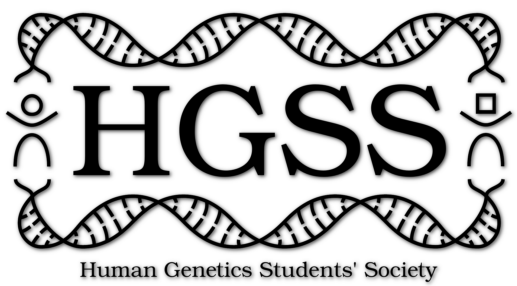
HGSS Workshop : R, data manipulation, visualization, genomic ranges

Jean Monlong

Human Genetics department

May 8th, 2015



McGill

Today's topic

- Advanced manipulation of *data.frames* .
- Advanced visualization.
- Analyzing large data.

Doodle

Advanced visualization	Genomic ranges manipulation	Accessing available genomic annotations .	Analyzing large data.	Automation and code reduction.	Advanced manipulation of data.frames	Data cleaning.
12	8	9	11	8	**14**	7

Let's get started !

1. Open R/Rstudio or whatever you use.
2. Prepare a folder for the workshop and set it as working directory.
3. Download dataWS2.RData from there

Today's packages

Installation

- Using `install.packages` for CRAN packages.
- Using `biocLite` for Bioconductor packages.

Run this

```
install.packages(c("data.table","dplyr","ggplot2","reshape"))
source("http://bioconductor.org/biocLite.R")
biocLite(c("GenomicRanges","AnnotationHub"))
```

Large *matrix*

Large *matrix* and row-by-row analysis

Don't do this !

Concatenate iteratively on big data.

```
myMatrix = NULL
for(i in 1:100000){
... Instructions on myOtherMatrix[i,]
myMatrix = rbind(myMatrix, myNewLine)
}
```

Instead do this

Create the data and then fill it.

```
myMatrix = matrix(NA,100000,100)
for(i in 1:100000){
... Instructions on myOtherMatrix[i,]
myMatrix[i,] = myNewLine
}
```

Large *matrix* and row-by-row analysis

Don't do this !

Create the data and then fill it.

```
myMatrix = matrix(NA,100000,100)
for(i in 1:100000){
... Instructions on myOtherMatrix[i,]
myMatrix[i,] = myNewLine
}
```

Instead do this

Use apply and a function.

```
myMatrix = apply(myOtherMatrix, 1, function(myOtherMatrix.row){
    ... Instructions on myOtherMatrix.row
   return(myNewLine)
})
```

data.frame

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.

data.frame basics

Create a *data.frame*

```
myDF = data.frame(gene=c("A","B","C"),id=1:3)
myDF = data.frame(gene=c("A","B","C"),id=1:3,
                                  stringsAsFactors=FALSE)
```

Add parameter `stringsAsFactors=FALSE}` to avoid annoying type conversion.

Exercise

Try both commands and observe the difference using `str` function.

Access columns

Use $ symbol for comfort and readability.

```
myDF[,1]
myDF[,"gene"]
myDF$gene
```

Same functions as *matrix*

colnames, dim, head, str, rbind.

Transform a *matrix* into a *data.frame*

Package reshape

melt function melts a *matrix* into a *data.frame* using the rows and columns names.

Example

```
> library(reshape)
> mat
     col1 col2 col3
row1    1    3    5
row2    2    4    6
> melt(mat)
    X1   X2 value
1 row1 col1     1
2 row2 col1     2
3 row1 col2     3
4 row2 col2     4
5 row1 col3     5
6 row2 col3     6
```

Exercise

- Load dataWS2.RData and have a look at mat.ge *matrix* .
- Create a new matrix with only the first 10 rows of mat.ge.
- Create a *data.frame* from this new matrix.
- Add relevant column names.

Merge two *data.frames*

merge function merges *data.frames* using their **common columns**.

Example

```
df12 = merge(df1, df2)
```

Example - Two *data.frames* merged

colA	colB	colC
2	43	france
4	87	france
1	100	spain

colA	colD	colE
2	TRUE	bonjour
1	FALSE	hola

colA	colB	colC	colD	colE
2	43	france	TRUE	bonjour
1	100	spain	FALSE	hola

Exercise

- Have a look at metadata.df *data.frame* .
- Update the expression *data.frame* (created previously) by merging metadata.df information.

Subset a *data.frame*

subset function retrieves a subset of a *data.frames* using a condition on column names.

Example

```
coldMontreal = weatherDF[which(weatherDF$city=="Montreal"
                               & weatherDF$celsius < -30),]

coldMontreal = subset(weatherDF, city=="Montreal"
                                        & celsius < -30)
```

Exercise

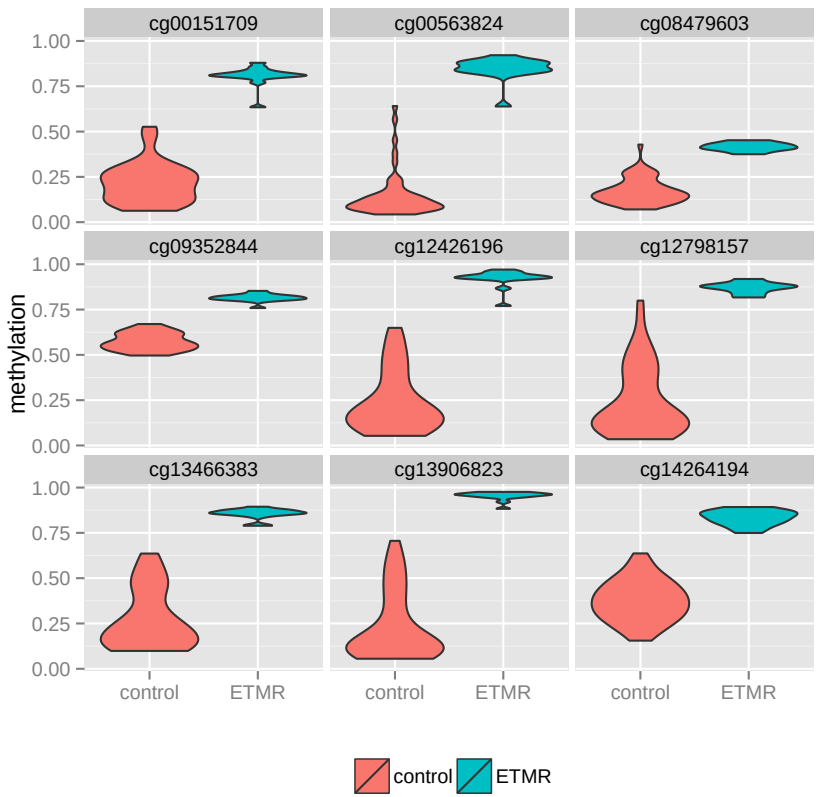
- Create a *data.frame* with expression of male samples only.
- Create a *data.frame* with expression of the first gene.

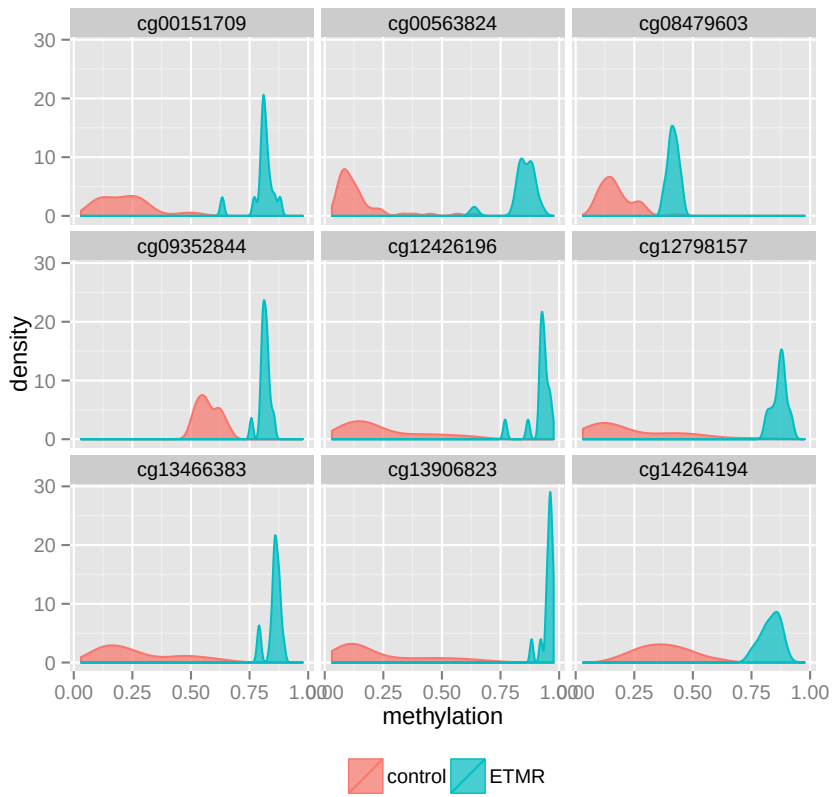
Visualization with ggplot2

ggplot2 package

Introduction

A package to construct pretty and/or complex graphs. Many aspects of the graph are arranged automatically but everything can be specified. 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(...)` part and the input *data.frame* .
- `aes(...)` defines how to use the input *data.frame* 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/`

Histogram

To represent distribution of continuous values

- `geom_histogram` function.
- `x=` to define the x-coordinate.

- `fill=` to define the bar color.
- `position="dodge"` to put different bars side-by-side.

Example

```
ggplot(myDF,aes(x=colA, fill=colB)) + geom_histogram()

ggplot(myDF,aes(x=colA)) + geom_histogram(fill="red")

ggplot(myDF,aes(x=colA, fill=colB)) +
                        geom_histogram(position="dodge")
```

Exercise

- Plot the distribution of the expression of all genes and all samples
- Same but coloring the bars by genes.

General functions

xlab/ylab	change x/y axis label.
xlim/ylim	change x/y axis limits (range).
ggtitle	adds a title.

Example

```
ggplot(myDF,aes(x=colA, fill=colB)) + geom_histogram() +
                                    xlab("column A") +
                                    ylab("column B") +
                                    ylim(0,10) +
                                    xlim(1,5) +
                                    ggtitle("An example")
```

Exercise

Add relevant axis labels and title

Themes

- `theme_bw()` or `theme_minimal()` for lighter graphs.
- `theme(...)` to change specific aspects.
  - `legend.position="bottom"`.

Example

```
ggplot(myDF,aes(x=colA,y=colB,colour=colC)) +
                    geom_point() +
                    theme_bw() +
                    theme(legend.position="bottom")
```

Exercise

Try these on the previous graphs.

Faceting : multi-panel graphs

- `facet_wrap` function.
  - A formula `~colName` to define the column to use.
  - `ncol=` to define a number of facet columns.
- `facet_grid` function.
  - A formula `colName1~colName2` to define the columns to use as facet row/column.
- `scales="free"` to allow different axis scales.

Example

```
ggplot(myDF,aes(x=colA)) + geom_histogram() +
                           facet_wrap(~colB, ncol=3)

ggplot(myDF,aes(x=colA)) + geom_histogram() +
                           facet_grid(colB~colC)
```

Exercise

Show separate histograms for each gene using `facet_wrap`.

Exploring Gencode annotation

Get the data

1. Download `ftp://ftp.sanger.ac.uk/pub/gencode/Gencode_human/release_22/gencode.v22.annotation.gtf.gz`.
2. Unzip it in your working directory.

Gencode file

- Human gene reference annotation.
- Genes, exons, transcipts, ...
- More than 2 million lines.
- GTF format (see `http://www.ensembl.org/info/website/upload/gff.html`).

Basic functions

Extra parameters in read.table

colClasses a vector with the data type of each column: e.g. *"character"*, *"numeric"*. Put `"NULL"` to skip a column.

nrows the number of rows to read.

Read a file line by line (or by chunk)

```
con = file(file.name)
while(length(line = readLines(con,n=1))>0){
... Instructions
}
```

To test the performance

```
system.time({
... Instructions
})
```

data.table package and fread

fread function

+ Very fast.
+ Almost no additional parameters.
- Cannot read compressed file.
- Has its specific format (*data.table*)...
+ ... which can be converted into *data.frame* .
+ Very fast.

Example

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

Useful parameters

skip= the number of lines to skip before starting to read.

select= a *vector* with the columns to read.

Exercise

1. Use read.table to read a few rows of gencode.v22.annotation.gtf.
2. Try reading the entire file (or say 500000 rows) using read.table.
3. Compare with fread (skipping the first 5 rows).

4. Use fread to read columns $\{1, 3, 4, 5, 9\}$ of the entire file.
5. Convert the output into a *data.frame* .
6. Add relevant column names (see GTF format in previous slides).

7. How many elements of each feature are there.

Bar plots

For a summary of categorical values

- `geom_bar` function.
- `x=` to define the x-coordinate.

- `fill=` to define the bar color.
- `position="dodge"` to put different bars side-by-side.

- `y=` to define the y-coordinate. If not defined, the number of observations is shown.
- `stat="identity"` if the y-coordinate is defined.

Example

```
ggplot(myDF,aes(x=colB)) + geom_bar()
ggplot(myDF,aes(x=colB)) + geom_bar(position="dodge")

ggplot(myDF,aes(x=colB,y=colC)) + geom_bar(stat="identity")
```

Exercise

- Create a bar plot of the number of elements in each feature.
- Create a bar plot of the number of elements in each chromosome.
- Same but coloring by feature. Try the *dodge* positioning.

data.frame manipulation with dplyr

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)
```

dplyr piping

Example

```
newDF = myDF %>% filter(colA < 1) %>%
              mutate(colAB=colA*colB) %>% arrange(colAB)
```

Exercise

- Create the following getAtt function.
- Try this function on a few of the *attribute* column values.
- Try changing the second argument. E.g. to *gene_type*.
- Create new columns for *gene_id* and *gene_type*, using mutate.
- Create a bar plot of the number of **genes** for each *gene_type*. Try adding `+coord_flip()` to the `ggplot` command.

getAtt function

- Parse a character to retrieve a value formatted as `gene_id ="value"`.
- E.g. retrieving `ENSG111` from `XXX; gene_id ="ENSG111"; XXX`.

```
getAtt <- function(attributes, att.name="gene_id"){
  sub(paste0(".*",att.name," \"([^\"]+)\";.*"), "\\1", attributes)
}
```

Exercise - Other questions

- Find the 10 longest pseudogenes. Hint: arrange, desc, filter.
- Find the 10 shortest exons in protein-coding genes.

- Show the distribution of gene size colored by gene type.
- Add `scale_x_log10()` for logarithmic scale.

- Same but using density curve : replace `geom_histogram` by `geom_density`.

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. Get the number of genes for each gene type...
2. ... ordered by descending number of genes.
3. Keep only genes in the 6 most common gene types OR label the rest as `"others"`.
4. Plot the new version of the gene size distribution.

5. Compute the number of exons per gene.
6. Plot the distribution.
7. Same zooming in the range $[0, 100]$.
8. Same coloring by gene type.
9. Which genes have the more exons.

Scatterplots

- `geom_point` function.
- `x=` to define the x-coordinate.
- `y=` to define the y-coordinate.

- `colour=` to define the point color.
- `alpha=` to define the point opaqueness.
- `shape=`, `size=` to define the point shape/size.

Example

```
ggplot(myDF,aes(x=colA, y=colB)) + geom_point()

ggplot(myDF,aes(x=colA,y=colB,colour=colC)) + geom_point()

ggplot(myDF,aes(x=colA, y=colB)) + geom_point(colour="red",
                                              alpha=.5)
```

Exercise

1. Compute the number of transcript per gene.
2. Plot the distribution colored by gene type.

3. Compute the size of each gene.
4. Merge this information with the number of transcript.

5. Plot number of transcript versus gene size.
6. Color and shape by gene type.
7. Add some transparency.
8. Use log-scale if necessary.
9. Change the legend position.

10. Same graph with one panel per gene type.

Genomic Ranges

GenomicRanges

Introduction

Represents genomic intervals. All annotation can be represented through *GenomicRanges* objects.

Creation

```
myGR = GRanges(chrs, IRanges(start=starts,end=ends))
```

Useful functions

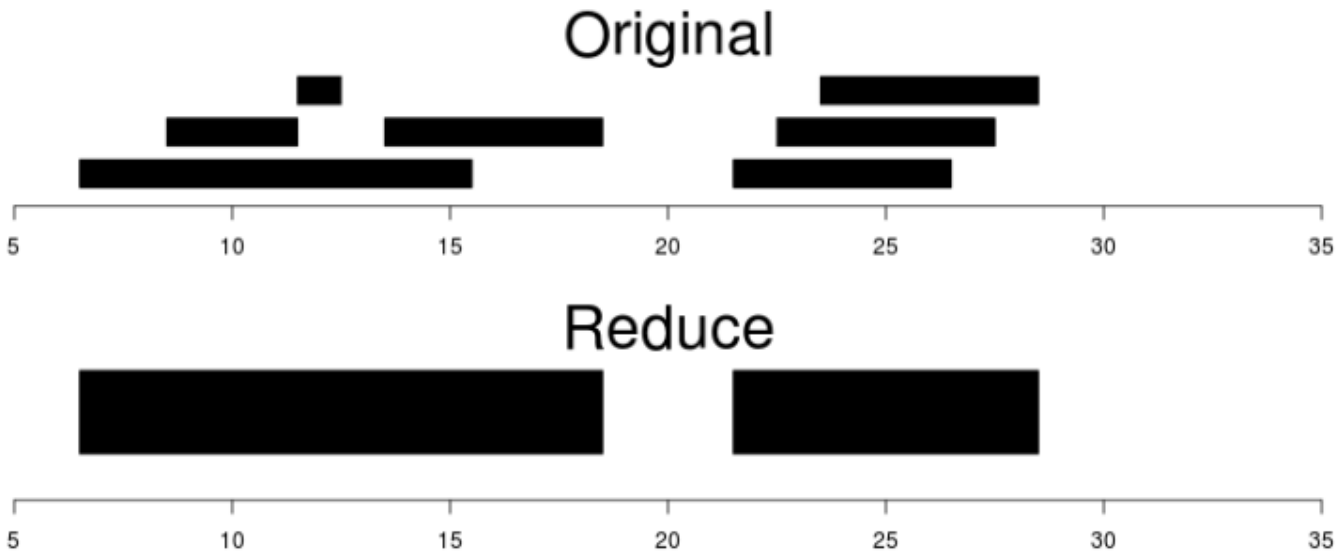
overlapsAny Test overlaps of one *GRanges* into second *GRanges* .

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.

Other functions

reduce function



Basic functions

width gets the regions size.

start/end gets the start/end positions.

range gets the range of region (i.e. smallest start + largest end).

Exercise - Exon density for one gene

- Get a *data.frame* of the exons for one gene.
- Create the corresponding *GRanges* .
- Try width/start/end/range functions.
- "Reduce" it.

- Compute the exon density : region covered by exons divided by total region.

Apply function on blocks

do function

- Apply a specific function to each block.
- Hence input of this function is a *data.frame* .
- The output as well.

- In the pipe chain . represents the input *data.frame* .

Example

```
myDF %>% group_by(colA) %>% head
myDF %>% group_by(colA) %>% do(head(.))

funFunFun <- function(df){
... Instructions on input data.frame 'df'
... creating output data.frame 'new.df'
return(new.df)
}
myDF %>% group_by(colA) %>% do(funFunFun(.))
```

Exercise

- Write a function that computes the exon density from a *data.frame* with exons information.
- Apply this function to each gene of chromosome 13.

Parallel processing

Easiest solution with parallel package

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

Example

```
lapply(1:10, function(ii){
... Instructions with 'ii'
})

mclapply(1:10, function(ii){
... Instructions with 'ii'
}, mc.cores=4)
```

Exercise

- Compute the exon density of genes in 4 chromosomes, parallelized by chromosome.
- Join the output list into one *data.frame* using `do.call(rbind, myOutList)`.

Annotation database

Introduction

Many annotation are already available directly from R, see Bioconductor website. Else you can create your own *GenomicRanges* object.

AnnotationHub package

Many different tracks, including most of Encode's.

```
library(AnnotationHub)
ah = AnnotationHub()
hist.prom = ah$goldenpath.hg19.encodeDCC.wgEncodeBroadHistone.
   wgEncodeBroadHistoneGm12878H3k4me3StdPk.broadPeak_0.0.1.RData
```

Exercise

1. Run these commands.
2. Have a look at the retrieved object.
3. How wide is a peak on average ?
4. Plot the distribution of the peaks width ?
5. Remove peaks wider than 10kb.

Test overlap of one *GRanges* in another

overlapsAny function

- Two *GRanges* objects as input.
- Returns TRUE/FALSE for each range of the first *GRanges* .
- TRUE means the range overlaps something in the second *GRanges* .
- FALSE if not.

Example

```
> any1in2 = overlapsAny(gr1,gr2)
> any1in2
[1]  TRUE  TRUE FALSE FALSE  TRUE FALSE
```

Exercise

- Add a column to the original data.frame with TRUE/FALSE if it overlaps a promoter region.
- Create some graphs using this new column.

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
```

Exercise

1. Find the overlaps between your genes and the promoters.
2. Same but allowing +/- 1Kbp for the overlap. See parameter `maxgap=`.

3. Create a new *data.frame* , manually merging the promoter *score* and your genes annotation *data.frame* , for the genes overlapping promoters.
4. Plot the distribution of the scores for different gene types.

Distance between two *GRanges* sets

distanceToNearest function

- Two *GRanges* objects as input.
- Returns the index of regions in object 1, the closest in 2 and the distance between them.
- queryHits and subjectHits functions to retrieves those index.
- mcols function to retrieve the distance information.

Example

```
> d12 = distanceToNearest(gr1, gr2)
> d12
Hits of length 6
queryLength: 6
subjectLength: 4
  queryHits subjectHits  distance
   <integer>   <integer> <integer>
 1         1           1         0
 2         2           2         0
 3         3           2        10
 4         4           3         6
 5         5           3         0
 6         6           3        21
> queryHits(d12)
[1] 1 2 3 4 5 6
> mcols(d12)$distance
[1]  0  0 10  6  0 21
```

Exercise

1. Compute the distance between each gene and the nearest promoter.
2. Add this information as a column in your gene annotation *data.frame* .

3. Plot the distribution of these distances for different gene types.

More *ggplot2* tricks

- Use `theme(text=element_text(size=22))`.
- Use `scale_fill_brewer(palette="Set1")`.
- Use `scale_x_discrete(limits=)`.

Boxplots

- `geom_boxplot` function.
- `x=` to define the x-coordinate.
- `y=` to define the y-coordinate.

- `fill=` to define the box color.
- Optional: `group=` to define what's in the box.

Example

```
ggplot(myDF,aes(x=colA, y=colB)) + geom_boxplot()

ggplot(myDF,aes(x=colA,y=colB,fill=colC)) + geom_boxplot()

ggplot(myDF,aes(x=colA,y=colB,group=colC)) + geom_boxplot()
```

Exercise

Make some boxplots, maybe using the gene expression *data.frame* .

Online resources

R basics

- `http://www.twotorials.com/` : small video-tutorials.
- `www.youtube.com/user/rdpeng/` : Coursera *Computing for Data Analysis* videos. Other interesting videos, e.g. *ggplot2*.

- `https://www.datacamp.com/` or `http://tryr.codeschool.com/` : Interactive tutorial of R basics.

- `http://www.r-tutor.com/` : R and statistics small web-tutorials.
- `http://www.computerworld.com/s/article/9239625/Beginner_s_guide_to_R_Introduction` : Beginner's guide with screenshots.
- `http://cran.r-project.org/manuals.html` : R manual.

Bioinformatics

- `http://stephenturner.us/p/edu` List of online resources for Bioinformatics.
- `http://bioinformatics.ca/workshops/2013/` : Bioinformatics workshop material.
- `http://manuals.bioinformatics.ucr.edu/home/R_BioCondManual` : Pieces of code for bioinformatics analysis, plots. Including Bioconductor.
- `http://bioconductor.org/help/course-materials/2013/` : Bioinformatics tutorials material: pdf and R scripts.