

RÉPUBLIQUE ALGÉRIENNE DÉMOCRATIQUE ET POPULAIRE

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

École nationale Supérieure d'Informatique (ESI ex. INI)

2^{ème} année cycle supérieur SIT (2CS - SIT1)

Spark

Réalisé par :

- ADLA ilyes chiheb eddine

Année 2019/2020

Table des matières

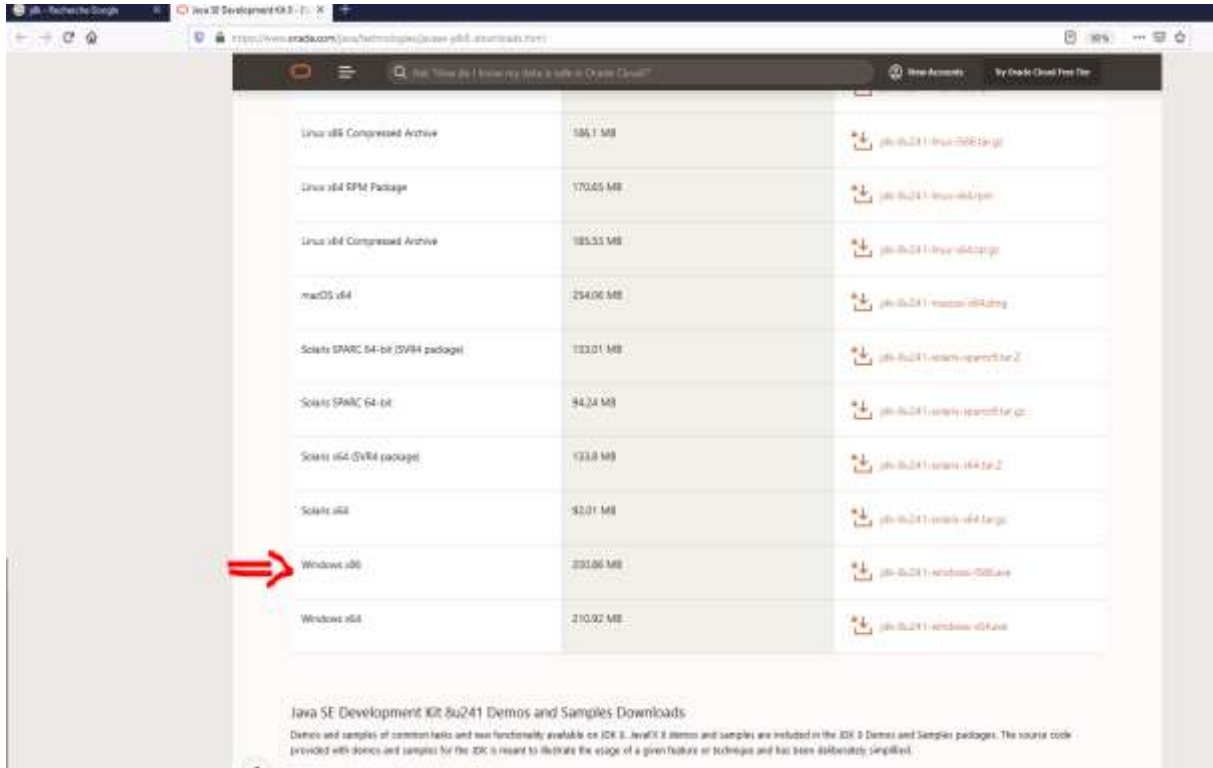
• Installation locale	3
Installation java	3
Installation du Hadoop	6
Installation du Eclipse	8
Installation du Maven	9
Installation du l'IDE Scala	10
Lancement du Spark	13
• Travail à faire	16
• Le choix de jeux de données et l'algorithme de résolution	17
• Codage de l'algorithme	17
Construction d'un modèle K-Means	26
Le test du modèle avec l'ensemble de données	27
• Parallélisation sur un cluster	31



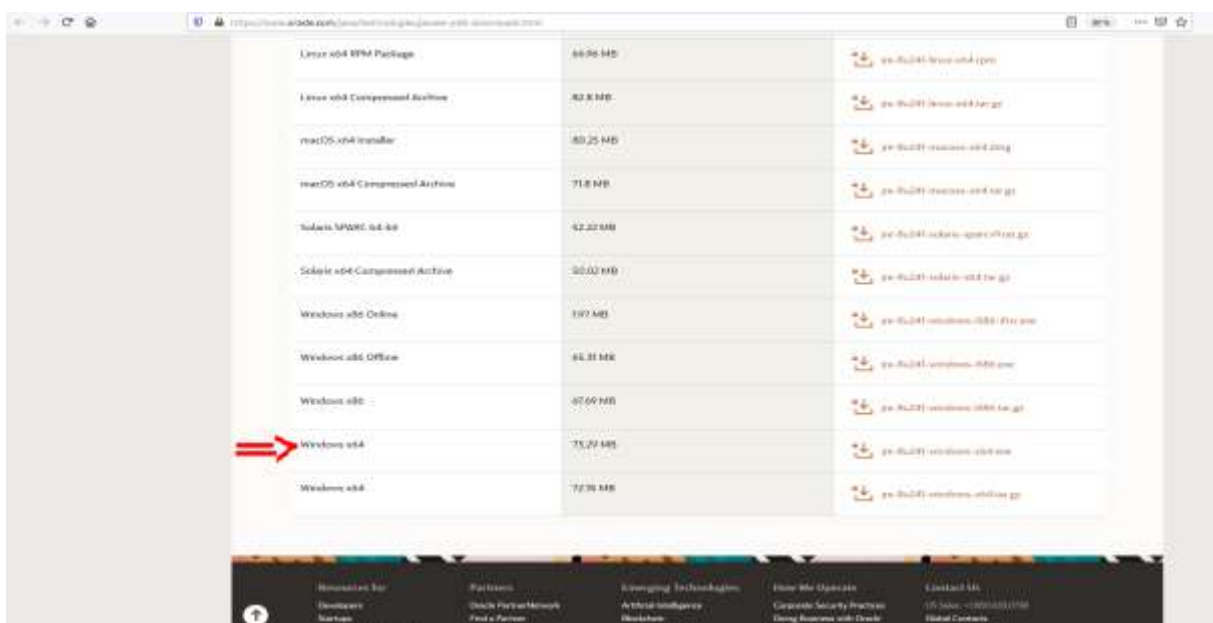
Installation locale

Installation java

Installation du JDK : tout d'abord afin d'accomplir cette tâche, on doit télécharger le JDK depuis le site d'oracle et ensuite procéder à son installation.

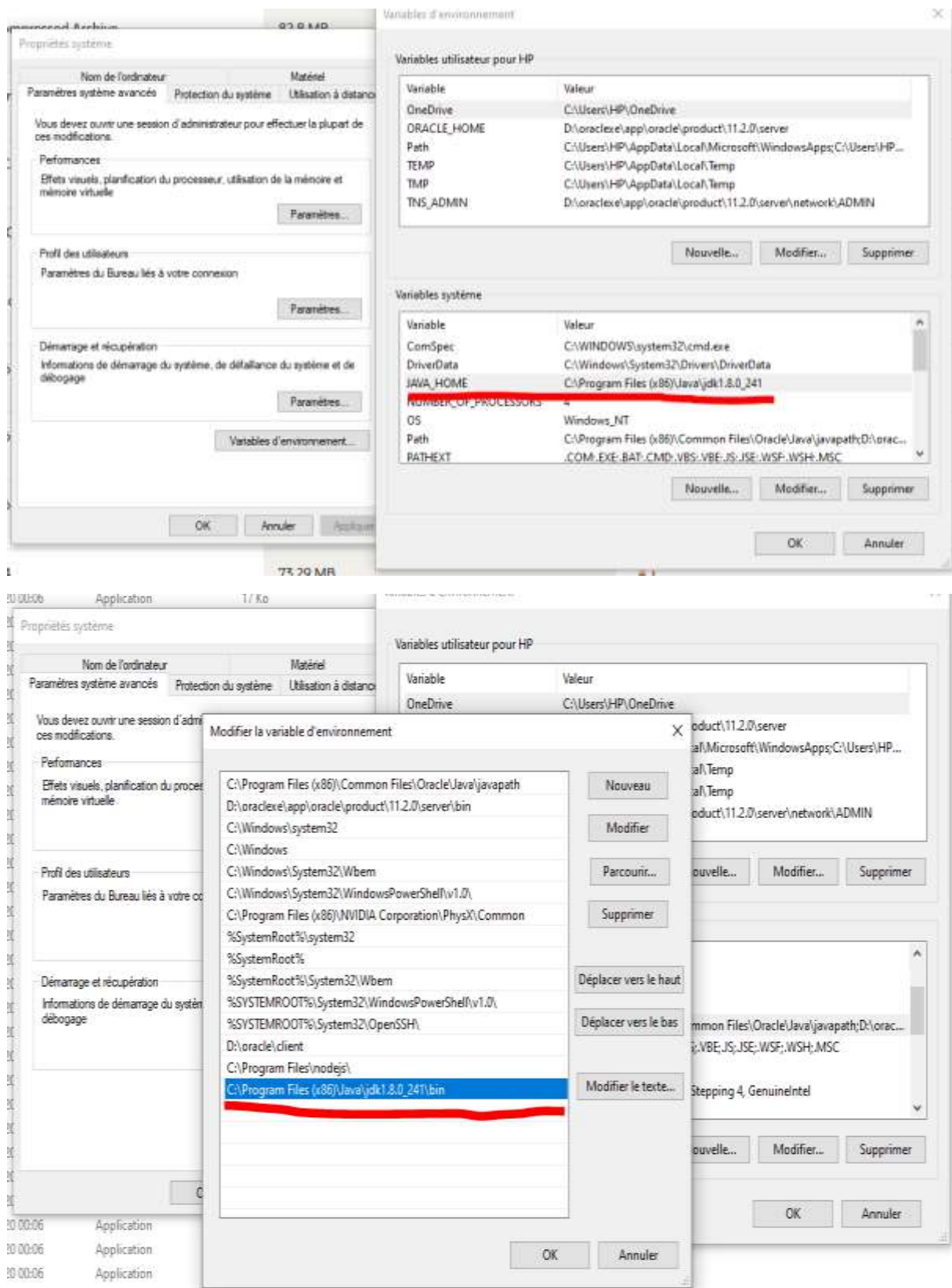


Installation du JRE : de même que JDK, on doit télécharger le JRE depuis le site d'oracle et l'installer.



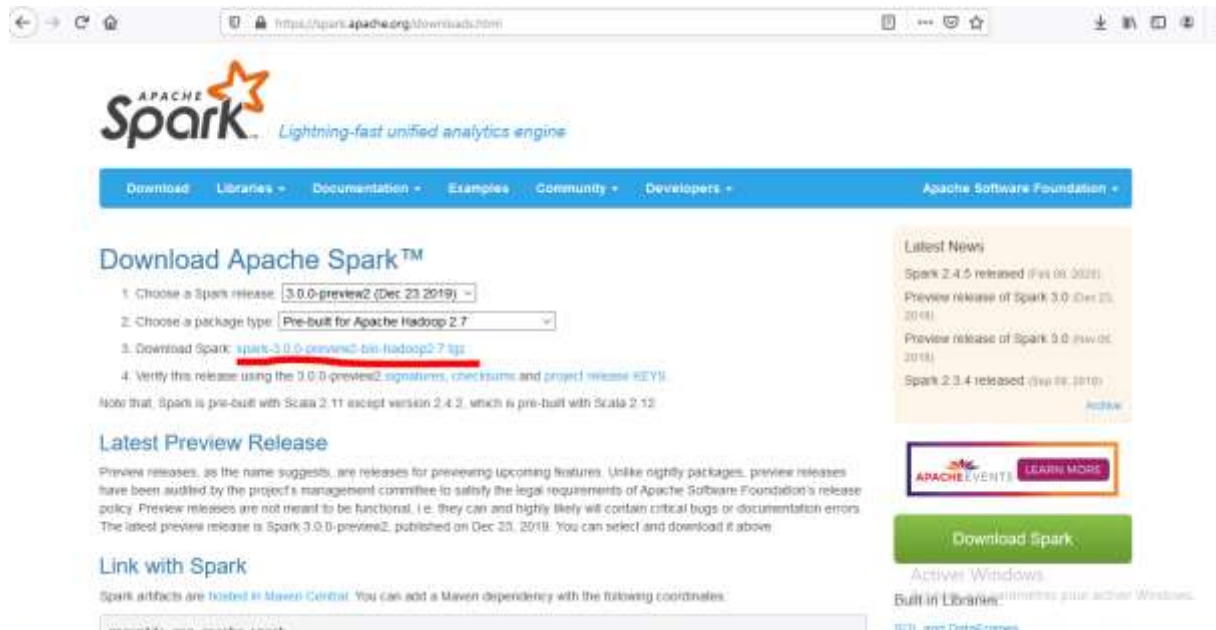
Configuration de l'environnement Java : l'ajout des variables d'environnement.

Premièrement, on ajoute la variable HOME_JAVA et puis on édite la variable path pour configurer l'environnement du JAVA.



Installation du SPARK

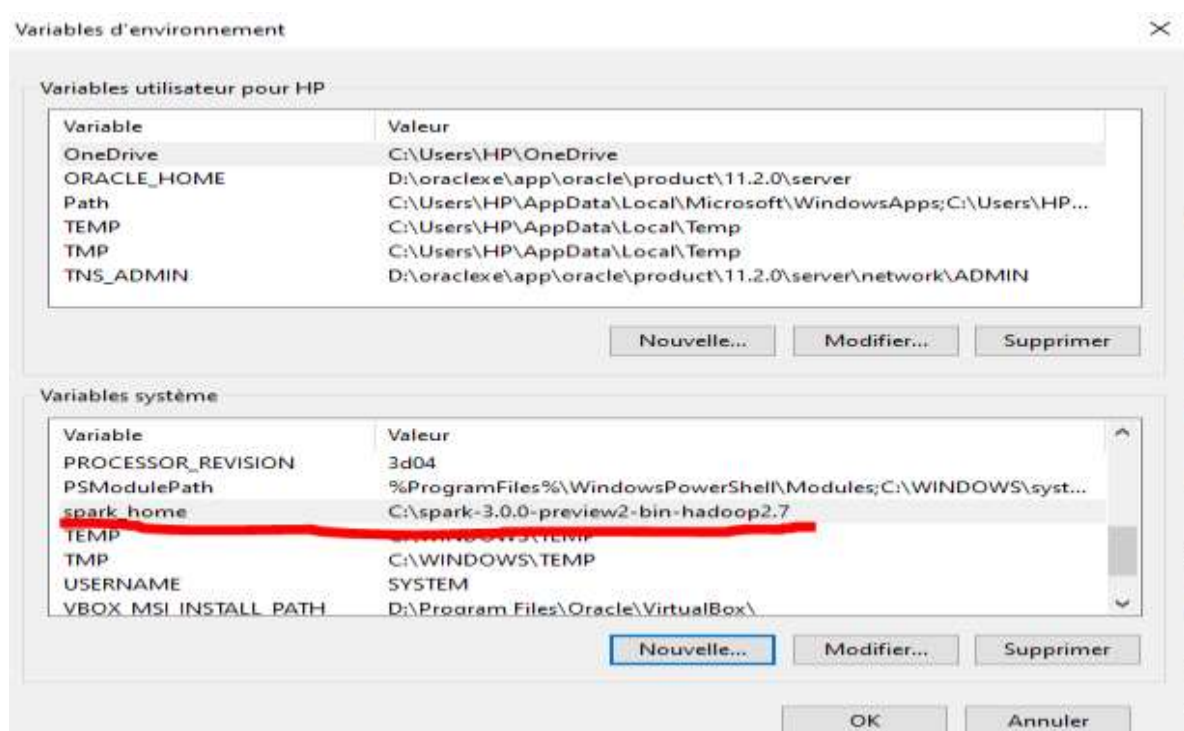
Afin de réaliser cette opération, on doit d'abord visiter le site d'apache SPARK et télécharger le fichier zip.



Configuration de l'environnement SPARK

Après le téléchargement et l'extraction, on peut copier le fichier sur le disque pour faciliter la suite de la configuration.

La configuration de l'environnement du Spark se fait par l'ajout de la variable d'environnement SPARK_HOME et la modification de la variable path comme suite :



Modifier la variable d'environnement

C:\Program Files (x86)\Common Files\Oracle\Java\javapath
D:\oracle\app\oracle\product\11.2.0\server\bin
C:\Windows\system32
C:\Windows
C:\Windows\System32\Wbem
C:\Windows\System32\WindowsPowerShell\v1.0\
C:\Program Files (x86)\NVIDIA Corporation\PhysX\Common
%SystemRoot%\system32
%SystemRoot%
%SystemRoot%\System32\Wbem
%SYSTEMROOT%\System32\WindowsPowerShell\v1.0\
%SYSTEMROOT%\System32\OpenSSH\
D:\oracle\client
C:\Program Files\nodejs\
C:\Program Files (x86)\Java\jdk1.8.0_241\bin
C:\spark-3.0.0-preview2-bin-hadoop2.7\bin

Nouveau

Modifier

Parcourir...

Supprimer

Déplacer vers le haut

Déplacer vers le bas

Modifier le texte...

OK

Annuler

Installation du Hadoop

Apache Hadoop

[Download](#)
[Documentation](#)
[Community](#)
[Development](#)
[Help](#)
[Outlets](#)
Apache Software Foundation

Download

Hadoop is released as source code tarballs with corresponding binary tarballs for convenience. The downloads are distributed via mirror sites and should be checked for tampering using GPG or SHA-512.

Version	Release date	Source download	Binary download	Release notes
3.19.0	2019 Oct 29	source (checksum signature)	binary (checksum signature)	Announcement
3.1.3	2019 Oct 21	source (checksum signature)	binary (checksum signature)	Announcement
3.2.1	2018 Sep 22	source (checksum signature)	binary (checksum signature)	Announcement
3.1.2	2018 Feb 6	source (checksum signature)	binary (checksum signature)	Announcement
2.9.2	2018 Nov 19	source (checksum signature)	binary (checksum signature)	Announcement

To verify Hadoop releases using GPG:

- Download the release `hadoop-X.Y.Z-src.tar.gz` from a mirror site.
- Download the signature file `hadoop-X.Y.Z-src.tar.gz.asc` from Apache.
- Download the `hadoop-X.Y.Z` file.
- `gpg --import KEYS`
- `gpg --verify hadoop-X.Y.Z-src.tar.gz.asc`

To perform a quick check using SHA-512:

- Download the release `hadoop-X.Y.Z-src.tar.gz` from a mirror site.
- Download the checksum `hadoop-X.Y.Z-src.tar.gz.sha512` or `hadoop-X.Y.Z-src.tar.gz.sha1` from Apache.
- `sha512sum -c hadoop-X.Y.Z-src.tar.gz.sha512`

All previous releases of Hadoop are available from the [Apache release archive](#) site.

Many third parties distribute products that include Apache Hadoop and related tools. Some of these are listed on the [Distributors](#) web page.

License

The software is licensed under [Apache License 2.0](#).

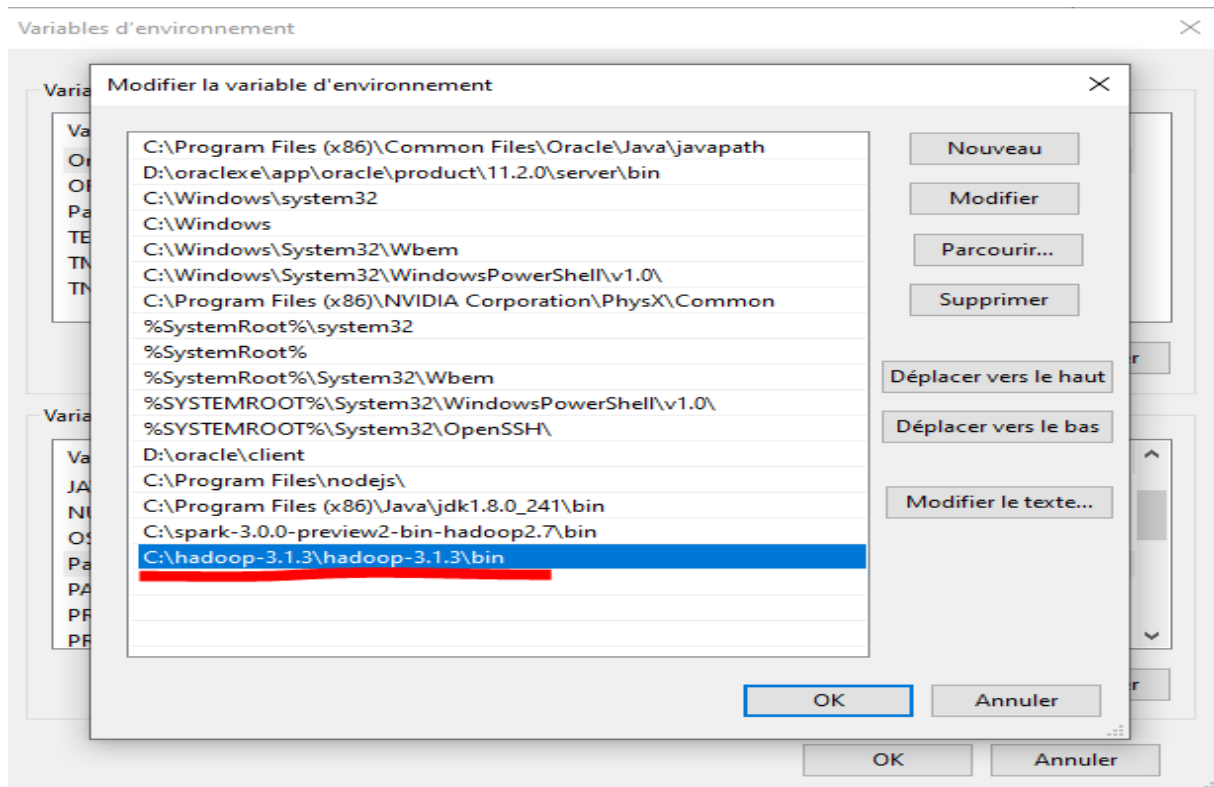
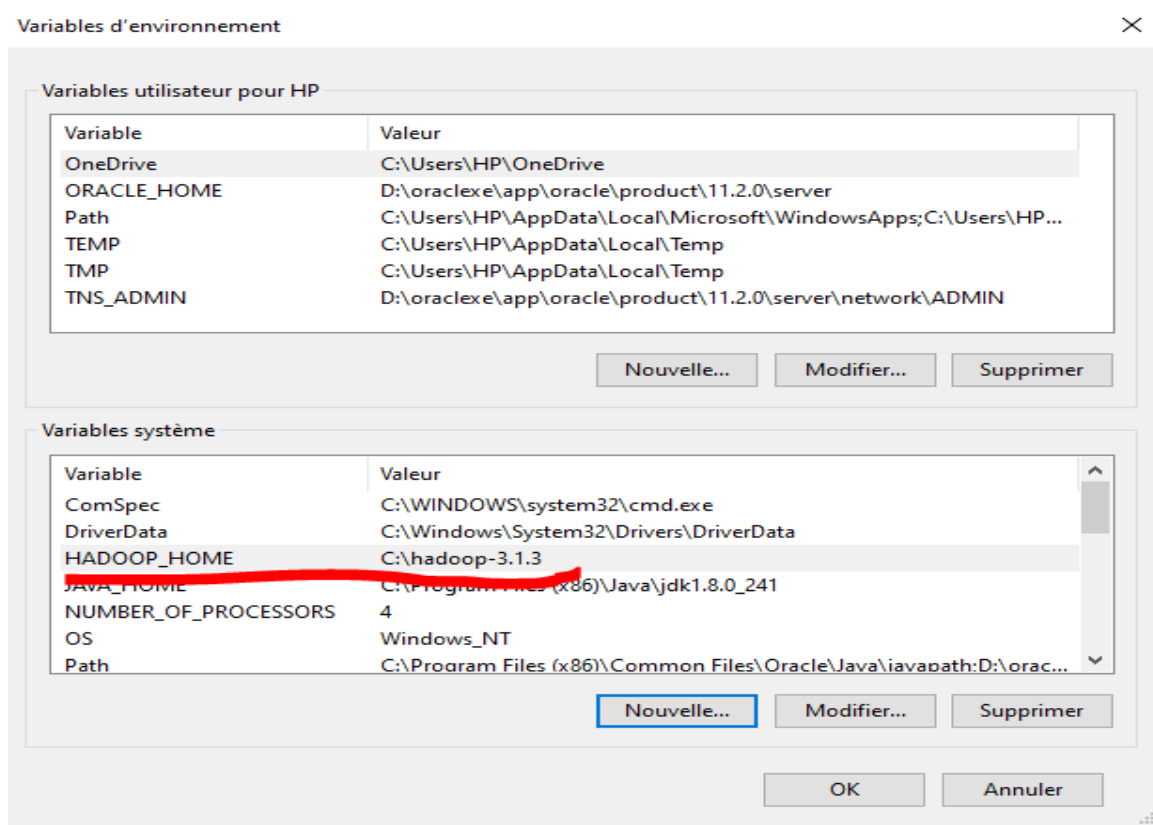
Apache Hadoop, Hadoop, Apache, the Apache feather logo, and the Apache Hadoop project logo are either registered trademarks or trademarks of the Apache Software Foundation in the United States and other countries.

Copyright © 2005-2020 The Apache Software Foundation

APACHE SOFTWARE FOUNDATION
<http://www.apache.org/>

A la suite du téléchargement de la version binaire de Hadoop depuis le site officiel d'apache, on peut le copier dans le disque C:/ et refaire la même configuration que les parties précédentes.

Voici les images qui montrent la configuration faite sur les variables d'environnement :



Installation du Eclipse

L'installation d'Eclipse est faite pour faciliter l'utilisation du JAVA ou SCALA dans la programmation des méthodes d'apprentissages.

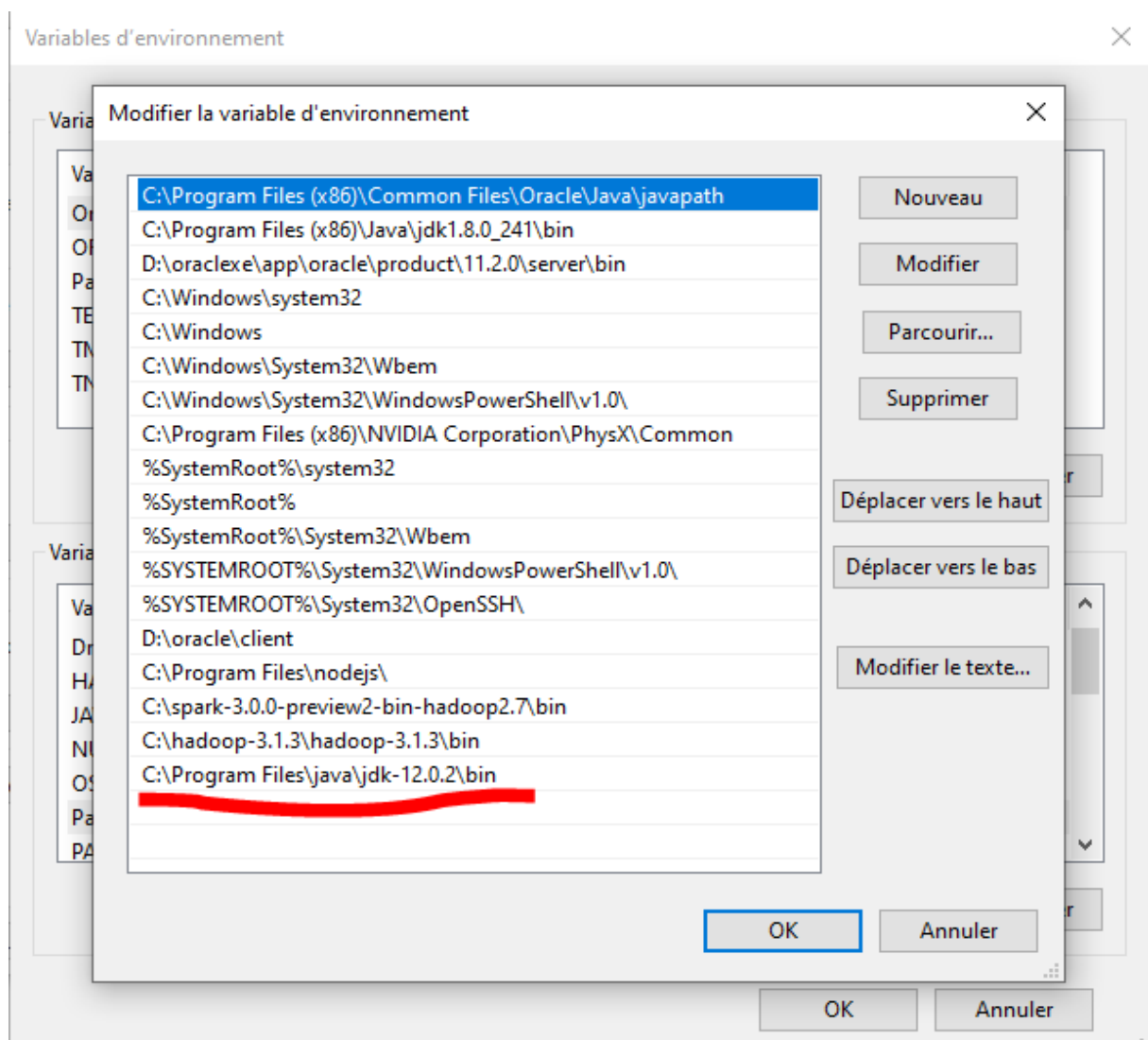
Problème : lors de l'installation du Eclipse, ce dernier ne marche pas et il affiche un message d'erreur de même type que celui-ci :

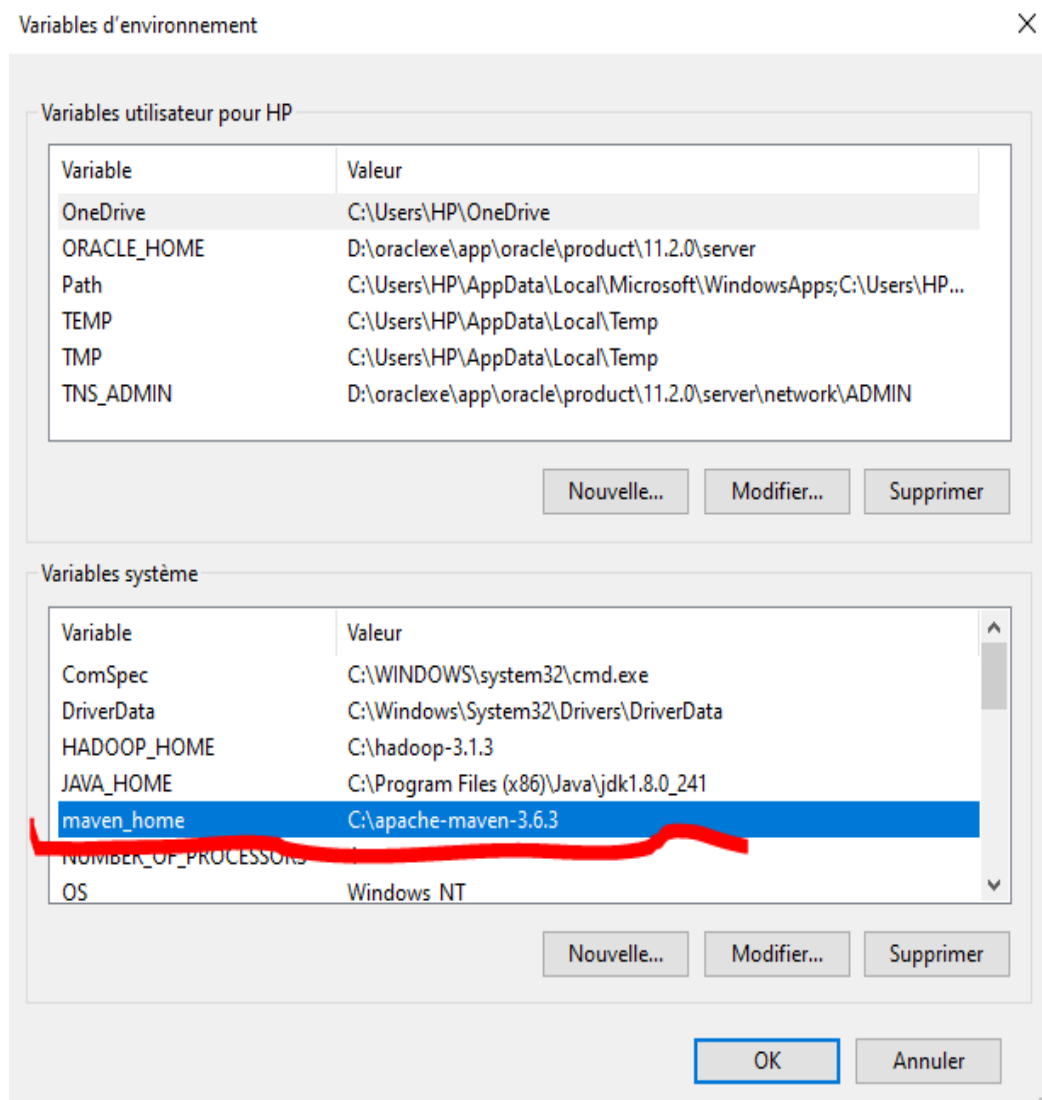
Failed to load the JNI shared library "C:\Program Files (x86)\Java\jre6\bin\client\jvm.dll".

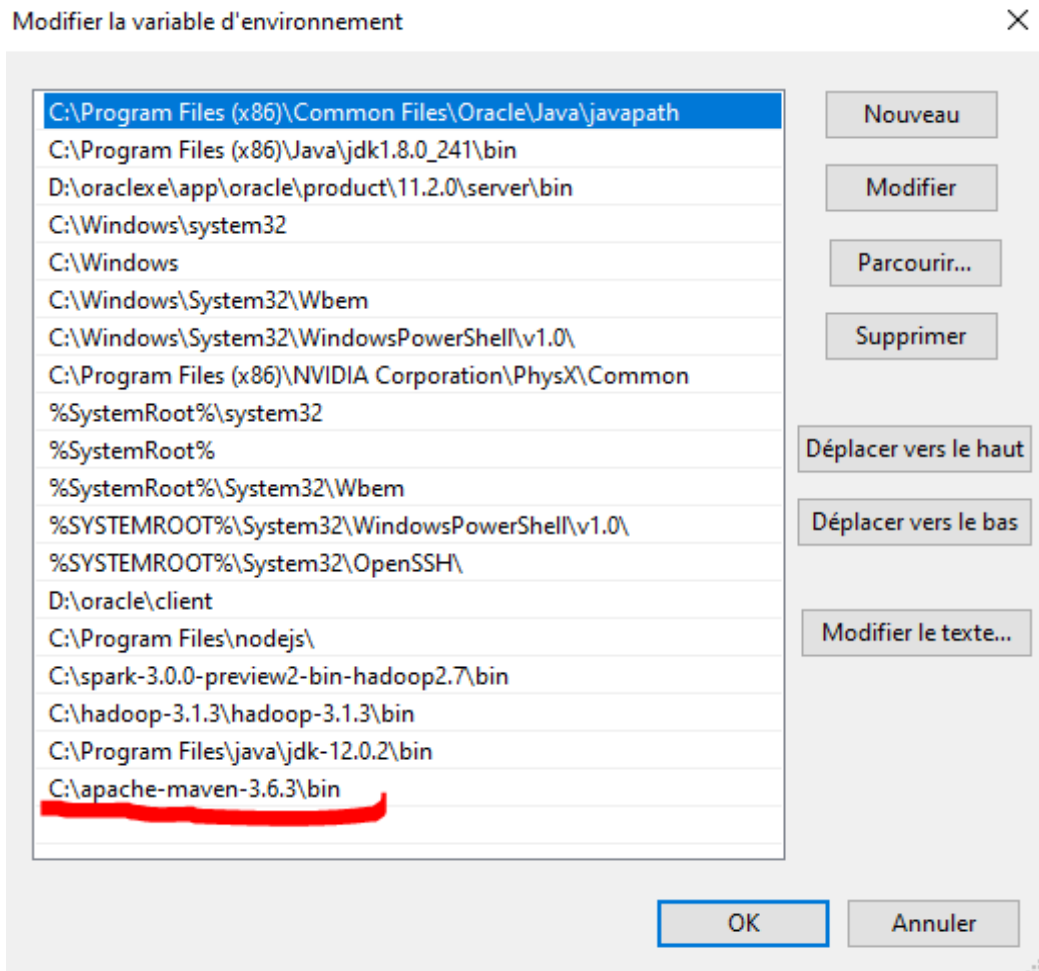
Ce problème revient à la non-détection du JDK lors de l'exécution du programme d'installation, j'ai trouvé la solution dans le forum communauté d'Eclipse

<https://www.eclipse.org/forums/index.php/t/171274/> .

Par contre j'ai résolu mon problème par le téléchargement de la dernière version du JDK puisqu'il est conseiller dans le site d'Eclipse d'utiliser les dernières versions du JDK, et j'ai ajouté la référence dans la variable d'environnement path.







Installation du l'IDE Scala

Généralement le travail sur [Hadoop](#) génère un temps de réponse relativement long. C'est la raison pour laquelle ce type de plateforme est conçue, pour être avant tout performante sur de gros volumes de données avec un concept qui assure que les données soient montées en mémoire et les traitements soient de fait jusqu'à 100 fois plus rapide que sur Hadoop.

En visitant le site du scala IDE, on peut télécharger le IDE pour Eclipse puis le lancer après l'extraction du fichier zip.

<http://scala-ide.org/download/sdk.html>

Problème : Erreur lors du lancement du scala IDE, la source du problème réside dans la variable d'environnement d'oracle. Ce type de problème avec oracle XE (base de données) et java est populaire lors des installations.

L'erreur est dans l'image suivante :



Solution :

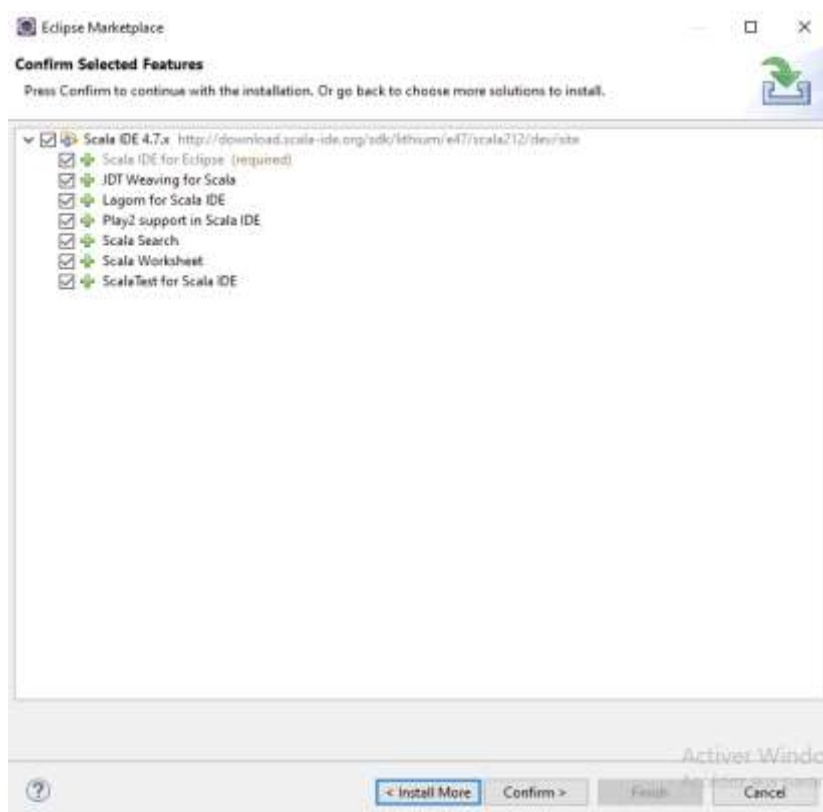
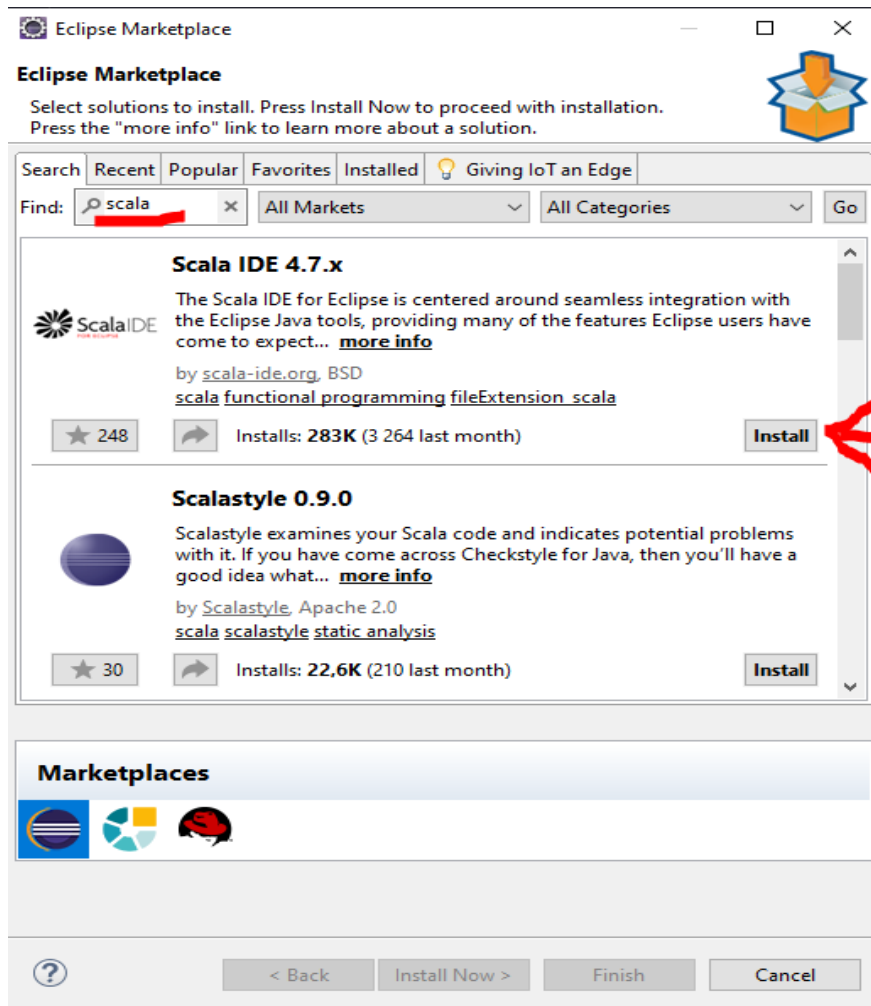
Dans le cas présent de ce travail, j'ai déjà téléchargé Eclipse, donc il me reste juste de trouver une solution pour intégrer scala dans mon Eclipse IDE.

L'IDE Scala pour Eclipse est centré sur une intégration transparente avec les outils Java d'Eclipse, offrant de nombreuses fonctionnalités que les utilisateurs d'Eclipse attendent.

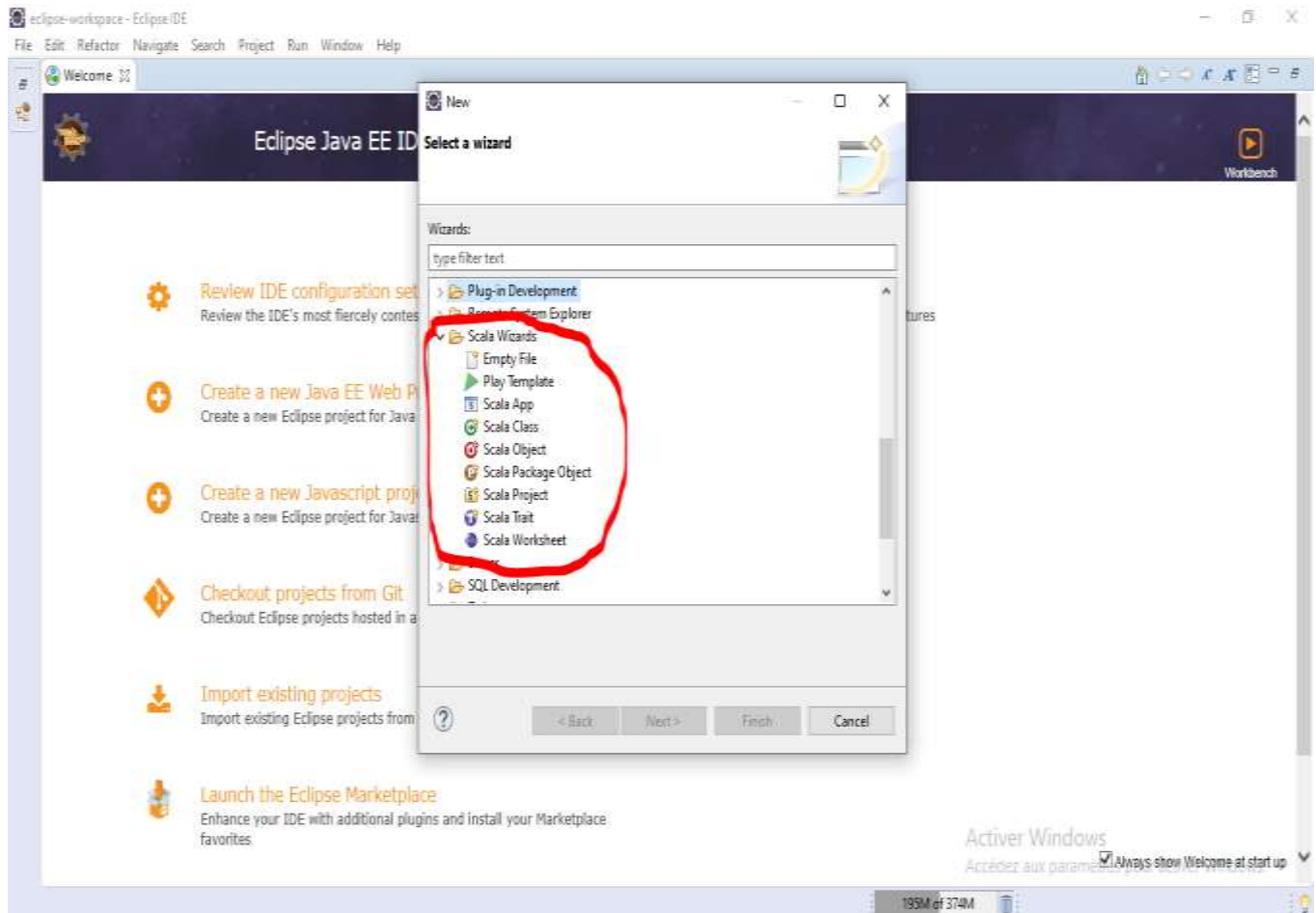
En suivant les étapes suivantes, on peut intégrer scala pour ouvrir des projets scala facilement dans Eclipse.

Dans Eclipse IDE :

1. Clic sur **help**.
2. Clic sur **Eclipse marketplace** pour ouvrir la fenêtre Eclipse MarketPlace.
3. Clic **Install**.
4. Cocher toutes les cases et confirmer.



Donc après l'intégration, on peut ouvrir des projets scala comme il est montré dans la fenêtre suivante :



Lancement du Spark

Dans des étapes précédentes, on n'a pas fait monter Spark sur un serveur, mais on a fait juste les configurations qui assurent son fonctionnement.

Pour lancer Spark on doit accéder avec la ligne de commande cmd au fichier Spark qu'on a copié dans le disque et lancer le Spark Shell mais comme il est basé sur le JAVA il va y'avoir automatiquement **un problème**.

Problème :

Lors de l'exécution de spark-shell dans la ligne de commande le résultat va être du type `spark-shell \Java\jdk1.8.0_241\bin\java` était inattendu comme illustré dans l'image suivante :

```
Administrator: C:\windows\system32\cmd.exe

C:\Users\dt203916>java -version
java version "1.8.0_162"
Java(TM) SE Runtime Environment (build 1.8.0_162-b12)
Java HotSpot(TM) Client VM (build 25.162-b12, mixed mode)

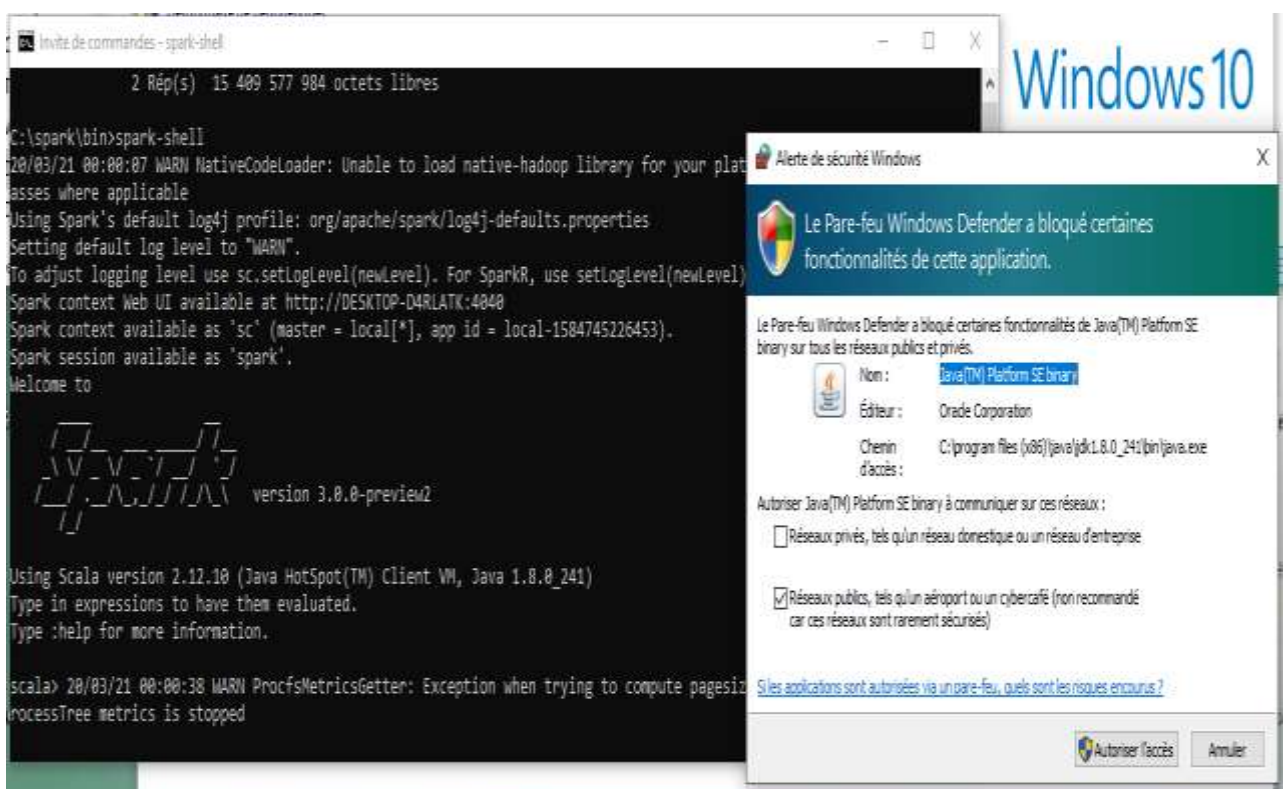
C:\Users\dt203916>spark-shell
\Java\jdk1.7.0\bin\java was unexpected at this time.
```

Ceci revient à la variable d'environnement JAVA_HOME qui pointe vers un chemin contenant des espaces, et scala ne marche pas avec les références qui contiennent des espaces.

Explication plus simple : La variable d'environnement JAVA_HOME contient un espace « Program Files (x86) » qui brise dans le chemin JAVA_HOME=C:\Program Files (x86)\Java\jdk1.8.0_162\bin.

La solution envisagée est de réinstaller Java dans un répertoire sans espace (créer un dossier JAVA et copier le contenu du dossier " Program Files (x86)\Java" dans ce dossier) puis de modifier la variable JAVA_HOME dans le dossier créé. = **le problème est réglé**

L'image suivante représente l'étape du lancement du Spark après le réglage du problème.



Pour assurer l'utilisation du Spark dans Eclipse, on doit assurer l'exécution de Spark dans un serveur local, pour le faire on doit suivre les étapes illustrées dans les images suivantes.

Donc on doit tout d'abord déployer Spark comme un master dans un serveur local comme le montre l'image suivante.

```
Invité de commandes - spark-class org.apache.spark.deploy.master.Master
Microsoft Windows [version 10.0.18362.720]
(c) 2019 Microsoft Corporation. Tous droits réservés.

C:\Users\HP>cd ..

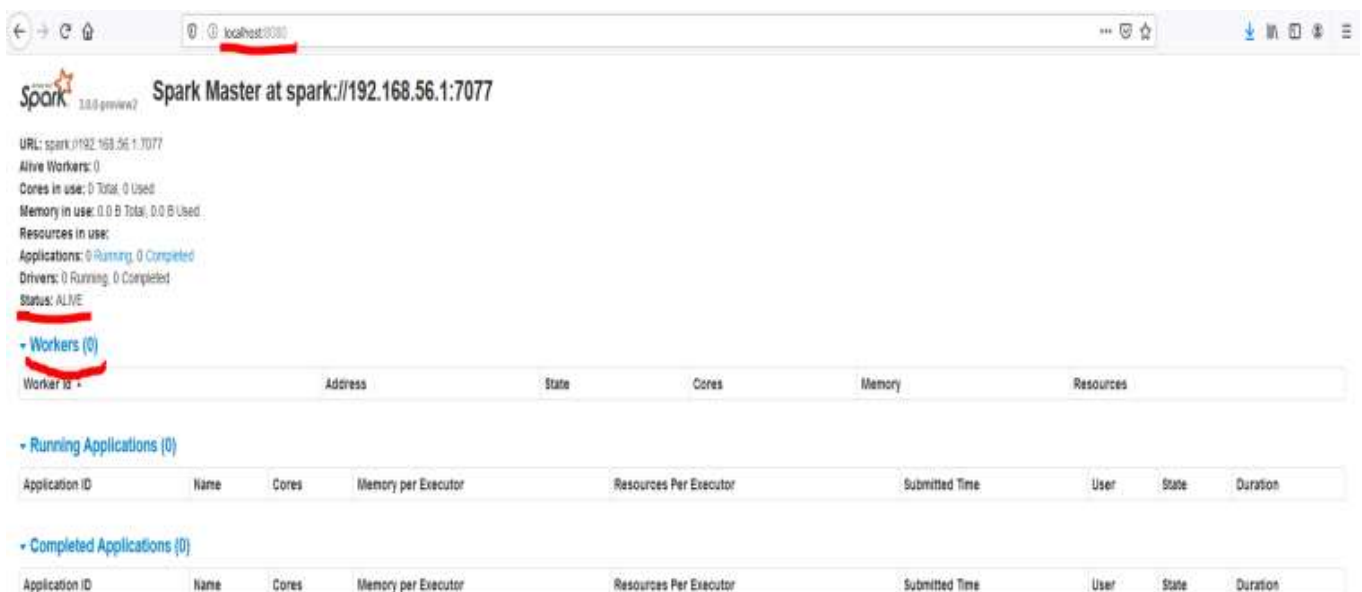
C:\Users>cd ..

C:\>cd spark

C:\spark>spark-class org.apache.spark.deploy.master.Master
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
20/03/21 01:05:39 INFO Master: Started daemon with process name: 17736@DESKTOP-D4RLATK
20/03/21 01:05:45 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
20/03/21 01:05:45 INFO SecurityManager: Changing view acls to: HP
20/03/21 01:05:45 INFO SecurityManager: Changing modify acls to: HP
20/03/21 01:05:45 INFO SecurityManager: Changing view acls groups to:
20/03/21 01:05:45 INFO SecurityManager: Changing modify acls groups to:
20/03/21 01:05:45 INFO SecurityManager: SecurityManager: authentication disabled; ui acls disabled; users with view permissions: Set(HP); groups with view permissions: Set(); users with modify permissions: Set(HP); groups with modify permissions: Set()
20/03/21 01:05:48 INFO Utils: Successfully started service 'sparkMaster' on port 7077.
20/03/21 01:05:48 INFO Master: Starting Spark master at spark://192.168.56.1:7077
20/03/21 01:05:48 INFO Master: Running Spark version 3.0.0-preview2
20/03/21 01:05:48 INFO Utils: Successfully started service 'MasterUI' on port 8080.
20/03/21 01:05:48 INFO MasterWebUI: Bound MasterWebUI to 0.0.0.0, and started at http://DESKTOP-D4RLATK:8080
20/03/21 01:05:49 INFO Master: I have been elected leader! New state: ALIVE
```

On peut voir dans l'image précédente que le service est lancé dans le localhost dans le port 8080.

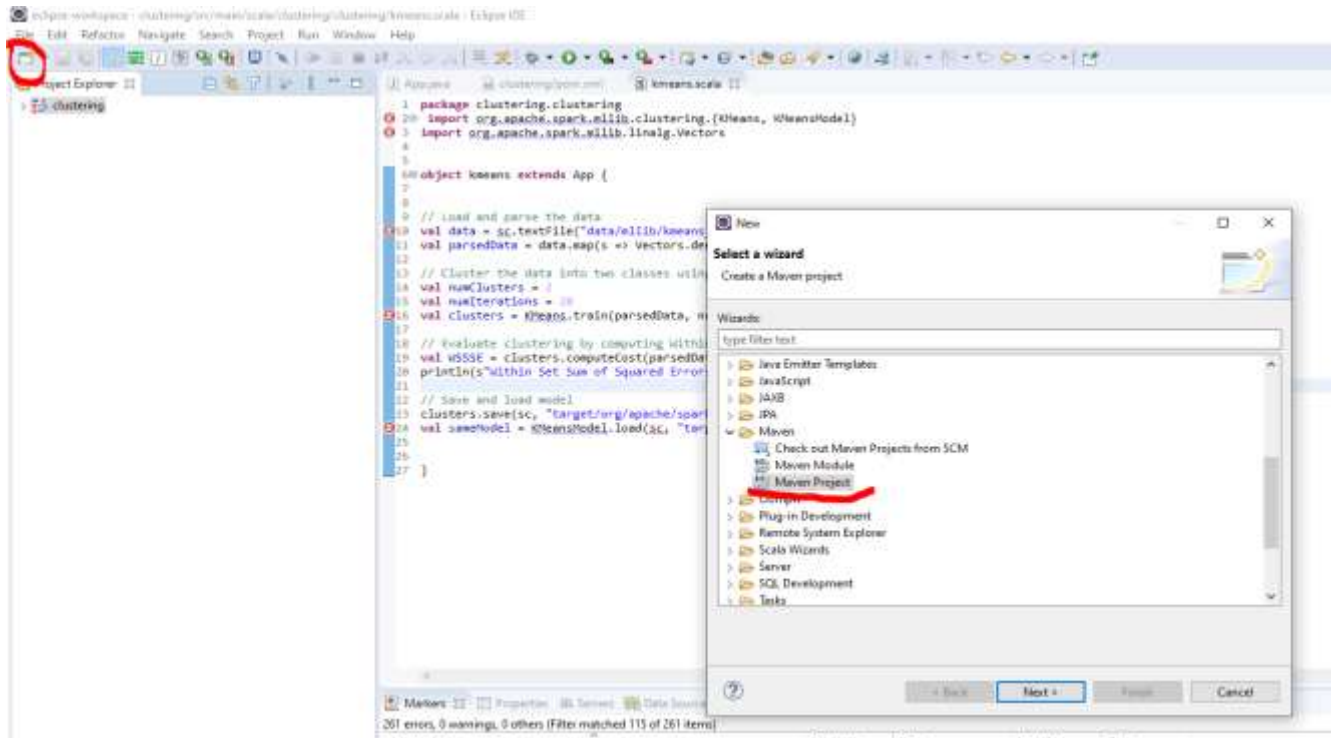
Dans l'image suivante en consultant le serveur local dans le port 8080, on constate que le statut du service est **ALIVE**.





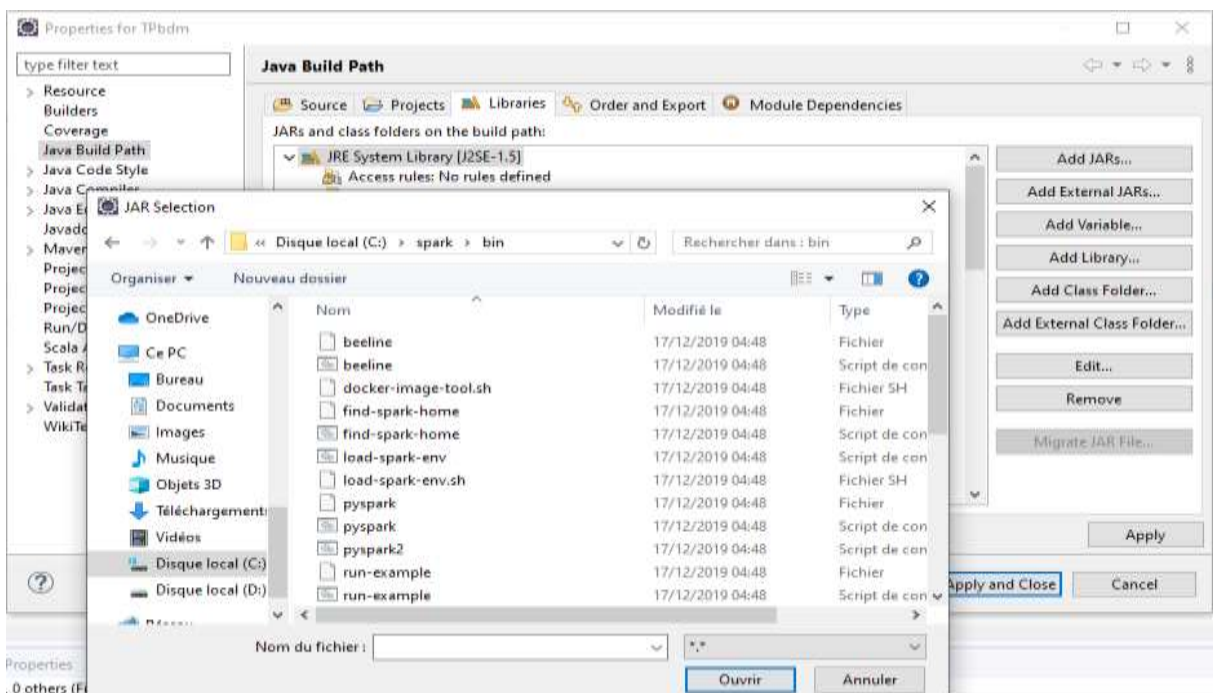
Travail à faire

Afin de réaliser ce présent travail, on doit lancer Eclipse et ouvrir un projet Maven comme il est montré dans l'image



Après on doit inclure notre Library Spark dans notre projet.

Tout d'abord, on doit cliquer droit sur JRE système Library | build path | configure build path | add external Jars (on doit trouver les JARs dans le fichier de Spark dans le disque C qu'on a déjà téléchargé)





Le choix de jeux de données et l'algorithme de résolution

Notre jeu de données va être des données existantes / historiques sur les taxis. Notre source de données est un document Excel qui contient les localisations des taxis dans une date (temps et date) et qui est rempli au fur et à mesure des déplacements des taxis , la localisation est faite par les coordonnées du géoréférencement (longitude et latitude) qui sont des paramètres déjà présents dans le fichier et qui aident à prédire les nouveaux trajets taxis.

Le choix de l'algorithme d'apprentissage automatique non supervisé est nécessaire puisqu'on a besoin de trouver des modèles sur des données sans étiquette (données sans catégories ou groupes définis).

On va utiliser l'algorithme K-Means pour construire un modèle d'apprentissage automatique avec Apache Spark. Ce modèle K-Means regroupe les données de voyage taxis en fonction des attributs de voyage. Ensuite, ce modèle peut être utilisé pour effectuer une analyse en temps réel des nouveaux trajets taxis.



Codage de l'algorithme

On doit tout d'abord lire notre source de données, on peut choisir un fichier text ou un CSV.

Dans notre cas il existe un fichier CSV qui contient les attributs « Date, Longitude , Latitude , Base »

Voici les commandes exécutées

On doit tout d'abord importer les différentes bibliothèques pour assurer l'exécution de K-means , Puis on doit importer notre source de données

```
scala> import org.apache.spark.mllib.linalg.Vectors
import org.apache.spark.mllib.linalg.Vectors

scala> import org.apache.spark.ml.feature.LabeledPoint
import org.apache.spark.ml.feature.LabeledPoint

scala> import org.apache.spark.ml.evaluation.ClusteringEvaluator
import org.apache.spark.ml.evaluation.ClusteringEvaluator

scala> import org.apache.spark.mllib.clustering.{KMeans, KMeansModel}
import org.apache.spark.mllib.clustering.{KMeans, KMeansModel}

scala> import org.apache.spark.ml.clustering.{KMeans, KMeansModel}
import org.apache.spark.ml.clustering.{KMeans, KMeansModel}
```

Problème : On remarque que dans dataset les colonnes date/time ; lat ;lon ;base sont de type string.

si on charge notre fichier CSV , les cases vont être du type string (ce qui ne marche pas avec le processus Kmeans) comme il est montré dans l'image suivante :

```
scala> val dataset = spark.read.format("csv").option("header", value = true).option("delimiter", ";").load("D:/ilyes.csv")
20/03/21 18:26:40 WARN SizeEstimator: Failed to check whether UseCompressedOops is set; assuming yes
dataset: org.apache.spark.sql.DataFrame = [Date/Time: string, Lat: string ... 3 more fields]

scala> dataset.show
+-----+-----+-----+-----+
|Date/Time| Lat| Lon| Base|features|
+-----+-----+-----+-----+
| 452524|45245| 425|45554| null|
| 45245| 4245| 4524| 4524| null|
| 4242| 4245| 42| 4524| null|
| 452| 524|254254| 425| null|
| 452| 452| 4254| 452| null|
+-----+-----+-----+-----+
```

Solution :

Donc la solution est de définir un schema (type de structure) pour charger notre fichier csv. On doit déclarer la structure dans une variable puis l'imposer dans les options du chargement de notre dataset.

Voici les commandes utilisées :

```
scala> val spark = SparkSession.builder.master("local[*]").appName("lambda").getOrCreate()
20/03/21 19:04:20 WARN SparkSession$Builder: Using an existing SparkSession; some configuration may not take effect.
spark: org.apache.spark.sql.SparkSession = org.apache.spark.sql.SparkSession@4e40a7

scala> import spark.implicits._
import spark.implicits._

scala> dataset.printSchema()
root
 |-- Date/Time: string (nullable = true)
 |-- Lat: string (nullable = true)
 |-- Lon: string (nullable = true)
 |-- Base: string (nullable = true)
 |-- features: string (nullable = true)

scala> import org.apache.spark.sql._
import org.apache.spark.sql._

scala> import org.apache.spark.sql.types._
import org.apache.spark.sql.types._

scala> val schema = StructType( StructField("time", TimestampType, nullable = true) :: StructField("lat", DoubleType, nullable = true) :: StructField("lon", DoubleType, nullable = true) :: StructField("base", StringType, nullable = true) :: Nil)
schema: org.apache.spark.sql.types.StructType = StructType(StructField(time,TimestampType,true), StructField(lat,DoubleType,true), StructField(lon,DoubleType,true), StructField(base,StringType,true))
```

L'importation des bibliothèques dans l'image précédente est nécessaire pour l'exécution de la fonction StructType et pour définir la structure de notre dataset.

Après la définition de notre structure, on a impliqué la structure dans le chargement de notre source CSV.

Voici la commande qui montre qu'évidemment la structure a été appliquée :

```
scala> val dataset = spark.read.format("csv").option("header", value = true).option("delimiter", ";").schema(schema).load("D:/ilyes.csv")
dataset: org.apache.spark.sql.DataFrame = [time: timestamp, lat: double ... 2 more fields]
```

Donc on peut remarquer que notre variable dataset contient une colonne de type date (Timestamp) et les autres de type double sauf l'attribut « base » qui est de type string.

Voici la commande utilisée pour afficher la structure de notre dataset :

```
scala> val dataset = spark.read.format("csv").option("header", value = true).option("delimiter", ";").schema(schema).load("D:/ilyes.csv")
dataset: org.apache.spark.sql.DataFrame = [time: timestamp, lat: double ... 2 more fields]

scala> dataset.printSchema()
root
 |-- time: timestamp (nullable = true)
 |-- lat: double (nullable = true)
 |-- lon: double (nullable = true)
 |-- base: string (nullable = true)

scala> _
```

Problème :

La Date qui est de type « Timestamp » ne s'affiche pas lors de l'exécution de la commande « Show », j'ai essayé toutes les méthodes mais aucune ne marche.

Ce problème ne va pas impacter notre travail car on est intéressé qu'aux colonnes « lat » et « lon » qui représentent les paramètres « longitude » et « latitude ».

Voici la solution que j'ai envisagée.

```
scala> val dataset= spark.read.format("csv").option("header", value = true).option("delimiter", "," ).option("timestampFormat", "dd/MM/yyyy HH:mm:ss").schema(schema).load("D:/uber.csv").cache()
20/03/24 15:44:47 WARN CacheManager: Asked to cache already cached data.
dataset: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [Date/Time: timestamp, lat: double ... 2 more fields]

scala> dataset.printSchema()
root
 |-- Date/Time: timestamp (nullable = true)
 |-- lat: double (nullable = true)
 |-- lon: double (nullable = true)
 |-- base: string (nullable = true)

scala> dataset.show(10)
+-----+-----+-----+-----+
|Date/Time|  lat|   lon|  base|
+-----+-----+-----+-----+
|    null|40.7521|-73.9914|B02512|
|    null|40.6965|-73.9715|B02512|
|    null|40.7464|-73.9838|B02512|
|    null|40.7463|-74.0011|B02512|
|    null|40.7594|-73.9734|B02512|
|    null|40.7685|-73.8625|B02512|
|    null|40.7637|-73.9962|B02512|
|    null|40.7252|-74.0023|B02512|
|    null|40.7607|-73.9625|B02512|
|    null|40.7212|-73.9879|B02512|
+-----+-----+-----+-----+
only showing top 10 rows
```

A la différence de la précédente commande « Read », j’ai ajouté une option qui impose un format sur les colonnes qui sont de type date « TimesTamp ».

Comme vous pouvez le voir dans la fonction option, la forme de la date est comme suit « dd/MM/yyyy HH:mm:ss».

Il nous reste de spécifier les colonnes qui sont concernées par l’algorithme de K-means (lon et lat) car lors de l’exécution simple de l’algorithme de K-means , scala oblige l’utilisateur d’avoir une colonne (ou vecteur) qui contient les informations des colonnes concernés (dans notre cas lon et lat) sous forme de points multidimensionnels (les axes des points sont lon et lat) .

Voici le problème affiché lors de l’exécution simple de l’algorithme de K-means , il affiche le manque de la colonne « Features ».

Voici le résultat dans l'image ci-dessous :

Un vecteur qui contient les coordonnées lon et lat dans une seule colonne Features dans notre dataset (on a nommé la variable du tableau feature).

```
scala> feature.show(10)
+-----+-----+-----+-----+-----+
|Date/Time| lat | lon | base | features |
+-----+-----+-----+-----+-----+
| null | 40.7521 | -73.9914 | B02512 | [40.7521, -73.9914] |
| null | 40.6965 | -73.9715 | B02512 | [40.6965, -73.9715] |
| null | 40.7464 | -73.9838 | B02512 | [40.7464, -73.9838] |
| null | 40.7463 | -74.0011 | B02512 | [40.7463, -74.0011] |
| null | 40.7594 | -73.9734 | B02512 | [40.7594, -73.9734] |
| null | 40.7685 | -73.8625 | B02512 | [40.7685, -73.8625] |
| null | 40.7637 | -73.9962 | B02512 | [40.7637, -73.9962] |
| null | 40.7252 | -74.0023 | B02512 | [40.7252, -74.0023] |
| null | 40.7607 | -73.9625 | B02512 | [40.7607, -73.9625] |
| null | 40.7212 | -73.9879 | B02512 | [40.7212, -73.9879] |
+-----+-----+-----+-----+-----+
only showing top 10 rows
```

L'application simple de K-means

Ensuite, nous pouvons construire le modèle K-Means en définissant le nombre de clusters, la colonne d'entités et la colonne de prédiction de sortie. Afin de former et de tester le modèle K-Means.

Problème :

Malheureusement la méthode ne marche pas à cause des valeurs null dans notre dataset. Après une longue recherche et modification dans la commande du read , j'ai trouvé que scala interprète le fichier Excel différemment.

Voici notre fichier Excel :

Les remarques constatées dans le fichier CSV:

1. Le header doit être simple, dans notre cas il y'a les "" pour spécifier que c'est un string mais ce n'est pas nécessaire dans scala.
2. La même remarque pour les valeurs de la colonne « Base » comme « B02512 » on a pas besoin des "" pour spécifier que c'est un string..
3. Une 2 -ème remarque : les valeurs de Date/Time ne sont pas par default « dd/MM/yyyy HH:mm:ss»

	A	B	C	D
1	Date/Time	"Lat", "Lon", "Base"		
2	05/01/2014 00:02:00	40.7521, -73.9914	"B02512"	
3	5/1/2014 0:06:00	40.6965, -73.9715	"B02512"	
4	5/1/2014 0:15:00	40.7464, -73.9838	"B02512"	
5	5/1/2014 0:17:00	40.7463, -74.0011	"B02512"	
6	5/1/2014 0:17:00	40.7594, -73.9734	"B02512"	
7	5/1/2014 0:20:00	40.7685, -73.8625	"B02512"	
8	5/1/2014 0:21:00	40.7637, -73.9962	"B02512"	
9	5/1/2014 0:21:00	40.7252, -74.0023	"B02512"	
10	5/1/2014 0:25:00	40.7607, -73.9625	"B02512"	
11	5/1/2014 0:25:00	40.7212, -73.9879	"B02512"	
12	5/1/2014 0:29:00	40.7255, -73.9986	"B02512"	
13	5/1/2014 0:32:00	40.6467, -73.7901	"B02512"	

Scala interprète le type Timestamp sur une valeur par default « dd/MM/yyyy HH:mm:ss» qui n'est pas respecté dans le fichier.

L'image suivante montre la différence après la modification dans le document.

Remarque : on a ajouté l'option « `option("mode", "DROPMALFORMED")` » dans la commande Read pour supprimer toutes les données qui ne respectent pas le format du schema (la structure) .

	A	B	C	D
1	Date/Time,Lat,Lon,Base			
2	05/01/2014 00:02:00	40.7521,-73.9914	B02512	
3	05/01/2014 00:06:00	40.6965,-73.9715	B02512	
4	5/1/2014 0:15:00	40.7464,-73.9838	B02512	
5	5/1/2014 0:17:00	40.7463,-74.0011	B02512	
6	5/1/2014 0:17:00	40.7594,-73.9734	B02512	
7				
8				

```
scala> val uberDf = spark.read.format("csv").option("header", value = true).option("delimiter", ",")
.option("mode", "DROPMALFORMED").option("timestampFormat", "dd/MM/yyyy HH:mm:ss").schema(schema).load(
  "hdfs://...")
uberDf: org.apache.spark.sql.DataFrame = [Date/Time: timestamp, Lat: double ... 2 more fields]

scala> uberDf.show
+-----+-----+-----+-----+
|   Date/Time   |   Lat   |   Lon   |   Base   |
+-----+-----+-----+-----+
|2014-01-05 00:02:00|40.7521|-73.9914|B02512|
|2014-01-05 00:06:00|40.6965|-73.9715|B02512|
+-----+-----+-----+-----+
```

On peut remarquer que les lignes 4,5 et 6 ne respectent pas le format « dd/MM/yyyy HH:mm:ss» imposé dans le Schema. Donc elles n'ont pas été chargées.

Alors que les 2 premières lignes respectent proprement le format déclaré dans Schema, ce qui explique leurs affichages par la fonction Show.

Voici les commandes utilisées pour vous montrer que notre dataset s'affiche effectivement après la modification:

```
scala> val uberDf = spark.read.format("csv").option("header", value = true).option("delimiter", ",")
.option("mode", "DROPMALFORMED").option("timestampFormat", "dd/MM/yyyy HH:mm:ss").schema(schema).load("D:/ilyes.csv")
uberDf: org.apache.spark.sql.DataFrame = [Date/Time: timestamp, Lat: double ... 2 more fields]

scala> uberDf.show
+-----+-----+-----+-----+
| Date/Time | Lat | Lon | Base |
+-----+-----+-----+-----+
| 2014-01-05 00:02:00 | 40.7521 | -73.9914 | B02512 |
| 2014-01-05 00:06:00 | 40.6965 | -73.9715 | B02512 |
| 2014-01-05 00:15:00 | 40.7464 | -73.9838 | B02512 |
| 2014-01-05 00:17:00 | 40.7463 | -74.0011 | B02512 |
| 2014-01-05 00:17:00 | 40.7594 | -73.9734 | B02512 |
+-----+-----+-----+-----+

scala> uberDf.printSchema
root
|-- Date/Time: timestamp (nullable = true)
|-- Lat: double (nullable = true)
|-- Lon: double (nullable = true)
|-- Base: string (nullable = true)
```

Maintenant, on va travailler avec le document ilyes.csv vu que l'autre ne respecte pas la forme du schema et il est long (25Mo), j'ai copier une partie dans le document **ilyes.csv** et j'ai assuré la modification nécessaire, voici le contenu du document ilyes.csv :

	A	B	C	D
1	Date/Time,Lat,Lon,Base			
2	05/01/2014 00:02:00,40.7521,-73.9914,B02512			
3	05/01/2014 00:06:00,40.6965,-73.9715,B02512			
4	05/01/2014 00:15:00,40.7464,-73.9838,B02512			
5	05/01/2014 00:17:00,40.7463,-74.0011,B02512			
6	05/01/2014 00:17:00,40.7594,-73.9734,B02512			
7				
8				
9				

Après cela, on doit refaire toutes les étapes expliquées dans le début de notre TP parce qu'on va utiliser le fichier ilyes.csv comme un dataset.

Voici une image qui montre les étapes réalisées:

```
scala> val dataset = spark.read.format("csv").option("header", value = true).option("delimiter", ",")
).option("mode", "DROPMALFORMED").option("timestampFormat", "dd/MM/yyyy HH:mm:ss").schema(schema).load("D:/ilyes.csv")
dataset: org.apache.spark.sql.DataFrame = [Date/Time: timestamp, Lat: double ... 2 more fields]

scala> dataset.printSchema
root
|-- Date/Time: timestamp (nullable = true)
|-- Lat: double (nullable = true)
|-- Lon: double (nullable = true)
|-- Base: string (nullable = true)

scala> dataset.show
+-----+-----+-----+-----+
| Date/Time | Lat | Lon | Base |
+-----+-----+-----+-----+
| 2014-01-05 00:02:00 | 40.7521 | -73.9914 | B02512 |
| 2014-01-05 00:06:00 | 40.6965 | -73.9715 | B02512 |
| 2014-01-05 00:15:00 | 40.7464 | -73.9838 | B02512 |
| 2014-01-05 00:17:00 | 40.7463 | -74.0011 | B02512 |
| 2014-01-05 00:17:00 | 40.7594 | -73.9734 | B02512 |
+-----+-----+-----+-----+

scala> val cols = Array("Lat", "Lon")
cols: Array[String] = Array(Lat, Lon)

scala> val assembler = new VectorAssembler().setInputCols(cols).setOutputCol("features")
assembler: org.apache.spark.ml.feature.VectorAssembler = VectorAssembler: uid=vecAssembler_3087cd4ed29e, handleInvalid=error, numInputCols=2

scala> val feature = assembler.transform(dataset)
feature: org.apache.spark.sql.DataFrame = [Date/Time: timestamp, Lat: double ... 3 more fields]

scala> feature.printSchema
root
|-- Date/Time: timestamp (nullable = true)
|-- Lat: double (nullable = true)
|-- Lon: double (nullable = true)
|-- Base: string (nullable = true)
|-- features: vector (nullable = true)

scala> feature.show
+-----+-----+-----+-----+-----+
| Date/Time | Lat | Lon | Base | features |
+-----+-----+-----+-----+-----+
| 2014-01-05 00:02:00 | 40.7521 | -73.9914 | B02512 | [40.7521,-73.9914] |
| 2014-01-05 00:06:00 | 40.6965 | -73.9715 | B02512 | [40.6965,-73.9715] |
| 2014-01-05 00:15:00 | 40.7464 | -73.9838 | B02512 | [40.7464,-73.9838] |
| 2014-01-05 00:17:00 | 40.7463 | -74.0011 | B02512 | [40.7463,-74.0011] |
| 2014-01-05 00:17:00 | 40.7594 | -73.9734 | B02512 | [40.7594,-73.9734] |
+-----+-----+-----+-----+-----+

scala>
```

Construction d'un modèle K-Means

La construction du modèle se fait en définissant le nombre de clusters, la colonne d'entités et la colonne de prédiction de sortie.

Dans notre cas, on a pris la variable feature (qui est le document Excel ilyes.csv + colonne imposée features) comme un ensemble de données d'apprentissage.

Puisque notre ensemble de données est petit, j'ai choisi de classifier selon 2 clusters, l'image suivante représente la démarche d'utilisation de la méthode k-means avec 2 clusters.

La dernière ligne représente les centres des 2 clusters.

```
scala> val kmeans = new KMeans().setK(2).setFeaturesCol("features").setPredictionCol("prediction")
kmeans: org.apache.spark.ml.clustering.KMeans = kmeans_082385997663

scala> val kmeansModel = kmeans.fit(feature)
kmeansModel: org.apache.spark.ml.clustering.KMeansModel = KMeansModel: uid=kmeans_082385997663, k=2,
  distanceMeasure=euclidean, numFeatures=2

scala> kmeansModel.clusterCenters.foreach(println)
[40.75105,-73.987425]
[40.6965,-73.9715]

scala>
```

Voici les clusters attribués à l'ensemble des données :

```
scala> val predict = kmeansModel.transform(feature)
predict: org.apache.spark.sql.DataFrame = [Date/Time: timestamp, Lat: double ... 4 more fields]

scala> predict.show
+-----+-----+-----+-----+-----+-----+
| Date/Time | Lat | Lon | Base | features | prediction |
+-----+-----+-----+-----+-----+-----+
| 2014-01-05 00:02:00 | 40.7521 | -73.9914 | 802512 | [40.7521,-73.9914] | 0 |
| 2014-01-05 00:06:00 | 40.6965 | -73.9715 | 802512 | [40.6965,-73.9715] | 1 |
| 2014-01-05 00:15:00 | 40.7464 | -73.9838 | 802512 | [40.7464,-73.9838] | 0 |
| 2014-01-05 00:17:00 | 40.7463 | -74.0011 | 802512 | [40.7463,-74.0011] | 0 |
| 2014-01-05 00:17:00 | 40.7594 | -73.9734 | 802512 | [40.7594,-73.9734] | 0 |
+-----+-----+-----+-----+-----+-----+
```

Le test du modèle avec l'ensemble de données

Dans la section précédente '**Construction d'un modèle K-Means**', on a donné un ensemble de données pour l'apprentissage du modèle, maintenant on peut prédire la classe de n'importe quelle information ajoutée dans notre source de donnée.

Pour le test, on est obligé d'ajouter d'autres lignes dans le fichier **ilyes.csv** pour confirmer que le modèle peut prédire de nouvelles informations, parce que les 5 premières lignes ont été déjà utilisées dans la **construction du modèle**.

Voici le document après l'ajout et la modification d'un ensemble de lignes pour respecter le Schema utilisé dans la lecture et éviter les problèmes cités dans les sections précédentes.

	A	B	C	D
1	Date/Time,Lat,Lon,Base			
2	05/01/2014 00:02:00,40.7521,-73.9914,B02512			
3	05/01/2014 00:06:00,40.6965,-73.9715,B02512			
4	05/01/2014 00:15:00,40.7464,-73.9838,B02512			
5	05/01/2014 00:17:00,40.7463,-74.0011,B02512			
6	05/01/2014 00:17:00,40.7594,-73.9734,B02512			
7	05/01/2014 00:20:00,40.7685,-73.8625,B02512			
8	05/01/2014 00:21:00,40.7637,-73.9962,B02512			
9	05/01/2014 00:21:00,40.7252,-74.0023,B02512			
10	05/01/2014 00:25:00,40.7607,-73.9625,B02512			
11	05/01/2014 00:25:00,40.7212,-73.9879,B02512			
12	05/01/2014 00:29:00,40.7255,-73.9986,B02512			
13	05/01/2014 00:32:00,40.6467,-73.7901,B02512			
14	05/01/2014 00:40:00,40.7613,-73.9788,B02512			
15				

Dans cette section du **test de notre modèle**, on va prédire l'ensemble des données ajoutées (ligne 7 jusqu'à la ligne 14).

On va tout d'abord charger à nouveau le document **ilyes.csv** qui contient des informations nouvelles concernant les taxis et mettre à jour la table feature.

L'image ci-dessous montre la différence entre l'ancien tableau feature et le nouveau après le chargement des nouvelles données.

Voici l'image des commandes utilisées :

```
scala> feature.show
```

Date/Time	Lat	Lon	Base	features
2014-01-05 00:02:00	40.7521	-73.9914	B02512	[40.7521,-73.9914]
2014-01-05 00:06:00	40.6965	-73.9715	B02512	[40.6965,-73.9715]
2014-01-05 00:15:00	40.7464	-73.9838	B02512	[40.7464,-73.9838]
2014-01-05 00:17:00	40.7463	-74.0011	B02512	[40.7463,-74.0011]
2014-01-05 00:17:00	40.7594	-73.9734	B02512	[40.7594,-73.9734]
2014-01-05 00:20:00	40.7685	-73.8625	B02512	[40.7685,-73.8625]

← ancien

```
scala> val dataset = spark.read.format("csv").option("header", value = true).option("delimiter", ",")
).option("mode", "DROPMALFORMED").option("timestampFormat", "dd/MM/yyyy HH:mm:ss").schema(schema).load("D:/ilyes.csv")
```

```
dataset: org.apache.spark.sql.DataFrame = [Date/Time: timestamp, Lat: double ... 2 more fields]
```

```
scala> dataset.show
```

Date/Time	Lat	Lon	Base
2014-01-05 00:02:00	40.7521	-73.9914	B02512
2014-01-05 00:06:00	40.6965	-73.9715	B02512
2014-01-05 00:15:00	40.7464	-73.9838	B02512
2014-01-05 00:17:00	40.7463	-74.0011	B02512
2014-01-05 00:17:00	40.7594	-73.9734	B02512
2014-01-05 00:20:00	40.7685	-73.8625	B02512
2014-01-05 00:21:00	40.7637	-73.9962	B02512
2014-01-05 00:21:00	40.7252	-74.0023	B02512
2014-01-05 00:25:00	40.7607	-73.9625	B02512
2014-01-05 00:25:00	40.7212	-73.9879	B02512
2014-01-05 00:29:00	40.7255	-73.9986	B02512
2014-01-05 00:32:00	40.6467	-73.7901	B02512
2014-01-05 00:40:00	40.7613	-73.9788	B02512

```
scala> val feature = assembler.transform(dataset)
```

```
feature: org.apache.spark.sql.DataFrame = [Date/Time: timestamp, Lat: double ... 3 more fields]
```

```
scala> feature.show
```

Date/Time	Lat	Lon	Base	features
2014-01-05 00:02:00	40.7521	-73.9914	B02512	[40.7521,-73.9914]
2014-01-05 00:06:00	40.6965	-73.9715	B02512	[40.6965,-73.9715]
2014-01-05 00:15:00	40.7464	-73.9838	B02512	[40.7464,-73.9838]
2014-01-05 00:17:00	40.7463	-74.0011	B02512	[40.7463,-74.0011]
2014-01-05 00:17:00	40.7594	-73.9734	B02512	[40.7594,-73.9734]
2014-01-05 00:20:00	40.7685	-73.8625	B02512	[40.7685,-73.8625]
2014-01-05 00:21:00	40.7637	-73.9962	B02512	[40.7637,-73.9962]
2014-01-05 00:21:00	40.7252	-74.0023	B02512	[40.7252,-74.0023]
2014-01-05 00:25:00	40.7607	-73.9625	B02512	[40.7607,-73.9625]
2014-01-05 00:25:00	40.7212	-73.9879	B02512	[40.7212,-73.9879]
2014-01-05 00:29:00	40.7255	-73.9986	B02512	[40.7255,-73.9986]
2014-01-05 00:32:00	40.6467	-73.7901	B02512	[40.6467,-73.7901]
2014-01-05 00:40:00	40.7613	-73.9788	B02512	[40.7613,-73.9788]

← nouveau

Maintenant le modèle va prédire l'affectation des nouvelles informations dans les clusters.

```
scala> feature.show
```

Date/Time	Lat	Lon	Base	features
2014-01-05 00:02:00	40.7521	-73.9914	B02512	[40.7521, -73.9914]
2014-01-05 00:06:00	40.6965	-73.9715	B02512	[40.6965, -73.9715]
2014-01-05 00:15:00	40.7464	-73.9838	B02512	[40.7464, -73.9838]
2014-01-05 00:17:00	40.7463	-74.0011	B02512	[40.7463, -74.0011]
2014-01-05 00:17:00	40.7594	-73.9734	B02512	[40.7594, -73.9734]
2014-01-05 00:20:00	40.7685	-73.8625	B02512	[40.7685, -73.8625]
2014-01-05 00:21:00	40.7637	-73.9962	B02512	[40.7637, -73.9962]
2014-01-05 00:21:00	40.7252	-74.0023	B02512	[40.7252, -74.0023]
2014-01-05 00:25:00	40.7607	-73.9625	B02512	[40.7607, -73.9625]
2014-01-05 00:25:00	40.7212	-73.9879	B02512	[40.7212, -73.9879]
2014-01-05 00:29:00	40.7255	-73.9986	B02512	[40.7255, -73.9986]
2014-01-05 00:32:00	40.6467	-73.7901	B02512	[40.6467, -73.7901]
2014-01-05 00:40:00	40.7613	-73.9788	B02512	[40.7613, -73.9788]

```
scala> val predict = kmeansModel.transform(feature)
predict: org.apache.spark.sql.DataFrame = [Date/Time: timestamp, Lat: double ... 4 more fields]

scala> predict.show
```

Date/Time	Lat	Lon	Base	features	prediction
2014-01-05 00:02:00	40.7521	-73.9914	B02512	[40.7521, -73.9914]	0
2014-01-05 00:06:00	40.6965	-73.9715	B02512	[40.6965, -73.9715]	1
2014-01-05 00:15:00	40.7464	-73.9838	B02512	[40.7464, -73.9838]	0
2014-01-05 00:17:00	40.7463	-74.0011	B02512	[40.7463, -74.0011]	0
2014-01-05 00:17:00	40.7594	-73.9734	B02512	[40.7594, -73.9734]	0
2014-01-05 00:20:00	40.7685	-73.8625	B02512	[40.7685, -73.8625]	0
2014-01-05 00:21:00	40.7637	-73.9962	B02512	[40.7637, -73.9962]	0
2014-01-05 00:21:00	40.7252	-74.0023	B02512	[40.7252, -74.0023]	0
2014-01-05 00:25:00	40.7607	-73.9625	B02512	[40.7607, -73.9625]	0
2014-01-05 00:25:00	40.7212	-73.9879	B02512	[40.7212, -73.9879]	1
2014-01-05 00:29:00	40.7255	-73.9986	B02512	[40.7255, -73.9986]	0
2014-01-05 00:32:00	40.6467	-73.7901	B02512	[40.6467, -73.7901]	1
2014-01-05 00:40:00	40.7613	-73.9788	B02512	[40.7613, -73.9788]	0

Donc les nouvelles informations ont été classifiées dans les 2 clusters (0 et 1) par rapport à l'ensemble de données d'apprentissage (les 5 premières lignes) .

Dans cette étude, on a classifié les données par rapport aux centres de 2 clusters en utilisant la distance euclidienne.

Si les données d'apprentissage ont été différentes , les résultats des prédictions vont différer eux aussi . Voici un exemple pour montrer la différence :


```

scala> val kmeans = new KMeans().setK(2).setFeaturesCol("features").setPredictionCol("prediction")
kmeans: org.apache.spark.ml.clustering.KMeans = kmeans_e3e4fdbcb394d

scala> val predict= kmeansModel.transform(feature)
predict: org.apache.spark.sql.DataFrame = [Date/Time: timestamp, Lat: double ... 4 more fields]

scala> val kmeansModel = kmeans.fit(feature)
kmeansModel: org.apache.spark.ml.clustering.KMeansModel = KMeansModel: uid=kmeans_e3e4fdbcb394d, k=2, distanceMeasure=euc
clidean, numFeatures=2

scala> val predict = kmeansModel.transform(feature)
predict: org.apache.spark.sql.DataFrame = [Date/Time: timestamp, Lat: double ... 4 more fields]

scala> kmeansModel.clusterCenters.foreach(println)
[40.74166363636364,-73.98613636363636]
[40.7076,-73.8263]

scala> val predict= kmeansModel.transform(feature)
predict: org.apache.spark.sql.DataFrame = [Date/Time: timestamp, Lat: double ... 4 more fields]

scala> predict.show
+-----+-----+-----+-----+-----+-----+
| Date/Time | Lat | Lon | Base | features | prediction |
+-----+-----+-----+-----+-----+-----+
| 2014-01-05 00:02:00 | 40.7521 | -73.9914 | 802512 | [40.7521,-73.9914] | 0 |
| 2014-01-05 00:06:00 | 40.6965 | -73.9715 | 802512 | [40.6965,-73.9715] | 0 |
| 2014-01-05 00:15:00 | 40.7464 | -73.9838 | 802512 | [40.7464,-73.9838] | 0 |
| 2014-01-05 00:17:00 | 40.7463 | -74.0011 | 802512 | [40.7463,-74.0011] | 0 |
| 2014-01-05 00:17:00 | 40.7594 | -73.9734 | 802512 | [40.7594,-73.9734] | 0 |
| 2014-01-05 00:20:00 | 40.7685 | -73.8625 | 802512 | [40.7685,-73.8625] | 1 |
| 2014-01-05 00:21:00 | 40.7637 | -73.9962 | 802512 | [40.7637,-73.9962] | 0 |
| 2014-01-05 00:21:00 | 40.7252 | -74.0023 | 802512 | [40.7252,-74.0023] | 0 |
| 2014-01-05 00:25:00 | 40.7607 | -73.9625 | 802512 | [40.7607,-73.9625] | 0 |
| 2014-01-05 00:25:00 | 40.7212 | -73.9879 | 802512 | [40.7212,-73.9879] | 0 |
| 2014-01-05 00:29:00 | 40.7255 | -73.9986 | 802512 | [40.7255,-73.9986] | 0 |
| 2014-01-05 00:32:00 | 40.6467 | -73.7901 | 802512 | [40.6467,-73.7901] | 1 |
| 2014-01-05 00:40:00 | 40.7613 | -73.9788 | 802512 | [40.7613,-73.9788] | 0 |
+-----+-----+-----+-----+-----+-----+

```

Dans le cas suivant, on a passé un grand nombre de lignes, et on peut constater que les centres des clusters ont changés, ce qui veut dire que les données d'apprentissage jouent un rôle très important dans la prédiction de classification des nouvelles informations, puisque ça peut donner des informations qui améliorent la prédiction et dès fois ça peut causer de faux résultats (perturbations) dans la méthode de prédiction.

Malgré qu'on a utilisé le même nombre de clusters mais les centres des clusters changent par le fait qu'il y'a plus d'informations qui donnent beaucoup plus d'informations d'apprentissage et de ce fait améliorent la prédiction.



Parallélisation sur un cluster

Malheureusement ce n'est pas possible de travailler sur un cloud puisque ce dernier est payant.

Même avec notre compte étudiant, Microsoft Azure ne dépose pas le service des machines virtuelles gratuitement.

Microsoft Azure

Rechercher dans les ressources, services et documents (G+)

Accueil > Créer une machine virtuelle > Sélectionner une taille de machine virtuelle

Sélectionner une taille de machine virtuelle

Parcourir les tailles de machine virtuelle disponibles et leurs caractéristiques

Rechercher par taille d... Effacer tous les filtres

Taille: Petite (0-5) Ajouter un filtre

Affichage de 86 tailles de machine virtuelle sur 234 | Abonnement: Azure pour les étudiants | Région: USA Centre Sud | Image: Ubuntu Server 18.04 LTS

Taille d...	Offre	Famille	Proce...	RAM (Go)	Disques	Opérations d'E/S par seconde maximales	Stockage temporaire (Go)	Prise en charge de disque Premium	coût/mois (estime)
A0	Standard	Usage général	1	0,75	1	1x500		Non	14,80 \$US
A0	De base	Usage général	1	0,75	1	1x300		Non	13,14 \$US
A1	Standard	Usage général	1	1,75	2	2x500		Non	43,80 \$US
A1	De base	Usage général	1	1,75	2	2x300		Non	18,25 \$US

Déploiement local

Pour assurer l'utilisation du Spark, on doit assurer l'exécution de spark dans un serveur local, pour le faire on doit suivre les étapes illustrées dans les images suivantes.

Donc on doit tout d'abord déployer Spark comme un master dans un serveur local comme le montre l'image suivante.

```
Invite de commandes - spark-class org.apache.spark.deploy.master.Master
Microsoft Windows [version 10.0.18362.720]
(c) 2019 Microsoft Corporation. Tous droits réservés.

C:\Users\HP>cd ..
C:\Users>cd ..
C:\>cd spark

C:\spark>spark-class org.apache.spark.deploy.master.Master
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
20/03/21 01:05:39 INFO Master: Started daemon with process name: 17736@DESKTOP-D4RLATK
20/03/21 01:05:45 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
20/03/21 01:05:45 INFO SecurityManager: Changing view acls to: HP
20/03/21 01:05:45 INFO SecurityManager: Changing modify acls to: HP
20/03/21 01:05:45 INFO SecurityManager: Changing view acls groups to:
20/03/21 01:05:45 INFO SecurityManager: Changing modify acls groups to:
20/03/21 01:05:45 INFO SecurityManager: SecurityManager: authentication disabled; ui acls disabled; users with view permissions: Set(HP); groups with view permissions: Set(); users with modify permissions: Set(HP); groups with modify permissions: Set()
20/03/21 01:05:48 INFO Utils: Successfully started service 'sparkMaster' on port 7077.
20/03/21 01:05:48 INFO Master: Starting Spark master at spark://192.168.56.1:7077
20/03/21 01:05:48 INFO Master: Running Spark version 3.0.0-preview2
20/03/21 01:05:48 INFO Utils: Successfully started service 'MasterUI' on port 8080.
20/03/21 01:05:48 INFO MasterWebUI: Bound MasterWebUI to 0.0.0.0, and started at http://DESKTOP-D4RLATK:8080
20/03/21 01:05:49 INFO Master: I have been elected leader! New state: ALIVE
```

On peut voir dans l'image précédente que le service MasterUI est lancé dans le localhost dans le port 8080.

En consultant le serveur local dans le port 8080 on constate que le statu du service est **ALIVE** mais il n'y'a aucun esclave (worker).

spark 3.0.0-preview2 Spark Master at spark://192.168.56.1:7077

URL: spark://192.168.56.1:7077
Alive Workers: 0
Cores in use: 0 Total, 0 Used
Memory in use: 0.0 B Total, 0.0 B Used
Resources in use:
Applications: 0 Running, 0 Completed
Drivers: 0 Running, 0 Completed
Status: ALIVE

Workers (0)

Worker ID	Address	State	Cores	Memory	Resources
-----------	---------	-------	-------	--------	-----------

Running Applications (0)

Application ID	Name	Cores	Memory per Executor	Resources Per Executor	Submitted Time	User	State	Duration
----------------	------	-------	---------------------	------------------------	----------------	------	-------	----------

Completed Applications (0)

Application ID	Name	Cores	Memory per Executor	Resources Per Executor	Submitted Time	User	State	Duration
----------------	------	-------	---------------------	------------------------	----------------	------	-------	----------

Remarque : Spark n'assure pas la sécurité lors du lancement des serveurs ce qui rend votre port ouvert vers l'extérieur et ouvre une vulnérabilité.

<https://spark.apache.org/docs/latest/spark-standalone.html>

Pour lancer Spark comme un worker (esclave), on peut exécuter la commande suivante :

```
Microsoft Windows [version 10.0.18362.720]
(c) 2019 Microsoft Corporation. Tous droits réservés.

C:\Users\HP>cd ..

C:\Users>cd ..

C:\>cd spark

C:\spark>spark-class org.apache.spark.deploy.worker.Worker spark://192.168.56.1:7077
```

Ce qui montre qu'on va déployer Spark comme un esclave dans notre serveur déjà vu dans les 2 images précédentes « **spark://192.168.56.1 :7077** » et voici le résultat de la commande :

```
Invite de commandes - spark-class org.apache.spark.deploy.worker.Worker spark://192.168.56.1:7077

C:\>cd spark

C:\spark>spark-class org.apache.spark.deploy.worker.Worker spark://192.168.56.1:7077
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
20/03/21 01:38:59 INFO Worker: Started daemon with process name: 7716@DESKTOP-D4RLATK
20/03/21 01:39:04 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java cl
asses where applicable
20/03/21 01:39:04 INFO SecurityManager: Changing view acls to: HP
20/03/21 01:39:04 INFO SecurityManager: Changing modify acls to: HP
20/03/21 01:39:04 INFO SecurityManager: Changing view acls groups to:
20/03/21 01:39:04 INFO SecurityManager: Changing modify acls groups to:
20/03/21 01:39:04 INFO SecurityManager: SecurityManager: authentication disabled; ui acls disabled; users with view per
missions: Set(HP); groups with view permissions: Set(); users with modify permissions: Set(HP); groups with modify perm
issions: Set()
20/03/21 01:39:07 INFO Utils: Successfully started service 'sparkWorker' on port 51138.
20/03/21 01:39:08 INFO Worker: Starting Spark worker 192.168.56.1:51138 with 4 cores, 4.9 GiB RAM
20/03/21 01:39:08 INFO Worker: Running Spark version 3.0.0-preview2
20/03/21 01:39:08 INFO Worker: Spark home: .
20/03/21 01:39:08 INFO ResourceUtils: =====
20/03/21 01:39:08 INFO ResourceUtils: Resources for spark.worker:

20/03/21 01:39:08 INFO ResourceUtils: =====
20/03/21 01:39:08 INFO Utils: Successfully started service 'WorkerUI' on port 8081.
20/03/21 01:39:08 INFO WorkerWebUI: Bound WorkerWebUI to 0.0.0.0, and started at http://DESKTOP-D4RLATK:8081
20/03/21 01:39:08 INFO Worker: Connecting to master 192.168.56.1:7077...
20/03/21 01:39:08 INFO TransportClientFactory: Successfully created connection to /192.168.56.1:7077 after 180 ms (0 ms
spent in bootstraps)
20/03/21 01:39:09 INFO Worker: Successfully registered with master spark://192.168.56.1:7077
```

Si on consulte de nouveau notre serveur localhost alors on peut remarquer qu'il existe un worker avec un status ALIVE et un port spéciale pour lui en plus un espace pour lui de 4Go , mais il n'existe pas une application qui est entrain de s'exécuter.

Spark Master at spark://192.168.56.1:7077

URL: spark://192.168.56.1:7077
Alive Workers: 1
Cores in use: 4 Total: 0 Used
Memory in use: 4.9 GiB Total: 0.0 B Used
Resources in use:
Applications: 0 Running, 0 Completed
Drivers: 0 Running, 0 Completed
Status: ALIVE

Workers (1)

Worker ID	Address	State	Cores	Memory	Resources
worker-20200321013907-192.168.56.1-51138	192.168.56.1:51138	ALIVE	4 (0 Used)	4.9 GiB (0.0 B Used)	

Running Applications (0)

Application ID	Name	Cores	Memory per Executor	Resources Per Executor	Submitted Time	User	State	Duration
----------------	------	-------	---------------------	------------------------	----------------	------	-------	----------

Completed Applications (0)

Application ID	Name	Cores	Memory per Executor	Resources Per Executor	Submitted Time	User	State	Duration
----------------	------	-------	---------------------	------------------------	----------------	------	-------	----------

Pour exécuter une application, on doit ouvrir Spark et commencer le travail de traitement des données avec Eclipse ou scala.

Pour exécuter une application dans notre nœud qui est local (dans notre cas on a pris notre machine locale comme un master et worker) veiller voir les tutoriels youtube sur le principe de travail de spark pour comprendre la relation qui existe entre le master et les workers et la nécessité de l'utilisation de Hadoop.

En résumé

Spark exécute la totalité des opérations d'analyse de données en mémoire et en temps réel. Il s'appuie sur des disques seulement lorsque sa mémoire n'est plus suffisante. À l'inverse, avec Hadoop, les données sont écrites sur le disque après chacune des opérations. Ce travail en mémoire permet de réduire les temps de latence entre les traitements, ce qui explique une telle rapidité.

Cependant, Spark ne dispose pas de système de gestion de fichier qui lui est propre. Il est nécessaire de lui en fournir un, par exemple Hadoop Distributed File System, Informix, Cassandra, Il est conseillé de l'utiliser avec Hadoop qui reste actuellement la meilleure solution globale de stockage grâce à ses outils d'administration.

Voici la commande pour lancer l'application :

```
Invite de commandes - spark-shell --master spark://192.168.56.1:7077
C:\Users\HP>cd ..
C:\Users>cd ..
C:\>cd spark
C:\spark>spark-shell --master spark://192.168.56.1:7077
20/03/21 01:48:25 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
Spark context Web UI available at http://DESKTOP-D4RLATK:4040
Spark context available as 'sc' (master = spark://192.168.56.1:7077, app id = app-20200321014839-0000).
Spark session available as 'spark'
Welcome to

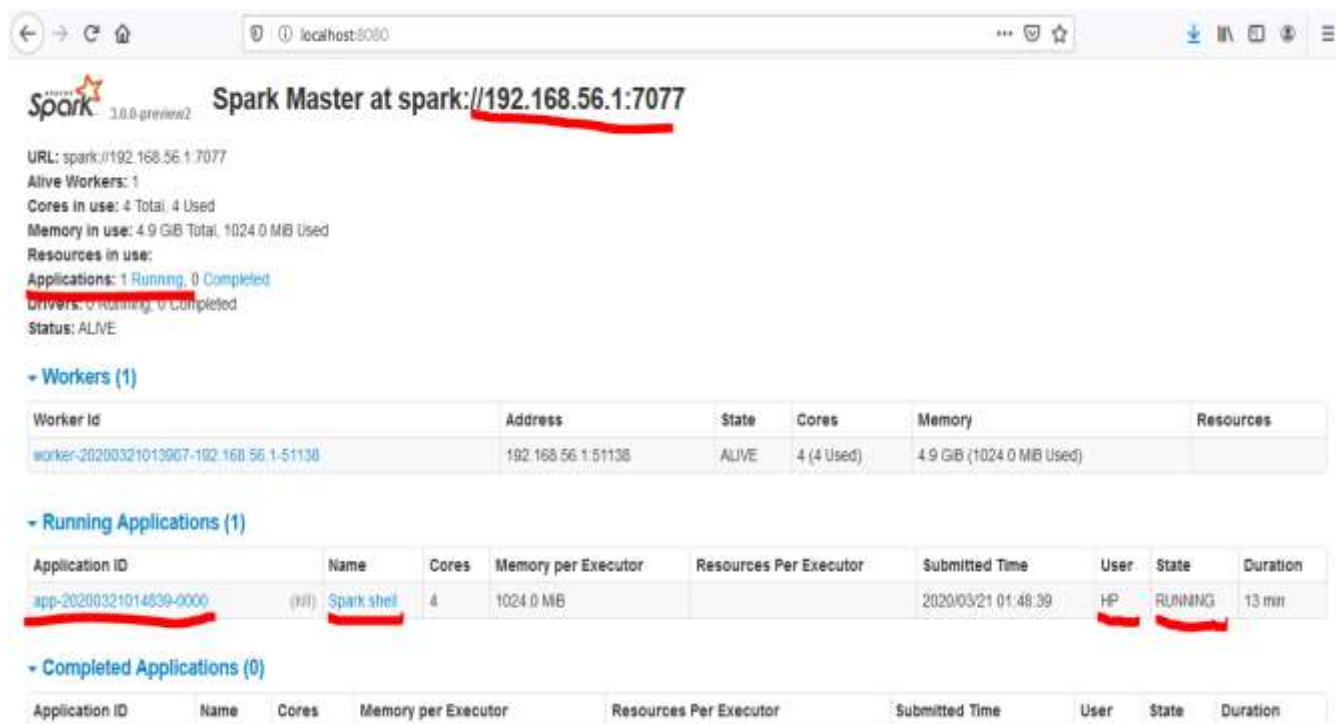
  ____ _
 / ___ \| | | |
/ /   \| |_| |
\ \   /| | | |
 \___/\_| |_|_|_|

 version 3.0.0-preview2

Using Scala version 2.12.10 (Java HotSpot(TM) Client VM, Java 1.8.0_241)
Type in expressions to have them evaluated.
Type :help for more information.

scala> 20/03/21 01:48:50 WARN ProcfsMetricsGetter: Exception when trying to compute pagesize, as a result reporting of ProcessTree metrics is stopped
```


En visitant notre serveur à nouveau, on peut remarquer qu'il y'a une application qui est entrain de s'exécuter



The screenshot shows the Spark Master web interface at localhost:8080. The title is "Spark Master at spark://192.168.56.1:7077". The status bar indicates: URL: spark://192.168.56.1:7077, Alive Workers: 1, Cores in use: 4 Total, 4 Used, Memory in use: 4.9 GiB Total, 1024.0 MiB Used, Resources in use: Applications: 1 Running, 0 Completed, Drivers: 0 Running, 0 Completed, Status: ALIVE.

Workers (1)

Worker Id	Address	State	Cores	Memory	Resources
worker-20200321013907-192.168.56.1-51138	192.168.56.1:51138	ALIVE	4 (4 Used)	4.9 GiB (1024.0 MiB Used)	

Running Applications (1)

Application ID	Name	Cores	Memory per Executor	Resources Per Executor	Submitted Time	User	State	Duration
app-20200321014639-0000	(Nil) Spark shell	4	1024.0 MiB		2020/03/21 01:48:39	HP	RUNNING	13 min

Completed Applications (0)

Application ID	Name	Cores	Memory per Executor	Resources Per Executor	Submitted Time	User	State	Duration
----------------	------	-------	---------------------	------------------------	----------------	------	-------	----------