Architecture Big Data

Année : 2020/2021

Professor : Mr KALLOUBI

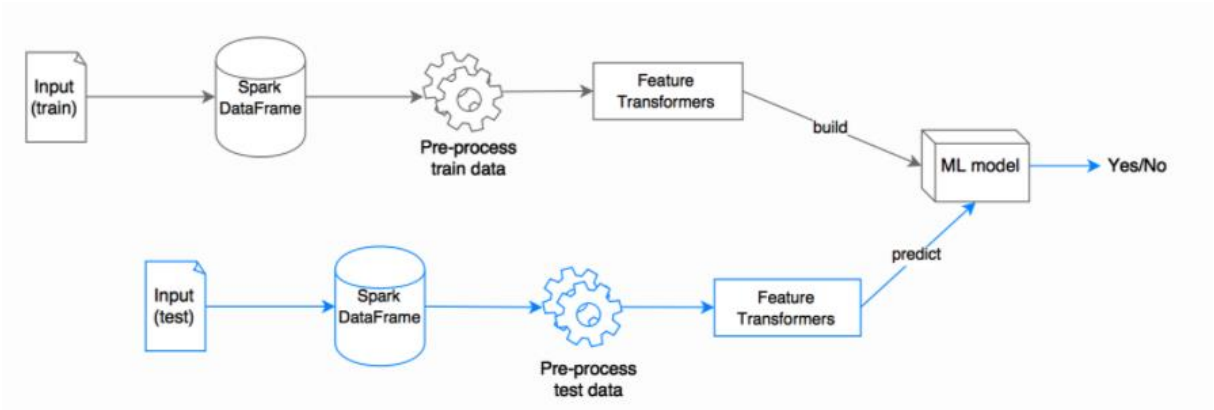
Prepared by : CHARHABIL SANAA

Project Scala & PYTHON ML

1. Concepts et Définitions:

Principaux concepts dans les pipelines :

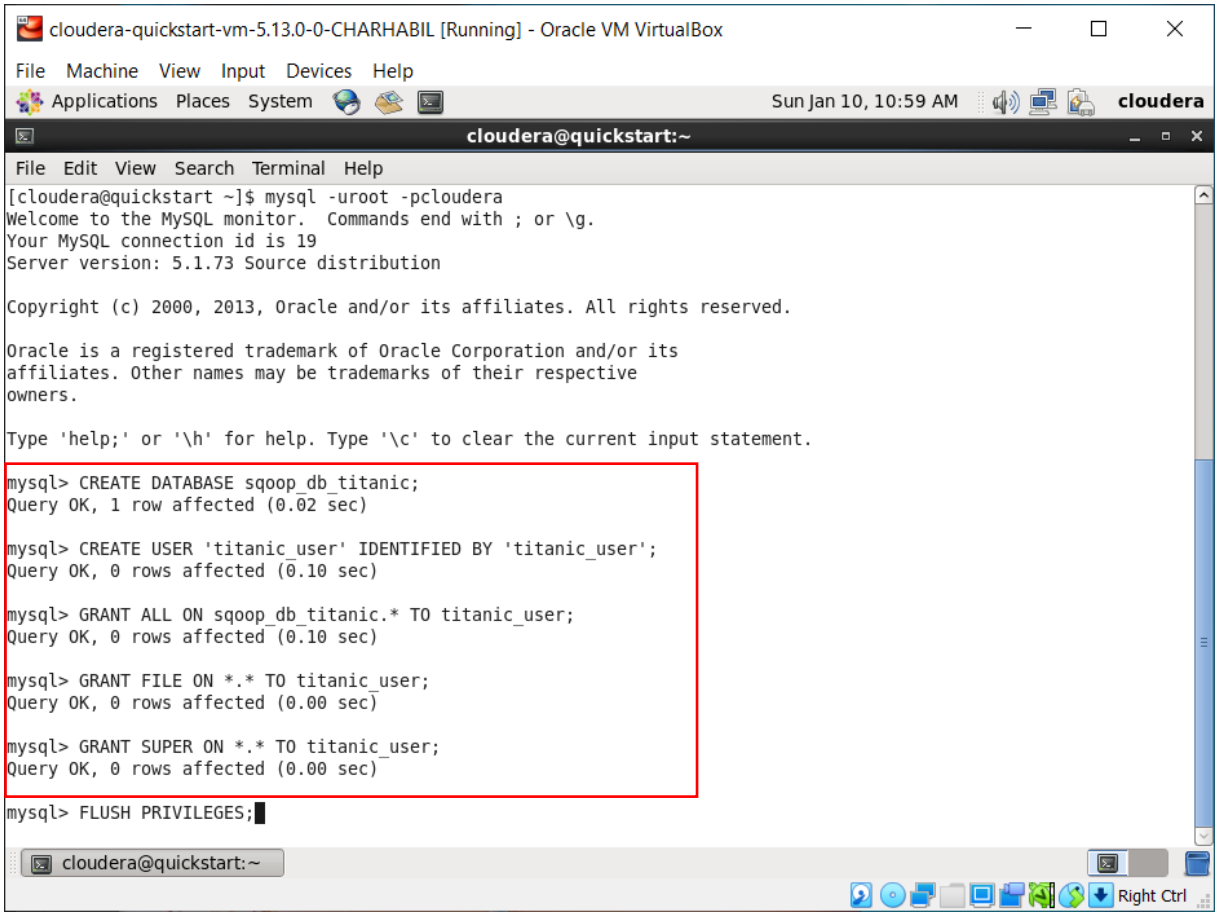
- **MLlib :** standardise les API pour les algorithmes d'apprentissage automatique afin de faciliter la combinaison de plusieurs algorithmes en un seul pipeline ou flux de travail. Cette section couvre les concepts clés introduits par l'API Pipelines, où le concept de pipeline est principalement inspiré du projet scikit-learn.
- **DataFrame:** cette API ML utilise DataFrame de Spark SQL en tant qu'ensemble de données ML, qui peut contenir une variété de types de données. Par exemple, un DataFrame peut avoir différentes colonnes stockant du texte, des vecteurs de caractéristiques, de vraies étiquettes et des prédictions.
- **Transformer:** Un Transformer est un algorithme qui peut transformer un DataFrame en un autre DataFrame. Par exemple, un modèle ML est un Transformer qui transforme un DataFrame avec des fonctionnalités en un DataFrame avec des prédictions.
- **Estimator:** Un Estimator est un algorithme qui peut être ajusté sur un DataFrame pour produire un Transformer. Par exemple, un algorithme d'apprentissage est un Estimator qui s'entraîne sur un DataFrame et produit un modèle.
- **Pipeline:** un pipeline enchaîne plusieurs transformateurs et estimateurs pour spécifier un flux de travail ML.
- **Paramètre:** tous les transformateurs et estimateurs partagent désormais une API commune pour la spécification des paramètres.



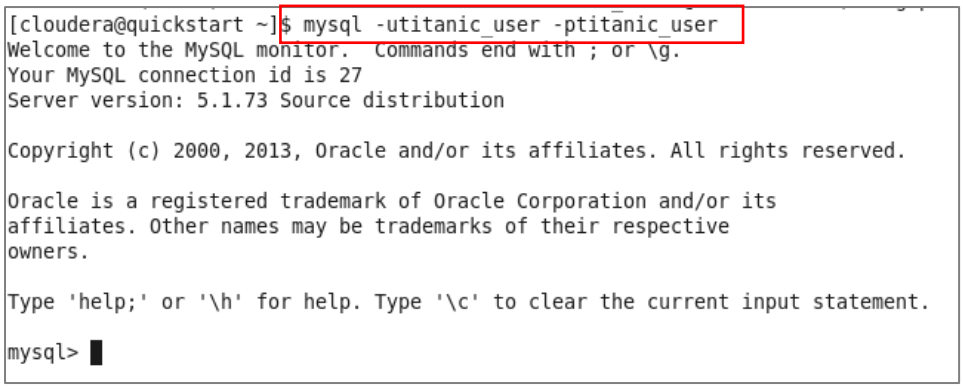
2. Création et insertion du Dataset au niveau de Spark:

1. Charger les données MySQL:

Connexion en tant que root en MySQL et création d'un nouveau utilisateur:



Connexion de notre nouveau utilisateur :



Création de la table titanic et importation des données :

```
mysql> create table if not exists titanic ( PassengerId integer primary key, Survived integer, Pclass integer, Nam
e varchar(225), Sex varchar(225), SibSp integer, Parch integer, Ticket varchar(225), Fare double, Cabin varchar(22
5), Embarked varchar(225) );
Query OK, 0 rows affected (0.11 sec)

mysql> LOAD DATA INFILE '/home/cloudera/Desktop/train.csv' INTO TABLE titanic
    -> FIELDS TERMINATED BY ',' ENCLOSED BY '"'
    -> LINES TERMINATED BY '\n';
Query OK, 892 rows affected, 1305 warnings (0.03 sec)
Records: 892  Deleted: 0  Skipped: 0  Warnings: 1123
```

Sélection des 20 premières lignes de notre table :

```
mysql> select * from titanic limit 20;
+-------------+----------+--------+-----------------------------------------------------+--------+-------+----
---+--------+--------+---------+----------+
| PassengerId | Survived | Pclass | Name                                                | Sex    | SibSp | Par
ch | Ticket | Fare   | Cabin   | Embarked |
+-------------+----------+--------+-----------------------------------------------------+--------+-------+----
---+--------+--------+---------+----------+
|           0 |        0 |      0 | Name                                                | Sex    |     0 |
 0 | Parch  |      0 | Fare    | Cabin    |
|           1 |        0 |      3 | Braund, Mr. Owen Harris                             | male   |    22 |
 1 | 0      |      0 | 7.25    |          |
|           2 |        1 |      1 | Cumings, Mrs. John Bradley (Florence Briggs Thayer) | female |    38 |
 1 | 0      |      0 | 71.2833 | C85      |
|           3 |        1 |      3 | Heikkinen, Miss. Laina                              | female |    26 |
 0 | 0      |      0 | 7.925   |          |
|           4 |        1 |      1 | Futrelle, Mrs. Jacques Heath (Lily May Peel)        | female |    35 |
 1 | 0      | 113803 | 53.1    | C123     |
|           5 |        0 |      3 | Allen, Mr. William Henry                            | male   |    35 |
 0 | 0      | 373450 | 8.05    |          |
|           6 |        0 |      3 | Moran, Mr. James                                    | male   |     0 |
 0 | 0      | 330877 | 8.4583  |          |
|           7 |        0 |      1 | McCarthy, Mr. Timothy J                             | male   |    54 |
 0 | 0      |  17463 | 51.8625 | E46      |
|           8 |        0 |      3 | Palsson, Master. Gosta Leonard                      | male   |     2 |
 3 | 1      | 349909 | 21.075  |          |
```

2. Charger les données SQOOP:

Afficher les bases de données avec **sqoop list-databases** :

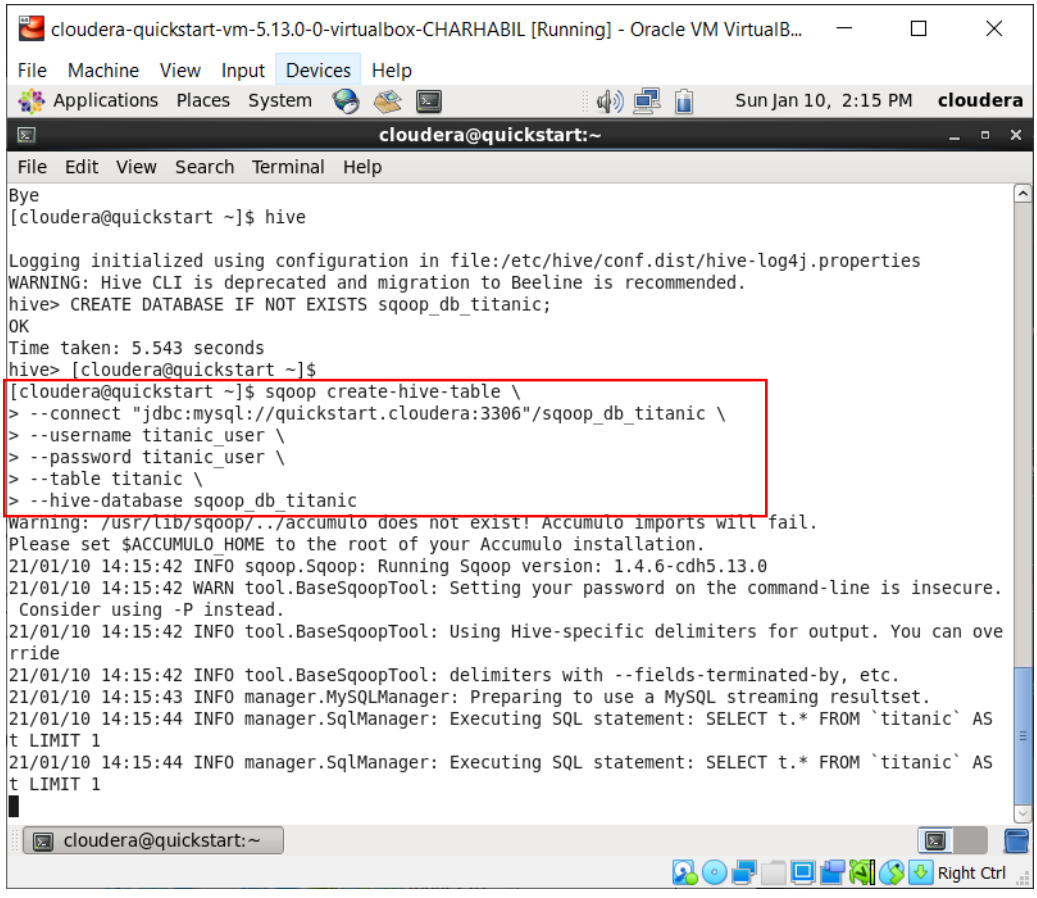
```
[cloudera@quickstart ~]$ sqoop list-databases \
> --connect "jdbc:mysql://quickstart.cloudera:3306" \
> --username titanic_user \
> --password titanic_user
Warning: /usr/lib/sqoop/../accumulo does not exist! Accumulo imports will fail.
Please set $ACCUMULO_HOME to the root of your Accumulo installation.
21/01/10 12:23:34 INFO sqoop.Sqoop: Running Sqoop version: 1.4.6-cdh5.13.0
21/01/10 12:23:34 WARN tool.BaseSqoopTool: Setting your password on the command-line is insecure. Consider using -
P instead.
21/01/10 12:23:35 INFO manager.MySQLManager: Preparing to use a MySQL streaming resultset.
information_schema
sqoop_db_titanic
[cloudera@quickstart ~]$
```

Afficher les tables avec **sqoop list-tables** :

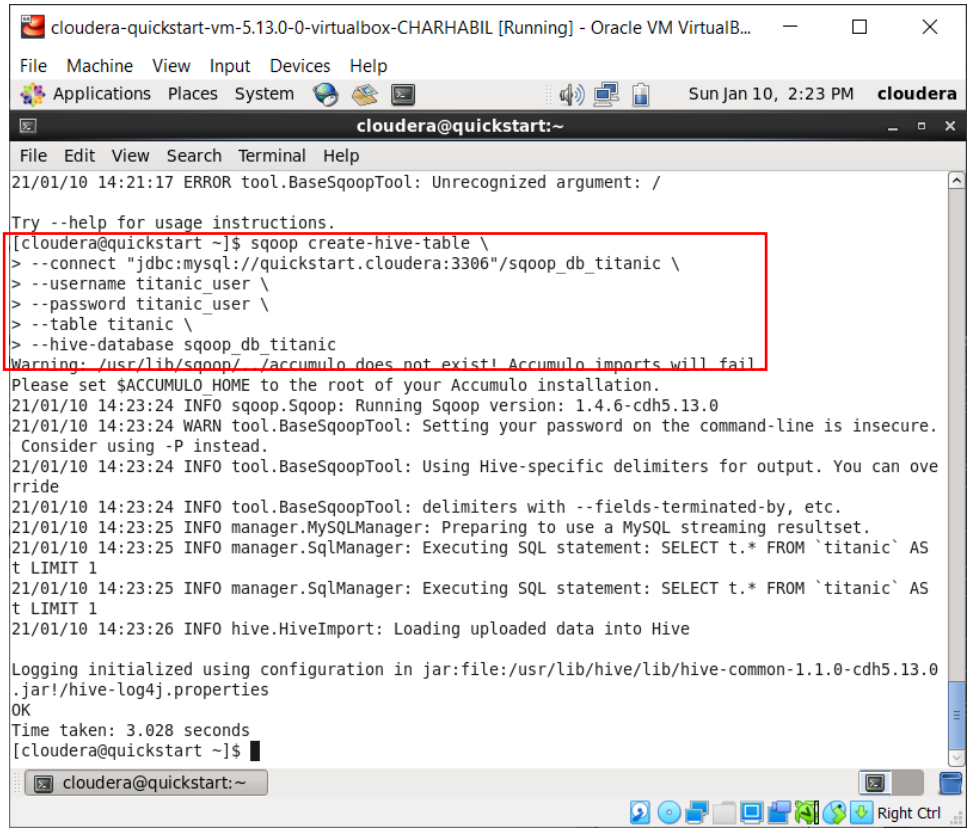
```
[cloudera@quickstart ~]$ sqoop list-tables \
> --connect "jdbc:mysql://quickstart.cloudera:3306"/sqoop_db_titanic \
> --username titanic_user \
> --password titanic_user
Warning: /usr/lib/sqoop/../accumulo does not exist! Accumulo imports will fail.
Please set $ACCUMULO_HOME to the root of your Accumulo installation.
21/01/10 12:29:39 INFO sqoop.Sqoop: Running Sqoop version: 1.4.6-cdh5.13.0
21/01/10 12:29:39 WARN tool.BaseSqoopTool: Setting your password on the command-line is insecure. Consider using -
P instead.
21/01/10 12:29:40 INFO manager.MySQLManager: Preparing to use a MySQL streaming resultset.
passengers
titanic
[cloudera@quickstart ~]$
```

3. Charger les données dans HIVE:

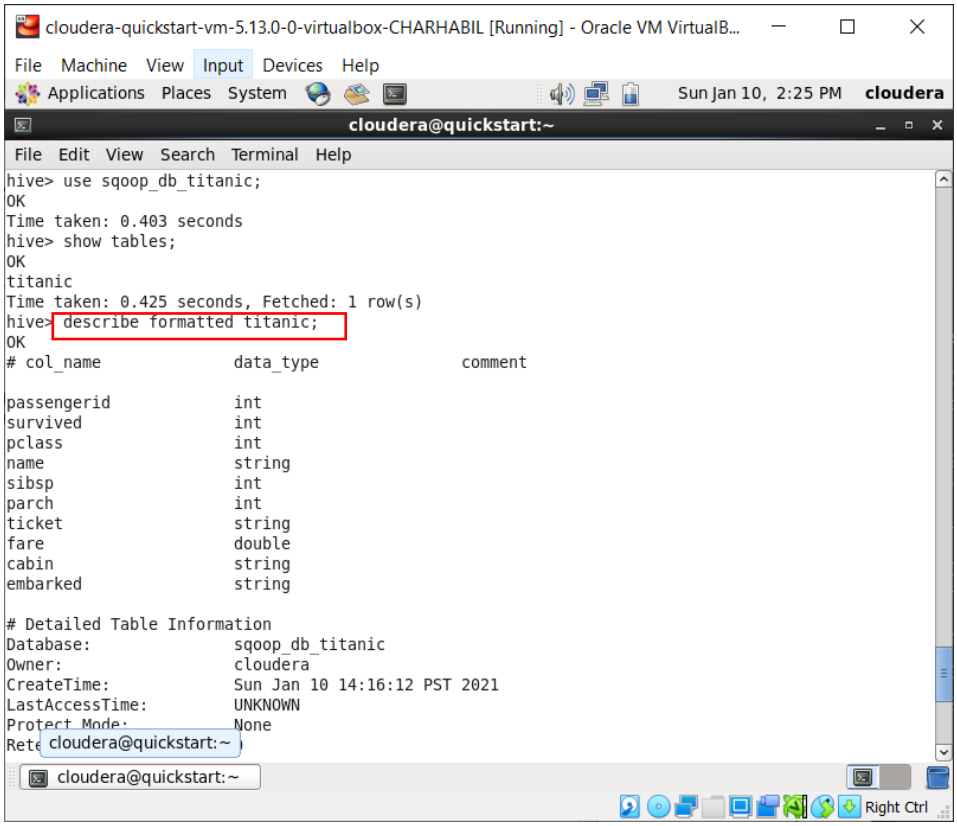
Création de la base de données dans Hive :



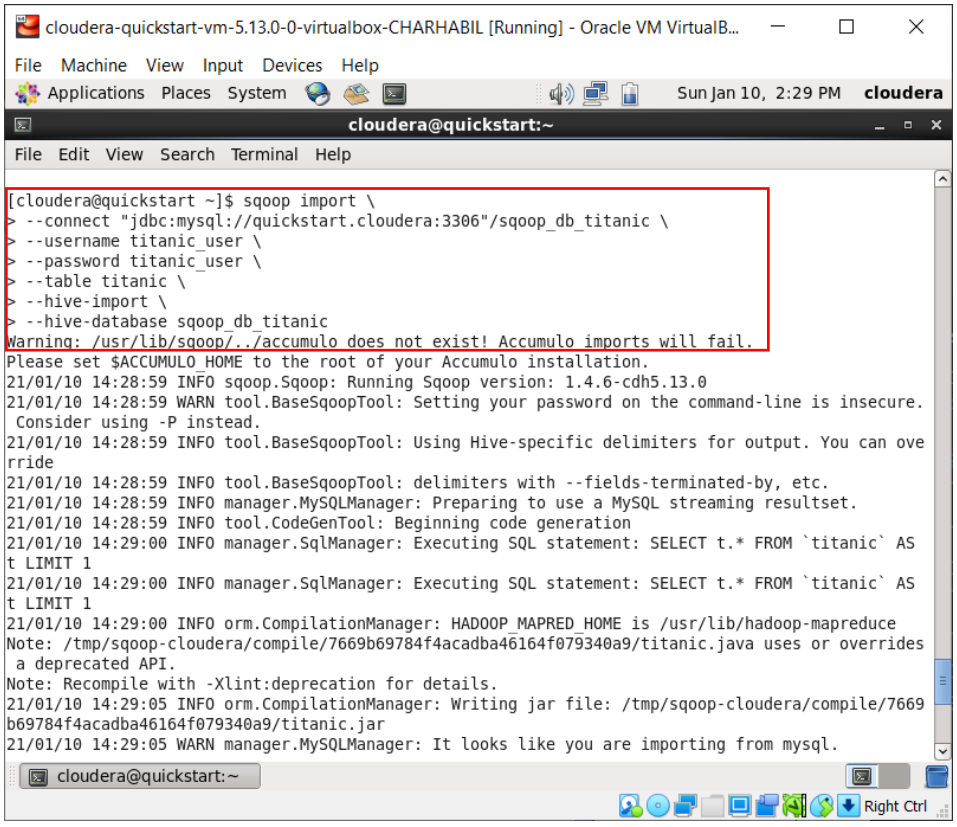
Création de la table titanic :

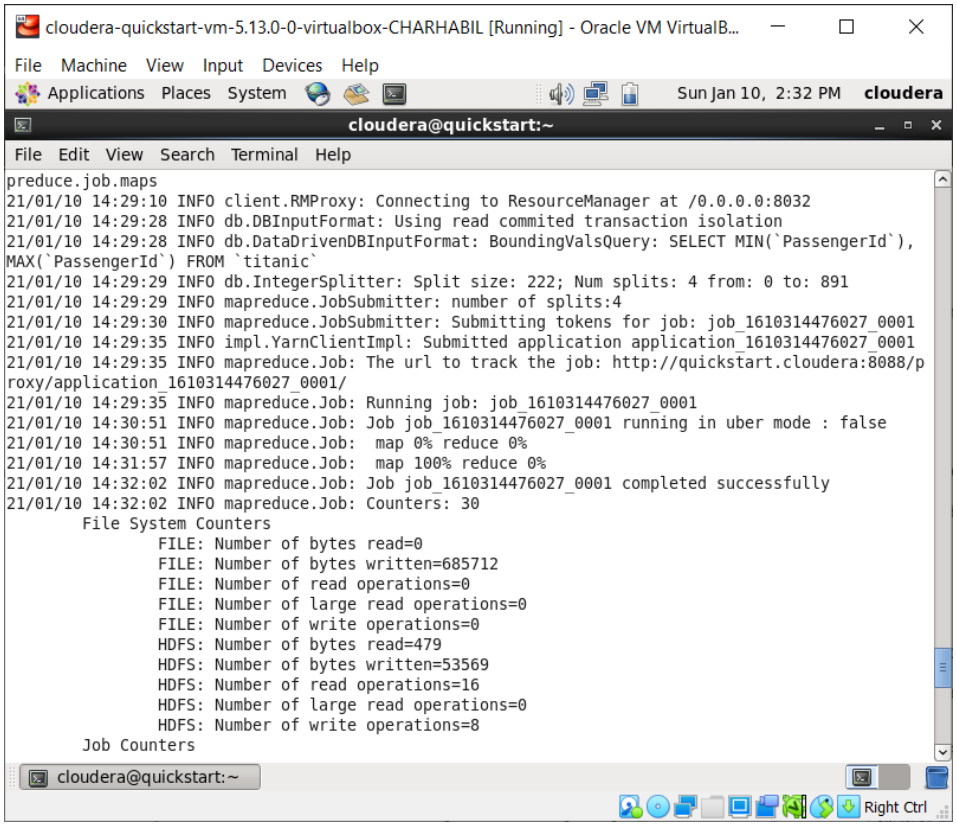


Vérification :

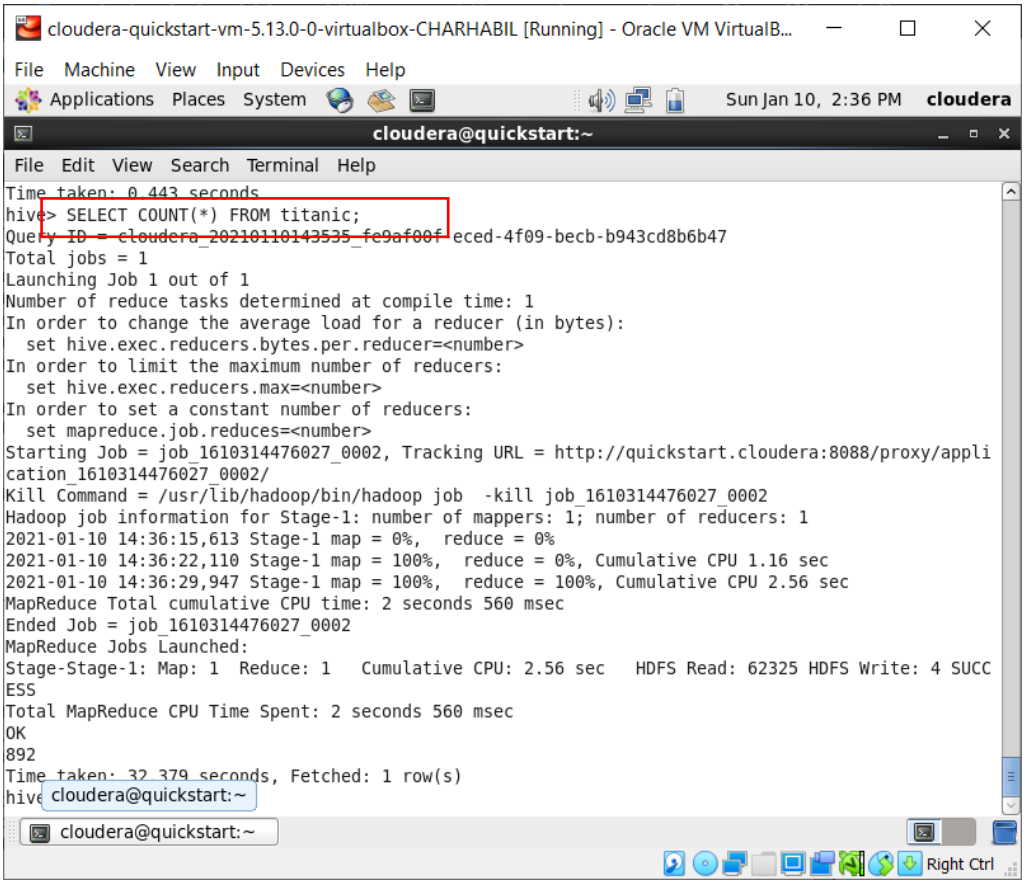


Importation des données et remplissage de table :





Compter le nombre d'enregistrements :



3. Construction du JAR - En SCALA:

1. Définition du Dataset:

J'ai choisi le csv Titanic qui contient un ensemble de détails sur les passagers tels que le nom, le sexe, le tarif, la cabine, etc. et si la personne a survécu à la catastrophe du Titanic. Sur cette base, nous devons construire un modèle qui peut prédire, étant donné un autre passager, s'il est susceptible de survivre. Ceci est un exemple de classification binaire où il n'y a que deux classes possibles (1 si le passager survit et 0 sinon).

PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
1	0	3	Braund, Mr. C	male	22	1	0	A/5 21171	7.25		S
2	1	1	Cumings, Mrs.	female	38	1	0	PC 17599	71.2833	C85	C
3	1	3	Heikkinen, Mi	female	26	0	0	STON/O2. 31(	7.925		S
4	1	1	Futrelle, Mrs.	female	35	1	0	113803	53.1	C123	S
5	0	3	Allen, Mr. Will	male	35	0	0	373450	8.05		S
6	0	3	Moran, Mr. Ja	male		0	0	330877	8.4583		Q
7	0	1	McCarthy, Mr	male	54	0	0	17463	51.8625	E46	S
8	0	3	Palsson, Mast	male	2	3	1	349909	21.075		S
9	1	3	Johnson, Mrs.	female	27	0	2	347742	11.1333		S
10	1	2	Nasser, Mrs. N	female	14	1	0	237736	30.0708		C
11	1	3	Sandstrom, M	female	4	1	1	PP 9549	16.7	G6	S
12	1	1	Bonnell, Miss.	female	58	0	0	113783	26.55	C103	S
13	0	3	Saundercock,	male	20	0	0	A/5. 2151	8.05		S
14	0	3	Andersson, M	male	39	1	5	347082	31.275		S
15	0	3	Vestrom, Miss	female	14	0	0	350406	7.8542		S
16	1	2	Hewlett, Mrs.	female	55	0	0	248706	16		S
17	0	3	Rice, Master.	male	2	4	1	382652	29.125		Q
18	1	2	Williams, Mr.	male		0	0	244373	13		S
19	0	3	Vander Planke	female	31	1	0	345763	18		S
20	1	3	Masselmani, N	female		0	0	2649	7.225		C
21	0	2	Fynney, Mr. Jo	male	35	0	0	239865	26		S
22	1	2	Beesley, Mr. L	male	34	0	0	248698	13	D56	S

Voici les étapes qu'on va appliquer sur notre Dataset afin de conclure les prédictions :

- La première étape lorsque vous essayez de créer un modèle d'apprentissage automatique consiste à analyser et à comprendre les données dont vous disposez. Pour que vous puissiez décider quelles fonctionnalités doivent être utilisées pour construire le modèle, si les fonctionnalités sont numériques ou catégoriques, quelle est la moyenne, maximum ou minimum de vos fonctionnalités numériques, etc.
- Une fois les données analysées, l'étape suivante est la sélection des fonctionnalités où nous décidons quelles fonctionnalités sont pertinentes pour la construction du modèle
- Vient ensuite le prétraitement des données. La plupart du temps, les données d'entrée que vous recevez pour la modélisation ne seront pas de bonnes données. Au cours de cette étape, par exemple, nous pouvons décider quoi faire avec les valeurs manquantes - supprimer les lignes ayant des valeurs nulles, ou remplir celles avec la valeur moyenne de la fonctionnalité (si la fonctionnalité est numérique), ou remplir avec la valeur la plus fréquente de la caractéristique (si la caractéristique est catégorielle) etc.
- Vient ensuite l'étape de l'ingénierie des fonctionnalités et de la transformation des fonctionnalités. Dans l'ingénierie des fonctionnalités, nous dérivons de nouvelles fonctionnalités à partir de celles existantes et pendant la transformation des fonctionnalités, nous transformons les fonctionnalités existantes afin qu'elles puissent être utilisées pour créer le modèle.
- Enfin, nous construisons le modèle en utilisant les fonctionnalités sélectionnées et faisons des prédictions sur un nouvel ensemble de données.

Nous implémenterons toutes les étapes ci-dessus à l'aide de Spark et Scala et créerons un pipeline d'apprentissage automatique - le flux global peut être illustré par le diagramme ci-dessous. La section grise du diagramme montre le flux de construction du modèle et la section bleue du diagramme montre le flux pour faire des prédictions.

Importation des librairies :

Importation des librairies :

```
import org.apache.spark.{SparkConf, SparkContext}
import org.apache.spark.ml.Pipeline
import org.apache.spark.ml.evaluation.RegressionEvaluator
import org.apache.spark.ml.feature.VectorAssembler
import org.apache.spark.ml.classification.{LogisticRegression, RandomForestClassifier, GBTClassifier,DecisionTreeClassifier, NaiveBayes}
import org.apache.spark.sql.functions._
import org.apache.spark.ml.classification.{RandomForestClassificationModel, RandomForestClassifier}
import org.apache.spark.ml.evaluation.MulticlassClassificationEvaluator
import org.apache.spark.ml.feature.{IndexToString, StringIndexer, VectorIndexer}
```

2. Configuration et importation des données:

Configuration de l'application Spark et sélection des données à manipuler :

```
object code_pipeline_titanic extends App{


  println(" Welcome PASSENGER ! ")

  //________PART1 :configuration fro a spark application :_____________________________________

  val conf = new SparkConf().setAppName("SparkHiveTitanic").setMaster("yarn-client")
  val sc = new SparkContext(conf)
  val sqlContext = new org.apache.spark.sql.hive.HiveContext(sc)

  //Create dataframe and select data from database
  val titanic_df = sqlContext.sql("select * from sqoop_db_titanic.titanic")
```

Visualisation des données :

Nous allons maintenant explorer le DataFrame chargé pour mieux comprendre les données. On peut vérifier le schéma des données chargées par exemple en appelant **printShema()** :

```
  //_______PART2 : EXPLORE DATAFRAMME_____________________________________

  //checking shema for our dataset:
  titanic_df.printSchema()
  //Display data:
  titanic_df.show()
  //show values of colomne Fare;
  titanic_df.describe("Fare").show()
```

EDA - exploratory data analysis :

L'analyse de données exploratoires est une approche d'analyse des ensembles de données pour résumer leurs principales caractéristiques, souvent avec des méthodes visuelles. Un modèle statistique peut être utilisé ou non, mais l'EDA sert principalement à voir ce que les données peuvent nous dire au-delà de la tâche de modélisation formelle ou de test d'hypothèse.

```
//EDA - exploratory data analysis:

//Knowing the number of Passengers Survived ?
titanic_df.groupBy("Survived").count().show()
gropuBy_output = titanic_df.groupBy("Survived").count()
display(gropuBy_output)

//Checking survival rate using feature Sex:
titanic_df.groupBy("Sex","Survived").count().show()
//Checking survival rate using feature Pclass:
titanic_df.groupBy("Pclass","Survived").count().show()
```

3. Pré-traitement :

Nettoyage des données-Remplacement des données manquantes :

En analysant les données, vous pouvez y voir quelques irrégularités. Par exemple, il manque quelques valeurs dans la colonne Age. De même, il existe des valeurs nulles / manquantes dans Cabin, Fare et Embarked. Il existe plusieurs techniques pour remplir les valeurs manquantes. Vous pouvez :

1. **Ignorez / supprimez les lignes ayant des valeurs manquantes. Cela peut être fait dans Spark en appelant :**

```
//fill missing values:
var titanic_df_removed = titanic_df.na.drop("any")

var avgAge = titanic_df.select(mean("Age")).first()(0).asInstanceOf[Double]
titanic_df = titanic_df.na.fill(avgAge, Seq("Age"))

var titanic_df_embarked_filled = titanic_df.na.fill("S", Seq("Embarked"))
```

2. **Si la colonne est numérique, remplissez la valeur manquante avec la valeur moyenne / moyenne de la colonne. Nous allons remplacer les valeurs manquantes dans la colonne Age en utilisant cette méthode :**

```
//fill missing values:
var titanic_df_removed = titanic_df.na.drop("any")

var avgAge = titanic_df.select(mean("Age")).first()(0).asInstanceOf[Double]
titanic_df = titanic_df.na.fill(avgAge, Seq("Age"))

var titanic_df_embarked_filled = titanic_df.na.fill("S", Seq("Embarked"))
```

3. **Si la colonne est catégorielle, remplissez avec la catégorie la plus courante :**

```
//fill missing values:
var titanic_df_removed = titanic_df.na.drop("any")

var avgAge = titanic_df.select(mean("Age")).first()(0).asInstanceOf[Double]
titanic_df = titanic_df.na.fill(avgAge, Seq("Age"))

var titanic_df_embarked_filled = titanic_df.na.fill("S", Seq("Embarked"))
```

Créez un modèle d'apprentissage automatique qui peut prédire ces valeurs manquantes.

Précision des Features :

Dans de nombreux cas, il y aura des fonctionnalités dans vos données d'entrée qui peuvent être utilisées pour dériver de nouvelles fonctionnalités qui aideront à créer un meilleur modèle. Cela s'appelle également l'ingénierie des fonctionnalités. Par exemple, si vous regardez de plus près la colonne «Nom», vous pouvez voir que le format est FirstName Title. Nom de famille. Nous n'avons pas pu faire de prédiction en fonction du nom du passager, mais il se peut qu'il y ait une relation entre le titre et la survie du passager. Donc, nous allons extraire le titre de chaque nom et former une nouvelle colonne / fonctionnalité. L'udf findTitle est utilisé pour extraire le titre d'une chaîne donnée.

```
val findTitle = sqlContext.udf.register("findTitle", (name: String) => {
    val pattern = "(Dr|Mrs?|Ms|Miss|Master|Rev|Capt|Mlle|Col|Major|Sir|Lady|Mme|Don)\\.".r
    val matchedStr = pattern.findFirstIn(name)
    var title = matchedStr match {
      case Some(s) => matchedStr.getOrElse("Other.")
      case None => "Other."
    }
    if (title.equals("Don.") || title.equals("Major.") || title.equals("Capt."))
      title = "Sir."
    if (title.equals("Mlle.") || title.equals("Mme."))
        title = "Miss."
    title
})
```

DataFrame fournit une méthode **withColumn()** qui peut être utilisée pour ajouter / remplacer une colonne existante. Il prend deux paramètres - le nom de la nouvelle colonne et une colonne du DataFrame actuel. c'est-à-dire si vous appelez :

```
var temp  = titanic_df.withColumn("test",titanic_df("PassengerId"))
//^will create a new column named test with same values as in the column PassengerId.
//we can also modify the value of the new column. e.g.,
temp = titanic_df.withColumn("test",titanic_df("PassengerId")-1)
temp.select("PassengerId","test").show(3)
```

Nous allons maintenant appliquer la fonction **findTitle()** sur la colonne Nom pour extraire le titre et créer une nouvelle colonne - Titre.

```
titanic_df = titanic_df.withColumn("Title", findTitle(titanic_df("Name")))
titanic_df.select("Name","Title").show()
```

De même, nous définirons 3 autres udfs, à l'aide desquels nous générerons de nouvelles fonctionnalités.

```
//Categorize a passenger as child if his/her age is less than 15
//(more chances of survival)
val addChild = sqlContext.udf.register("addChild", (sex: String, age: Double) => {
  if (age < 15)
    "Child"
  else
      sex

})

//withFamily is true(1) if the family size excluding self is > 3
//(large family may have more/less chance of survival)
val withFamily = sqlContext.udf.register("withFamily", (sib: Int, par: Int) => {
    if (sib + par > 3)
      1.0
    else
      0.0

})
//for converting integer columns to double. Requires since few of the
//columns of our DataFrame are of Int type.
val toDouble = sqlContext.udf.register("toDouble", ((n: Int) => { n.toDouble }))

//apply the udfs
train_data = train_data.withColumn("Sex", addChild(train_data("Sex"), train_data("Age")))
train_data = train_data.withColumn("Pclass", toDouble(train_data("Pclass")))
train_data = train_data.withColumn("Family", withFamily(train_data("SibSp"), train_data("Parch")))
train_data = train_data.withColumn("Survived", toDouble(train_data("Survived")))
```

4. Composants du pipeline :

Le pipeline ML aura une séquence de composants de pipeline. Il existe deux types de composants: les transformateurs et les estimateurs. Transformers transforme le Dataframe d'entrée en un nouveau DataFrame à l'aide de la méthode **transform().** Un Estimator ajuste d'abord un modèle aux données, à l'aide de la méthode **fit(),** puis transforme. Celles-ci seront plus claires une fois que vous aurez parcouru les composants ci-dessous.

StringIndexer :

Pour créer un modèle dans Spark, les fonctionnalités doivent être du type Double mais nous avons quelques fonctionnalités qui sont du type String. Spark fournit un Feature Transformer - **StringIndexer** qui peut être utilisé pour cette transformation.

```
val titleInd = new StringIndexer().setInputCol("Title").setOutputCol("TitleIndex")
var strIndModel = titleInd.fit(train_data)
strIndModel.transform(train_data).select("Title","TitleIndex").show(5)

val sexInd = new StringIndexer().setInputCol("Sex").setOutputCol("SexIndex")
```

Ici, StringIndexer est un Estimator qui transforme la colonne Title, génère des indices pour les mots et crée une nouvelle colonne nommée TitleIndex. La méthode Fit de StringIndexer convertit la colonne en **StringType** (si ce n'est pas StringType), puis compte l'occurrence de chaque mot. Il trie ensuite ces mots par ordre décroissant de leur fréquence et attribue un index à chaque mot. La méthode **StringIndexer.fit** () renvoie un **StringIndexerModel** qui est un Transformer.

```
val titleInd = new StringIndexer().setInputCol("Title").setOutputCol("TitleIndex")
var strIndModel = titleInd.fit(train_data)
strIndModel.transform(train_data).select("Title","TitleIndex").show(5)

val sexInd = new StringIndexer().setInputCol("Sex").setOutputCol("SexIndex")
```

M. est le mot le plus fréquent dans ces données, il reçoit donc l'index 0. De même, nous allons également créer un indexeur pour la fonctionnalité – Sexe

```
val titleInd = new StringIndexer().setInputCol("Title").setOutputCol("TitleIndex")
var strIndModel = titleInd.fit(train_data)
strIndModel.transform(train_data).select("Title","TitleIndex").show(5)

val sexInd = new StringIndexer().setInputCol("Sex").setOutputCol("SexIndex")
```

5. Rangement / regroupement :

Pendant le Binning / Bukceting, une colonne avec des valeurs continues est convertie en buckets. Nous définissons la valeur de début et de fin de chaque bucket lors de la création du Bucketizer - qui est un Transformer. Nous allons regrouper la colonne "Tarif".

```
//binning/Bucketing:
//define the buckets/splits
val fareSplits = Array(0.0,10.0,20.0,30.0,40.0,Double.PositiveInfinity)
val fareBucketize = new Bucketizer().setInputCol("Fare").setOutputCol("FareBucketed").setSplits(fareSplits)
fareBucketize.transform(train_data).select("Fare","FareBucketed").show(10)
```

Assembleur de vecteur

VectorAssembler est utilisé pour assembler des entités dans un vecteur. Nous passerons toutes les colonnes que nous allons utiliser pour la prédiction au VectorAssembler et il créera une nouvelle colonne vectorielle.

```
//vector assembler:

val assembler = new VectorAssembler().setInputCols(Array("SexIndex", "Age", "TitleIndex", "Pclass", "Family","FareBucketed")).setOutputCol("features_temp")
```

Normaliseur

Ensuite, nous normaliserons ou normaliserons les données à l'aide du transformateur - Normalizer. Le normalisateur prendra la colonne créée par VectorAssembler, la normalisera et produira une nouvelle colonne.

```
//normalizer:
val normalizer = new Normalizer().setInputCol("features_temp").setOutputCol("features")
```

Training set & Test set :

Pour évaluer le modèle, nous diviserons nos données en deux: ensemble d'apprentissage (80%) et ensemble de test (20%). Nous allons construire notre modèle à l'aide de l'ensemble d'entraînement et l'évaluer à l'aide de l'ensemble de test.

```
//training set and test set
val splits = titanic_df.randomSplit(Array(0.8, 0.2), seed = 11L)
val train = splits(0).cache()
val test = splits(1).cache()
```

6. Création et Application des modèles de machine learning :

Nous allons construire notre modèle en utilisant les algorithmes qui sont utilisés pour la classification. La variable classée est appelée variable dépendante et les autres variables qui déterminent la valeur de la variable dépendante sont appelées variables indépendantes.

- **Régression Logistique :**

Dans la régression logistique, basée sur les valeurs des variables indépendantes, elle prédit la probabilité que la variable dépendante prenne l'une de ses valeurs catégorielles (classes). Dans notre exemple, il existe deux classes possibles 0 ou 1.

```
//PIPELINE 1 : LOGISTIC REGRESSION:
val lr = new LogisticRegression().setLabelCol("Survived").setFeaturesCol("features")
val pipeline = new Pipeline().setStages(Array(sexInd, titleInd, fareBucketize, assembler, normalizer,lr))
val lrModel = pipeline.fit(train)
```

Evaluation du model :

```
val predictions = lrModel.transform(test)
println("RMSE on test data = " + rmse.evaluate(predictions))
val lrrmse = rmse.evaluate(predictions)
```

- **RANDOM FOREST Classifier:**

Random Forest est un algorithme d'apprentissage automatique robuste qui peut être utilisé pour une variété de tâches, y compris la régression et la classification. Il s'agit d'une méthode d'ensemble, ce qui signifie qu'un modèle de forêt aléatoire est composé d'un grand nombre de petits arbres de décision, appelés estimateurs, qui produisent chacun leurs propres prédictions.

Application du model :

```
//PIPELINE 2 : RANDOM FOREST Classifier:
val rf = new RandomForestClassifier().setLabelCol("Survived").setFeaturesCol("features").setNumTrees(10).setMaxDepth(10).setSeed(1L)
val pipeline2 = new Pipeline().setStages(Array(sexInd, titleInd, fareBucketize, assembler, normalizer,rf))
val rfModel = pipeline2.fit(train)
```

Evaluation du model :

```
val predictions2 = rfModel.transform(test)
println("RMSE on test data = " + rmse.evaluate(predictions2))
val rfrmse = rmse.evaluate(predictions2)
```

- **GBT classifier :**

Gradient-Boosted Trees (GBT) est un algorithme d'apprentissage pour la classification. Il prend en charge les étiquettes binaires, ainsi que les fonctionnalités continues et catégorielles

Application du model :

```
// PIPELINE 3 : GBT Classifier:
val gbt = new GBTClassifier().setLabelCol("Survived").setFeaturesCol("features").setMaxIter(100).setSeed(1L)
val pipeline3 = new Pipeline().setStages(Array(sexInd, titleInd, fareBucketize, assembler, normalizer,gbt))
val gbtModel = pipeline3.fit(train)
```

Evaluation du model :

```
val predictions3 = gbtModel.transform(test)
println("RMSE on test data = " + rmse.evaluate(predictions3))
val gbtrmse = rmse.evaluate(predictions3)
```

- **Decision Tree Classifier :**

L'apprentissage par arbre de décision désigne une méthode basée sur l'utilisation d'un arbre de décision comme modèle prédictif. On l'utilise notamment en fouille de données et en apprentissage automatique.

Application du model :

```
// PIPELINE 4 : Decision Tree Classifier:
val dt = new DecisionTreeClassifier().setLabelCol("Survived").setFeaturesCol("features")
val pipeline4 = new Pipeline().setStages(Array(sexInd, titleInd, fareBucketize, assembler, normalizer,dt))
val dtModel = pipeline4.fit(train)
```

Evaluation du model :

```
val predictions4 = dtModel.transform(test)
println("RMSE on test data = " + rmse.evaluate(predictions4))
val dtrmse = rmse.evaluate(predictions4)
```

- **NAIF BAYES :**

La classification naïve bayésienne est un type de classification bayésienne probabiliste simple basée sur le théorème de Bayes avec une forte indépendance des hypothèses. Elle met en œuvre un classifieur bayésien naïf, ou classifieur naïf de Bayes, appartenant à la famille des classifieurs linéaires.

Application du model :

```
// PIPELINE 5 : NaiveBayes :
val nb = NaiveBayes().setlabelCol("Survived").setFeaturesCol("features")
val pipeline5 = new Pipeline().setStages(Array(sexInd, titleInd, fareBucketize, assembler, normalizer,nb))
val nbModel = pipeline5.fit(train)
```

Evaluation du model :

```
val predictions5 = nbModel.transform(test)
println("RMSE on test data = " + rmse.evaluate(predictions5))
val nbrmse = rmse.evaluate(predictions5)
```

7. Enregistrement des évaluations des modèles dans un nouveau DataFrame :

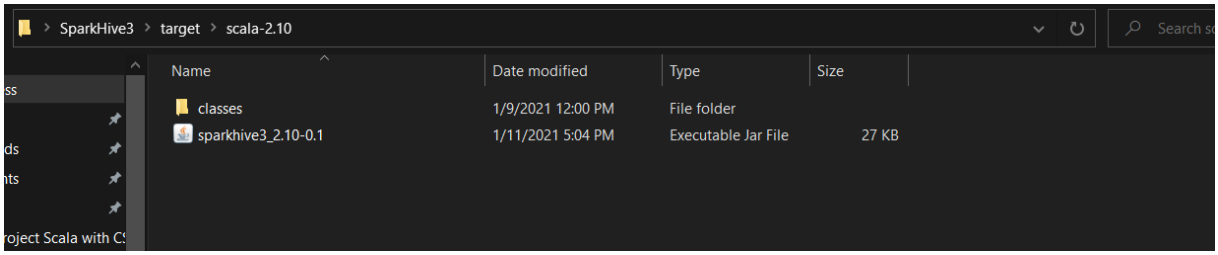
```
//send predictors into a new dataframe :
 var data = sqlContext.createDataFrame(Seq((1,"Logistic Regression",lrrmse),
   (2,"Random Forest classifier ",rfrmse),(3,"Gradient Boost Tree Classifier",gbtrmse),
   (4,"Decision Tree classifier",dtrmse),(5,"Naive Bayes",nbrmse))).toDF("id", "name", "rmse")
 data.write.mode("append").saveAsTable("predictors")
```

Création du jar avec la commande **sbt package** :

```
C:\Users\pc\Desktop\SparkHive3>sbt package
[info] Loading project definition from C:\Users\pc\Desktop\SparkHive3\project
[info] Set current project to SparkHive3 (in build file:/C:/Users/pc/Desktop/SparkHive3/)
[info] Compiling 1 Scala source to C:\Users\pc\Desktop\SparkHive3\target\scala-2.10\classes...
[info] Packaging C:\Users\pc\Desktop\SparkHive3\target\scala-2.10\sparkhive3_2.10-0.1.jar ...
[info] Done packaging.
[success] Total time: 7 s, completed Jan 11, 2021 5:04:13 PM

C:\Users\pc\Desktop\SparkHive3>
```

Vérification de l'emplacement du jar :



1. Code PYTHON:

1. Premiers pas :

Installation de pyspark :

```
(base) C:\Users\pc>pip install pyspark
Collecting pyspark
  Downloading pyspark-3.0.1.tar.gz (204.2 MB)
     |████████████████████████████████| 204.2 MB 12 kB/s
Collecting py4j==0.10.9
  Downloading py4j-0.10.9-py2.py3-none-any.whl (198 kB)
     |████████████████████████████████| 198 kB 19 kB/s
Building wheels for collected packages: pyspark
  Building wheel for pyspark (setup.py) ... done
  Created wheel for pyspark: filename=pyspark-3.0.1-py2.py3-none-any.whl size=204612242 sha256=8ea424934b4760de0127033f4
6b22f13e3b72c25e2eb004a501598f099176896
  Stored in directory: c:\users\pc\appdata\local\pip\cache\wheels\ea\21\84\970b03913d0d6a96ef51c34c878add0de9e4ecbb7c764
ea21f
Successfully built pyspark
Installing collected packages: py4j, pyspark
Successfully installed py4j-0.10.9 pyspark-3.0.1
```

Importation des librairies :

Entrée [1]:
```
from pyspark.sql import SparkSession
from pyspark.ml import Pipeline
from pyspark.sql.functions import mean,col,split, col, regexp_extract, when, lit
from pyspark.ml.feature import StringIndexer
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
from pyspark.ml.feature import QuantileDiscretizer
```

Création d'une session spark et importation des données:

Entrée [2]:
```
#Beginning with SparkSession
spark = SparkSession \
    .builder \
    .appName("Spark ML example on titanic data ") \
    .getOrCreate()
```

Entrée [7]:
```
s3_bucket_path = "./train.csv"
```

Entrée [8]:
```
titanic_df = spark.read.csv(s3_bucket_path,header = 'True',inferSchema='True')
```

Exploration du dataset et visualisation des données:

Entrée [10]:
```
titanic_df.printSchema()
```

```
root
 |-- PassengerId: integer (nullable = true)
 |-- Survived: integer (nullable = true)
 |-- Pclass: integer (nullable = true)
 |-- Name: string (nullable = true)
 |-- Sex: string (nullable = true)
 |-- Age: double (nullable = true)
 |-- SibSp: integer (nullable = true)
 |-- Parch: integer (nullable = true)
 |-- Ticket: string (nullable = true)
 |-- Fare: double (nullable = true)
 |-- Cabin: string (nullable = true)
 |-- Embarked: string (nullable = true)
```

Entrée [11]:
```
passengers_count = titanic_df.count()
```

Entrée [12]:
```
print(passengers_count)
```

```
891
```

Affichage des 5 premiers enregistrements :

Entrée [13]:
```
titanic_df.show(5)
```

```
+-----------+--------+------+--------------------+------+----+-----+-----+----------------+-------+-----+--------+
|PassengerId|Survived|Pclass|                Name|   Sex| Age|SibSp|Parch|          Ticket|   Fare|Cabin|Embarked|
+-----------+--------+------+--------------------+------+----+-----+-----+----------------+-------+-----+--------+
|          1|       0|     3|Braund, Mr. Owen ...|  male|22.0|    1|    0|       A/5 21171|   7.25| null|       S|
|          2|       1|     1|Cumings, Mrs. Joh...|female|38.0|    1|    0|        PC 17599|71.2833|  C85|       C|
|          3|       1|     3|Heikkinen, Miss. ...|female|26.0|    0|    0|STON/O2. 3101282|  7.925| null|       S|
|          4|       1|     1|Futrelle, Mrs. Ja...|female|35.0|    1|    0|          113803|   53.1| C123|       S|
|          5|       0|     3|Allen, Mr. Willia...|  male|35.0|    0|    0|          373450|   8.05| null|       S|
+-----------+--------+------+--------------------+------+----+-----+-----+----------------+-------+-----+--------+
only showing top 5 rows
```

Entrée [16]:
```
titanic_df.select("Survived","Pclass","Embarked").show()
```

```
+--------+------+--------+
|Survived|Pclass|Embarked|
+--------+------+--------+
|       0|     3|       S|
|       1|     1|       C|
|       1|     3|       S|
|       1|     1|       S|
|       0|     3|       S|
|       0|     3|       Q|
|       0|     1|       S|
|       0|     3|       S|
|       1|     3|       S|
|       1|     2|       C|
|       1|     3|       S|
|       1|     1|       S|
|       0|     3|       S|
|       0|     3|       S|
|       0|     3|       S|
|       1|     2|       S|
|       0|     3|       Q|
|       1|     2|       S|
|       0|     3|       S|
|       1|     3|       C|
+--------+------+--------+
only showing top 20 rows
```

2. EDA :

Connaître le nombre de passagers ont survécu?

Entrée [18]:
```
titanic_df.groupBy("Survived").count().show()
```

```
+--------+-----+
|Survived|count|
+--------+-----+
|       1|  342|
|       0|  549|
+--------+-----+
```

Entrée [19]:
```
gropuBy_output = titanic_df.groupBy("Survived").count()
```

Entrée [20]:
```
display(gropuBy_output)
```

```
DataFrame[Survived: int, count: bigint]
```

Entrée [21]:
```
titanic_df.groupBy("Sex","Survived").count().show()
```

```
+------+--------+-----+
|   Sex|Survived|count|
+------+--------+-----+
|  male|       0|  468|
|female|       1|  233|
|female|       0|   81|
|  male|       1|  109|
+------+--------+-----+
```

3. Les valeurs manquantes et nettoyage :

Création d'une fonction qui cherche les valeurs manquantes/nulles dans notre dataframe et appel de cette dernière :

Entrée [23]:
```
#Checking Null values
```

Entrée [24]:
```
# This function use to print feature with null values and null count
def null_value_count(df):
  null_columns_counts = []
  numRows = df.count()
  for k in df.columns:
    nullRows = df.where(col(k).isNull()).count()
    if(nullRows > 0):
      temp = k,nullRows
      null_columns_counts.append(temp)
  return(null_columns_counts)
```

Entrée [25]:
```
# Calling function
null_columns_count_list = null_value_count(titanic_df)
```

Voici le nombre de données manquantes par colonne :

Entrée [26]:
```
spark.createDataFrame(null_columns_count_list, ['Column_With_Null_Value', 'Null_Values_Count']).show()
```

```
+----------------------+-----------------+
|Column_With_Null_Value|Null_Values_Count|
+----------------------+-----------------+
|                   Age|              177|
|                 Cabin|              687|
|              Embarked|                2|
+----------------------+-----------------+
```

Remplacement par les méthodes suivantes vues dans le cours comme moyenne pour l'âge :

Entrée [27]:
```
mean_age = titanic_df.select(mean('Age')).collect()[0][0]
print(mean_age)
```

Nous pouvons vérifier la fonctionnalité Nom. En regardant la fonctionnalité, nous pouvons voir que les noms ont une salutation comme M. ou Mme Ainsi nous pouvons attribuer les valeurs moyennes de M. et Mme aux groupes respectifs

```
Entrée [29]: titanic_df = titanic_df.withColumn("Initial",regexp_extract(col("Name"),"([A-Za-z]+)\.",1))
```

```
Entrée [30]: titanic_df.show()
```

```
+-----------+--------+------+--------------------+------+----+-----+-----+----------------+-------+-----+--------+-------+
|PassengerId|Survived|Pclass|                Name|   Sex| Age|SibSp|Parch|          Ticket|   Fare|Cabin|Embarked|Initial|
+-----------+--------+------+--------------------+------+----+-----+-----+----------------+-------+-----+--------+-------+
|          1|       0|     3|Braund, Mr. Owen ...|  male|22.0|    1|    0|       A/5 21171|   7.25| null|       S|     Mr|
|          2|       1|     1|Cumings, Mrs. Joh...|female|38.0|    1|    0|        PC 17599|71.2833|  C85|       C|    Mrs|
|          3|       1|     3|Heikkinen, Miss. ...|female|26.0|    0|    0|STON/O2. 3101282|  7.925| null|       S|   Miss|
|          4|       1|     1|Futrelle, Mrs. Ja...|female|35.0|    1|    0|          113803|   53.1| C123|       S|    Mrs|
|          5|       0|     3|Allen, Mr. Willia...|  male|35.0|    0|    0|          373450|   8.05| null|       S|     Mr|
```

Il y a des initiales mal orthographiées comme Mlle ou Mme qui représentent Mlle. Je les remplacerai par Mlle et la même chose pour d'autres valeurs.

```
Entrée [32]: titanic_df = titanic_df.replace(['Mlle','Mme', 'Ms', 'Dr','Major','Lady','Countess','Jonkheer','Col','Rev','Capt','Sir','Don'],
                                              ['Miss','Miss','Miss','Mr','Mr',  'Mrs',  'Mrs',  'Other',  'Other','Other','Mr','Mr','Mr'])
```

```
Entrée [33]: titanic_df.select("Initial").distinct().show()
```

```
+-------+
|Initial|
+-------+
|   Miss|
|  Other|
| Master|
|     Mr|
|    Mrs|
+-------+
```

Vérifier l'âge moyen par initiales :

```
Entrée [34]: titanic_df.groupby('Initial').avg('Age').collect()
```

```
Out[34]: [Row(Initial='Miss', avg(Age)=21.86),
          Row(Initial='Other', avg(Age)=45.888888888888886),
          Row(Initial='Master', avg(Age)=4.574166666666667),
          Row(Initial='Mr', avg(Age)=32.73960880195599),
          Row(Initial='Mrs', avg(Age)=35.981818181818184)]
```

```
Entrée [35]: nic_df.withColumn("Age",when((titanic_df["Initial"] == "Miss") & (titanic_df["Age"].isNull()), 22).otherwise(titanic_df["Age"]))
             nic_df.withColumn("Age",when((titanic_df["Initial"] == "Other") & (titanic_df["Age"].isNull()), 46).otherwise(titanic_df["Age"]))
             nic_df.withColumn("Age",when((titanic_df["Initial"] == "Master") & (titanic_df["Age"].isNull()), 5).otherwise(titanic_df["Age"]))
             nic_df.withColumn("Age",when((titanic_df["Initial"] == "Mr") & (titanic_df["Age"].isNull()), 33).otherwise(titanic_df["Age"]))
             nic_df.withColumn("Age",when((titanic_df["Initial"] == "Mrs") & (titanic_df["Age"].isNull()), 36).otherwise(titanic_df["Age"]))
```

Utilisation de la méthode **na.fill()** :

```
Entrée [38]: titanic_df.groupBy("Embarked").count().show()
```

```
+--------+-----+
|Embarked|count|
+--------+-----+
|       Q|   77|
|    null|    2|
|       C|  168|
|       S|  644|
+--------+-----+
```

```
Entrée [39]: titanic_df = titanic_df.na.fill({"Embarked" : 'S'})
```

Suppression des données inutiles :

```
Entrée [40]: titanic_df = titanic_df.drop("Cabin")

Entrée [41]: titanic_df.printSchema()
```

```
root
 |-- PassengerId: integer (nullable = true)
 |-- Survived: integer (nullable = true)
 |-- Pclass: integer (nullable = true)
 |-- Name: string (nullable = true)
 |-- Sex: string (nullable = true)
 |-- Age: double (nullable = true)
 |-- SibSp: integer (nullable = true)
 |-- Parch: integer (nullable = true)
 |-- Ticket: string (nullable = true)
 |-- Fare: double (nullable = true)
 |-- Embarked: string (nullable = false)
 |-- Initial: string (nullable = true)
```

4. Features & spliting & Pipeline:

Précision des features et les étapes du pipeline avec **StringIndexer ().**Nous mettons toutes les fonctionnalités dans le vecteur :

```
Entrée [47]: indexers = [StringIndexer(inputCol=column, outputCol=column+"_index").fit(titanic_df) for column in ["Sex","Embarked","Initial"]]
pipeline = Pipeline(stages=indexers)
titanic_df = pipeline.fit(titanic_df).transform(titanic_df)

Entrée [48]: titanic_df.show()
```

```
+-----------+--------+------+--------------------+------+----+-----+-----+----------------+-------+--------+-------+-----------
+-----+---------+--------------+-------------+
|PassengerId|Survived|Pclass|                Name|   Sex| Age|SibSp|Parch|          Ticket|   Fare|Embarked|Initial|Family_Size
|Alone|Sex_index|Embarked_index|Initial_index|
+-----------+--------+------+--------------------+------+----+-----+-----+----------------+-------+--------+-------+-----------
+-----+---------+--------------+-------------+
|          1|       0|     3|Braund, Mr. Owen ...|  male|22.0|    1|    0|       A/5 21171|   7.25|       S|     Mr|          1
|    0|      0.0|           0.0|          0.0|
|          2|       1|     1|Cumings, Mrs. Joh...|female|38.0|    1|    0|        PC 17599|71.2833|       C|    Mrs|          1
|    0|      1.0|           1.0|          2.0|
|          3|       1|     3|Heikkinen, Miss. ...|female|26.0|    0|    0|STON/O2. 3101282|  7.925|       S|   Miss|          0
|    1|      1.0|           0.0|          1.0|
|          4|       1|     1|Futrelle, Mrs. Ja...|female|35.0|    1|    0|          113803|   53.1|       S|    Mrs|          1
|    0|      1.0|           0.0|          2.0|
|          5|       0|     3|Allen, Mr. Willia...|  male|35.0|    0|    0|          373450|   8.05|       S|     Mr|          0
|    1|      0.0|           0.0|          0.0|
```

Vector Assembler :

Entrée [51]:
```
titanic_df = titanic_df.drop("PassengerId","Name","Ticket","Cabin","Embarked","Sex","Initial")
```

Entrée [52]:
```
feature = VectorAssembler(inputCols=titanic_df.columns[1:],outputCol="features")
feature_vector= feature.transform(titanic_df)
```

Entrée [53]:
```
feature_vector.show()
```

```
+--------+------+----+-----+-----+-------+-----------+-----+---------+-------------+-------------+--------------------+
|Survived|Pclass| Age|SibSp|Parch|   Fare|Family_Size|Alone|Sex_index|Embarked_index|Initial_index|            features|
+--------+------+----+-----+-----+-------+-----------+-----+---------+-------------+-------------+--------------------+
|       0|     3|22.0|    1|    0|   7.25|          1|    0|      0.0|          0.0|          0.0|(10,[0,1,2,4,5],[...|
|       1|     1|38.0|    1|    0|71.2833|          1|    0|      1.0|          1.0|          2.0|[1.0,38.0,1.0,0.0...|
|       1|     3|26.0|    0|    0|  7.925|          0|    1|      1.0|          0.0|          1.0|[3.0,26.0,0.0,0.0...|
|       1|     1|35.0|    1|    0|   53.1|          1|    0|      1.0|          0.0|          2.0|[1.0,35.0,1.0,0.0...|
|       0|     3|35.0|    0|    0|   8.05|          0|    1|      0.0|          0.0|          0.0|(10,[0,1,4,6],[3....|
|       0|     3|33.0|    0|    0| 8.4583|          0|    1|      0.0|          2.0|          0.0|(10,[0,1,4,6,8],[...|
|       0|     1|54.0|    0|    0|51.8625|          0|    1|      0.0|          0.0|          0.0|(10,[0,1,4,6],[1....|
|       0|     3| 2.0|    3|    1| 21.075|          4|    0|      0.0|          0.0|          3.0|[3.0,2.0,3.0,1.0,...|
|       1|     3|27.0|    0|    2|11.1333|          2|    0|      1.0|          0.0|          2.0|[3.0,27.0,0.0,2.0...|
|       1|     2|14.0|    1|    0|30.0708|          1|    0|      1.0|          1.0|          2.0|[2.0,14.0,1.0,0.0...|
```

Maintenant que les données sont toutes définies, divisons-les en entraînement et test. J'en utiliserai 80%.

Entrée [54]:
```
(trainingData, testData) = feature_vector.randomSplit([0.8, 0.2],seed = 11)
```

5. Machine learning Models :

Régression Logistique :

Application du model :

Entrée [57]:
```
from pyspark.ml.classification import LogisticRegression
lr = LogisticRegression(labelCol="Survived", featuresCol="features")
#Training algo
lrModel = lr.fit(trainingData)
lr_prediction = lrModel.transform(testData)
lr_prediction.select("prediction", "Survived", "features").show()
evaluator = MulticlassClassificationEvaluator(labelCol="Survived", predictionCol="prediction", metricName="accuracy")
```

```
+----------+--------+--------------------+
|prediction|Survived|            features|
+----------+--------+--------------------+
|       1.0|       0|(10,[0,1,4,6,8],[...|
|       1.0|       0|(10,[0,1,4,6,8],[...|
|       0.0|       0|(10,[0,1,4,6],[1....|
|       0.0|       0|(10,[0,1,2,4,5],[...|
|       0.0|       0|(10,[0,1,4,6,8],[...|
|       0.0|       0|(10,[0,1,2,4,5],[...|
|       0.0|       0|(10,[0,1,6],[1.0,...|
|       0.0|       0|(10,[0,1,4,6],[1....|
|       0.0|       0|(10,[0,1,4,6],[1....|
|       0.0|       0|(10,[0,1,4,6],[1....|
|       1.0|       0|(10,[0,1,4,6],[1....|
|       0.0|       0|(10,[0,1,2,4,5],[...|
|       1.0|       0|(10,[0,1,3,4,5],[...|
|       0.0|       0|(10,[0,1,2,4,5],[...|
|       0.0|       0|(10,[0,1,4,6],[1....|
|       0.0|       0|[1.0,58.0,0.0,2.0...|
|       0.0|       0|(10,[0,1,4,6],[1....|
|       0.0|       0|(10,[0,1,4,6],[1....|
|       0.0|       0|(10,[0,1,4,6,8],[...|
```

Evaluation :

Entrée [58]:
```
#Evaluating accuracy of LogisticRegression.
lr_accuracy = evaluator.evaluate(lr_prediction)
print("Accuracy of LogisticRegression is = %g"% (lr_accuracy))
print("Test Error of LogisticRegression = %g " % (1.0 - lr_accuracy))
```

```
Accuracy of LogisticRegression is = 0.771277
Test Error of LogisticRegression = 0.228723
```

DecisionTreeClassifier:

Application du model :

Entrée [61]:
```
from pyspark.ml.classification import DecisionTreeClassifier
dt = DecisionTreeClassifier(labelCol="Survived", featuresCol="features")
dt_model = dt.fit(trainingData)
dt_prediction = dt_model.transform(testData)
dt_prediction.select("prediction", "Survived", "features").show()
```

```
+----------+--------+--------------------+
|prediction|Survived|            features|
+----------+--------+--------------------+
|       0.0|       0|(10,[0,1,4,6,8],[...|
|       0.0|       0|(10,[0,1,4,6,8],[...|
|       1.0|       0|(10,[0,1,4,6],[1....|
|       0.0|       0|(10,[0,1,2,4,5],[...|
|       1.0|       0|(10,[0,1,4,6,8],[...|
|       0.0|       0|(10,[0,1,2,4,5],[...|
|       0.0|       0|(10,[0,1,6],[1.0,...|
|       0.0|       0|(10,[0,1,4,6],[1....|
|       0.0|       0|(10,[0,1,4,6],[1....|
|       0.0|       0|(10,[0,1,4,6],[1....|
|       0.0|       0|(10,[0,1,4,6],[1....|
|       0.0|       0|(10,[0,1,2,4,5],[...|
```

Evaluation :

Entrée [62]:
```
#Evaluating accuracy of DecisionTreeClassifier.
dt_accuracy = evaluator.evaluate(dt_prediction)
print("Accuracy of DecisionTreeClassifier is = %g"% (dt_accuracy))
print("Test Error of DecisionTreeClassifier = %g " % (1.0 - dt_accuracy))
```

```
Accuracy of DecisionTreeClassifier is = 0.819149
Test Error of DecisionTreeClassifier = 0.180851
```

RandomForestClassifier:

Application du model :

Entrée [64]:
```
from pyspark.ml.classification import RandomForestClassifier
rf = DecisionTreeClassifier(labelCol="Survived", featuresCol="features")
rf_model = rf.fit(trainingData)
rf_prediction = rf_model.transform(testData)
rf_prediction.select("prediction", "Survived", "features").show()
```

```
+----------+--------+--------------------+
|prediction|Survived|            features|
+----------+--------+--------------------+
|       0.0|       0|(10,[0,1,4,6,8],[...|
|       0.0|       0|(10,[0,1,4,6,8],[...|
|       1.0|       0|(10,[0,1,4,6],[1....|
|       0.0|       0|(10,[0,1,2,4,5],[...|
|       1.0|       0|(10,[0,1,4,6,8],[...|
|       0.0|       0|(10,[0,1,2,4,5],[...|
|       0.0|       0|(10,[0,1,6],[1.0,...|
|       0.0|       0|(10,[0,1,4,6],[1....|
|       0.0|       0|(10,[0,1,4,6],[1....|
|       0.0|       0|(10,[0,1,4,6],[1....|
|       0.0|       0|(10,[0,1,4,6],[1....|
|       0.0|       0|(10,[0,1,2,4,5],[...|
|       0.0|       0|(10,[0,1,3,4,5],[...|
|       0.0|       0|(10,[0,1,2,4,5],[...|
```

Evaluation :

Entrée [65]:
```
#Evaluating accuracy of RandomForestClassifier.
```

Entrée [66]:
```
rf_accuracy = evaluator.evaluate(rf_prediction)
print("Accuracy of RandomForestClassifier is = %g"% (rf_accuracy))
print("Test Error of RandomForestClassifier  = %g " % (1.0 - rf_accuracy))
```

```
Accuracy of RandomForestClassifier is = 0.819149
Test Error of RandomForestClassifier  = 0.180851
```

Gradient-boosted tree classifier:

Application du model :

Entrée [68]:
```
from pyspark.ml.classification import GBTClassifier
gbt = GBTClassifier(labelCol="Survived", featuresCol="features",maxIter=10)
gbt_model = gbt.fit(trainingData)
gbt_prediction = gbt_model.transform(testData)
gbt_prediction.select("prediction", "Survived", "features").show()
```

```
+----------+--------+--------------------+
|prediction|Survived|            features|
+----------+--------+--------------------+
|       1.0|       0|(10,[0,1,4,6,8],[...|
|       0.0|       0|(10,[0,1,4,6,8],[...|
|       1.0|       0|(10,[0,1,4,6],[1....|
|       0.0|       0|(10,[0,1,2,4,5],[...|
|       1.0|       0|(10,[0,1,4,6,8],[...|
|       0.0|       0|(10,[0,1,2,4,5],[...|
|       0.0|       0|(10,[0,1,6],[1.0,...|
|       0.0|       0|(10,[0,1,4,6],[1....|
|       0.0|       0|(10,[0,1,4,6],[1....|
|       0.0|       0|(10,[0,1,4,6],[1....|
|       0.0|       0|(10,[0,1,4,6],[1....|
|       0.0|       0|(10,[0,1,2,4,5],[...|
|       0.0|       0|(10,[0,1,3,4,5],[...|
|       0.0|       0|(10,[0,1,2,4,5],[...|
|       0.0|       0|(10,[0,1,4,6],[1....|
|       0.0|       0|[1.0,58.0,0.0,2.0...|
|       0.0|       0|(10,[0,1,4,6],[1....|
|       0.0|       0|(10,[0,1,4,6],[1....|
```

Evaluation :

Entrée [69]:
```
#Evaluate accuracy of Gradient-boosted.
```

Entrée [70]:
```
gbt_accuracy = evaluator.evaluate(gbt_prediction)
print("Accuracy of Gradient-boosted tree classifie is = %g"% (gbt_accuracy))
print("Test Error of Gradient-boosted tree classifie %g"% (1.0 - gbt_accuracy))
```

```
Accuracy of Gradient-boosted tree classifie is = 0.803191
Test Error of Gradient-boosted tree classifie 0.196809
```

NaiveBayes:

Application du model :

Entrée [72]:
```
from pyspark.ml.classification import NaiveBayes
nb = NaiveBayes(labelCol="Survived", featuresCol="features")
nb_model = nb.fit(trainingData)
nb_prediction = nb_model.transform(testData)
nb_prediction.select("prediction", "Survived", "features").show()
```

```
+----------+--------+--------------------+
|prediction|Survived|            features|
+----------+--------+--------------------+
|       1.0|       0|(10,[0,1,4,6,8],[...|
|       1.0|       0|(10,[0,1,4,6,8],[...|
|       0.0|       0|(10,[0,1,4,6],[1....|
|       1.0|       0|(10,[0,1,2,4,5],[...|
|       0.0|       0|(10,[0,1,4,6,8],[...|
|       1.0|       0|(10,[0,1,2,4,5],[...|
|       0.0|       0|(10,[0,1,6],[1.0,...|
|       0.0|       0|(10,[0,1,4,6],[1....|
|       0.0|       0|(10,[0,1,4,6],[1....|
|       1.0|       0|(10,[0,1,4,6],[1....|
|       1.0|       0|(10,[0,1,4,6],[1....|
|       1.0|       0|(10,[0,1,2,4,5],[...|
|       1.0|       0|(10,[0,1,3,4,5],[...|
|       1.0|       0|(10,[0,1,2,4,5],[...|
|       0.0|       0|(10,[0,1,4,6],[1....|
```

Evaluation :

Entrée [73]:
```
#Evaluating accuracy of NaiveBayes.
nb_accuracy = evaluator.evaluate(nb_prediction)
print("Accuracy of NaiveBayes is  = %g"% (nb_accuracy))
print("Test Error of NaiveBayes  = %g " % (1.0 - nb_accuracy))
```

```
Accuracy of NaiveBayes is  = 0.739362
Test Error of NaiveBayes  = 0.260638
```

Support Vector Machine:

Application du model :

Entrée [75]:
```
from pyspark.ml.classification import LinearSVC
svm = LinearSVC(labelCol="Survived", featuresCol="features")
svm_model = svm.fit(trainingData)
svm_prediction = svm_model.transform(testData)
svm_prediction.select("prediction", "Survived", "features").show()
```

```
+----------+--------+--------------------+
|prediction|Survived|            features|
+----------+--------+--------------------+
|       0.0|       0|(10,[0,1,4,6,8],[...|
|       0.0|       0|(10,[0,1,4,6,8],[...|
|       0.0|       0|(10,[0,1,4,6],[1....|
|       0.0|       0|(10,[0,1,2,4,5],[...|
|       0.0|       0|(10,[0,1,4,6,8],[...|
|       0.0|       0|(10,[0,1,2,4,5],[...|
|       0.0|       0|(10,[0,1,6],[1.0,...|
|       0.0|       0|(10,[0,1,4,6],[1....|
|       0.0|       0|(10,[0,1,4,6],[1....|
|       0.0|       0|(10,[0,1,4,6],[1....|
|       0.0|       0|(10,[0,1,4,6],[1....|
|       0.0|       0|(10,[0,1,2,4,5],[...|
|       0.0|       0|(10,[0,1,3,4,5],[...|
|       0.0|       0|(10,[0,1,2,4,5],[...|
```

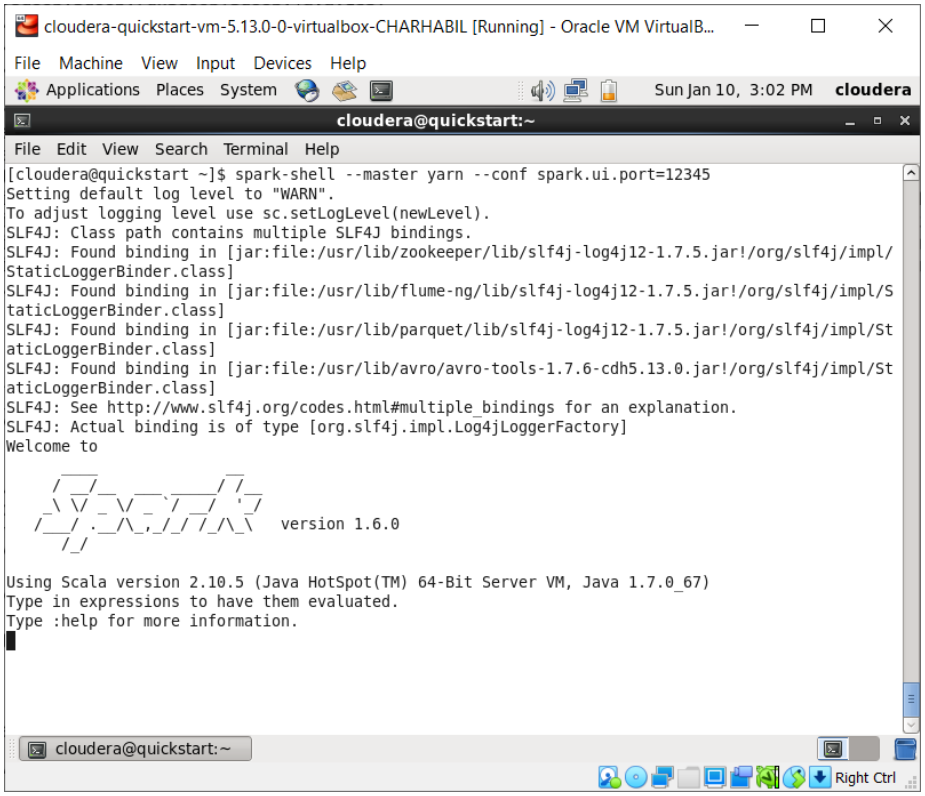
Evaluation :

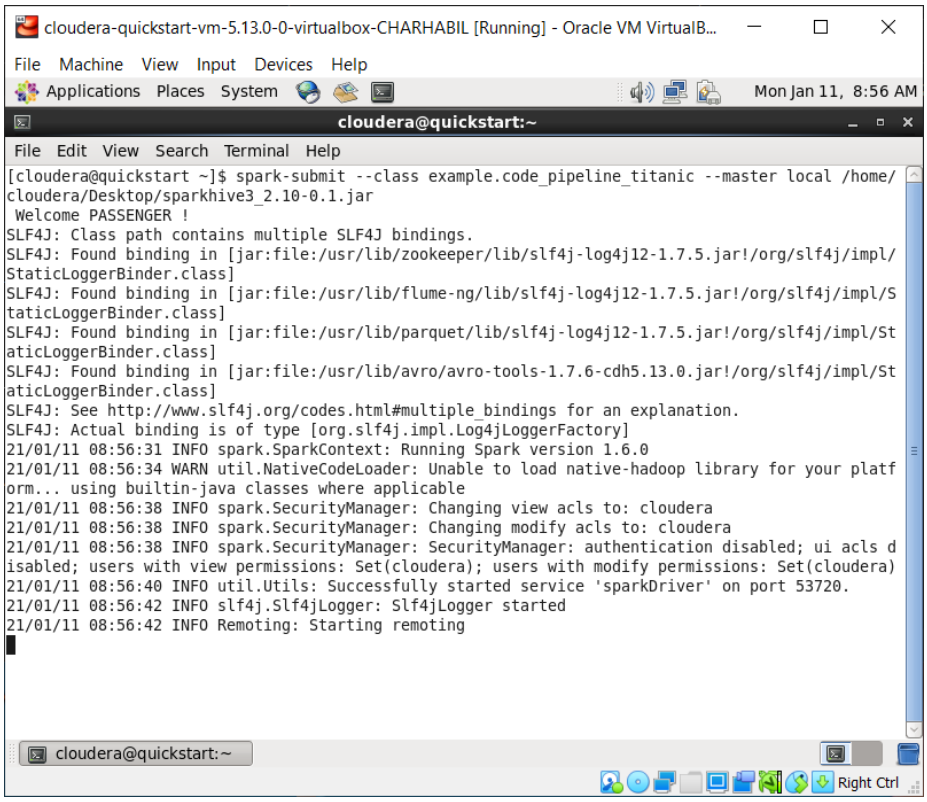
Entrée [76]:
```
#Evaluating the accuracy of Support Vector Machine.
svm_accuracy = evaluator.evaluate(svm_prediction)
print("Accuracy of Support Vector Machine is = %g"% (svm_accuracy))
print("Test Error of Support Vector Machine = %g " % (1.0 - svm_accuracy))
```

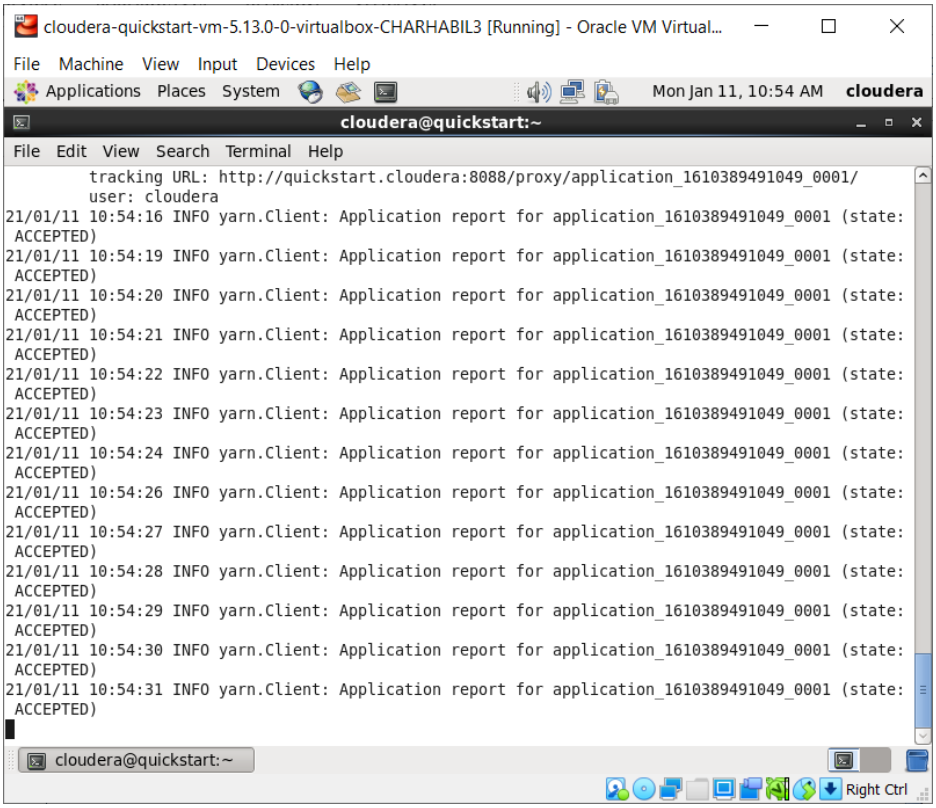
```
Accuracy of Support Vector Machine is = 0.808511
Test Error of Support Vector Machine = 0.191489
```

1. Exécution du JAR avec Spark:

Vérifier le bon fonctionnement de Spark :


cloudera-quickstart-vm-5.13.0-0-virtualbox-CHARHABIL [Running] - Oracle VM VirtualB...
File Machine View Input Devices Help
Applications Places System
Sun Jan 10, 3:02 PM cloudera
cloudera@quickstart:~
File Edit View Search Terminal Help
[cloudera@quickstart ~]$ spark-shell --master yarn --conf spark.ui.port=12345
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel).
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/lib/zookeeper/lib/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/lib/flume-ng/lib/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/lib/parquet/lib/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/lib/avro/avro-tools-1.7.6-cdh5.13.0.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
Welcome to
version 1.6.0
Using Scala version 2.10.5 (Java HotSpot(TM) 64-Bit Server VM, Java 1.7.0_67)
Type in expressions to have them evaluated.
Type :help for more information.
cloudera@quickstart:~
Right Ctrl


cloudera-quickstart-vm-5.13.0-0-virtualbox-CHARHABIL [Running] - Oracle VM VirtualB...
File Machine View Input Devices Help
Applications Places System
Mon Jan 11, 8:56 AM
cloudera@quickstart:~
File Edit View Search Terminal Help
[cloudera@quickstart ~]$ spark-submit --class example.code_pipeline_titanic --master local /home/cloudera/Desktop/sparkhive3_2.10-0.1.jar
Welcome PASSENGER !
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/lib/zookeeper/lib/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/lib/flume-ng/lib/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/lib/parquet/lib/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/usr/lib/avro/avro-tools-1.7.6-cdh5.13.0.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
21/01/11 08:56:31 INFO spark.SparkContext: Running Spark version 1.6.0
21/01/11 08:56:34 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
21/01/11 08:56:38 INFO spark.SecurityManager: Changing view acls to: cloudera
21/01/11 08:56:38 INFO spark.SecurityManager: Changing modify acls to: cloudera
21/01/11 08:56:38 INFO spark.SecurityManager: SecurityManager: authentication disabled; ui acls disabled; users with view permissions: Set(cloudera); users with modify permissions: Set(cloudera)
21/01/11 08:56:40 INFO util.Utils: Successfully started service 'sparkDriver' on port 53720.
21/01/11 08:56:42 INFO slf4j.Slf4jLogger: Slf4jLogger started
21/01/11 08:56:42 INFO Remoting: Starting remoting
cloudera@quickstart:~
Right Ctrl


cloudera-quickstart-vm-5.13.0-0-virtualbox-CHARHABIL3 [Running] - Oracle VM Virtual...
File Machine View Input Devices Help
Applications Places System
Mon Jan 11, 10:54 AM cloudera
cloudera@quickstart:~
File Edit View Search Terminal Help
tracking URL: http://quickstart.cloudera:8088/proxy/application_1610389491049_0001/
user: cloudera
21/01/11 10:54:16 INFO yarn.Client: Application report for application_1610389491049_0001 (state: ACCEPTED)
21/01/11 10:54:19 INFO yarn.Client: Application report for application_1610389491049_0001 (state: ACCEPTED)
21/01/11 10:54:20 INFO yarn.Client: Application report for application_1610389491049_0001 (state: ACCEPTED)
21/01/11 10:54:21 INFO yarn.Client: Application report for application_1610389491049_0001 (state: ACCEPTED)
21/01/11 10:54:22 INFO yarn.Client: Application report for application_1610389491049_0001 (state: ACCEPTED)
21/01/11 10:54:23 INFO yarn.Client: Application report for application_1610389491049_0001 (state: ACCEPTED)
21/01/11 10:54:24 INFO yarn.Client: Application report for application_1610389491049_0001 (state: ACCEPTED)
21/01/11 10:54:26 INFO yarn.Client: Application report for application_1610389491049_0001 (state: ACCEPTED)
21/01/11 10:54:27 INFO yarn.Client: Application report for application_1610389491049_0001 (state: ACCEPTED)
21/01/11 10:54:28 INFO yarn.Client: Application report for application_1610389491049_0001 (state: ACCEPTED)
21/01/11 10:54:29 INFO yarn.Client: Application report for application_1610389491049_0001 (state: ACCEPTED)
21/01/11 10:54:30 INFO yarn.Client: Application report for application_1610389491049_0001 (state: ACCEPTED)
21/01/11 10:54:31 INFO yarn.Client: Application report for application_1610389491049_0001 (state: ACCEPTED)
cloudera@quickstart:~
Right Ctrl