Before we begin

1. Open R/Rstudio or whatever you use.
2. Prepare a folder for the workshop and set it as working directory.
3. Install packages and download the data.

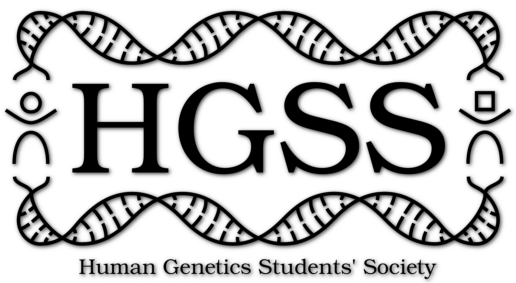
Instructions and slides available at
https://github.com/jmonlong/HGSS_Rworkshops/ (in the Advanced-Tidyverse-Bioconductor-2018 folder).

HGSS R Workshop : Tidyverse and Bioconductor

Jean Monlong

Human Genetics department

March 26, 2018



Sponsored by the McGill Initiative in Computational Medicine (MiCM)

Today's topic

- Manipulating, analyzing and visualizing large *data.frame* in the Tidyverse.
- Tricks: cleaner and faster code.

- Access and manipulate genomics ranges.
- Tricks: Gene enrichment analysis, heatmaps.

Disclaimer

- Some things might be too technical. Follow what you can.
- Feel free to interrupt or suggest other ways.

- Lunch break: 12:15pm - 1:15pm

data.table package and fread

fread function

+ Very fast.
+ Usually no need for additional parameters.
- Has its specific format (*data.table*)...
+ ... which can be converted into *data.frame* .
+ Very fast.

Example

```
library(data.table)
myDT = fread("myFile.tsv")
myDF = as.data.frame(myDT)

myDT = fread("gunzip -c myFile.tsv.gz")
```

Exercise

1. Read the Gencode file (`gencodeForWorkshop.tsv.gz`) with `read.table`.
2. Same with `fread`.
3. Have a look at the data.
4. For each gene type, compute
   - the number of genes
   - the average gene size
   - the proportion of genes larger than 1 Kbp

   Hint: one data.frame with the results.
5. 4. again but for chr Y only.

Avoid loops, use sapply/lapply

+ Avoid manual init/update of objects.
+ No temporary object polluting the environment.
+ More optimized.
+ Easy to parallelize.
- More painful to debug.
- An error and everything must be computed again.

```
perm.l = lapply(1:1000, function(ii){
  data.frame(perm=ii, est=..SOMETHING..)
})

perm.df = do.call(rbind, perm.l)
## or
perm.df = as.data.frame(rbindlist(perm.l))
```

Minimize copy-pasting, use functions

+ Easier to propagate changes.
+ Easier to reuse in another analysis.
+ Cleaner/smaller code.

```
superFun <- function(df, arg1, opt2=0){
  ## Something with df, arg1 and opt2
  return(XXX)
}
```

Exercise

Improve the previous exercise with functions.

Parallel processing

Easiest solution with *parallel* package

- Using `mclapply` instead of `lapply`.
- `mc.cores=` the number of processors to use.

Example

```
perm.l = mclapply(1:1000, function(ii){
  data.frame(perm=ii, est=..SOMETHING..)
}, mc.cores=4)
```

Exercise

Parallelize the previous exercise.

The Tidyverse

https://www.tidyverse.org/

R packages for data science

The tidyverse is an opinionated **collection of R packages** designed for data science. All packages share an underlying design philosophy, grammar, and data structures.

Install the complete tidyverse with:

```
install.packages("tidyverse")
```

data.frames

- Mix between *matrix* and *list*
- Array form.
- Columns can have different data types.

matrix

```
      samp1 samp2 samp3
gene1  -1.3  -1.8  -4.1
gene2  -1.5  -1.2   4.9
```

data.frame

```
 gene sample expression
gene1  samp1       -1.3
gene2  samp1       -1.5
gene1  samp2       -1.8
gene2  samp2       -1.2
gene1  samp3       -4.1
gene2  samp3        4.9
```

Pros/Cons

+ Dense representation of large data.
- Accepts only one data type.
- manual combination with other information often required.

+ Flexible.
+ Accepts several data types.
+ Can represent all the data needed for an analysis.
- Takes more space/memory due to repetitions.

dplyr package

"A Grammar of Data Manipulation"

dplyr provides functions which can be combined for data manipulation.

mutate	add a new column using others.
filter	filter rows (similar as `subset` function).
select	select specific columns only.
arrange	order rows using specific columns.
group_by	groups rows according to specific columns.
summarize	summarizes each group of rows.
do	applies a function to a group of rows.

\+ Works with pipes.

\+ Fast.

\- Has its own format *tbl_df*...

\+ ... which is almost the same as *data.frame* .

Pipes are cool !

- Pipe functions instead of embedding them.
- More readable.
- Easier to combine several functions.
- Avoid temporary objects.
- Pipe argument %>%.

Example

```
head(sort(round(sqrt(myVec))))
myVec %>% sqrt %>% round %>% sort %>% head

head(sort(round(sqrt(myVec),digits=3),decreasing=TRUE),10)
myVec %>% sqrt %>% round(digits=3) %>%
                          sort(decreasing=TRUE) %>% head(10)
```

Grouping rows

Operation by block

- Using `group_by()` function.
- Further operations are applied separately per group of rows.

Example

```
myDF %>% group_by(colA) %>% summarize(colB.mean=mean(colB))

myDF %>% group_by(colA, colB) %>% summarize(nbAB=n())

myDF %>% group_by(colAB) %>% summarize(nbAB=n()) %>%
                  ungroup %>% arrange(desc(nbAB)) %>% head
```

Tips

- `n()` gives the number of rows in the group.
- `ungroup` removes groups.
- `desc()` means descending order (in `arrange()`).

Exercise

1. Print the top 10 genes with the most exons.

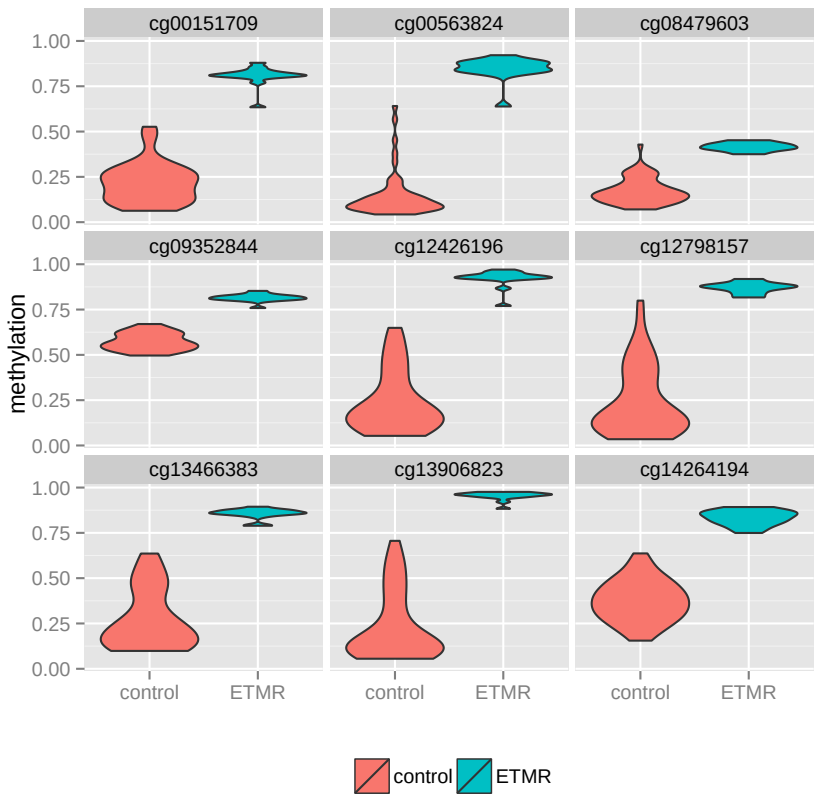
2. Same exercise as before with `group_by` instead of `lapply`.
   - For each gene type, compute
     - the number of genes
     - the average gene size
     - the proportion of genes larger than 1 Kbp
   - Idem for chr Y only.

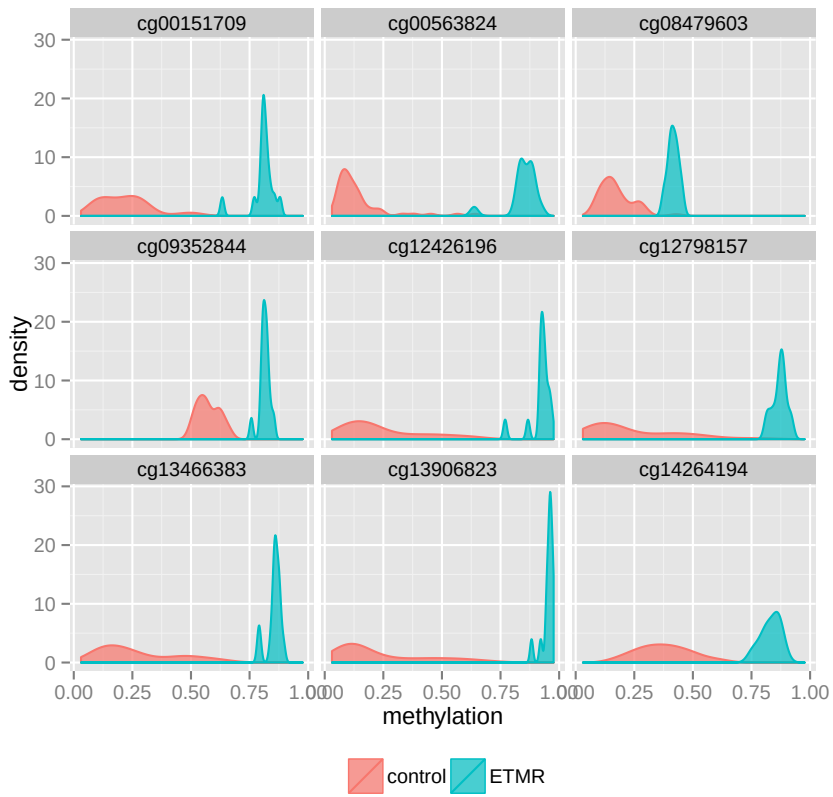
3. Idem with `summarize` (i.e. no functions).

ggplot2 package

Introduction

A package to construct pretty and/or complex graphs. Many aspects of the graph are arranged automatically but everything can be customized. Easy layers addition.





ggplot2

Input : *data.frame*

- Each row represents one *"observation"*.
- Columns represent the different information about the *"observations"*.

Concept

- Start with a `ggplot(...)` and the input *data.frame* .
- `aes(...)` defines how to use the input's columns.
- Add layers : `geom_*(...)`, `scale_*(...)`, ...

Example

```
library(ggplot2)
ggplot(myDf, aes(x=colA, y=colB, colour=colC, linetype=colD))
   + geom_point() + geom_line() + scale_y_log10()
```

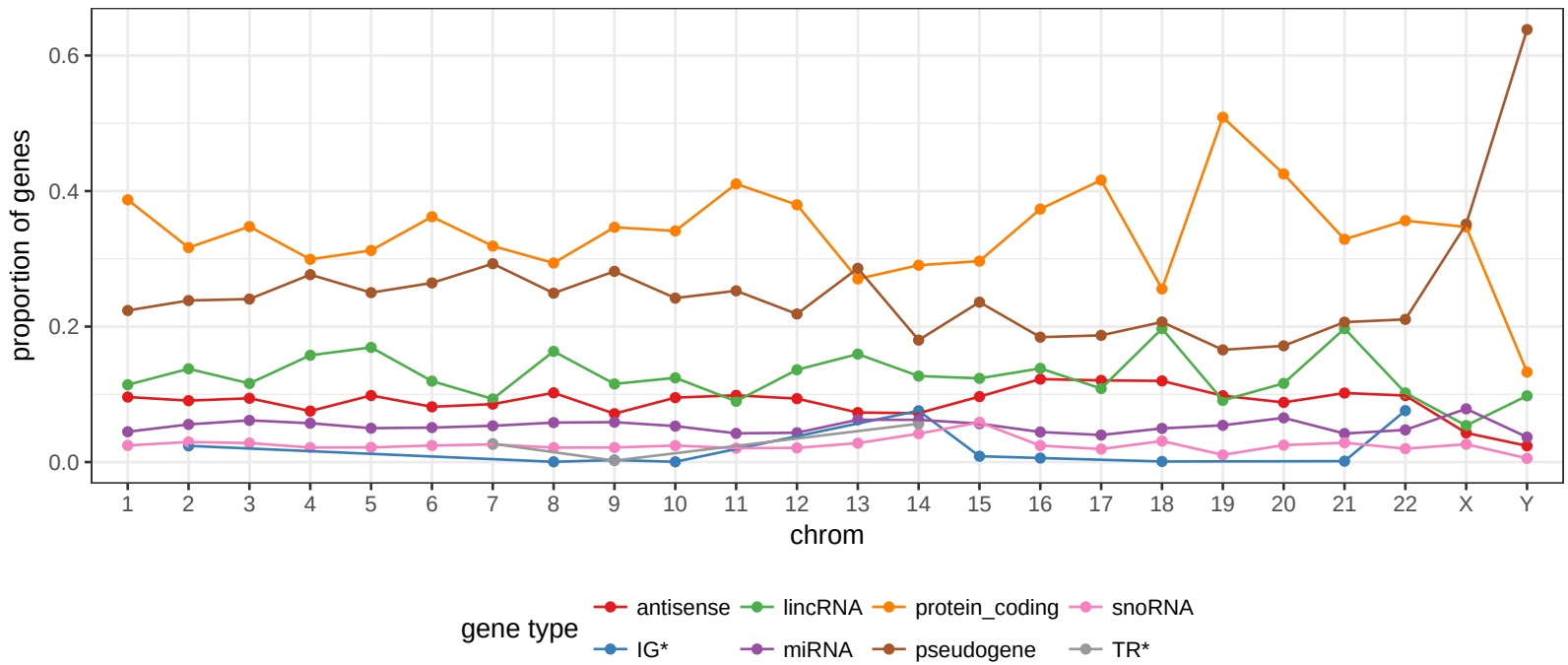
Useful online resources

- http://docs.ggplot2.org/current/
- http://www.cookbook-r.com/Graphs/

Exercise

1. Show the distribution of the gene size (histogram), colored by gene type.
2. Show for each chromosome the number of genes, colored by gene type.

⁂ Plot the proportion of gene types in each chromosome.



Tips: simplify (e.g. IG/TR*) gene types and keep most common ones.*

Lunch Break

Take-home messages

- `fread` instead of `read.table`.
- `lapply` instead of loops (also easy to parallelize).
- Functions instead of copy-pasting.
- *data.frames* and the Tidyverse for exploration and visualization.

More in appendix

- Reading a file chunk by chunk.
- Working with file slices for BED/VCF/BAM files using indexing.
- Using computing clusters in R.

GenomicRanges package

Genomic regions can be represented by *GRanges* objects. Used in many packages/analysis.

```
library(GenomicRanges)

## Creating GRanges
gr = GRanges('chr1:103-404')
gr = GRanges('chr1', IRanges(103, 404))
gr = makeGRangesFromDataFrame(df, keep.extra.columns = TRUE)

## Changing the chromosome labels
seqlevels(genc) = gsub("chr","",seqlevels(genc))
## or
seqlevels(genc) = paste0("chr",seqlevels(genc))

## Other
width(gr)
promoters(gr)
resize(gr, width=width(gr)+1000, fix='center')
```

AnnotationHub

The AnnotationHub package allows you to access genomic files and anotations (e.g. UCSC files, Epigenome RoadMap, Encode).

- `query` function to search/list resources.
- Download by accessing the element.
- Eventually, `import` to import BigWig files.

```
library(AnnotationHub)
ah = AnnotationHub()
his.q = query(ah, c('Gm12878','H3K4me3','hg19'))
his.gr = his.q[[1]]

## If BigWig file, import only relevant region.
his.gr = import(his.q[[1]], which=reg.gr)
```

Exercise

1. Create a *GRanges* with the protein-coding genes in chr 1.
2. Create a *GRanges* with the promoters protein-coding genes in chr 1. Promoter regions must be 1bp-long.

3. Find and download broad H3K4me3 peaks for Gm12878.

4. Find and download CpG islands for Hg19.
5. Expand CpG islands by 10 Kbp using the `resize` function.

6. Find and download Whole Genome Bisulfite Sequencing methylation data for Hg19.
7. Retrieve methylation information around 1000 randomly selected CpG islands (from question 5).

Enrichment heatmaps

Bioconductor package EnrichedHeatmap creates heatmap of overlaps between two *GRanges*. Typical example: Binding sites on known domains.

Example

```
mathm = normalizeToMatrix(H3K4me3, tss, value_column = "coverage",
    extend = 5000, mean_mode = "w0", w = 50)
EnrichedHeatmap(mathm, name = "H3K4me3")

mathm = normalizeToMatrix(meth, cgi, value_column = "meth", mean_
    mode = "absolute", extend = 5000, w = 50, background = NA)
EnrichedHeatmap(mathm)
```

Exercise

1. Heatmap of the histone mark around genes promoters in chr 1.
2. Heatmap of the methylation level in and around the 1000 random CpG islands.

Overlaps between two *GRanges* sets

Which function fit your exact need ?

overlapsAny Test overlaps of one *GRanges* into second *GRanges* .

countOverlaps For each region in one *GRanges* , count how many overlaps from another.

findOverlaps Finds overlaps between two *GRanges* objects.

distanceToNearest Computes the distance from each regions in a *GRanges* object to the nearest in another *GRanges* object.

subsetByOverlaps Keep the regions from one *GRanges* that overlaps another.

Overlaps between two *GRanges* sets

findOverlaps function

- Two *GRanges* objects as input.
- Extra parameters available for specific overlaps.
- Returns the index of regions in object 1 and 2 that overlap.
- `queryHits` and `subjectHits` functions to retrieves those index.

Example

```
> ol12 = findOverlaps(gr1, gr2)
> ol12
Hits of length 3
queryLength: 6
subjectLength: 4
  queryHits subjectHits
   <integer>   <integer>
 1         1           1
 2         2           2
 3         5           3
> queryHits(ol12)
[1] 1 2 5
```

Better one big overlap than many small ones

Exercise

For each gene type, how many genes overlap an H3K4me3 histone mark in Gm12878 ?

Hints

`lapply`, `tapply`, *dplyr*, GRanges$\leftrightarrow$*data.frame*

GenomicRanges + dplyr

Convert the results of `findOverlaps` to a *data.frame* and pipe it to *dplyr*.

```
findOverlaps(gr1, gr2) %>% as.data.frame %>%
    mutate(col1=gr1$col1[queryHits], ...subjectHits...) %>%
       group_by(col1) %>% summarize(...)
```

Exercise

1. In one pipe/line: For each gene type, how many genes overlap an H3K4me3 histone mark in Gm12878 ?

Gene Ontology enrichment

The clusterProfiler package provides functions for Gene Ontology enrichment and Gene Set Enrichment Analysis.

```
library(clusterProfiler)
library(org.Hs.eg.db)

go.enr = enrichGO(gene=sig.entrez, 'org.Hs.eg.db', ont="BP",
   universe=all.entrez, readable=TRUE)

go.enr.s = as.data.frame(go.enr)
head(go.enr.s[,c("Description","GeneRatio","qvalue")],20)

## With gene names (in theory)
go.enr = enrichGO(gene=sig.symbol, 'org.Hs.eg.db', ont="BP",
   universe=all.symbol, readable=TRUE, keyType='SYMBOL')
```

Convert gene names to Entrez IDs

Some functions work better with Entrez IDs.

```
library(clusterProfiler)
library(org.Hs.eg.db)

conv.df <- bitr(gene.symbols, fromType = "SYMBOL",
        toType = c("ENTREZID"),
        OrgDb = org.Hs.eg.db)

symbolToEntrez = conv.df$ENTREZID
names(symbolToEntrez) = conv.df$SYMBOL

sig.entrez = symbolToEntrez[sig.symbol]
```

Exercise

1. Find genes overlapping H3K4me3 marks with a score larger than 900.
2. GO enrichment analysis on these genes.

3. If you get an error (or for "fun"), try to convert the gene names to Entrez IDs.

Gviz for multi-track graphs

```
library(Gviz)

region.gr = GRanges('chr3:110e6-116e6')
annot.reg.gr = subsetByOverlaps(annot.all.gr, region.gr)
data.reg.gr = subsetByOverlaps(data.all.gr, region.gr)

ga.t = GenomeAxisTrack()
annot.t = AnnotationTrack(annot.reg.gr, name = "Annot", feature=
    annot.reg.gr$color, group=annot.reg.gr$name)
data.t = DataTrack(data.reg.gr, data="score", type='h', name="Data")

plotTracks(list(ga.t, annot.t, data.t), showId=TRUE, my_feature_name
    ='red')
```

More info

See slides/code/links from MonBUG Gviz demo.

Exercise

Try to make a graph of the gene annotation and histone mark scores in chr3 between 110 Mbp and 116 Mbp.

Final recommendations

- Use **names** that makes sense (to you and future you).
- **Nothing in the console**, everything in an organized script.
- The script should be **sequential and commented** when complex.
- Save the graphs **in the code**, not manually through RStudio.
- **Split** long scripts and **save** temporary files.
- **Overwriting** objects is fine if in the same paragraph.
- Use **functions/pipes** to avoid environment/code pollution.
- Use **R Markdown** to produce a readable report while keeping the code.

Appendix

Chunk-by-chunk approach

When you can analyze the data in slices.

+ Only a slice of the file in memory.
- A bit painful/ugly.

```
con = file(file.name)
while(length((chunk.df = read.table(con,nrows=1000)))>0){
... Instructions
}
close(con)
```

Bioconductor packages

- GFF format with `import` (*rtracklayer* package).
- VCF format with `readVcf` (*VariantAnnotation* package).

\+ Parse the format.

\- Sometimes parse too much → complicated object.

\+ Read indexed files.

Exercise

1. Read `gencode.gtf.gz` in another object using `import`.
2. Have a look at the data.

Using file indexing

Why indexing ?

To **quickly import a slice** of a file. In genomics, one region.

Indexing workflow

1. Order file by position (chr + start).
2. Compress with bgzip.
3. Index.

How ?

With command lines or with R functions.

Indexing files with R

data.table package is faster to order large files than conventional R.

```
library(data.table)
dt = fread("file.bed")
setkey(dt, chr, start)
write.table(dt, file="file-ordered.bed")

library(Rsamtools)
bgzip("file-ordered.bed")
indexTabix("file-ordered.bed.bgz")
```

Using indexed files

```
reg = GRanges(...)

library(VariantAnnotation)
vcf <- readVcf("variants.vcf.gz", "hg19", reg)

library(rtracklayer)
gtf = import(TabixFile("annotation.gtf.bgz"), which=reg)
```

Exercise

1. Read variants in VCF between coordinates 30 Mb and 31 Mb.
2. Order, write, compress and index the gencode file.

⁎ Tile the Mb in 10 bins. In each bin count the number of variants in the VCF.

Using computing clusters directly with *BatchJobs*

What you need

- A function that can independently the code
  - Load packages.
  - Load necessary data.
  - Run the code.
- A parameter list (used to define the jobs).
- Some global objects (same for all jobs). Optional.
- Your favorite cluster configured (or your computer).

More info

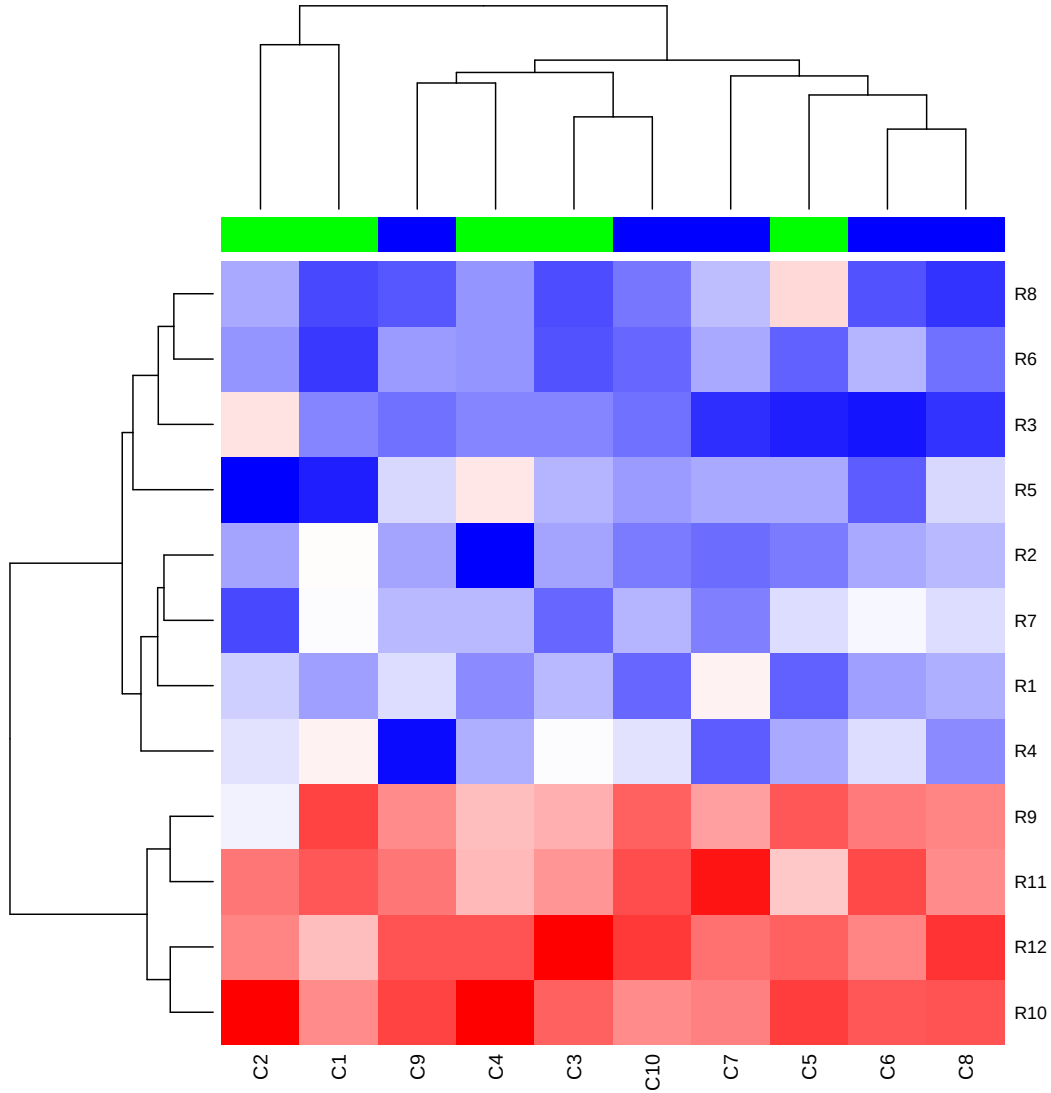
Checkout PopSV documentation on BatchJobs. To use Guillimin, Abacus, Briaree or Mammouth ask me for the configuration files.

BatchJobs example

```
library(BatchJobs)

reg = makeRegistry("perm")

jobFun <- function(ii, necessaryData){
    library(...)
    ... Instructions using 'ii' and 'necessaryData'
    data.frame(perm=ii, ...)
}

batchMap(reg, jobFun, 1:10, more.args=list(necessaryData=myData))

submitJobs(reg))

showStatus(reg)

perm.l = reduceResultsList(reg)
```

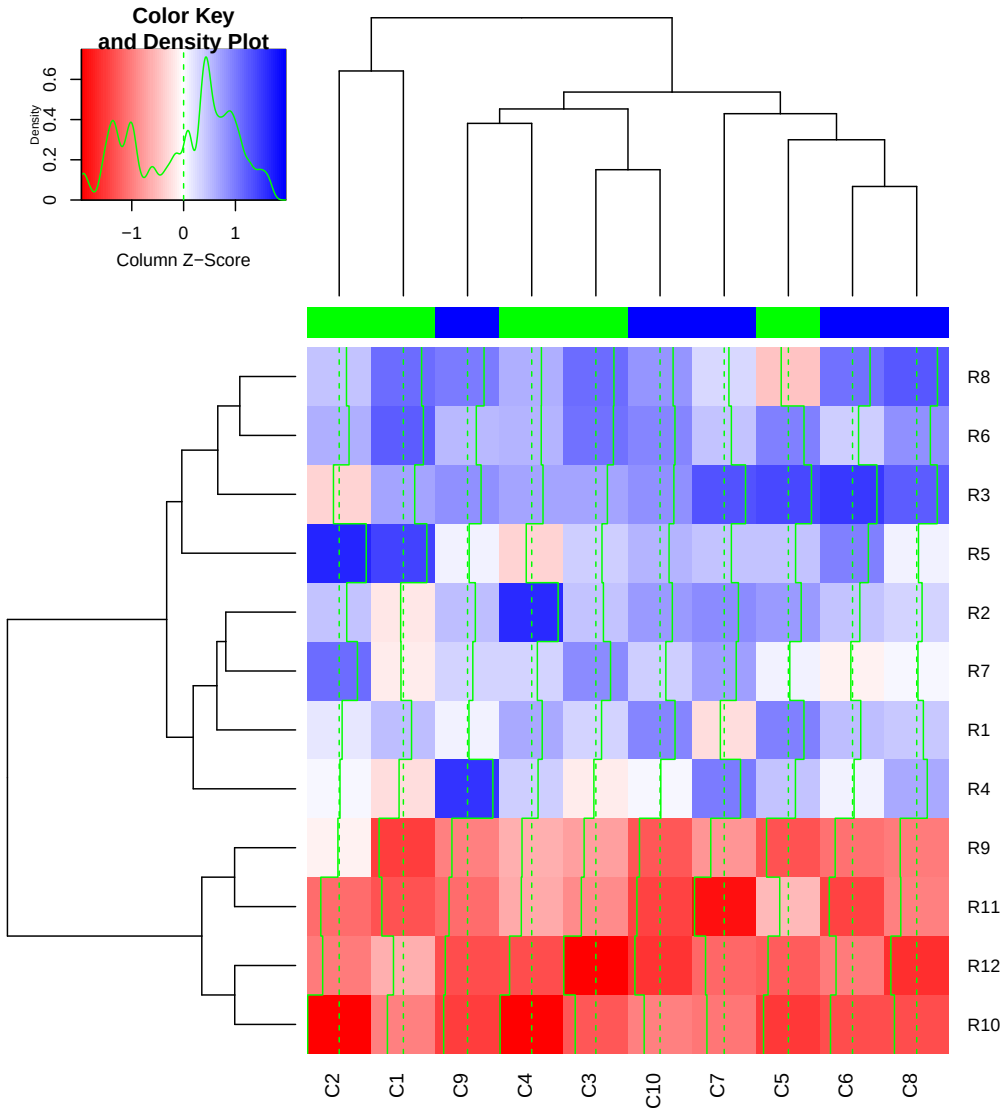
Heatmaps

`heatmap` built-in function: heatmap, dendogram, one information annotation for the columns/rows.



Heatmaps

`heatmap.2` function from *gplots* package: density distribution in legend and heatmap.



Heatmaps

`Heatmap` function from *ComplexHeatmap* package: more columns/rows annotation with graphs and panels.