```
%md

### The [Apache Mahout](http://mahout.apache.org/)â„¢ project's goal is to build an environment for quickly creating scalable performant machine learning applications.

#### Apache Mahout software provides three major features:

- A simple and extensible programming environment and framework for building scalable algorithms
- A wide variety of premade algorithms for Scala + Apache Spark, H2O, Apache Flink
- Samsara, a vector math experimentation environment with R-like syntax which works at scale

#### In other words:

*Apache Mahout provides a unified API for quickly creating machine learning algorithms on a variety of engines.*

#### Getting Started

Apache Mahout is a collection of Libraries that enhance Apache Flink, Apache Spark, and others. Currently Zeppelin support the Flink and Spark Engines. A convenience script is provided to setup the nesse

We can use Apache Mahout's R-Like Domain Specific Language (DSL) inline with native Flink or Spark code.  We must however, first declare a few imports that are different for Spark and Flink

__References:__

[Mahout-Samsara's In-Core Linear Algebra DSL Reference](http://mahout.apache.org/users/environment/in-core-reference.html)
[Mahout-Samsara's Distributed Linear Algebra DSL Reference](http://mahout.apache.org/users/environment/out-of-core-reference.html)
[Getting Started with the Mahout-Samsara Shell](http://mahout.apache.org/users/sparkbindings/play-with-shell.html)

<h3>The <a href="http://mahout.apache.org/">Apache Mahout</a>â„¢ project's goal is to build an environment for quickly creating scalable performant machine learning applications.</h3>
<h4>Apache Mahout software provides three major features:</h4>
<ul>
<li>A simple and extensible programming environment and framework for building scalable algorithms</li>
<li>A wide variety of premade algorithms for Scala + Apache Spark, H2O, Apache Flink</li>
<li>Samsara, a vector math experimentation environment with R-like syntax which works at scale</li>
</ul>
<h4>In other words:</h4>
<p><em>Apache Mahout provides a unified API for quickly creating machine learning algorithms on a variety of engines.</em></p>
<p><strong>References:</strong></p>
<p><a href="http://mahout.apache.org/users/environment/in-core-reference.html">Mahout-Samsara's In-Core Linear Algebra DSL Reference</a>
<br  /><a href="http://mahout.apache.org/users/environment/out-of-core-reference.html">Mahout-Samsara's Distributed Linear Algebra DSL Reference</a>
<br  /><a href="http://mahout.apache.org/users/sparkbindings/play-with-shell.html">Getting Started with the Mahout-Samsara Shell</a></p>

%md

#### "Installing" the Apache Mahout dependencies and configuring a new Spark and Flink interpreter

The following two paragraphs are convenience paragraphs. You **only need to run them once** to create two new interpreters `%spark.mahout` and `%flink.mahout`. These are intended for users who don't have

They both run a python script which may be found at `ZEPPELIN_HOME/scripts/mahout/add_mahout.py`

In short this script:
- Downloads Apache Mahout
- Creates a new Flink interpreter with dependencies.
- Creates a new Spark interpreter with dependencies and modified configuration to use Kryo serialization.

__You only need to run this script once ever.__ (Maybe again if for some reason you delete `conf/interpreter.json`)

<h4>&ldquo;Installing&rdquo; the Apache Mahout dependencies and configuring a new Spark and Flink interpreter</h4>
<p>The following two paragraphs are convenience paragraphs. You <strong>only need to run them once</strong> to create two new interpreters <code>%spark.mahout</code> and <code>%flink.mahout</code>. These
<p>They both run a python script which may be found at <code>ZEPPELIN_HOME/scripts/mahout/add_mahout.py</code></p>
<p>In short this script:</p>
<ul>
<li>Downloads Apache Mahout</li>
<li>Creates a new Flink interpreter with dependencies.</li>
<li>Creates a new Spark interpreter with dependencies and modified configuration to use Kryo serialization.</li>
</ul>
<p><strong>You only need to run this script once ever.</strong> (Maybe again if for some reason you delete <code>conf/interpreter.json</code>)</p>

%sh

python ../scripts/mahout/add_mahout.py

%sh
python scripts/mahout/add_mahout_interpreters.py

%md

After the interpreters are created you will need to 'bind' them by clicking on the little gear in the top right corner, scrolling to the top, and clicking on `mahoutFlink` and `mahoutSpark` so that they a

#### Running Mahout code

You will need to import certain libraries, and declare the _Mahout Distributed Context_ when you first start your notebook using the interpreters.

If using Apache Flink the code you need to run is:
```scala
%flinkMahout

import org.apache.flink.api.scala._
import org.apache.mahout.math.drm._
import org.apache.mahout.math.drm.RLikeDrmOps._
import org.apache.mahout.flinkbindings._
import org.apache.mahout.math._
import scalabindings._
import RLikeOps._


implicit val ctx = new FlinkDistributedContext(benv)
```

If using Apache Spark the code you need to run is
```scala
%sparkMahout

import org.apache.mahout.math._
import org.apache.mahout.math.scalabindings._
import org.apache.mahout.math.drm._
import org.apache.mahout.math.scalabindings.RLikeOps._
import org.apache.mahout.math.drm.RLikeDrmOps._
import org.apache.mahout.sparkbindings._

implicit val sdc: org.apache.mahout.sparkbindings.SparkDistributedContext = sc2sdc(sc)
```

__Note: For Apache Mahout on Apache Spark you must be running Spark 1.5.x or 1.6.x. We are working hard on supporting Spark 2.0__
In the meantime, feel free to play with Mahout on Flink and then simple _copy and paste your Mahout code to Spark once it is supported!_

A Side by Side Example

<p>After the interpreters are created you will need to 'bind' them by clicking on the little gear in the top right corner, scrolling to the top, and clicking on <code>mahoutFlink</code> and <code>mahoutS
<h4>Running Mahout code</h4>
<p>You will need to import certain libraries, and declare the Mahout Distributed Context when you first start your notebook using the interpreters.</p>
<p>If using Apache Flink the code you need to run is:</p>
<pre><code class="scala">%flinkMahout

import org.apache.flink.api.scala._
import org.apache.mahout.math.drm._
import org.apache.mahout.math.drm.RLikeDrmOps._
import org.apache.mahout.flinkbindings._
import org.apache.mahout.math._
import scalabindings._
import RLikeOps._

@transient implicit val ctx = new FlinkDistributedContext(benv)
</code></pre>
<p>If using Apache Spark the code you need to run is</p>
<pre><code class="scala">%sparkMahout
```

```
import org.apache.mahout.math._
import org.apache.mahout.math.scalabindings._
import org.apache.mahout.math.drm._
import org.apache.mahout.math.scalabindings.RLikeOps._
import org.apache.mahout.math.drm.RLikeDrmOps._
import org.apache.mahout.sparkbindings._

implicit val sdc: org.apache.mahout.sparkbindings.SparkDistributedContext = sc2sdc(sc)
</code></pre>
<p><strong>Note: For Apache Mahout on Apache Spark you must be running Spark 1.5.x or 1.6.x.  We are working hard on supporting Spark 2.0</strong>
<br  />In the meantime, feel free to play with Mahout on Flink and then simple <em>copy and paste your Mahout code to Spark once it is supported!</em></p>
<h3>A Side by Side Example</h3>

%flinkMahout

// Imports and creating the distributed context, similar but not exactly the same ////////////////////////////////////////
import org.apache.flink.api.scala._
import org.apache.mahout.math.drm._
import org.apache.mahout.math.drm.RLikeDrmOps._
import org.apache.mahout.flinkbindings._
import org.apache.mahout.math._
import scalabindings._
import RLikeOps._


implicit val ctx = new FlinkDistributedContext(benv)

// CODE IS EXACTLY THE SAME FROM HERE ON - R-Like DSL ////////////////////////////////////////////////////////////////////////////

val drmData = drmParallelize(dense(
  (2, 2, 10.5, 10, 29.509541),  // Apple Cinnamon Cheerios
  (1, 2, 12,   12, 18.042851),  // Cap'n'Crunch
  (1, 1, 12,   13, 22.736446),  // Cocoa Puffs
  (2, 1, 11,   13, 32.207582),  // Froot Loops
  (1, 2, 12,   11, 21.871292),  // Honey Graham Ohs
  (2, 1, 16,   8,  36.187559),  // Wheaties Honey Gold
  (6, 2, 17,   1,  50.764999),  // Cheerios
  (3, 2, 13,   7,  40.400208),  // Clusters
  (3, 3, 13,   4,  45.811716)), numPartitions = 2)

drmData.collect(::, 0 until 4)

val drmX = drmData(::, 0 until 4)
val y = drmData.collect(::, 4)
val drmXtX = drmX.t %*% drmX
val drmXty = drmX.t %*% y


val XtX = drmXtX.collect
val Xty = drmXty.collect(::, 0)
val beta = solve(XtX, Xty)

import org.apache.flink.api.scala._
import org.apache.mahout.math.drm._
import org.apache.mahout.math.drm.RLikeDrmOps._
import org.apache.mahout.flinkbindings._
import org.apache.mahout.math._
import scalabindings._
import RLikeOps._
ctx: org.apache.mahout.flinkbindings.FlinkDistributedContext = org.apache.mahout.flinkbindings.FlinkDistributedContext@4452b0a5
warning: Class it.unimi.dsi.fastutil.ints.Int2DoubleOpenHashMap not found - continuing with a stub.
drmData: org.apache.mahout.math.drm.CheckpointedDrm[Int] = org.apache.mahout.flinkbindings.drm.CheckpointedFlinkDrm@445242be
(5,9)
res1: org.apache.mahout.math.Matrix =
{
 0 =>   {0:2.0,1:2.0,2:10.5,3:10.0}
 1 =>   {0:1.0,1:2.0,2:12.0,3:12.0}
 2 =>   {0:1.0,1:1.0,2:12.0,3:13.0}
 3 =>   {0:2.0,1:1.0,2:11.0,3:13.0}
 4 =>   {0:1.0,1:2.0,2:12.0,3:11.0}
 5 =>   {0:2.0,1:1.0,2:16.0,3:8.0}
 6 =>   {0:6.0,1:2.0,2:17.0,3:1.0}
 7 =>   {0:3.0,1:2.0,2:13.0,3:7.0}
 8 =>   {0:3.0,1:3.0,2:13.0,3:4.0}
}
drmX: org.apache.mahout.math.drm.DrmLike[Int] = OpMapBlock(org.apache.mahout.flinkbindings.drm.CheckpointedFlinkDrm@445242be,<function1>,4,-1,true)
(5,9)
y: org.apache.mahout.math.Vector = {0:29.509541,1:18.042851,2:22.736446,3:32.207582,4:21.871292,5:36.187559,6:50.764999,7:40.400208,8:45.811716}
drmXtX: org.apache.mahout.math.drm.DrmLike[Int] = OpABAnyKey(OpAt(OpMapBlock(org.apache.mahout.flinkbindings.drm.CheckpointedFlinkDrm@445242be,<function1>,4,-1,true)),OpMapBlock(org.apache.mahout.flinkbi
drmXty: org.apache.mahout.math.drm.DrmLike[Int] = OpAx(OpAt(OpMapBlock(org.apache.mahout.flinkbindings.drm.CheckpointedFlinkDrm@445242be,<function1>,4,-1,true)),{0:29.509541,1:18.042851,2:22.736446,3:32.
(4,4)
XtX: org.apache.mahout.math.Matrix =
{
 0 =>   {0:69.0,1:40.0,2:291.0,3:137.0}
 1 =>   {0:40.0,1:32.0,2:207.0,3:128.0}
 2 =>   {0:291.0,1:207.0,2:1546.25,3:968.0}
 3 =>   {0:137.0,1:128.0,2:968.0,3:833.0}
}
(1,4)
Xty: org.apache.mahout.math.Vector = {0:821.6857190000001,1:549.744517,2:3978.7015895000004,3:2272.7799889999997}
beta: org.apache.mahout.math.Vector = {0:5.247349465378393,1:2.7507945784675067,2:1.1527813010791783,3:0.10312017617607437}

%sparkMahout

// Imports and creating the distributed context, similar but not exactly the same ////////////////////////////////////////

import org.apache.mahout.math._
import org.apache.mahout.math.scalabindings._
import org.apache.mahout.math.drm._
import org.apache.mahout.math.scalabindings.RLikeOps._
import org.apache.mahout.math.drm.RLikeDrmOps._
import org.apache.mahout.sparkbindings._

implicit val sdc: org.apache.mahout.sparkbindings.SparkDistributedContext = sc2sdc(sc)


// CODE IS EXACTLY THE SAME FROM HERE ON - R-Like DSL ////////////////////////////////////////////////////////////////////////////

val drmData = drmParallelize(dense(
  (2, 2, 10.5, 10, 29.509541),  // Apple Cinnamon Cheerios
  (1, 2, 12,   12, 18.042851),  // Cap'n'Crunch
  (1, 1, 12,   13, 22.736446),  // Cocoa Puffs
  (2, 1, 11,   13, 32.207582),  // Froot Loops
  (1, 2, 12,   11, 21.871292),  // Honey Graham Ohs
  (2, 1, 16,   8,  36.187559),  // Wheaties Honey Gold
  (6, 2, 17,   1,  50.764999),  // Cheerios
  (3, 2, 13,   7,  40.400208),  // Clusters
  (3, 3, 13,   4,  45.811716)), numPartitions = 2)

drmData.collect(::, 0 until 4)

val drmX = drmData(::, 0 until 4)
val y = drmData.collect(::, 4)
val drmXtX = drmX.t %*% drmX
val drmXty = drmX.t %*% y


val XtX = drmXtX.collect
val Xty = drmXty.collect(::, 0)
val beta = solve(XtX, Xty)

import org.apache.mahout.math._

import org.apache.mahout.math.scalabindings._
```

```
import org.apache.mahout.math.drm._

import org.apache.mahout.math.scalabindings.RLikeOps._

import org.apache.mahout.math.drm.RLikeDrmOps._

import org.apache.mahout.sparkbindings._

sdc: org.apache.mahout.sparkbindings.SparkDistributedContext = org.apache.mahout.sparkbindings.SparkDistributedContext@32c46474

drmData: org.apache.mahout.math.drm.CheckpointedDrm[Int] = org.apache.mahout.sparkbindings.drm.CheckpointedDrmSpark@783484b9
```

```
res2: org.apache.mahout.math.Matrix =
{
 0 =>   {0:2.0,1:2.0,2:10.5,3:10.0}
 1 =>   {0:1.0,1:2.0,2:12.0,3:12.0}
 2 =>   {0:1.0,1:1.0,2:12.0,3:13.0}
 3 =>   {0:2.0,1:1.0,2:11.0,3:13.0}
 4 =>   {0:1.0,1:2.0,2:12.0,3:11.0}
 5 =>   {0:2.0,1:1.0,2:16.0,3:8.0}
 6 =>   {0:6.0,1:2.0,2:17.0,3:1.0}
 7 =>   {0:3.0,1:2.0,2:13.0,3:7.0}
 8 =>   {0:3.0,1:3.0,2:13.0,3:4.0}
}

drmX: org.apache.mahout.math.drm.DrmLike[Int] = OpMapBlock(org.apache.mahout.sparkbindings.drm.CheckpointedDrmSpark@783484b9,<function1>,4,-1,true)

y: org.apache.mahout.math.Vector = {0:29.509541,1:18.042851,2:22.736446,3:32.207582,4:21.871292,5:36.187559,6:50.764999,7:40.400208,8:45.811716}

drmXtX: org.apache.mahout.math.drm.DrmLike[Int] = OpABAnyKey(OpAt(OpMapBlock(org.apache.mahout.sparkbindings.drm.CheckpointedDrmSpark@783484b9,<function1>,4,-1,true)),OpMapBlock(org.apache.mahout.sparkbi

drmXty: org.apache.mahout.math.drm.DrmLike[Int] = OpAx(OpAt(OpMapBlock(org.apache.mahout.sparkbindings.drm.CheckpointedDrmSpark@783484b9,<function1>,4,-1,true)),{0:29.509541,1:18.042851,2:22.736446,3:32.
```

```
XtX: org.apache.mahout.math.Matrix =
{
 0 =>   {0:69.0,1:40.0,2:291.0,3:137.0}
 1 =>   {0:40.0,1:32.0,2:207.0,3:128.0}
 2 =>   {0:291.0,1:207.0,2:1546.25,3:968.0}
 3 =>   {0:137.0,1:128.0,2:968.0,3:833.0}
}

Xty: org.apache.mahout.math.Vector = {0:821.6857190000001,1:549.744517,2:3978.7015894999995,3:2272.779989}

beta: org.apache.mahout.math.Vector = {0:5.247349465378446,1:2.750794578467531,2:1.1527813010791554,3:0.10312017617608908}
```

```
%md

### Taking advantage of Zeppelin Resource Pools

One of the major motivations for integrating Apache Mahout with Apache Zeppelin was the many benefits that come from leveraging the resource pools.  A resource pool is a block of memory that can be accce

The Spark interpreter has a simple interface for accessing the ResourcePools, the Flink interface is less documented but can be reverse engineered from code (thanks open source!)


Collect betas from Spark and Flink- compare in Python

Create Matrix in Flink and Spark - visualize with R

<h3>Taking advantage of Zeppelin Resource Pools</h3>
<p>One of the major motivations for integrating Apache Mahout with Apache Zeppelin was the many benefits that come from leveraging the resource pools.  A resource pool is a block of memory that can be ac
<p>The Spark interpreter has a simple interface for accessing the ResourcePools, the Flink interface is less documented but can be reverse engineered from code (thanks open source!)</p>
<p>Collect betas from Spark and Flink- compare in Python</p>
<p>Create Matrix in Flink and Spark - visualize with R</p>
```

```
%flinkMahout

import org.apache.zeppelin.interpreter.InterpreterContext

val resourcePool = InterpreterContext.get().getResourcePool()

resourcePool.put("flinkBeta", beta.asFormatString)
```

```
import org.apache.zeppelin.interpreter.InterpreterContext
resourcePool: org.apache.zeppelin.resource.ResourcePool = org.apache.zeppelin.resource.DistributedResourcePool@3fdd93cc
```

```
%sparkMahout




z.put("sparkBeta", beta.asFormatString)
```

```
%spark.pyspark

import ast

flinkBetaDict = ast.literal_eval(z.get("flinkBeta"))
sparkBetaDict = ast.literal_eval(z.get("sparkBeta"))

print "----------------- differences between betas calulated in Flink and Spark-----------------"
for i in range(0,4):
    print "beta", i, ": " , flinkBetaDict[i] - sparkBetaDict[i]
```

```
----------------- differences between betas calulated in Flink and Spark-----------------
beta 0 :  -5.24025267623e-14
beta 1 :  -2.44249065418e-14
beta 2 :  2.28705943073e-14
beta 3 :  -1.47104550763e-14
```

```
%md

## Plotting Mahout with R

The following examples show how we can leverage R to plot our results from Mahout

<h2>Plotting Mahout with R</h2>
<p>The following examples show how we can leverage R to plot our results from Mahout</p>
```

```
%flinkMahout
val mxRnd = Matrices.symmetricUniformView(5000, 2, 1234)
val drmRand = drmParallelize(mxRnd)


val drmSin = drmRand.mapBlock() {case (keys, block) =>
  val blockB = block.like()
  for (i <- 0 until block.nrow) {
    blockB(i, 0) = block(i, 0)
    blockB(i, 1) = Math.sin((block(i, 0) * 8))
```

```
  }
  keys -> blockB
}

resourcePool.put("flinkSinDrm", drm.drmSampleToTSV(drmSin, 0.85))

mxRnd: org.apache.mahout.math.Matrix =
{
 0 =>   {0:0.4586377101191827,1:0.07261898163580698}
 1 =>   {0:0.48977896201757654,1:0.2695201068510176}
 2 =>   {0:0.33215452109376786,1:0.2148377346657124}
 3 =>   {0:0.4497098649240723,1:0.4331127334380502}
 4 =>   {0:-0.03782634247193647,1:-0.32353833540588983}
 5 =>   {0:0.15137106418749705,1:0.422446220403861}
 6 =>   {0:0.2714115385692545,1:-0.4495233989067956}
 7 =>   {0:0.02468155133492185,1:0.49474128114887833}
 8 =>   {0:-0.2269662536373416,1:-0.14808249195411455}
 9 =>   {0:0.050870692759856756,1:-0.4797329808849356}
... }
drmRand: org.apache.mahout.math.drm.CheckpointedDrm[Int] = org.apache.mahout.flinkbindings.drm.CheckpointedFlinkDrm@72c5b7be
drmSin: org.apache.mahout.math.drm.DrmLike[Int] = OpMapBlock(org.apache.mahout.flinkbindings.drm.CheckpointedFlinkDrm@72c5b7be,<function1>,-1,-1,true)
(2,5000)

%sparkMahout
val mxRnd = Matrices.symmetricUniformView(5000, 2, 1234)
val drmRand = drmParallelize(mxRnd)


val drmSin = drmRand.mapBlock() {case (keys, block) =>
  val blockB = block.like()
  for (i <- 0 until block.nrow) {
    blockB(i, 0) = block(i, 0)
    blockB(i, 1) = Math.sin((block(i, 0) * 8))
  }
  keys -> blockB
}

z.put("sparkSinDrm", org.apache.mahout.math.drm.drmSampleToTSV(drmSin, 0.85))

mxRnd: org.apache.mahout.math.Matrix =
{
 0 =>   {0:0.4586377101191827,1:0.07261898163580698}
 1 =>   {0:0.48977896201757654,1:0.2695201068510176}
 2 =>   {0:0.33215452109376786,1:0.2148377346657124}
 3 =>   {0:0.4497098649240723,1:0.4331127334380502}
 4 =>   {0:-0.03782634247193647,1:-0.32353833540588983}
 5 =>   {0:0.15137106418749705,1:0.422446220403861}
 6 =>   {0:0.2714115385692545,1:-0.4495233989067956}
 7 =>   {0:0.02468155133492185,1:0.49474128114887833}
 8 =>   {0:-0.2269662536373416,1:-0.14808249195411455}
 9 =>   {0:0.050870692759856756,1:-0.4797329808849356}
... }

drmRand: org.apache.mahout.math.drm.CheckpointedDrm[Int] = org.apache.mahout.sparkbindings.drm.CheckpointedDrmSpark@1d6a6ecf

drmSin: org.apache.mahout.math.drm.DrmLike[Int] = OpMapBlock(org.apache.mahout.sparkbindings.drm.CheckpointedDrmSpark@1d6a6ecf,<function1>,-1,-1,true)

%spark.r {"imageWidth": "400px"}

library("ggplot2")

flinkSinStr = z.get("flinkSinDrm")
sparkSinStr = z.get("sparkSinDrm")

flinkData <- read.table(text= flinkSinStr, sep="\t", header=FALSE)
sparkData <- read.table(text= sparkSinStr, sep="\t", header=FALSE)

plot(flinkData,  col="red")
# Graph trucks with red dashed line and square points
points(sparkData, col="blue")

# Create a title with a red, bold/italic font
title(main="Sampled Mahout Sin Graph in R", col.main="black", font.main=4)

legend("bottomright", c("Apache Flink", "Apache Spark"), col= c("red", "blue"), pch= c(22, 22))
```

<p><img src="

```
%flinkMahout

val mxRnd3d = Matrices.symmetricUniformView(5000, 3, 1234)
val drmRand3d = drmParallelize(mxRnd3d)

val drmGauss = drmRand3d.mapBlock() {case (keys, block) =>
  val blockB = block.like()
  for (i <- 0 until block.nrow) {
    val x: Double = block(i, 0)
    val y: Double = block(i, 1)
    val z: Double = block(i, 2)

    blockB(i, 0) = x
    blockB(i, 1) = y
    blockB(i, 2) = Math.exp(-((Math.pow(x, 2)) + (Math.pow(y, 2)))/2)
  }
  keys -> blockB
}

resourcePool.put("flinkGaussDrm", drm.drmSampleToTSV(drmGauss, 50.0))

mxRnd3d: org.apache.mahout.math.Matrix =
{
 0 =>   {0:0.4586377101191827,1:0.07261898163580698,2:-0.4120814898385057}
 1 =>   {0:0.48977896201757654,1:0.2695201068510176,2:0.2035624121801051}
 2 =>   {0:0.33215452109376786,1:0.2148377346657124,2:0.22923597484837382}
 3 =>   {0:0.4497098649240723,1:0.4331127334380502,2:-0.26063522630725094}
 4 =>   {0:-0.03782634247193647,1:-0.32353833540588983,2:-0.4423256266785404}
 5 =>   {0:0.15137106418749705,1:0.422446220403861,2:-0.20452218901606223}
 6 =>   {0:0.2714115385692545,1:-0.4495233989067956,2:0.13402344186662743}
 7 =>   {0:0.02468155133492185,1:0.49474128114887833,2:-0.484577970998106}
 8 =>   {0:-0.2269662536373416,1:-0.14808249195411455,2:-0.16159073199184967}
 9 =>   {0:0.050870692759856756,1:-0.4797329808849356,2:0.30230792168515175}
... }
drmRand3d: org.apache.mahout.math.drm.CheckpointedDrm[Int] = org.apache.mahout.flinkbindings.drm.CheckpointedFlinkDrm@448a1f4e
drmGauss: org.apache.mahout.math.drm.DrmLike[Int] = OpMapBlock(org.apache.mahout.flinkbindings.drm.CheckpointedFlinkDrm@448a1f4e,<function1>,-1,-1,true)
(3,5000)

%spark.r {"imageWidth": "400px"}

library(scatterplot3d)


flinkGaussStr = z.get("flinkGaussDrm")
flinkData <- read.table(text= flinkGaussStr, sep="\t", header=FALSE)

scatterplot3d(flinkData, color="green")



%md

**NOTE** To install `scatterplot3d` on Ubuntu use:

```sh
sudo apt-get install r-cran-scatterplot3d
```
```

```
<p><strong>NOTE</strong> To install <code>scatterplot3d</code> on Ubuntu use:</p>
<pre><code class="sh">sudo apt-get install r-cran-scatterplot3d
</code></pre>

%md
```