Making your packages accessible to non-programmer collaborators using the VisRseq framework

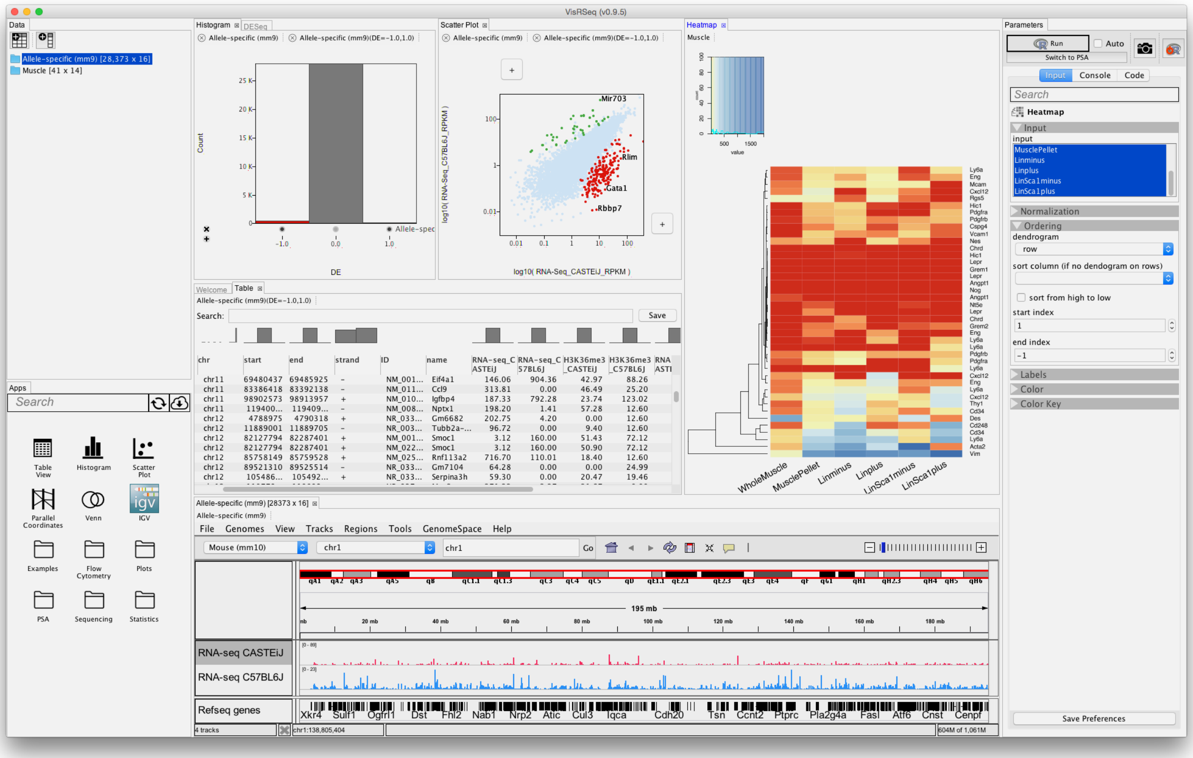
Hamid Younesy

June 26, 2016 | BioC 2016

Abstract

I will introduce the VisRseq framework and walk the participants through the quick process of creating modules called R-apps from R packages. I expect this to be specially useful for bioinformaticians and package developers that develop R-based analysis tools and would like to make them accessible to their non-programmer collaborators or to the public without having to spend significant time on creating extensive graphical user interfaces. I will walk the participants through several examples of creating diverse types of apps, from simple plotting (e.g. plots) to intermediate (e.g. clustering) to more advanced (e.g. edgeR and DEseq) packages. I will also show how several R-apps can be linked together to create more complex workflows. Participants will require having beginner/intermediate knowledge of R and a machine with R and Java installation.

VisRseq Overview



Allele-specific (mm9)(DE=-1.0,1.0)

chr	start	end	strand	ID	name	RNA-seq_CASTEiJ	RNA-seq_C57BL6J	H3K36me3_CASTEiJ	H3K36me3_C57BL6J
chr11	69480437	69485925	-	NM_001...	Eif4a1	146.06	904.36	42.97	88.26
chr11	83386418	83392138	-	NM_011...	Ccl9	313.81	0.00	46.49	25.20
chr11	98902573	98913957	+	NM_010...	Igfbp4	187.33	792.28	23.74	123.02
chr11	119400...	119409...	-	NM_008...	Nptx1	198.20	1.41	57.28	12.60
chr12	4788975	4790318	+	NR_033...	Gm6682	202.75	4.20	0.00	12.60
chr12	11889001	11889705	-	NR_003...	Tubb2a-...	96.72	0.00	9.40	12.60
chr12	82127794	82287401	+	NM_001...	Smoc1	3.12	160.00	51.43	72.12
chr12	82127794	82287401	+	NM_022...	Smoc1	3.12	160.00	50.90	72.12
chr12	85758149	85759528	+	NM_025...	Rnf113a2	716.70	110.01	18.40	12.60
chr12	89521310	89525514	-	NR_033...	Gm7104	64.28	0.00	0.00	24.99
chr12	105486...	105492...	+	NR_033...	Serpina3h	59.30	0.00	20.47	19.46

Allele-specific (mm9) [28373 x 16]

Figure 1:

Goals for this workshop

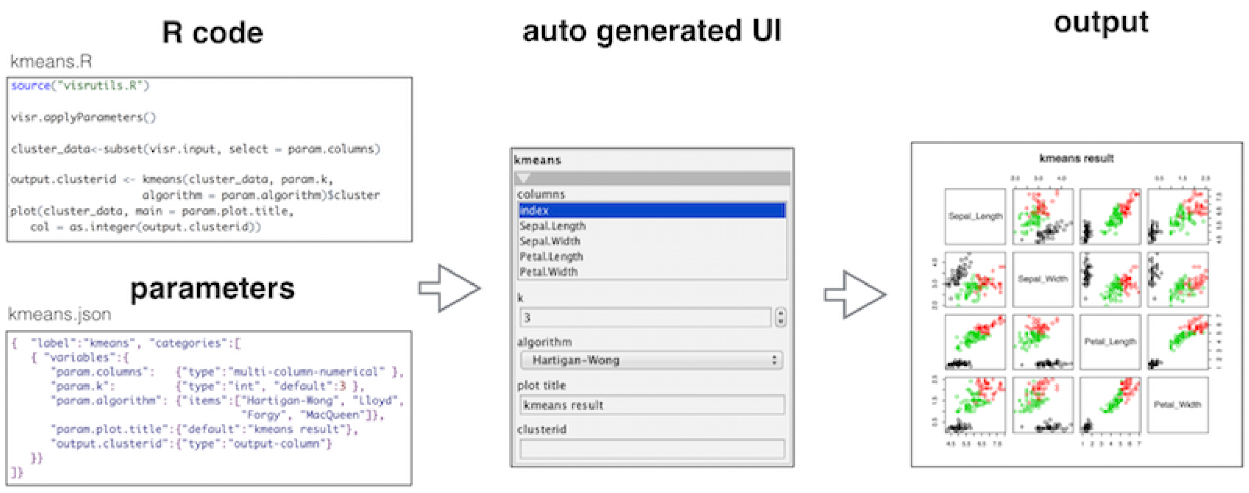
- Learn about VisRseq framework.

- Learn how to create R-apps through several examples.

Requirements

To get the most out of this workshop you are recommended to install the latest version of VisRseq from visrseq.github.io/downloads. You can still follow the instructions and create R-apps using plain R, however you will require VisRseq to be able to run the apps and see the results.

Structure of an R-App



kmeans.R

```
source("visrutils.R")

visr.applyParameters()

cluster_data<-subset(visr.input, select = param.columns)

output.clusterid <- kmeans(cluster_data, param.k,
                    algorithm = param.algorithm)$cluster
plot(cluster_data, main = param.plot.title,
   col = as.integer(output.clusterid))
```

kmeans.json

```
{  "label":"kmeans", "categories":[
   { "variables":{
     "param.columns":   {"type":"multi-column-numerical" },
     "param.k":         {"type":"int", "default":3 },
     "param.algorithm": {"items":["Hartigan-Wong", "Lloyd",
                                  "Forgy", "MacQueen"]},
     "param.plot.title":{"default":"kmeans result"},
     "output.clusterid":{"type":"output-column"}
   }}
]}
```

kmeans
columns: index, Sepal.Length, Sepal.Width, Petal.Length, Petal.Width
k: 3
algorithm: Hartigan-Wong
plot title: kmeans result
clusterid:

Figure 2:

To create an R app you need a `.R` code containing the main script together with a `.json` file specifying parameters. Each R app can have any number of "parameters", grouped into "categories". The `.json` file can be created manually or using the helper function in `visrutils.R`

```
source("https://raw.githubusercontent.com/hyounesy/visr-apps/master/visrutils.R")
```

which defines the following functions among other things.

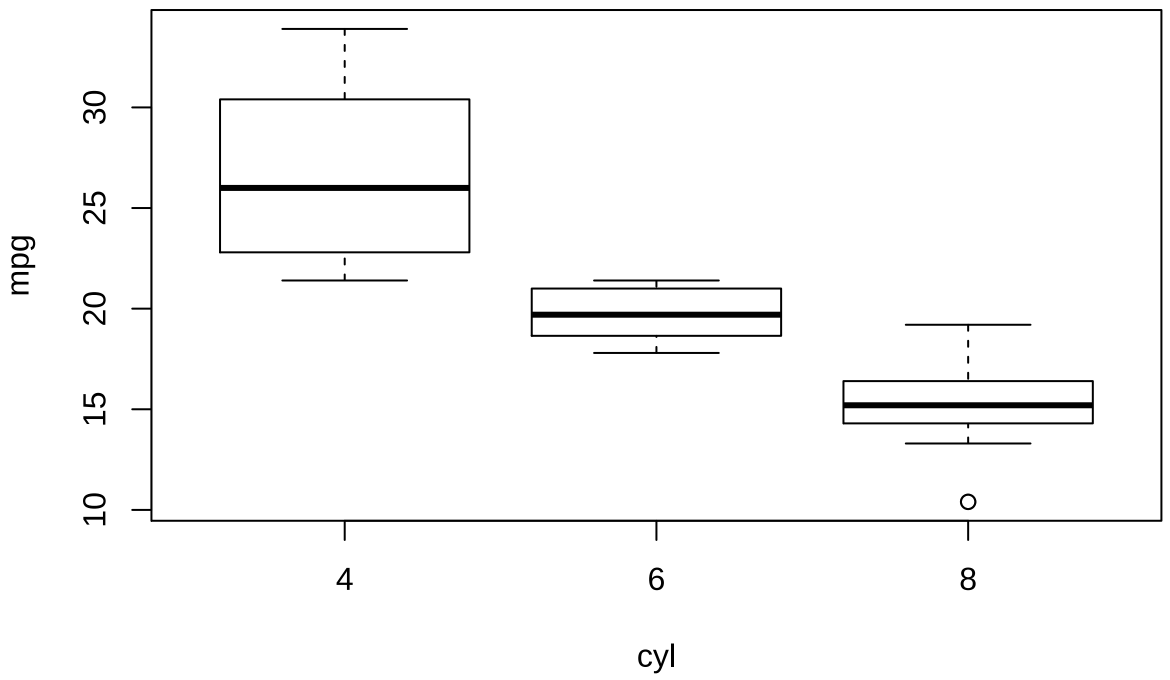
```
visr.app.start(name, info = "", debugdata = NULL)
visr.category(label, info = "")
visr.app.end(printjson = FALSE, writefile = FALSE, filename = NULL)
visr.param (name, label = NULL, info = NULL,
            default = NULL, min = NULL, max = NULL,
            items = NULL, item.labels = NULL,
            type = c("string", "character", "int", "integer",
                     "double", "boolean", "logical", "multi-string",
                     "column", "multi-column",
                     "column-numerical", "multi-column-numerical",
                     "color", "multi-color", "filename",
                     "output-column", "output-multi-column", "output-table"),
            filename.mode = c("load", "save", "dir"), debugvalue = NULL)
```

Let's get started and create our first app!

Example 1: Boxplot

Let's start simple by creating a basic app that draws a barplot. Let's first write a basic boxplot script in a file `simple_boxplot.R`:

```
boxplot(mtcars[["mpg"]]~mtcars[["cyl"]], xlab = "cyl", ylab = "mpg")
```



Assume we would like to allow user to use their own data set and be able to specify x and y columns from their data.

We start the app definition by calling `visr.app.start` and specifying the app name. If we want be able to debug this app in R, we pass a test data set to `debugdata`. This parameter will be ignored when running the app in VisRseq. The data assigned to an app will be stored in `visr.input` variable.

```
visr.app.start("Simple Boxplot", debugdata=mtcars)
```

We then add two parameters: `y` the numeric vector of data values that will be split into groups according to the parameter `group`. The parameter `y` has to be a numerical column in the input data, so we specify the `type="column-numerical"`. The parameter `group` can be any column, so we specify `type="column"`. In order to be able the debug the app inside the R, we also specify the values using `debugvalue`. This is optional but recommended.

```
visr.param("y", type="column-numerical", debugvalue="mpg")
visr.param("group", type="column", debugvalue="cyl")
```

This will create two parameters `visr.param.y` and `visr.param.group`. When running the script in R, they will be initialized with their corresponding debugvalue column names. When running the app in VisRseq, they will contain the string name of the column selected in the GUI.

We end the definition by calling `visr.app.end`. If argument `printjson == TRUE`, the json data for the app parameters will be printed to console. If argument `writefile == TRUE`, the output `.json` file will be written to a file specified by `filename` argument. If `filename` is not specified, the json file will be created at the same path of the R script, and same name but `.json` extension (i.e. `simple_boxplot.json`).

```
visr.app.end(printjson=TRUE, writefile = FALSE, filename = NULL)
```

```
## {
##   "label": "Simple Boxplot",
##   "info": "",
##   "categories":[  {
##     "label": "",
##     "info": "",
##     "variables": {
##       "visr.param.y": {
##         "type": "column-numerical"
##       },
##       "visr.param.group": {
##         "type": "column"
##       }
##     }
##   }]
## }
```

We now have the json file. So we just have to modify our original R script to use the correct parameter names.

```
visr.applyParameters()

boxplot(visr.input[[visr.param.y]]~visr.input[[visr.param.group]],
        xlab = visr.param.group, ylab = visr.param.y)
```

Done! now if we place the `simple_boxplot.R` and `simple_boxplot.json` inside `VisRseq/visr/srcR/bioc2016` and hit the `Refresh` icon above the apps pane, we will have our first custom app in VisRseq under the `bioc2016` folder. For demonstration, here we have used the app to draw a boxplot for the iris dataset.

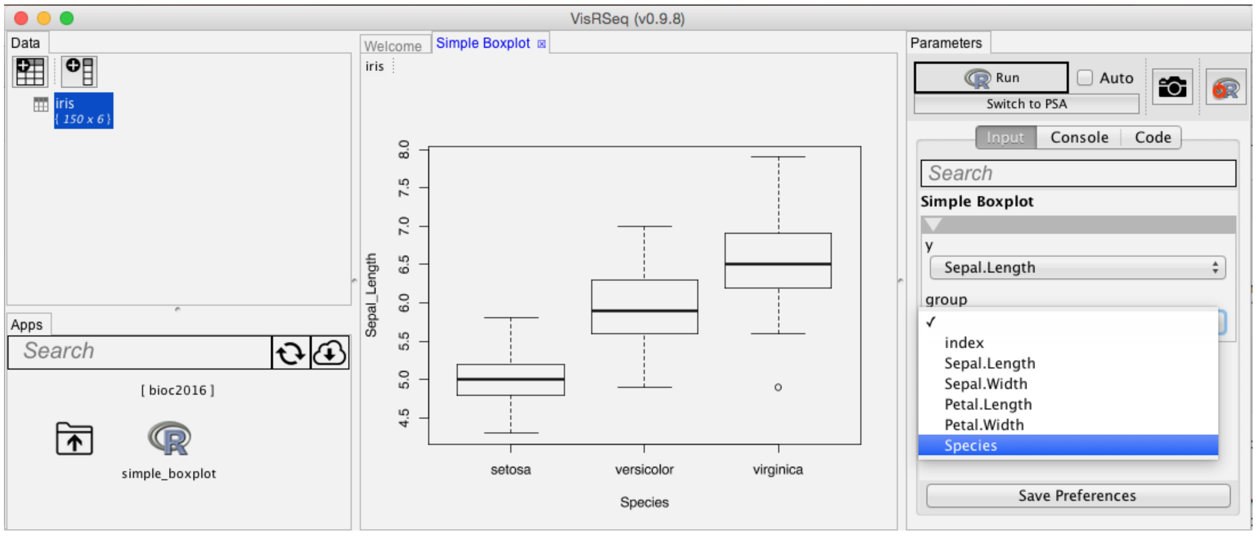


Figure 3:

Below is the complete source code for `simple_boxplot.R` app:

```
source("visrutils.R")
# parameters
visr.app.start("Simple Boxplot", debugdata=mtcars)
visr.param("y", type="column-numerical", debugvalue="mpg")
visr.param("group", type="column", debugvalue="cyl")
visr.app.end(printjson=TRUE, writefile = FALSE, filename = NULL)
visr.applyParameters()

# code
boxplot(visr.input[[visr.param.y]]~visr.input[[visr.param.group]],
        xlab = visr.param.group,
        ylab = visr.param.y)
```

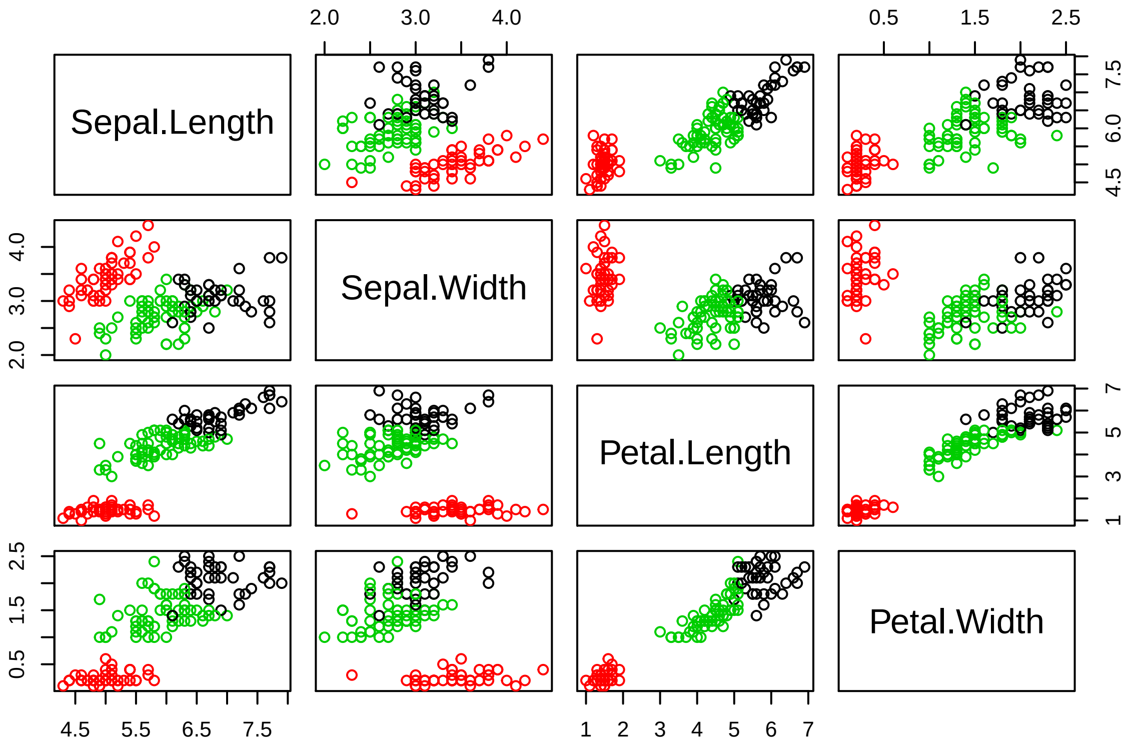
Exercise: Violin Plot

Let's do an exercise and create another app to draw violin plots.

Example 2: Kmeans

We will now make something more interesting: A kmeans clustering app. Like the previous example, we will start with the R script for kmeans clustering.

```
input <- iris
columns <- c("Sepal.Length", "Sepal.Width", "Petal.Length", "Petal.Width")
cluster_data<-subset(input, select = columns)
clusterid <- kmeans(cluster_data, centers = 3)$cluster
plot(cluster_data, col = as.integer(clusterid))
```



```
# start parameter definition
visr.app.start("Simple Kmeans", debugdata = iris)
visr.category("clustering parameters")
visr.param("columns", type = "multi-column-numerical",
           debugvalue = c("Sepal.Length", "Sepal.Width", "Petal.Length", "Petal.Width"))
visr.param("k", default = 3)
visr.param("algorithm", items = c("Hartigan-Wong", "Lloyd", "Forgy", "MacQueen"))
```

```r
visr.category("output")
visr.param("plot.title", default = "kmeans results")
visr.param("output.clusterid", type = "output-column")
visr.app.end(printjson=TRUE)
```

```
## {
##   "label": "Simple Kmeans",
##   "info": "",
##   "categories":[  {
##     "label": "clustering parameters",
##     "info": "",
##     "variables": {
##       "visr.param.columns": {
##         "type": "multi-column-numerical"
##       },
##       "visr.param.k": {
##         "type": "int",
##         "default": 3
##       },
##       "visr.param.algorithm": {
##         "type": "string",
##         "items": [ "Hartigan-Wong","Lloyd","Forgy","MacQueen" ]
##       }
##     }
##   },
##   {
##     "label": "output",
##     "info": "",
##     "variables": {
##       "visr.param.plot.title": {
##         "type": "string",
##         "default": "kmeans results"
##       },
##       "visr.param.output.clusterid": {
##         "type": "output-column"
##       }
##     }
##   }]
## }
```

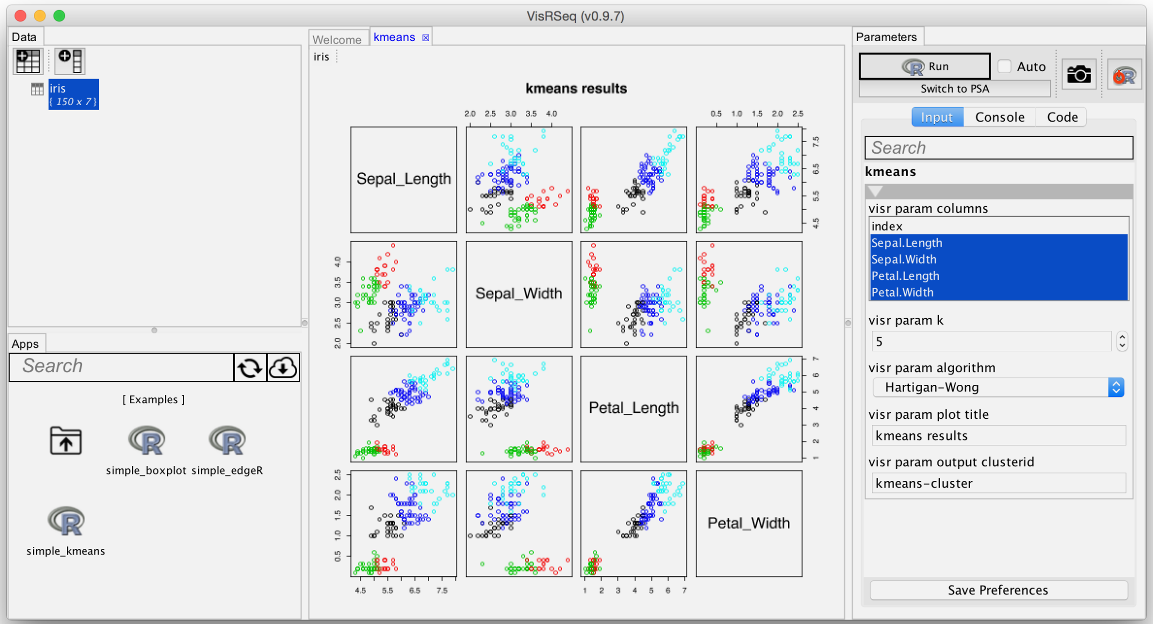
```r
visr.applyParameters()

cluster_data<-subset(visr.input, select = visr.param.columns)
visr.param.output.clusterid <- kmeans(cluster_data,
                                      centers = visr.param.k,
                                      algorithm = visr.param.algorithm)$cluster
plot(cluster_data, main = visr.param.plot.title,
     col = as.integer(visr.param.output.clusterid))
```

Below is the complete source code for `simple_kmeans.R` app:

```r
source("visrutils.R")

#  parameters
visr.app.start("Simple Kmeans", debugdata = iris)
```



visr param columns
index
Sepal.Length
Sepal.Width
Petal.Length
Petal.Width

visr param k
5

visr param algorithm
Hartigan-Wong

visr param plot title
kmeans results

visr param output clusterid
kmeans-cluster

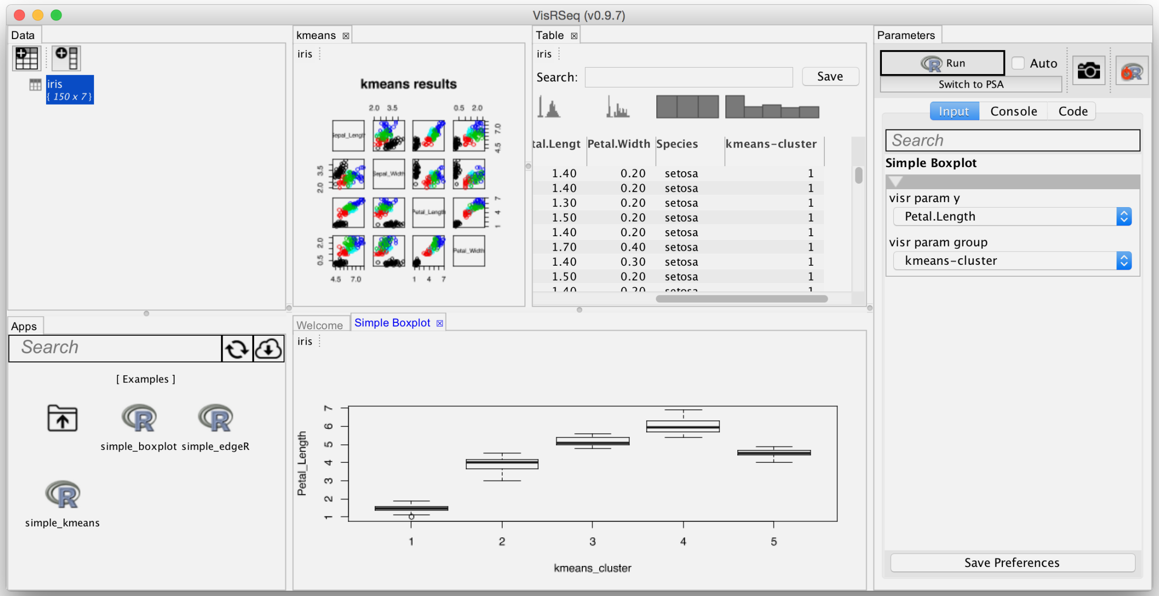
Figure 4:

```r
visr.category("clustering parameters")
visr.param("columns", type = "multi-column-numerical",
           debugvalue = c("Sepal.Length", "Sepal.Width", "Petal.Length", "Petal.Width"))
visr.param("k", default = 3)
visr.param("algorithm", items = c("Hartigan-Wong", "Lloyd", "Forgy", "MacQueen"))
visr.category("output")
visr.param("plot.title", default = "kmeans results")
visr.param("output.clusterid", type = "output-column")
visr.app.end(printjson=TRUE)
visr.applyParameters()

# kmeans code
cluster_data<-subset(visr.input, select = visr.param.columns)
visr.param.output.clusterid <- kmeans(cluster_data,
                                      centers = visr.param.k,
                                      algorithm = visr.param.algorithm)$cluster
plot(cluster_data, main = visr.param.plot.title,
     col = as.integer(visr.param.output.clusterid))
```

Chaining apps

When running kmeans inside VisRseq on a test data, the result of the clustering will be stored in a new column specified by output cluster id. We can preview this new column in a table view app, or even better we can now use that with the boxplot app we created earlier.



tal.Lengt	Petal.Width	Species	kmeans-cluster
1.40	0.20	setosa	1
1.40	0.20	setosa	1
1.30	0.20	setosa	1
1.50	0.20	setosa	1
1.40	0.20	setosa	1
1.70	0.40	setosa	1
1.40	0.30	setosa	1
1.50	0.20	setosa	1

visr param y: Petal.Length

visr param group: kmeans-cluster

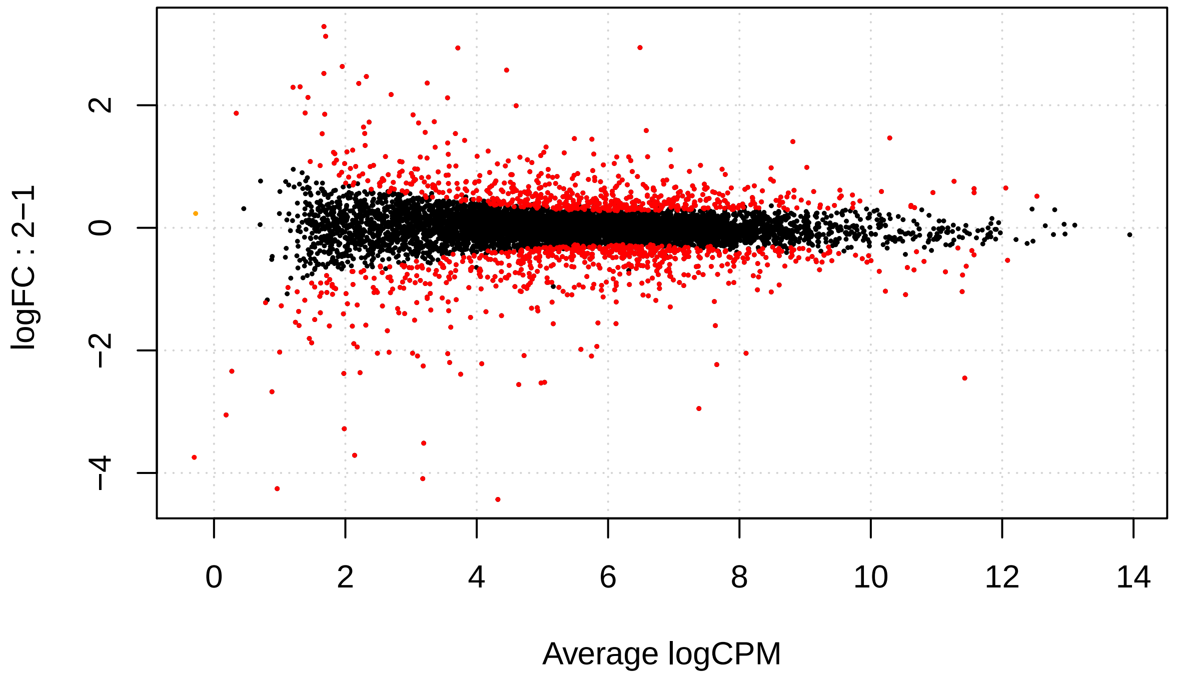
Example 3: Differntial expression analysis using edgeR

```
library("edgeR", quietly=T)
countdata = read.table(
  "https://raw.githubusercontent.com/hyounesy/bioc2016.visrseq/master/data/counts.txt",
  header=T, row.names = 1)
x <- countdata
head(x)
```

```
##             CT.PA.1 CT.PA.2 KD.PA.3 KD.PA.4 CT.SI.5 KD.SI.6 CT.SI.7
## FBgn0000008      76      71      87      68     137     115      82
## FBgn0000017    3498    3087    3029    3264    7014    4322    3926
## FBgn0000018     240     306     288     307     613     528     485
## FBgn0000032     611     672     694     757    1479    1361    1351
## FBgn0000042   40048   49144   70574   72850   97565   95760   99372
## FBgn0000043   15910   18194   31086   34085   34171   42389   29671
```

```
group1 <- c("CT.PA.1", "CT.PA.2")
group2 <- c("KD.PA.3", "KD.PA.4")
groups <- factor(c(rep(1, length(group1)), rep(2, length(group2)))) # c(1, 1, 2, 2)
# create edgeR's container for RNA-seq count data
y <- DGEList(counts=x[, c(group1, group2)], group = groups)
# estimate normalization factors
y <- calcNormFactors(y)
# estimate tagwise dispersion (simple design)
y <- estimateCommonDisp(y)
y <- estimateTagwiseDisp(y)
# test for differential expression using classic edgeR approach
et <- exactTest(y)
# total number of DE genes in each direction
is.de <- decideTestsDGE(et, adjust.method = "BH", p.value = 0.05, lfc = 0)
```

```r
# The log-fold change for each gene is plotted against the average abundance
plotSmear(y, de.tags=rownames(y)[is.de!=0])
```

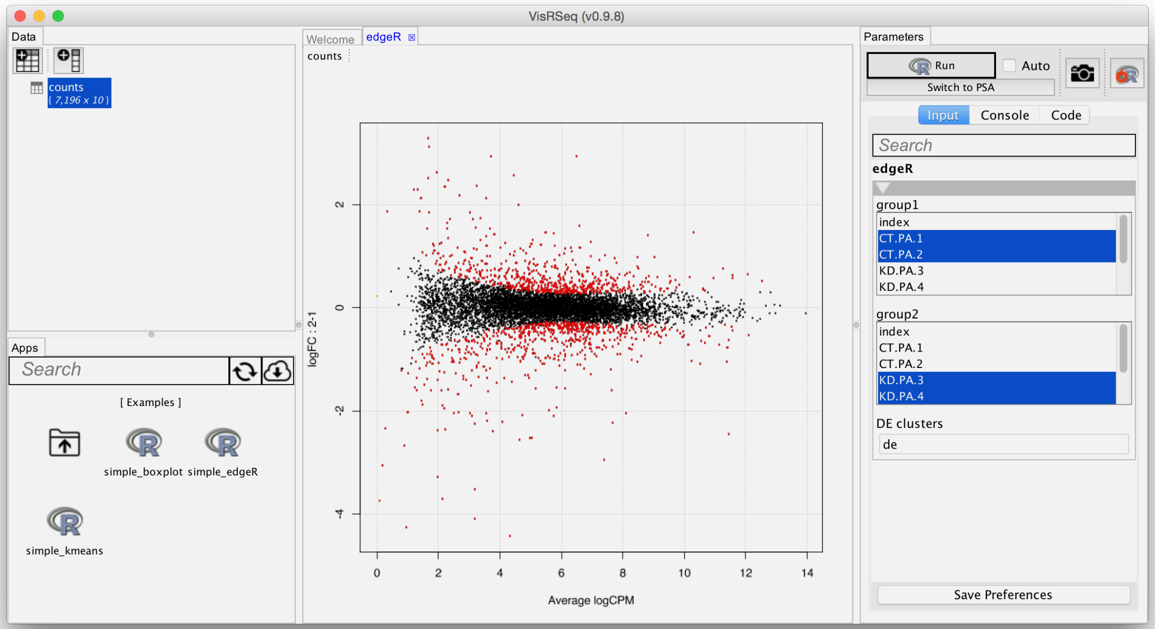


```r
## edgeR parameters
visr.app.start("edgeR", debugdata = countdata)
visr.param("group1", type = "multi-column-numerical", debugvalue = c("CT.PA.1", "CT.PA.2"))
visr.param("group2", type = "multi-column-numerical", debugvalue = c("KD.PA.3", "KD.PA.4"))
visr.param("output.de", label = "DE clusters", type = "output-column")
visr.app.end(printjson=TRUE, writefile=T)
```

```
## {
##   "label": "edgeR",
##   "info": "",
##   "categories":[  {
##     "label": "",
##     "info": "",
##     "variables": {
##       "visr.param.group1": {
##         "type": "multi-column-numerical"
##       },
##       "visr.param.group2": {
##         "type": "multi-column-numerical"
##       },
##       "visr.param.output.de": {
##         "label": "DE clusters",
##         "type": "output-column"
##       }
##     }
##   }]
## }
```

```r
visr.applyParameters()
```

```
## edgeR code
library("edgeR")
x <- visr.input
groups <- factor(c(rep(1, length(visr.param.group1)),  # e.g. 1,1
                   rep(2, length(visr.param.group2)))) # e.g. 2,2
# create edgeR's container for RNA-seq count data
y <- DGEList(counts=x[, c(visr.param.group1, visr.param.group2)], group = groups)
# estimate normalization factors
y <- calcNormFactors(y)
# estimate tagwise dispersion (simple design)
y <- estimateCommonDisp(y)
y <- estimateTagwiseDisp(y)
# test for differential expression using classic edgeR approach
et <- exactTest(y)
# total number of DE genes in each direction
is.de <- decideTestsDGE(et, adjust.method = "BH", p.value = 0.05, lfc = 0)
# export the results to VisRseq
visr.param.output.de <- as.factor(is.de)
# The log-fold change for each gene is plotted against the average abundance
plotSmear(y, de.tags = rownames(y)[is.de != 0])
```



edgeR

group1

index
CT.PA.1
CT.PA.2
KD.PA.3
KD.PA.4

group2

index
CT.PA.1
CT.PA.2
KD.PA.3
KD.PA.4

DE clusters

de

Figure 5:

Below is the complete source code for `simple_edgeR.R` app:

```
source("visrutils.R")

## edgeR parameters
visr.app.start("edgeR", debugdata = countdata)
visr.param("group1", type = "multi-column-numerical", debugvalue = c("CT.PA.1", "CT.PA.2"))
```

```r
visr.param("group2", type = "multi-column-numerical", debugvalue = c("KD.PA.3", "KD.PA.4"))
visr.param("output.de", label = "DE clusters", type = "output-column")
visr.app.end(printjson=TRUE, writefile=T)
visr.applyParameters()


## edgeR code
library("edgeR")
x <- visr.input
groups <- factor(c(rep(1, length(visr.param.group1)),  # e.g. 1,1
                   rep(2, length(visr.param.group2)))) # e.g. 2,2
# create edgeR's container for RNA-seq count data
y <- DGEList(counts=x[, c(visr.param.group1, visr.param.group2)], group = groups)
# estimate normalization factors
y <- calcNormFactors(y)
# estimate tagwise dispersion (simple design)
y <- estimateCommonDisp(y)
y <- estimateTagwiseDisp(y)
# test for differential expression using classic edgeR approach
et <- exactTest(y)
# total number of DE genes in each direction
is.de <- decideTestsDGE(et, adjust.method = "BH", p.value = 0.05, lfc = 0)
# export the results to VisRseq
visr.param.output.de <- as.factor(is.de)
# The log-fold change for each gene is plotted against the average abundance
plotSmear(y, de.tags = rownames(y)[is.de != 0])
```