Prérequis pour les utilisateurs de Windows uniquement

Si vous utilisez une distribution Linux ou Mac OS passez directement à la section LAB

Afin de pouvoir utiliser correctement `Spark` avec le système de fichier Windows il est nécessaire de disposer du binaire `winutils.exe`. Ce binaire permet d'écrire des données depuis `Spark` le système de fichier Windows.

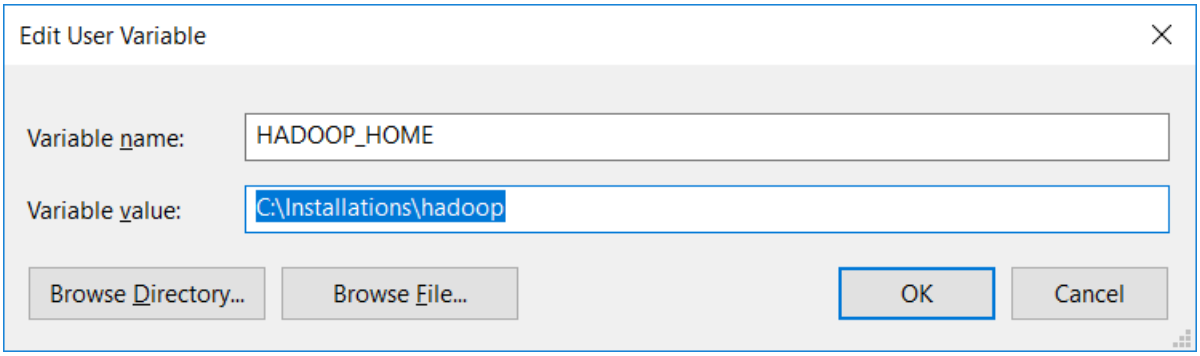
`Spark` attend `winutils.exe` dans l'installation `Hadoop "<Répertoire d'installation Hadoop>/bin/winutils.exe" (notez le dossier "bin")`

Création de HADOOP_HOME

- Créer un un répertoire pour les binaire hadoop: e.g: `C:\Program Files\hadoop\bin\`
- Télécharger le fichier `winutils.exe depuis le github :` https://github.com/yacineab/ESGI-spark-core/blob/master/hadoop/bin/winutils.exe
- Placer le fichier `winutils.exe` dans le répertoire `..\hadoop\bin\`

Définir la variable d'environnement *HADOOP_HOME*

- Sous Windows 10 et Windows 8 accédez à `Panneau de configuration> Système> Paramètres système avancés.`
- Windows 7, cliquez avec le bouton droit sur Poste de travail et sélectionnez `Propriétés> Avancé.`
- Cliquez sur le bouton Variables d'environnement.
- Sous Variables système, cliquez sur Nouveau.
- Dans le champ Nom de variable, entrez : `HADOOP_HOME`
- Dans le champ Valeur, entrez votre chemin du répertoire `hadoop`. **e.g:** `C:\Program Files\hadoop`
- Cliquez sur OK et appliquez les modifications



LAB

Les données utilisées dans ce TP se trouve sur le github dans le répertoire data

Lab 1 : Flight-Data

Nous allons travailler sur les données des vols entre différents pays pour l'année 2015. Ce jeu de données comptabilise le nombre de vols effectué entre deux destinations au cours de l'année 2015. Le dataset se trouve dans ce lien : 2015-summary.csv

1. Télécharger le fichier `csv` et le placer dans un répertoire `data`
2. Créer un nouveau projet `sbt` depuis votre `IDE` avec la **version `scala 2.12.x`**
3. Ajouter les dépendances `spark-core` et `spark-sql` dans fichier `build.sbt`
4. Créer une nouvelle class de type object `nommée FligthData` qui va contenir votre code Spark
5. Créer une variable `spark` et Instancier le `SparkSession`

Le point d'entrée dans l'API Spark DataFrame est le SparkSession, qui est instancier ici dans la variable appelée spark. L'appel de la fonction pour créer le DataFrame se fait donc à partir de cette variable.

6. Créer le DataFrame `flightData2015` à partir des données du fichier précèdent. Pour cela :
   a. Créer une variable immuable qui s'appelle `flightData2015`
   b. Appeler les fonctions pour lire un fichier csv : `spark.read.option("inferSchema", "true").option("header","true").csv("/chemin/vers/le/fichier.csv")`
7. Utiliser la fonction `printSchema()` pour afficher le schéma de données de votre DataFrame
8. Afficher les 20 premières lignes de votre DataFrame

Quelques transformations

1. Créer un nuveaux DataFrame où le pays de destination est égal au pays d'origine. Utiliser la fonction `where(…)`
2. Faire un `show()` sur ce DataFrame
3. Faire `explain` sur ce DataFrame. Que compreniez-vous de retour de la fonction `explain` ?

Sorting

1. Trier (fonction sort) le dataframe par le nombre de vol par destination i.e. la colonne `count`. Pour Cela utiliser la fonction `sort("Nom de la clone")`
2. Afficher les 20 premières lignes de ce DataFrame
3. Appeler la fonction `explain` sur ce dernier DataFrame. Que constateriez-vous ?

DataFrame à partir d'un fichier JSON

1. Télécharger le fichier `JSON` et le placer dans un répertoire `data:` `2010-summary.json`
2. Créer un DataFrame `flightData2010Json` à partir des données du fichier `JSON`
3. Afficher les premières lignes de votre DataFrame

DataFrame à partir d'un fichier PARQUET

1. Télécharger le fichier `parquet` et le placer dans un répertoire `data:` `flights.parquet`
2. Créer un DataFrame `flightsParquet` à partir du fichier précédent
3. Afficher les premières lignes de votre DataFrame

Lab 2 : DataFrames Schema et colonnes

Chaque DataFrame possède un schéma.

1. Afficher la liste des colonnes du dataframe flightData2014 en appelant la fonction : `.columns`

Un DataFrame peut être soit créer à partir des données en entrée soit à partir d'une séquence de données.

Suivez les étapes ci-dessous pour créer votre data frame à partir de :

2. Une séquence de rows
3. Une Séquence

```
1. val myRows = Seq(Row("Hello", null, 1L))


2. Seq(("Hello", null, 1L))
```

Select

Appeler la fonction `select` sur le dataframe pour selectionner la colonne `DEST_COUNTRY_NAME`.Utiliser les différentes manière d'appeler la colonne pour faire des select:

```
flightData2014.col("DEST_COUNTRY_NAME"),
col("DEST_COUNTRY_NAME"),
column("DEST_COUNTRY_NAME"),
$"DEST_COUNTRY_NAME",
'DEST_COUNTRY_NAME,
expr("DEST_COUNTRY_NAME")
```

SelectExpr

1. Créer un nouveau Dataframe withinCountry en utilisant SelectExpr pour ajouter une colonne au DataFrame qui a pour valeur: true si le vol est dans le meme pays et false sinon
2. Créer un nouveau dataframe filtredWithinCountry en filtrant par les vols interne (pays origin = pays destination)
3. Faire un show()
4. Créer un nouveau dataframe en calculant la moyenne de nombre de vol et faire un count sur les distinct "DEST_COUNTRY_NAME

Add a column

1. Créer un nouveau dataframe en ajoutant une colonne au DataFrame pour les vol locaux

Rename column

1. rename DEST_COUNTRY_NAME par dest et ORIGIN_COUNTRY_NAME par origin en utilsant .withColumnRenamed("colToRename", "newColName")
2. appeler la fonction .columns sur le nouveau dataframe

TP1 : Spark Core – Manipulation DataFrame

Removing column

1. remove the column ORIGIN_COUNTRY_NAME en utilsant .drop
2. Utiliser la fonction .columns sur ce dataframe pour afficher la liste des colonnes

Filtring rows

1. En utilisant .filter Filtrer le dataframe sur les vols où le count est < 15
2. En utilisant le where Filtrer le dataframe sur les vols où le count est > 50

Sorting rows

3. import org.apache.spark.sql.functions.{desc,asc}
4. faire un orderby desc sur le "count"

Some aggregation

5. Calculer le nombre d'élement dans le dataframe en utilisant le ".count"
6. importer org.apache.spark.sql.functions.countDistinct org.apache.spark.sql.functions.{desc,asc} et org.apache.spark.sql.functions.sumet org.apache.spark.sql.functions.countDistinct
7. compter le nombre de distinct row du dataframe
8. calculer la somme des count et faite un collectpour collecter le résultat dans le driver