logocover

**SMOKE TEST DOCUMENT**

Spark Smoke Test Cases

Date Prepared: August 2019

**Document Information**

|  |  |  |  |
| --- | --- | --- | --- |
| **Project Name** | **Spark Smoke Test Document** | | |
| **Project Owner** |  | **Document Version No** | 1.0 |
| **Quality Review Method** | By email/HP SharePoint |  |  |
| **Prepared By** |  | **Preparation Date** | August 2019 |
| **Reviewed By** | Refer to version history | **Review Date** |  |

**Table of Contents**

[1 Note 4](#_Toc16091154)

[2 Sample test case for Spark-submit 5](#_Toc16091155)

[3 Sample test case for Spark-shell 7](#_Toc16091156)

[3.1 Interactive analysis with the Spark Shell 7](#_Toc16091157)

[3.2 Caching operations on Spark Shell 8](#_Toc16091158)

[3.3 Example for Scala Word Count program: 9](#_Toc16091159)

[4 Test cases for Jupyterhub 11](#_Toc16091160)

[4.1 Spark Scala testing 11](#_Toc16091161)

[4.2 PySpark testing 13](#_Toc16091162)

[4.3 Execute Spark Sumbit job on JupterHub 15](#_Toc16091163)

[5 Sample test cases for Spark with notebooks 17](#_Toc16091164)

[5.1 PySpark testing 17](#_Toc16091165)

[5.2 Spark Scala testing 19](#_Toc16091166)

[6 Test cases for Spark with RStudio 22](#_Toc16091167)

[6.1 Base-R testing on RStudio GUI 22](#_Toc16091168)

[6.2 Sparklyr testing on RStudio GUI 23](#_Toc16091169)

[6.3 Simple test on RStudio GUI 24](#_Toc16091170)

[6.4 MLLib usage test on RSudio GUI 24](#_Toc16091171)

[7 Test cases for Spark with SQL 26](#_Toc16091172)

[7.1 Testing with user defined functions 26](#_Toc16091173)

[7.2 Starting a Spark Session and displaying DataFrame of people.json 27](#_Toc16091174)

[7.3 Creating a Dataset 28](#_Toc16091175)

# Note

Set SPARK\_HOME to $PATH or “cd /usr/lib/spark/spark-2.4.0-bin-hadoop2.7/” and run the following examples

All the Spark examples related to Python, Scala, Java & R are under this location “/usr/lib/spark/spark-2.4.0-bin-hadoop2.7/examples/src/main”

Replace <Spark-master-IP> with the actual IP address

Reference link: <https://spark.apache.org/docs/latest/submitting-applications.html>

# Sample test case for Spark-submit

* Run application locally on 8 cores

./bin/spark-submit \  
 --class org.apache.spark.examples.SparkPi \  
 --master local[8] \  
 /usr/lib/spark/spark-2.4.0-bin- hadoop2.7/examples/jars/spark-examples\_2.11-2.4.0.jar \  
 100

* Run below command on a Spark standalone cluster in client deploy mode

./bin/spark-submit \  
 --class org.apache.spark.examples.SparkPi \  
 --master spark://<Spark-master-IP>:7077 \  
 --executor-memory 20G \  
 --total-executor-cores 100 \  
 /usr/lib/spark/spark-2.4.0-bin-hadoop2.7/examples/jars/spark-examples\_2.11-2.4.0.jar \  
 1000

* Run below command on a Spark standalone cluster in cluster deploy mode with supervise

./bin/spark-submit \  
 --class org.apache.spark.examples.SparkPi \  
 --master spark://<Spark-master-IP>:7077 \  
 --deploy-mode cluster \  
 --supervise \

* Run a python application on a Spark standalone cluster

./bin/spark-submit \  
 --class org.apache.spark.examples.SparkPi \  
 --master spark://<Spark-master-IP>:7077 \  
 --deploy-mode cluster \  
 --supervise \

# Sample test case for Spark-shell

We will first introduce the API through Spark’s interactive shell (in Python or Scala) and then show how to write applications in Java, Scala and Python.

## Interactive analysis with the Spark Shell

Spark’s shell provides a simple way to learn the API, as well as a powerful tool to analyze data interactively. It is available in either Scala (which runs on the Java VM and is thus a good way to use existing Java libraries) or Python.

Start it by running the following in the Spark directory

./bin/spark-shell

* Make a new Dataset from the text of the README file in the Spark source directory:

scala> val textFile = spark.read.textFile("README.md")

textFile: org.apache.spark.sql.Dataset[String] = [value: string]

* Get values from Dataset directly, by calling some actions, or transform the Dataset to get new one

scala> textFile.count() // Number of items in this Dataset

res0: Long = 126 // May be different from yours as README.md

scala> textFile.first() // First item in this Dataset

res1: String = # Apache Spark

* Transform this Dataset into a new one

scala> val linesWithSpark = textFile.filter(line =>line.contains("Spark"))

linesWithSpark: org.apache.spark.sql.Dataset[String] =[value: string]

* Chain together transformations and actions

scala> textFile.filter(line =>line.contains("Spark")).count() // How many lines contain "Spark"?

res3: Long = 15

## Caching operations on Spark Shell

Spark also supports pulling data sets into a cluster-wide in-memory cache. This is very useful when data is accessed repeatedly.

scala> linesWithSpark.cache()

res7: linesWithSpark.type = [value: string]

scala> linesWithSpark.count()

res8: Long = 15

scala> linesWithSpark.count()

res9: Long = 15

Reference Link: <https://spark.apache.org/docs/latest/quick-start.html#basics>

## Example for Scala Word Count program:

Following are the commands that we shall use for Word Count Example in Spark Shell

* Using Spark context variable, sc to read a text file

scala> sc.textFile("usr/lib/spark/spark-2.4.0-bin-hadoop2.7/word.txt”)

* Split each line using space ” ” as separator

scala> flatMap(line => line.split(" "))

* Map each word to a tuple (word, 1), 1 being the number of occurrences of word

scala> map(word => (word,1))

* Reduce all the words based on Key

scala> var counts = map.reduceByKey(\_ + \_);

* Save counts to local file

scala> counts.saveAsTextFile("usr/lib/spark/spark-2.4.0-bin-hadoop2.7/result.txt”)

# Test cases for Jupyterhub

* Create a Linux user on the master controller node or login as AD user.
* Login to Jupyterhub.

Note: All the Spark examples related to Python, Scala, Java & R are under this location “/usr/lib/spark/spark-2.4.0-bin-hadoop2.7/examples/src/main/”

## Spark Scala testing

Start a toree scala kernel -> Wait till kernel creates a spark shell. Run following Pearson’s correlation. You can run up to 4 Spark shells with current configurations. If your shell doesn’t start, you may have used up all the cores. Kill unused Kernels to release resources

Code: Running Pearson’s correlation using mllib

**Note:** You can copy the sample code below from this link: <https://spark.apache.org/docs/latest/mllib-statistics.html>

import org.apache.spark.mllib.linalg.\_

import org.apache.spark.mllib.stat.Statistics

import org.apache.spark.rdd.RDD

val seriesX: RDD[Double] = sc.parallelize(Array(1, 2, 3, 3, 5)) // a series

// must have the same number of partitions and cardinality as seriesX

val seriesY: RDD[Double] = sc.parallelize(Array(11, 22, 33, 33, 555))

// compute the correlation using Pearson's method. Enter "spearman" for Spearman's method. If a

// method is not specified, Pearson's method will be used by default.

val correlation: Double = Statistics.corr(seriesX, seriesY, "pearson")

println(s"Correlation is: $correlation")

val data: RDD[Vector] = sc.parallelize(

Seq(

Vectors.dense(1.0, 10.0, 100.0),

Vectors.dense(2.0, 20.0, 200.0),

Vectors.dense(5.0, 33.0, 366.0))

) // note that each Vector is a row and not a column

// calculate the correlation matrix using Pearson's method. Use "spearman" for Spearman's method

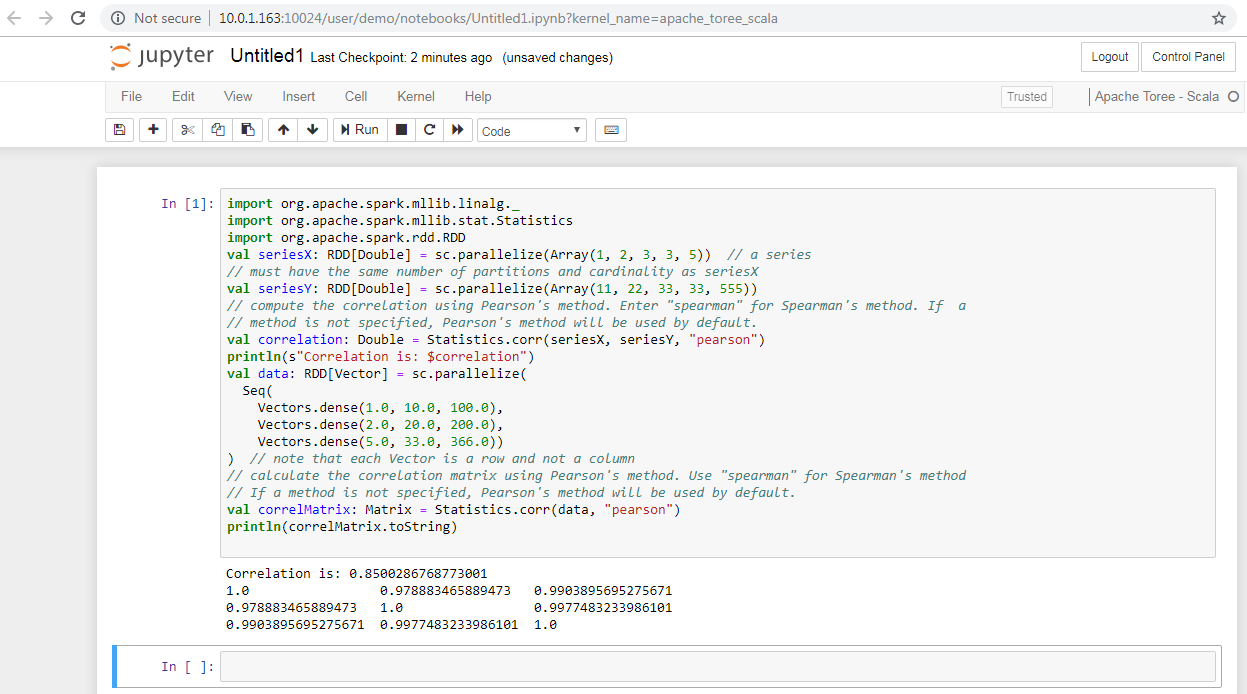
// If a method is not specified, Pearson's method will be used by default.

val correlMatrix: Matrix = Statistics.corr(data, "pearson")

println(correlMatrix.toString)

Input: Input is generated within the code. No external input is provided.

Output: Sample output is as given below.

****

## PySpark testing

Start a toree pySpark kernel -> Wait till kernel creates a spark shell. You can run up to 4 Spark shells with current configurations. If your shell doesn’t start, you may have used up all the cores. Kill unused Kernels to release resources.

Code:

from pyspark.mllib.linalg import Matrices, Vectors

from pyspark.mllib.regression import LabeledPoint

from pyspark.mllib.stat import Statistics

vec = Vectors.dense(0.1, 0.15, 0.2, 0.3, 0.25) # a vector composed of the frequencies of events

# compute the goodness of fit. If a second vector to test against

# is not supplied as a parameter, the test runs against a uniform distribution.

goodnessOfFitTestResult = Statistics.chiSqTest(vec)

# summary of the test including the p-value, degrees of freedom,

# test statistic, the method used, and the null hypothesis.

print("%s\n" % goodnessOfFitTestResult)

mat = Matrices.dense(3, 2, [1.0, 3.0, 5.0, 2.0, 4.0, 6.0]) # a contingency matrix

# conduct Pearson's independence test on the input contingency matrix

independenceTestResult = Statistics.chiSqTest(mat)

# summary of the test including the p-value, degrees of freedom,

# test statistic, the method used, and the null hypothesis.

print("%s\n" % independenceTestResult)

obs = sc.parallelize(

[LabeledPoint(1.0, [1.0, 0.0, 3.0]),

LabeledPoint(1.0, [1.0, 2.0, 0.0]),

LabeledPoint(1.0, [-1.0, 0.0, -0.5])]

) # LabeledPoint(label, feature)

# The contingency table is constructed from an RDD of LabeledPoint and used to conduct

# the independence test. Returns an array containing the ChiSquaredTestResult for every feature

# against the label.

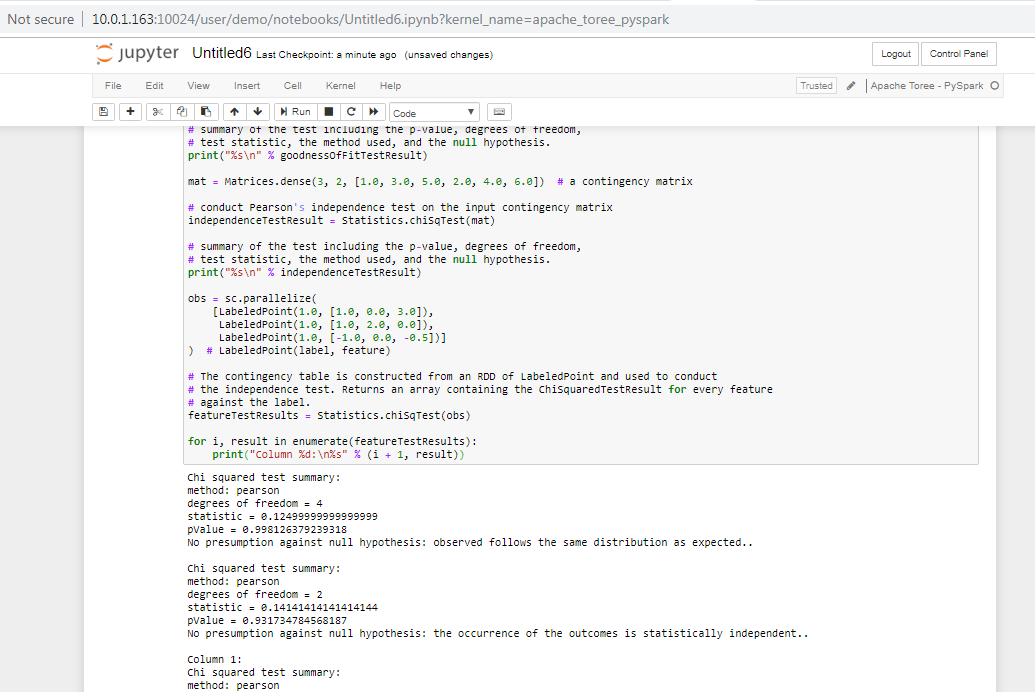
featureTestResults = Statistics.chiSqTest(obs)

for i, result in enumerate(featureTestResults):

print("Column %d:\n%s" % (i + 1, result))

Input: No input files used. Data is generated in the code.

Output: Sample output is as given below.



## Execute Spark Sumbit job on JupterHub

Start a toree pySpark kernel -> Wait till kernel creates a spark shell. You can run up to 4 Spark shells with current configurations. If your shell doesn’t start, you may have used up all the cores. Kill unused Kernels to release resources.

Code:

##sh

./bin/spark-submit \  
 --class org.apache.spark.examples.SparkPi \  
 --master local[8] \  
 /usr/lib/spark/spark-2.4.0-bin-hadoop2.7/examples/jars/spark-examples\_2.11-2.4.0.jar \  
 100

Output:

Check Spark master GUI that job is running under Running Applications section

# Sample test cases for Spark with notebooks

## PySpark testing

Start a toree pySpark kernel -> Wait till kernel creates a spark shell. You can run up to 4 Spark shells with current configurations. If your shell doesn’t start, you may have used up all the cores. Kill unused Kernels to release resources.

Code:

from pyspark import SparkConf, SparkContext

from sklearn.datasets import make\_classification

from sklearn.ensemble import ExtraTreesClassifier

import pandas as pd

import numpy as np

# Build a classification task using 3 informative features

X, y = make\_classification(n\_samples=12000,

n\_features=10,

n\_informative=3,

n\_redundant=0,

n\_repeated=0,

n\_classes=2,

random\_state=0,

shuffle=False)

# Partition data

def dataPart(X, y, start, stop): return dict(X=X[start:stop, :], y=y[start:stop])

def train(data):

X = data['X']

y = data['y']

return ExtraTreesClassifier(n\_estimators=100,random\_state=0).fit(X,y)

# Merge 2 Models

from sklearn.base import copy

def merge(left,right):

new = copy.deepcopy(left)

new.estimators\_ += right.estimators\_

new.n\_estimators = len(new.estimators\_)

return new

data = [dataPart(X, y, 0, 4000), dataPart(X,y,4000,8000), dataPart(X,y,8000,12000)]

forest = sc.parallelize(data).map(train).reduce(merge)

importances = forest.feature\_importances\_

std = np.std([tree.feature\_importances\_ for tree in forest.estimators\_],

axis=0)

indices = np.argsort(importances)[::-1]

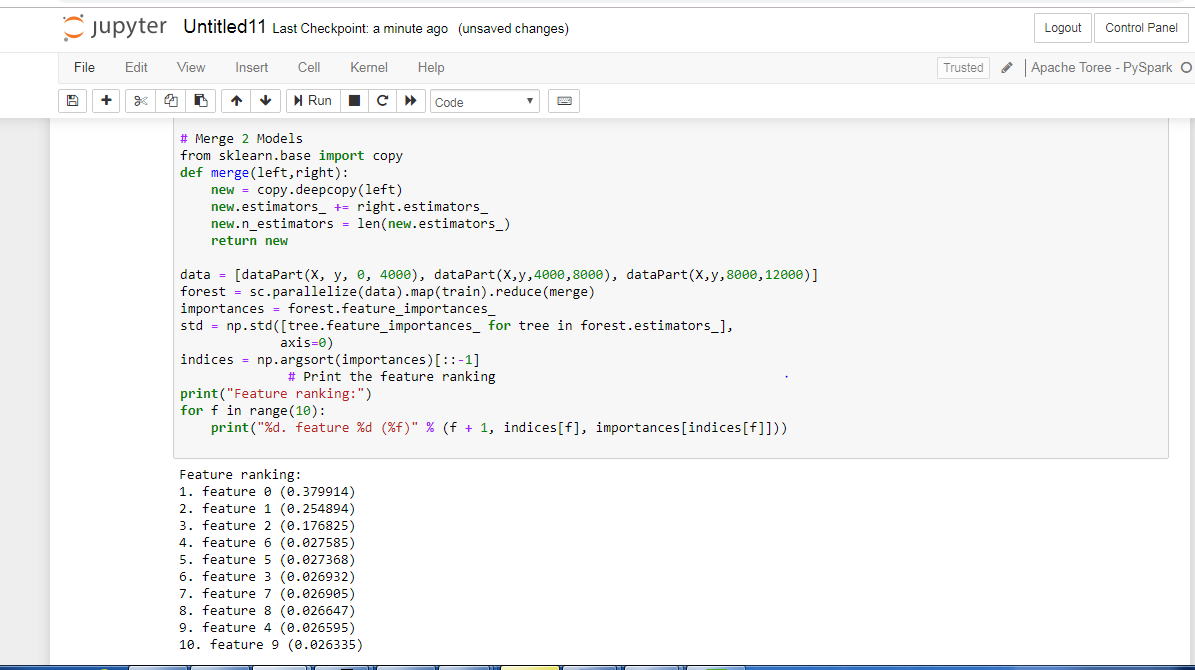
# Print the feature ranking

print("Feature ranking:")

for f in range(10):

print("%d. feature %d (%f)" % (f + 1, indices[f], importances[indices[f]]))

Output: Sample output is as given below.



## Spark Scala testing

Start a toree scala kernel -> Wait till kernel creates a spark shell. Run following Pearson’s correlation. You can run up to 4 Spark shells with current configurations. If your shell doesn’t start, you may have used up all the cores. Kill unused Kernels to release resources

Code:

import org.apache.spark.mllib.linalg.\_

import org.apache.spark.mllib.stat.Statistics

import org.apache.spark.rdd.RDD

val seriesX: RDD[Double] = sc.parallelize(Array(1, 2, 3, 3, 5)) // a series

// must have the same number of partitions and cardinality as seriesX

val seriesY: RDD[Double] = sc.parallelize(Array(11, 22, 33, 33, 555))

// compute the correlation using Pearson's method. Enter "spearman" for Spearman's method. If a

// method is not specified, Pearson's method will be used by default.

val correlation: Double = Statistics.corr(seriesX, seriesY, "pearson")

println(s"Correlation is: $correlation")

val data: RDD[Vector] = sc.parallelize(

Seq(

Vectors.dense(1.0, 10.0, 100.0),

Vectors.dense(2.0, 20.0, 200.0),

Vectors.dense(5.0, 33.0, 366.0))

) // note that each Vector is a row and not a column

// calculate the correlation matrix using Pearson's method. Use "spearman" for Spearman's method

// If a method is not specified, Pearson's method will be used by default.

val correlMatrix: Matrix = Statistics.corr(data, "pearson")

println(correlMatrix.toString)

Spark R-studio test on jupyter notebook

Open R-studio GUI and execute the following scrip

library(data.table)

dt <- data.table(1:3)

print(dt)

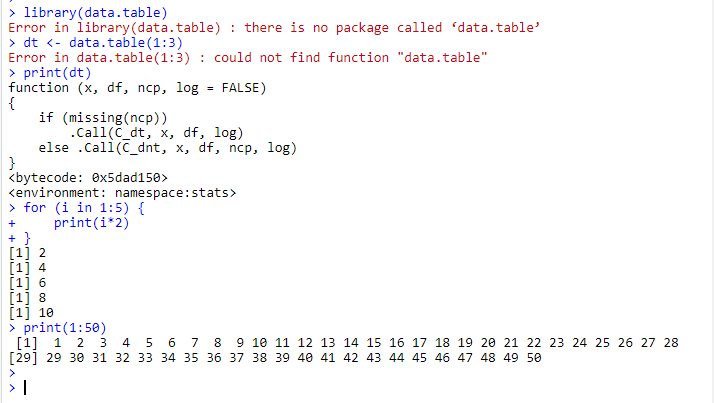
for (i in 1:5) {

print(i\*2)

}

print(1:50)

Output: Sample output is as given below.



# Test cases for Spark with RStudio

* Create a Linux user on the master controller node or login as AD user.
* Login to R-studio

## Base-R testing on RStudio GUI

data(iris) # Load the dataset iris

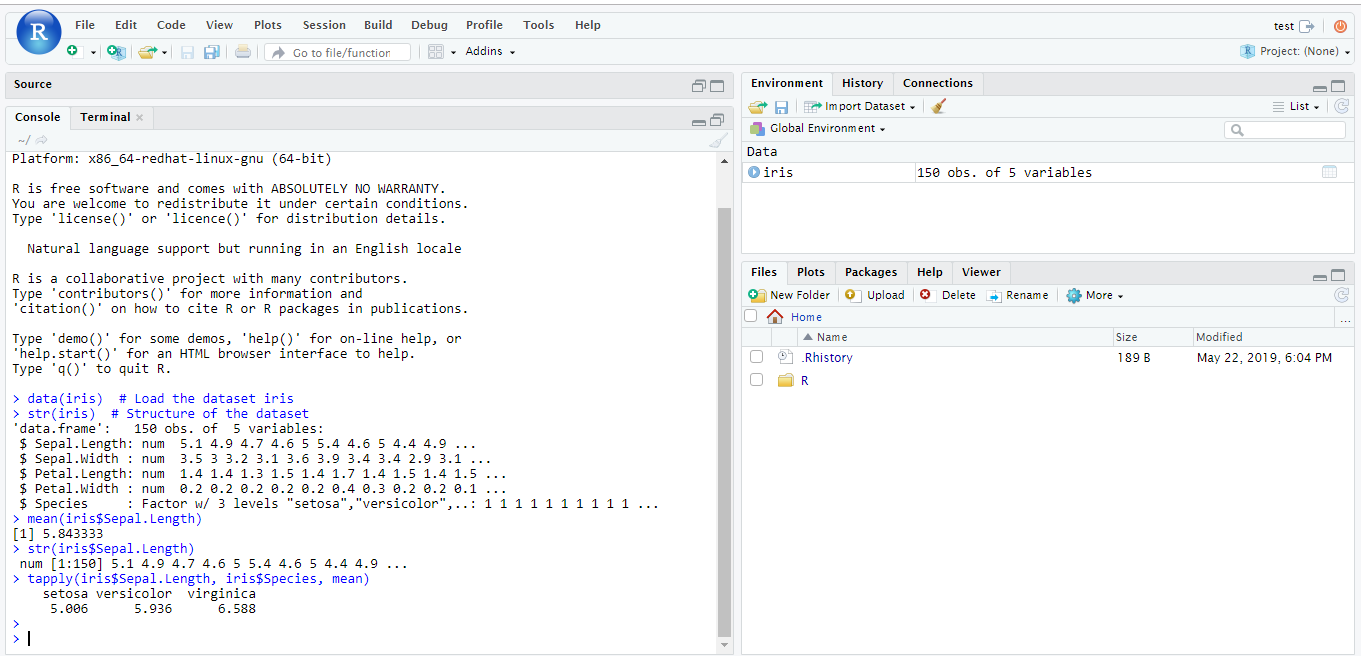
str(iris) # Structure of the dataset

mean(iris$Sepal.Length)

str(iris$Sepal.Length)

tapply(iris$Sepal.Length, iris$Species, mean)

Output: Sample output is as given below.



## Sparklyr testing on RStudio GUI

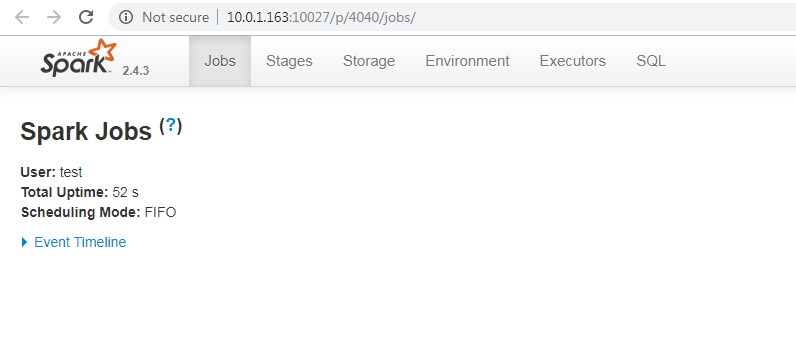
>install.packages("sparklyr")

>sparklyr::spark\_install()

>library(sparklyr)

>sc <- spark\_connect(master = 'local')

Output: Sample output is as given below.



## Simple test on RStudio GUI

data(iris) # Load the dataset iris

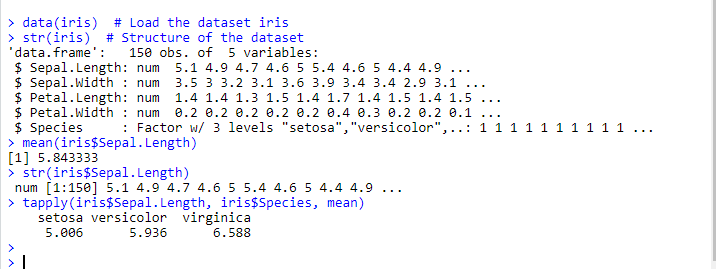
str(iris) # Structure of the dataset

mean(iris$Sepal.Length)

str(iris$Sepal.Length)

tapply(iris$Sepal.Length, iris$Species, mean)

Output: Sample output is as given below



## MLLib usage test on RSudio GUI

>install.packages("sparklyr")

>sparklyr::spark\_install()

>library(sparklyr)

>sc <- spark\_connect(master = 'local')

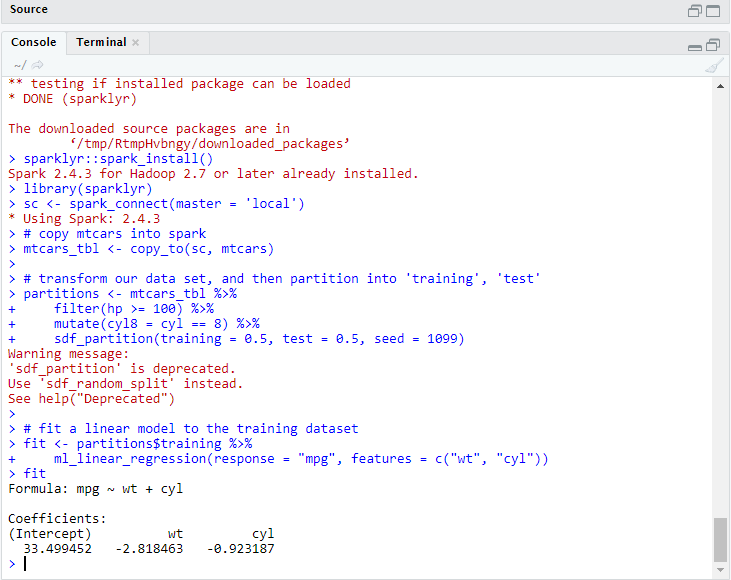
> library(dplyr)

# copy mtcars into spark  
> mtcars\_tbl <- copy\_to(sc, mtcars)

# \*\* May show an error regarding problem with database. Seems to work OK after that  
>src\_tbls(sc)  
# transform our data set, and then partition into 'training', 'test'  
> partitions <- mtcars\_tbl %>%  
 filter(hp >= 100) %>%  
 mutate(cyl8 = cyl == 8) %>%  
 sdf\_partition(training = 0.5, test = 0.5, seed = 1099)  
  
# fit a linear model to the training dataset  
> fit <- partitions$training %>%  
 ml\_linear\_regression(response = "mpg", features = c("wt", "cyl"))

> summary(fit)

Output: Sample output is as given below



# Test cases for Spark with SQL

Spark SQL allows relational queries expressed in SQL or Scala to be executed using Spark. At the core of this component is a new type of RDD, [SchemaRDD](https://spark.apache.org/docs/1.1.0/api/scala/index.html#org.apache.spark.sql.SchemaRDD). SchemaRDDs are composed of [Row](https://spark.apache.org/docs/1.1.0/api/scala/index.html#org.apache.spark.sql.catalyst.expressions.Row) objects, along with a schema that describes the data types of each column in the row. A SchemaRDD is similar to a table in a traditional relational database. The SchemaRDD can be created from an existing RDD, a [Parquet](http://parquet.io/) file, a JSON dataset.

Follow these tests for testing spark-sql for your cluster:

## Testing with user defined functions

* Creating a dataset “hello world”

val dataset = Seq((0, "hello"),(1, "world")).toDF("id","text")

* Defining a function ‘upper’ which converts a string into upper case

val upper: String => String =\_.toUpperCase

* We now import the ‘udf’ package into Spark

import org.apache.spark.sql.functions.udf

* Defining our UDF, ‘upperUDF’ and importing our function ‘upper’

val upperUDF = udf(upper)

* Displaying the results of our User Defined Function in a new column ‘upper’

dataset.withColumn("upper", upperUDF('text)).show

## Starting a Spark Session and displaying DataFrame of people.json

For the querying examples, we will be using files, ’people.txt’ and ’people.json’. These file stored at‘/usr/lib/spark/spark-2.4.0-bin-hadoop2.7/examples/src/main/resources/’

* We first import a Spark Session into Apache Spark

import org.apache.spark.sql.SparkSession

* Creating a Spark Session ‘spark’ using the ‘builder()’ function

val spark = SparkSession.builder().appName("Spark SQL basic example").config("spark.some.config.option", "some-value").getOrCreate()

* Importing the Implicts class into our ‘spark’ Session.

import spark.implicits.\_

* We now create a DataFrame ‘df’ and import data from the ’employee.json’ file.

val df = spark.read.json("examples/src/main/resources/people.json ")

* Displaying the DataFrame ‘df’. The result is a table of 5 rows of ages and names from our ’employee.json’ file.

df.show()

## Creating a Dataset

* Creating a class ‘Employee’ to store name and age of an employee

case class Employee(name: String, age: Long)

* Assigning a Dataset ‘caseClassDS’ to store the record of Andrew

val caseClassDS = Seq(Employee("Andrew", 55)).toDS()

* Displaying the Dataset ‘caseClassDS’

caseClassDS.show()

* Creating a primitive Dataset to demonstrate mapping of DataFrames into Datasets

val primitiveDS = Seq(1, 2, 3).toDS

* Assigning the above sequence into an array

primitiveDS.map(\_ + 1).collect()