Map Reduce car nous utilisons un environnement qui intègre ce modèle. Bref, tous ce que nous avons vu sur Map Reduce reste vrai mais tout est transparent (invisible) du point de vue utilisateur (vous). Cela signifie que vous interagissez avec un master (maitre) du cluster et qu'il gère lui-même comment les fragments de vos tables ou vos fichiers sont dispatchés au sein des nœuds, mais que certaines fonctionnalités comme des calcul de comptage (comme par exemple WordCount) sont déjà implémentés…

Partie 1 : DataFrame, et Tables (Spark SQL) ; Mise en bouche…

Vous allez pour ce TP interagir avec Spark via l'interpréteur PySpark du cours en ligne proposé par https://app.datacamp.com/

L'objectif est de découvrir comment programmer une interaction avec Spark et comment faire des calculs sur des tables. Dans ce cours en ligne, il y a 10 sections :

Le cours se nomme « Introduction à PySpark ; Chapitre 1. Apprendre à connaître PySpark » et est proposé par Lore Dirick - Director of Data Science Education at Flatiron School, Nick Solomon - Data Scientist et en collaboration avec Colin Ricardo.

Allez à l'adresse suivante et faites les 10 sections :
https://app.datacamp.com/learn/courses/introduction-to-pyspark

> ***TRAVAIL A RENDRE :*** *Pour cette partie, vous rendrez un compte rendu qui contiendra les copiers-collers de chacune des sections, le(s) question(s) et le(s) réponse(s). Ce compte rendu vous servira également de support de cours. Pour rappel, tous ce que nous voyons en cours/td est à connaitre.*

Partie 2 : Maintenant que vous avez l'interpréteur PySpark en main, déroulez le tutorial fourni par Apache :

Allez sur les pages du tutoral fourni par Apache :

1. Quickstart: DataFrame
   https://spark.apache.org/docs/latest/api/python/getting_started/quickstart_df.html#,
2. Quickstart: Spark Connect
   https://spark.apache.org/docs/latest/api/python/getting_started/quickstart_connect.html
3. Quickstart: Pandas API on Spark
   https://spark.apache.org/docs/latest/api/python/getting_started/quickstart_ps.html
4. Testing PySpark
   https://spark.apache.org/docs/latest/api/python/getting_started/testing_pyspark.html

et **testez les méthodes sur l'interpréteur de app.datacamp.com** … dans la mesure du possible.

Tout ne peut pas être testé via l'interpréteur de app.datacamp. Par exemple, les tests de la section « Applying a Function » ne fonctionnent pas car la librairie PyArrow n'est pas à jour.

A noter que les tests peuvent également être réalisé sur les "live notebooks" proposés par Apache :

- Live Notebook: DataFrame
  https://mybinder.org/v2/gh/apache/spark/a6f220d9517?filepath=python%2Fdocs%2Fsource%2Fgetting_started%2Fquickstart_df.ipynb

- Live Notebook: Spark Connect
  https://mybinder.org/v2/gh/apache/spark/a6f220d9517?filepath=python%2Fdocs%2Fsource%2Fgetting_started%2Fquickstart_connect.ipynb

- Live Notebook: pandas API on Spark
  https://mybinder.org/v2/gh/apache/spark/a6f220d9517?filepath=python%2Fdocs%2Fsource%2Fgetting_started%2Fquickstart_ps.ipynb

Dans ce tutorial vous (re-)verrez :

- DataFrame Creation
- Viewing Data
- Selecting and Accessing Data
- Applying a Function
- Grouping Data
- Getting Data In/Out
- Working with SQL

- Launch Spark server with Spark Connect
- Connect to Spark Connect server
- Create DataFrame

- Object Creation
- Missing Data
- Operations
- Grouping
- Plotting
- Getting data in/out

- Build a PySpark Application
- Testing your PySpark Application
- Putting It All Together!

TRAVAIL A RENDRE : *Vous rendrez « un résumé / compaction des notions » associées à vos tests sous la forme d'un rapport en pdf qui contiendra :*

1. *Le nom/objectif du test (pour décrire ce qui est testé ainsi que la notion) :*

2. *Dans un encadré d'une couleur : le code testé*
3. *Dans un autre encadré d'une autre couleur : le résultat de l'exécution*

Partie 3 : Et s'il reste du temps …

Cette partie est issue du TP Introduction à Spark du CNAM RCP216

https://cedric.cnam.fr/vertigo/Cours/RCP216/tpSparkPython.html

Tout d'abord vous aller installer `pyspark` sur votre machine. Taper ceci en ligne de commande :

```
>> pip install pyspark
```

Note : si cela ne marche pas … il va falloir chercher sur Internet. Pas de panique, cette partie du TD/TP est optionnelle (vous pouvez tout de même lire … la suite…)

Lancement de l'interpréteur de commandes en Python et opérations simples

Nous allons travailler avec un fichier texte contenant la première partie du livre *Les Malheurs de Sophie* de la comtesse de Ségur, téléchargeable ici. Commençons par ouvrir un terminal (Windows ou MacOS/Linux).

Dataset et *DataFrame*

Un *Dataset* est une collection distribuée de données. Il peut être vu comme une évolution conceptuelle des RDD (*Resilient Distributed DataFrames*), historiquement la première structure de données distribuée employée par Spark. Un *DataFrame* est un *Dataset* organisé en colonnes qui portent des noms, comme les tables d'une base de données. Avec l'interface de programmation en python, le type `DataFrame` est simplement l'alias du type `Dataset[Row]`.

Il peut être utile de lire ces explications préparées par les créateurs de Spark pour mieux comprendre l'intérêt de chacune des interfaces de programmation (API) *RDD* et *Dataset / DataFrame*.

Le lancement de `pyspark` crée implicitement l'instance `spark` de `SparkSession` qui comporte un certain nombre d'informations de configuration (par ex., indique à Spark

comment accéder à un *cluster* pour exécuter les commandes) et qui peut être employée directement.

Créez maintenant un *DataFrame* à partir du fichier texte `sophie.txt` :

```
texteSophie = spark.read.text("sophie.txt")
```

Dans la configuration actuelle de Spark dans la salle de travaux pratiques, c'est le système de fichiers (*filesystem*) **local** et non HDFS (distribué) qui est utilisé par défaut. Si vous entrez simplement `.text("tpintro/sophie.txt")`, Spark cherchera le fichier dans le système de fichiers par défaut, en suivant le chemin spécifié. Avec le préfixe `file://` on indique que le fichier à lire est sur le système de fichiers local, avec le préfixe `hdfs://` on indique qu'il s'agit plutôt de HDFS. L'utilisation d'un de ces préfixes exige l'emploi du chemin absolu vers le fichier correspondant.

Il est possible d'appliquer aux *DataFrames* des **actions**, qui retournent des valeurs, et des **transformations**, qui modifient les *DataFrames* ou en produisent de nouveaux.

Quelques **actions** simples à tester :

```
texteSophie.count()
  # retourne le nombre de lignes dans ce DataFrame

texteSophie.first()
  # retourne la première ligne de ce DataFrame

texteSophie.take(5)
  # retourne les 5 premières lignes de ce DataFrame

texteSophie.collect()
  # retourne sur le *driver* le contenu complet de ce
  #    DataFrame et en affiche une partie à la console
```

Avec `.collect()`, lorsque le DataFrame en question est distribué, toutes ses données situées sur les nœuds de calcul sont d'abord regroupées sur le nœud *driver*. Cette opération **très** coûteuse si le DataFrame est grand, et qui peut engendrer des débordements de la mémoire sur le *driver*, donc **à éviter** autant que possible. Ensuite, une petite partie des données est affichée à la console.

L'exécution d'une **action** comme count sur un très grand *DataFrame* se déroule de la manière suivante : le *DataFrame* est composé de fragments, chacun stocké sur un nœud de calcul ; chaque fragment contient un (grand) nombre de lignes ; le programme *driver* de Spark envoie à chaque nœud le traitement à faire ; chaque nœud de calcul exécute le traitement sur le fragment local et transmet les résultats au *driver*. Pour d'autres actions (comme first) seul un fragment est nécessaire donc un seul nœud est sollicité.

Quelques **transformations** simples :

- Construire un *DataFrame* `lignesAvecPoupee` comprenant seulement les lignes de `texteSophie` qui contiennent « poupée », puis afficher les 2 premières lignes :

```
lignesAvecPoupee    =    texteSophie.filter(texteSophie.value.contains("poupée"))
lignesAvecPoupee.take(2) # affiche les 2 premières lignes de ce DataFrame
```

- Construire un *DataFrame* `longueursLignes` composé des longueurs des lignes de `texteSophie`, retourner un tableau avec ses 5 premières lignes :

```
from pyspark.sql.functions import *  # pour utiliser fonctions comme length()

longueursLignes = texteSophie.select(length(texteSophie.value))

longueursLignes.show(5) # affiche les 5 premières lignes de ce DataFrame
```

Dans ces exemples, l'expression `texteSophie.value` retourne le contenu de l'unique colonne de `texteSophie`, appelée `value`.

L'exécution d'une **transformation** sur un très grand *DataFrame* se déroule de la manière suivante : le *DataFrame* est composé de fragments, chacun stocké sur un nœud de calcul ; le programme *driver* de Spark envoie à chaque nœud le traitement à faire ; chaque nœud de calcul exécute le traitement sur le fragment local et **conserve les résultats localement**.

Enchaînements et évaluation « paresseuse »

Transformations et actions peuvent s'enchaîner :

Combien de lignes contiennent `poupée` ? Écrire la commande nécessaire dans `pyspark` et vérifier dans un terminal (`bash`) Linux.

```
# .filter() permet de ne garder que les lignes vérifiant la condition
désirée
# .count() permet de compter le nombre de lignes

texteSophie.filter(texteSophie.value.contains("poupée")).count()
```

Combien de caractères contient le fichier ?

```
texteSophie.select(sum(length(texteSophie.value))).show()
```

Important : l'évaluation est « paresseuse » : les opérations sont effectuées seulement quand un résultat doit être retourné.