Bioconductor

Maja Kuzman

2018-11-08

Bioconductor

Maja Kuzman

2018-11-08

Bioconductor

basics

Bioconductor

OPEN SOURCE software for bioinformatics

1649 packages:

Bioconductor

Install the bioconductor and biocLite:

```
source("http://bioconductor.org/biocLite.R")
biocLite()
```

- DNA sequencing

Bioconductor

OPEN SOURCE software for bioinformatics

1649 packages:

Bioconductor

Install the bioconductor and biocLite:

```
source("http://bioconductor.org/biocLite.R")
biocLite()
```

- DNA sequencing
- Microarray

Bioconductor

OPEN SOURCE software for bioinformatics

1649 packages:

Bioconductor

Install the bioconductor and biocLite:

```
source("http://bioconductor.org/biocLite.R")
biocLite()
```

- DNA sequencing
- Microarray
- Flow cytometry

Bioconductor

OPEN SOURCE software for bioinformatics

1649 packages:

Bioconductor

Install the bioconductor and biocLite:

```
source("http://bioconductor.org/biocLite.R")
biocLite()
```

- DNA sequencing
- Microarray
- Flow cytometry
- Proteomics

Bioconductor

OPEN SOURCE software for bioinformatics

1649 packages:

Bioconductor

Install the bioconductor and biocLite:

```
source("http://bioconductor.org/biocLite.R")
biocLite()
```

- DNA sequencing
- Microarray
- Flow cytometry
- Proteomics
- Statistical analysis and comprehension

Bioconductor

OPEN SOURCE software for bioinformatics

1649 packages:

Bioconductor

Install the bioconductor and biocLite:

```
source("http://bioconductor.org/biocLite.R")
biocLite()
```

- DNA sequencing
- Microarray
- Flow cytometry
- Proteomics
- Statistical analysis and comprehension
- Work flows

Bioconductor

OPEN SOURCE software for bioinformatics

1649 packages:

Bioconductor

Install the bioconductor and biocLite:

```
source("http://bioconductor.org/biocLite.R")
biocLite()
```

- DNA sequencing
- Microarray
- Flow cytometry
- Proteomics
- Statistical analysis and comprehension
- Work flows
- Reproducible research

Bioconductor

OPEN SOURCE software for bioinformatics

1649 packages:

Bioconductor

Install the bioconductor and biocLite:

```
source("http://bioconductor.org/biocLite.R")
biocLite()
```

- DNA sequencing
- Microarray
- Flow cytometry
- Proteomics
- Statistical analysis and comprehension
- Work flows
- Reproducible research
- Great help in extensive documentation: vignettes

Bioconductor packages

IRanges

IRanges

```
library(IRanges)
oneRange <- IRanges(start = 5,end = 10)
```

IRanges

```
library(IRanges)
oneRange <- IRanges(start = 5,end = 10)
```

What is oneRange?

IRanges

```
library(IRanges)
oneRange <- IRanges(start = 5,end = 10)
```

What is oneRange?

```
oneRange
```

```
## IRanges object with 1 range and 0 metadata columns:
##           start       end     width
##       <integer> <integer> <integer>
##   [1]         5        10         6
```

How would you access "start" of oneRange?

IRanges

```
library(IRanges)
oneRange <- IRanges(start = 5,end = 10)
```

What is oneRange?

```
oneRange
```

```
## IRanges object with 1 range and 0 metadata columns:
##           start       end     width
##       <integer> <integer> <integer>
##   [1]         5        10         6
```

How would you access "start" of oneRange?

```
oneRange$start
```

Look at its structure. Do you have an idea how you could access start?

```
str(oneRange)
```

```
## Formal class 'IRanges' [package "IRanges"] with 6 slots
##   ..@ start          : int 5
##   ..@ width          : int 6
##   ..@ NAMES          : NULL
##   ..@ elementType    : chr "ANY"
##   ..@ elementMetadata: NULL
##   ..@ metadata       : list()
```

Look at its structure. Do you have an idea how you could access start?

```
str(oneRange)
```

```
## Formal class 'IRanges' [package "IRanges"] with 6 slots
##   ..@ start          : int 5
##   ..@ width          : int 6
##   ..@ NAMES          : NULL
##   ..@ elementType    : chr "ANY"
##   ..@ elementMetadata: NULL
##   ..@ metadata       : list()
```

```
oneRange@start
```

```
## [1] 5
```

Slots can be accessed via @, but this is not recommended. Objects are ment to be used in a certain way, and this is always controlled by functions defined on objects. For example, if you want to get start of IRanges object, look at the documentation and see how this can be done:

```
#browseVignettes("IRanges")
start(oneRange)
```

```
## [1] 5
```

See how this class is defined:

```
showClass("IRanges")
```

```
## Class "IRanges" [package "IRanges"]
##
## Slots:
##
## Name:               start             width             NAMES
## Class:            integer           integer character_OR_NULL
##
## Name:        elementType   elementMetadata          metadata
## Class:         character DataTable_OR_NULL              list
##
## Extends:
## Class "IPosRanges", directly
## Class "IntegerRanges", by class "IPosRanges", distance 2
## Class "Ranges", by class "IPosRanges", distance 3
## Class "List", by class "IPosRanges", distance 4
## Class "Vector", by class "IPosRanges", distance 5
## Class "list_OR_List", by class "IPosRanges", distance 5
## Class "Annotated", by class "IPosRanges", distance 6
## Class "IntegerRanges_OR_missing", by class "IntegerRanges", distance 3
##
## Known Subclasses: "NormalIRanges", "GroupingIRanges"
```

Make another range :

```
ir <- IRanges(c(5,7,1),width = c(6,9,3))
ir
```

```
## IRanges object with 3 ranges and 0 metadata columns:
##           start       end     width
##       <integer> <integer> <integer>
##   [1]         5        10         6
##   [2]         7        15         9
##   [3]         1         3         3
```

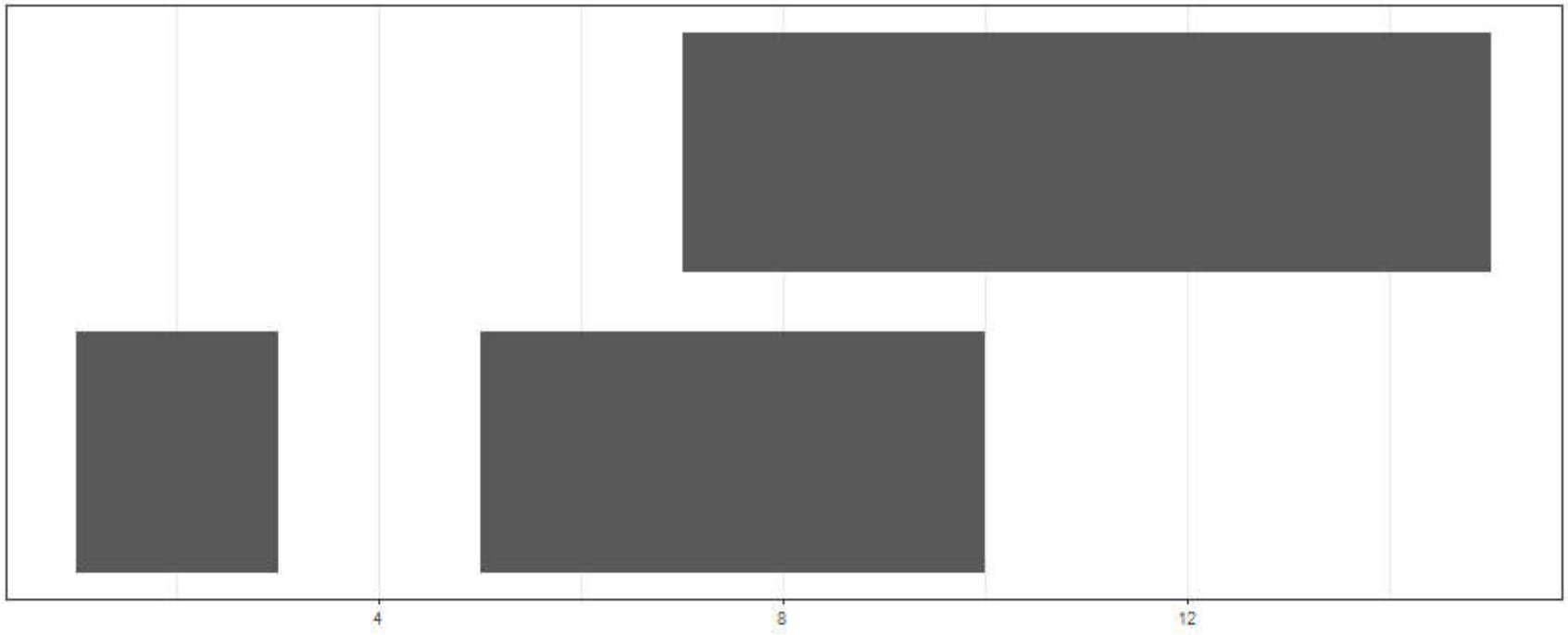
Acessing elements:

```
start(ir)
end(ir)
width(ir)
```

You can plot iranges with ggplot, use ggbio package!

```
library(ggbio)
library(GenomicRanges)
plotIRanges <- function(ir){
     ggr <- GRanges("1", ir)
    ggplot(ggr) +
         geom_rect() +
    theme_bw()
}
```

```
plotIRanges(ir)
```



Why is this useful?

Inter range methods :

transform all the ranges together as a set to produce a new set of ranges.

union(), intersect(), setdiff(), gaps(), pgap()

range() , reduce(), disjoin(), disjointBins(), restrict()

Intra range methods :

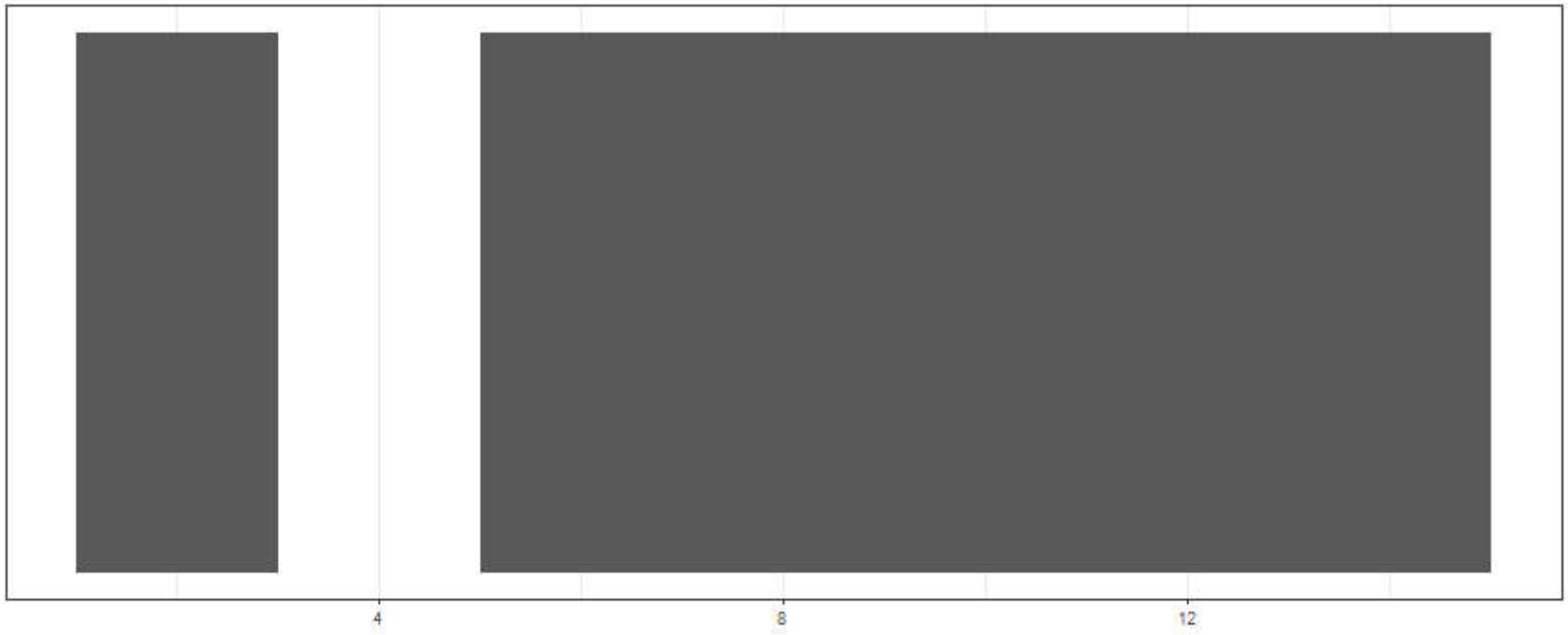
transform each range individually (and independently of the other ranges).

punion(), pintersect(), psetdiff(), pgap()
shift(), narrow(), flank(), promoters()

Complete the following function so it works in the examples below:

```
plotFunctionOnIr <- function(???){
     ???
}
```

```
plotFunctionOnIr(intersect, ir,ir)
```



Solution:

```
plotFunctionOnIr <- function(f,...){
     plotIRanges(f(...))
}
plotFunctionOnIr(intersect, ir,ir)
```

Solution:

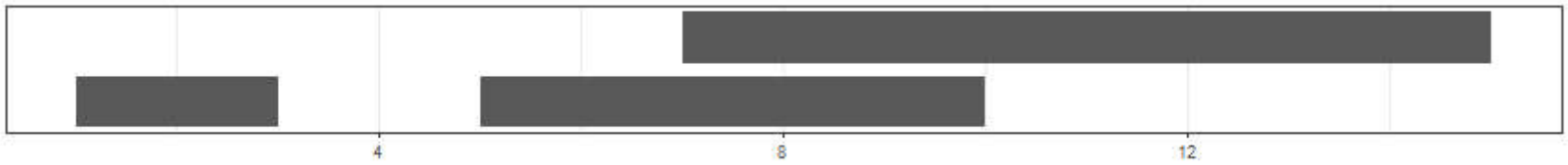
```
plotFunctionOnIr <- function(f,...){
     plotIRanges(f(...))
}
plotFunctionOnIr(intersect, ir,ir)
```

Use that function (plotFunctionOnIr) to understand what the functions mentioned above do!

- union(), intersect(), setdiff(), gaps(), pgap(), range() , reduce(), disjoin(), disjointBins(), restrict()
- punion(), pintersect(), psetdiff(), pgap(), shift(), narrow(), flank(), promoters()

Solution: union

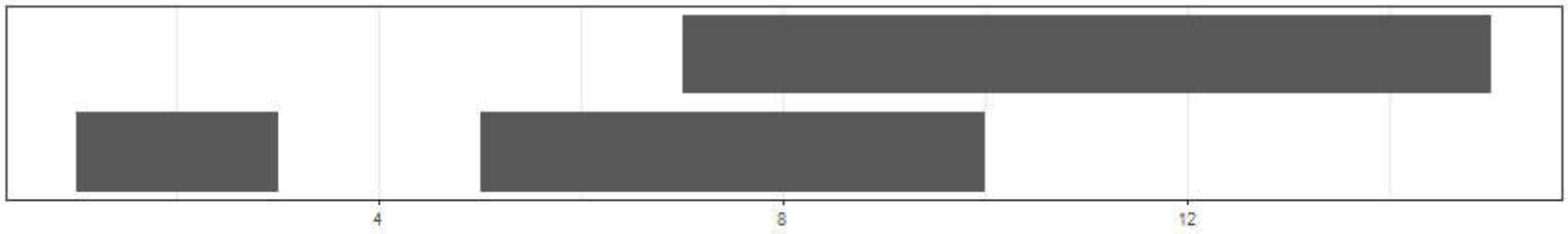
```
plotIRanges(ir)
```



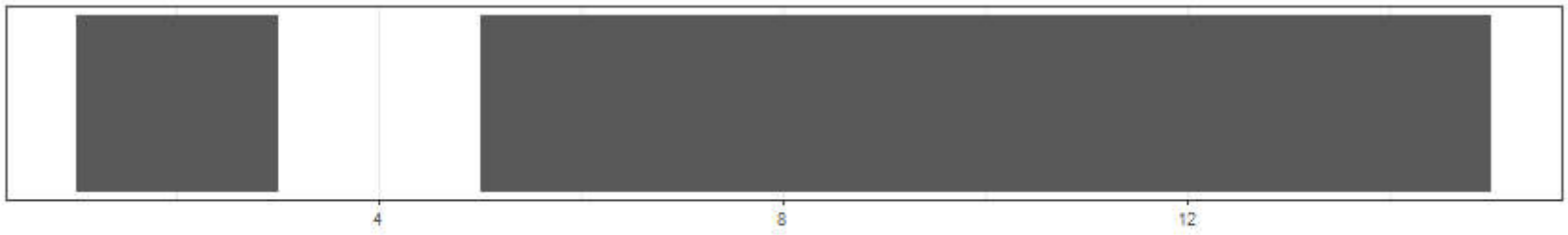

```
plotFunctionOnIr(union, ir,ir)
```

Solution: intersect

```
plotIRanges(ir)
```

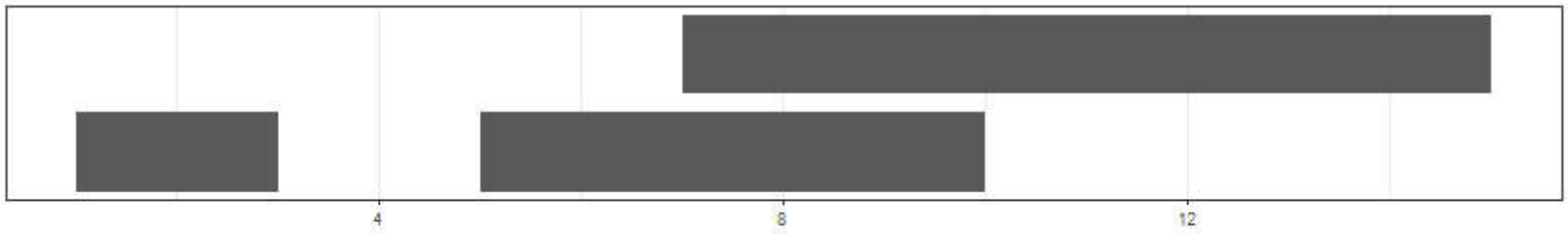



```
plotFunctionOnIr(intersect, ir,ir)
```

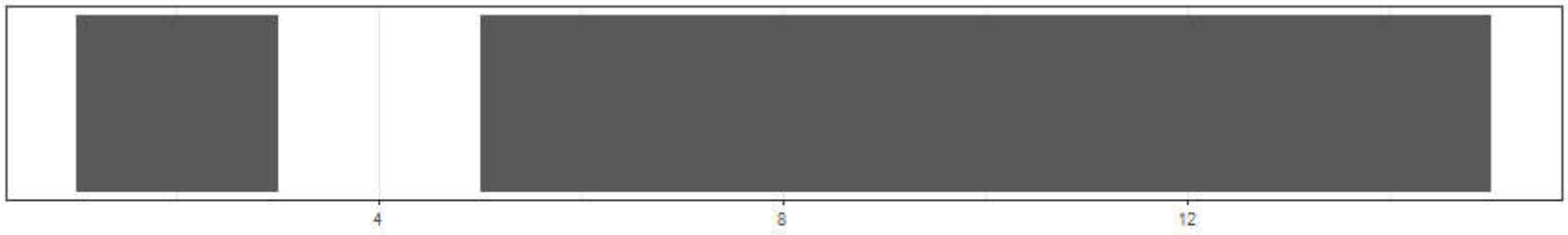


Solution: intersect

```
plotIRanges(ir)
```




```
plotFunctionOnIr(intersect, ir,ir)
```



Solution: shift

```
plotIRanges(ir)
```


```
shift(ir, 2)
```

```
## IRanges object with 3 ranges and 0 metadata columns:
##           start       end     width
##       <integer> <integer> <integer>
##   [1]         7        12         6
##   [2]         9        17         9
##   [3]         3         5         3
```

```
plotFunctionOnIr(shift, ir,2)
```

Solution: disjoin

```
plotIRanges(ir)
```


```
disjoin(ir)
```

```
## IRanges object with 4 ranges and 0 metadata columns:
##           start       end     width
##       <integer> <integer> <integer>
##   [1]         1         3         3
##   [2]         5         6         2
##   [3]         7        10         4
##   [4]        11        15         5
```

```
plotFunctionOnIr(disjoin, ir)
```

Solution: restrict

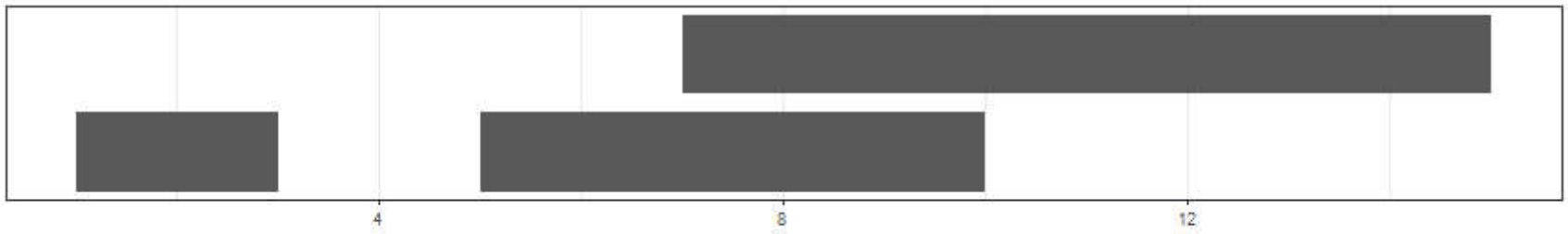
```
plotIRanges(ir)
```


```
plotFunctionOnIr(restrict, ir, 2)
```

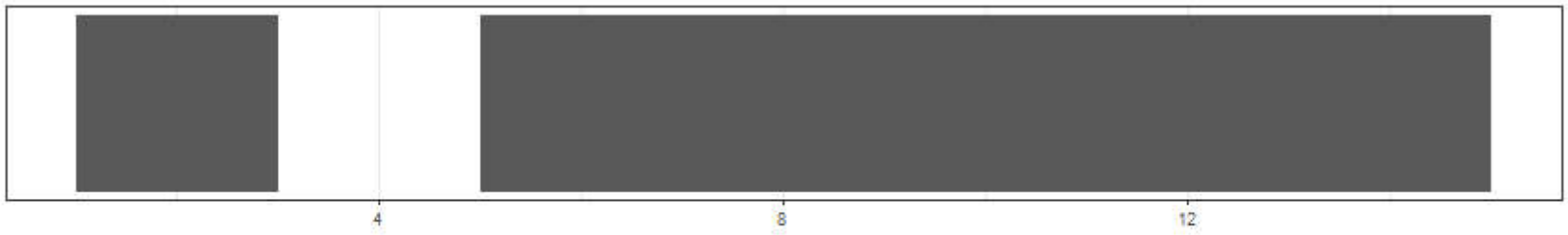

```
plotFunctionOnIr(restrict, ir, 2,7)
```

Solution: reduce

```
plotIRanges(ir)
```




```
plotFunctionOnIr(reduce, ir)
```



Solution: pintersect

```
plotIRanges(ir)
```


```
plotIRanges(ir+2)
```


```
pintersect(ir, ir+2)
```

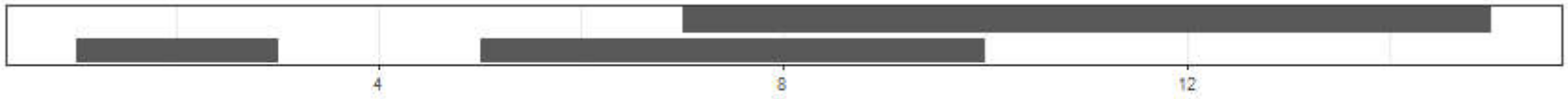
```
## IRanges object with 3 ranges and 0 metadata columns:
##           start       end     width
##       <integer> <integer> <integer>
##   [1]         5        10         6
##   [2]         7        15         9
##   [3]         1         3         3
```

```
plotFunctionOnIr(pintersect, ir, ir+2)
```

Finding overlaps:

```
plotIRanges(oneRange)
```


```
plotIRanges(ir)
```



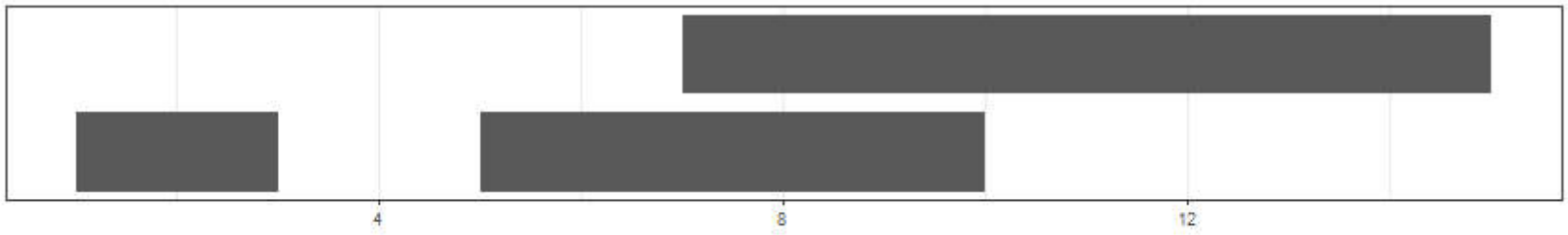

```
findOverlaps(oneRange,ir)
```

```
## Hits object with 2 hits and 0 metadata columns:
##       queryHits subjectHits
##       <integer>   <integer>
##   [1]         1           1
##   [2]         1           2
##   -------
##   queryLength: 1 / subjectLength: 3
```

Counting overlaps:

```
plotIRanges(oneRange)
```


```
plotIRanges(ir)
```




```
countOverlaps(oneRange,ir)
```

```
## [1] 2
```

Exercise (1 point) Get coverage withouth using coverage function!

```
plotIRanges(ir)
```


```
ir
```

```
## IRanges object with 3 ranges and 0 metadata columns:
##           start       end     width
##       <integer> <integer> <integer>
##   [1]         5        10         6
##   [2]         7        15         9
##   [3]         1         3         3
```

```
coverage(ir)
```

```
## integer-Rle of length 15 with 5 runs
##   Lengths: 3 1 2 4 5
##   Values : 1 0 1 2 1
```

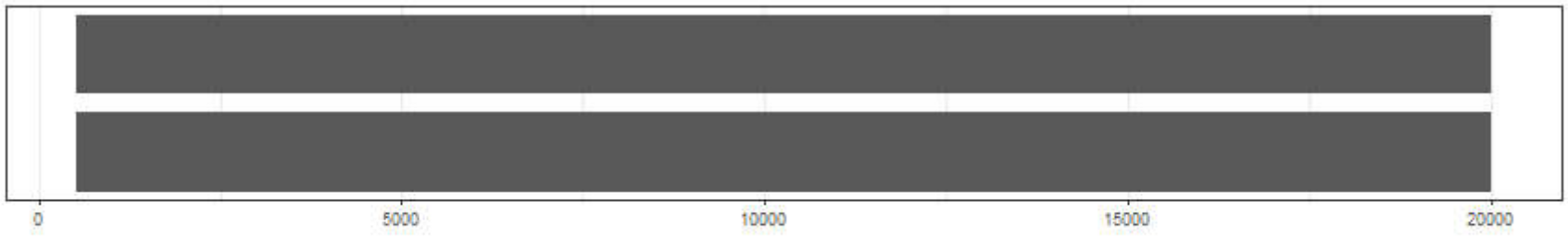
Bioconductor packages

GenomicRanges

This can be very useful, especially when combined with chromosomes! Genomic Ranges enable us to do exactly that!

```
genomepart1 <- GRanges("chr2", IRanges(517,20000), strand="+")
genomepart2 <- GRanges("chr2", IRanges(517,20000), strand="-")
```

```
plotIRanges(c(ranges(genomepart1),ranges(genomepart2)))
```




```
findOverlaps(genomepart1, genomepart2)
```

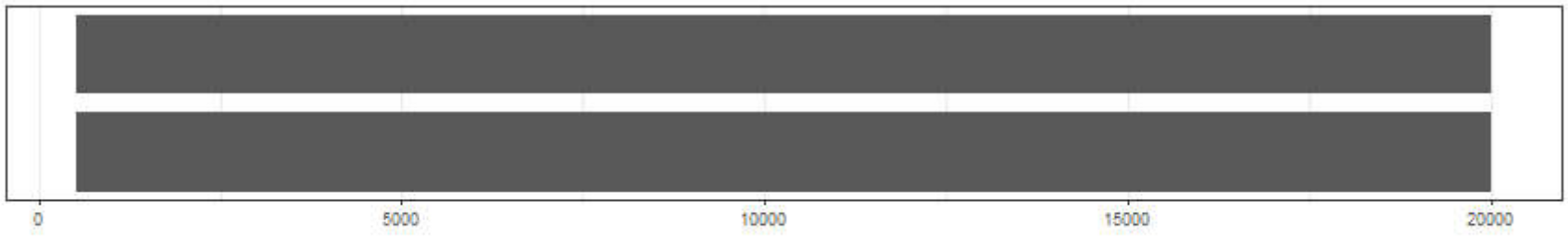
```
## Hits object with 0 hits and 0 metadata columns:
##     queryHits subjectHits
##     <integer>   <integer>
##    -------
##    queryLength: 1 / subjectLength: 1
```

Why are there no overlaps??

This can be very useful, especially when combined with chromosomes! Genomic Ranges enable us to do exactly that!

```
genomepart1 <- GRanges("chr2", IRanges(517,20000), strand="+")
genomepart2 <- GRanges("chr2", IRanges(517,20000), strand="-")
```

```
plotIRanges(c(ranges(genomepart1),ranges(genomepart2)))
```




```
findOverlaps(genomepart1, genomepart2, ignore.strand=TRUE)
```

```
## Hits object with 1 hit and 0 metadata columns:
##       queryHits subjectHits
##       <integer>   <integer>
##   [1]         1           1
##   -------
##   queryLength: 1 / subjectLength: 1
```

Why are there no overlaps??

Exercise: how many areas on chromosome 2 are overlapping at least 2 genes?? (1 point)

```
library(data.table)
mygenes <- fread("http://hex.bioinfo.hr/~mfabijanic/Chr2Geneshg38.txt")

chr2genes <- GRanges(mygenes$chromosome_name,
                     IRanges(mygenes$start_position, mygenes$end_position),
                     strand = mygenes$strand,
                     hgnc = mygenes$hgnc_symbol)
chr2genes
```

```
## GRanges object with 4006 ranges and 1 metadata column:
##          seqnames              ranges strand |        hgnc
##             <Rle>           <IRanges>  <Rle> | <character>
##      [1]        2         38814-46870      - |     FAM110C
##      [2]        2       197569-202605      + |
##      [3]        2       217730-266398      - |      SH3YL1
##      [4]        2       264140-278283      + |        ACP1
##      [5]        2       279558-288851      - |      ALKAL2
##      ...      ...                 ...    ... .         ...
##   [4002]        2 242087351-242088457      - |   LINC01238
##   [4003]        2 242088633-242160153      + |   LINC01881
##   [4004]        2 242119856-242120053      - |      CICP10
##   [4005]        2 242122287-242122469      + |    SEPT14P2
##   [4006]        2 242175181-242175634      + |   RPL23AP88
##   -------
##   seqinfo: 1 sequence from an unspecified genome; no seqlengths
```

Exercise: how many areas on chromosome 2 are overlapping at least 2 genes?? (1 point)

```
sum(coverage(chr2genes)>=2)
```

```
##        2
## 15858821
```

Exercise: how many areas on chromosome 2 are overlapping at least 2 genes?? (1 point)

```
sum(coverage(chr2genes)>=2)
```

```
##        2
## 15858821
```

```
ovls <- countOverlaps(disjoin(chr2genes), chr2genes)
sum(ovls>=2)
```

```
## [1] 686
```

Exercise (2 points):

Remember blastResult table? Make a consensus of qfrom-qto per group again, use GRanges this time! Do it for full data frame and for best 3 results! Are consensuses very different?

```
blastRes <- fread("http://hex.bioinfo.hr/~mfabijanic/blastResultSmall.txt")
blastRes
```

```
##                  id       species perc length mismatches gap_opens qfrom
##    1:  scaffold260 aac:Aaci_0411 59.6    938        376          2  4406
##    2:  scaffold260  acl:ACL_1347 60.0    937        374          1  4412
##    3:  scaffold260 afl:Aflv_2546 62.9    940        345          3  4409
##    4:  scaffold260 amt:Amet_4126 58.4    939        388          1  4406
##    5:  scaffold260 aoe:Clos_2257 59.3    941        379          2  4406
##   ---
## 4893: scaffold1369  lpc:LPC_1259 51.0     51         25          0   399
## 4894: scaffold1369   lpf:lpl1779 51.0     51         25          0   399
## 4895: scaffold1369   lpn:lpg1815 51.0     51         25          0   399
## 4896: scaffold1369   lpp:lpp1778 51.0     51         25          0   399
## 4897: scaffold3458 aau:AAur_0068 24.8    121         88          1   447
##        qto sfrom sto    eval bitscore      KEGG
##    1: 1596     4 939 0.0e+00   1125.2 ko:K03701
##    2: 1602     2 937 0.0e+00   1129.4 ko:K03701
##    3: 1596     1 938 0.0e+00   1207.2 ko:K03701
##    4: 1599     4 942 0.0e+00   1134.8 ko:K03701
##    5: 1596     4 944 0.0e+00   1151.7 ko:K03701
##   ---
## 4893:  551     1  51 8.6e-06     53.5 ko:K04761
## 4894:  551     1  51 8.6e-06     53.5 ko:K04761
## 4895:  551     1  51 8.6e-06     53.5 ko:K04761
## 4896:  551     1  51 8.6e-06     53.5 ko:K04761
## 4897:  809    26 143 9.4e-06     54.3 ko:K01195
```

Exercise (2 points):

Remember blastResult table? Make a consensus of qfrom-qto per group again, use GRanges this time! Do it for full data frame and for best 3 results! Are consensuses very different?

```
blastRes[qfrom>qto, ':='(qfrom=qto, qto=qfrom)]
gr <- GRanges(blastRes$id, IRanges(blastRes$qfrom, blastRes$qto))
gr
```

```
## GRanges object with 4897 ranges and 0 metadata columns:
##               seqnames    ranges strand
##                  <Rle> <IRanges>  <Rle>
##      [1]  scaffold260 1596-4406      *
##      [2]  scaffold260 1602-4412      *
##      [3]  scaffold260 1596-4409      *
##      [4]  scaffold260 1599-4406      *
##      [5]  scaffold260 1596-4406      *
##      ...           ...       ...    ...
##   [4893] scaffold1369   399-551      *
##   [4894] scaffold1369   399-551      *
##   [4895] scaffold1369   399-551      *
##   [4896] scaffold1369   399-551      *
##   [4897] scaffold3458   447-809      *
##   -------
##   seqinfo: 532 sequences from an unspecified genome; no seqlengths
```

Solution for consensus: Make a consensus of qfrom-qto per group again, use GRanges this time!

```
reduce(gr)
```

```
## GRanges object with 653 ranges and 0 metadata columns:
##             seqnames      ranges strand
##                <Rle>   <IRanges>  <Rle>
##     [1]  scaffold260   1596-4418      *
##     [2]  scaffold467   1080-4655      *
##     [3]  scaffold467   4796-8263      *
##     [4]  scaffold467 13962-16769      *
##     [5]   scaffold15 10640-13852      *
##     ...          ...         ...    ...
##   [649]  scaffold212       2-196      *
##   [650]  scaffold879      30-353      *
##   [651] scaffold2587      16-522      *
##   [652] scaffold1369     399-551      *
##   [653] scaffold3458     447-809      *
##   -------
##   seqinfo: 532 sequences from an unspecified genome; no seqlengths
```

Solution for best 3 results!

```
miniblast <- blastRes[,.SD[1:3], by=id][!is.na(KEGG)]
minigr <- GRanges(miniblast$id, IRanges(miniblast$qfrom, miniblast$qto))
reduce(minigr)
```

```
## GRanges object with 594 ranges and 0 metadata columns:
##             seqnames      ranges strand
##                <Rle>   <IRanges>  <Rle>
##     [1]  scaffold260   1596-4412      *
##     [2]  scaffold467   1101-4655      *
##     [3]  scaffold467   4829-8242      *
##     [4]  scaffold467 13962-16766      *
##     [5]   scaffold15 10664-13852      *
##     ...          ...         ...    ...
##   [590]  scaffold212       2-196      *
##   [591]  scaffold879      30-353      *
##   [592] scaffold2587      16-522      *
##   [593] scaffold1369     399-551      *
##   [594] scaffold3458     447-809      *
##   -------
##   seqinfo: 532 sequences from an unspecified genome; no seqlengths
```

Bioconductor packages

biomaRt

Ensembl

biomaRt

```
library(biomaRt)
```

Package for aquisition of ENSEMBL data sets! Ensembl

What version would you like to use?

```
listEnsembl()
```

```
##               biomart                version
## 1            ensembl       Ensembl Genes 94
## 2 ENSEMBL_MART_MOUSE       Mouse strains 94
## 3                snp  Ensembl Variation 94
## 4         regulation Ensembl Regulation 94
```

```
listMarts()
```

```
##                 biomart                version
## 1 ENSEMBL_MART_ENSEMBL       Ensembl Genes 94
## 2   ENSEMBL_MART_MOUSE       Mouse strains 94
## 3     ENSEMBL_MART_SNP  Ensembl Variation 94
## 4 ENSEMBL_MART_FUNCGEN Ensembl Regulation 94
```

```
listEnsembl(GRCh="37")
```

```
##      biomart                version
## 1    ensembl       Ensembl Genes 94
## 2        snp  Ensembl Variation 94
## 3 regulation Ensembl Regulation 94
```

Lets say we want to use ensemble genes version GRCh38 (current one). For a given mart, which organism do you choose?

```
ensembl <- useEnsembl(biomart = "ensembl")
listDatasets(ensembl)
```

```
##                              dataset
## 1          acalliptera_gene_ensembl
## 2       acarolinensis_gene_ensembl
## 3        acitrinellus_gene_ensembl
## 4        amelanoleuca_gene_ensembl
## 5          amexicanus_gene_ensembl
## 6          anancymaae_gene_ensembl
## 7          aocellaris_gene_ensembl
## 8            apercula_gene_ensembl
## 9      aplatyrhynchos_gene_ensembl
## 10      apolyacanthus_gene_ensembl
## 11       atestudineus_gene_ensembl
## 12            btaurus_gene_ensembl
## 13            caperea_gene_ensembl
## 14              catys_gene_ensembl
## 15         ccapucinus_gene_ensembl
## 16         cchok1gshd_gene_ensembl
## 17            ccrigri_gene_ensembl
## 18           celegans_gene_ensembl
## 19        cfamiliaris_gene_ensembl
## 20            chircus_gene_ensembl
## 21         choffmanni_gene_ensembl
## 22      cintestinalis_gene_ensembl
## 23           cjacchus_gene_ensembl
## 24          clanigera_gene_ensembl
## 25         cpalliatus_gene_ensembl
## 26         cporcellus_gene_ensembl
```

Lets say we want to use ensemble genes version GRCh38 (current one). For a given mart, which organism do you choose?

```
ensembl <- useEnsembl(biomart = "ensembl")
listDatasets(ensembl)
```

```
##                              dataset
## 1          acalliptera_gene_ensembl
## 2       acarolinensis_gene_ensembl
## 3        acitrinellus_gene_ensembl
## 4        amelanoleuca_gene_ensembl
## 5          amexicanus_gene_ensembl
## 6          anancymaae_gene_ensembl
## 7          aocellaris_gene_ensembl
## 8            apercula_gene_ensembl
## 9      aplatyrhynchos_gene_ensembl
## 10      apolyacanthus_gene_ensembl
## 11       atestudineus_gene_ensembl
## 12            btaurus_gene_ensembl
## 13            caperea_gene_ensembl
## 14              catys_gene_ensembl
## 15         ccapucinus_gene_ensembl
## 16         cchok1gshd_gene_ensembl
## 17            ccrigri_gene_ensembl
## 18           celegans_gene_ensembl
## 19        cfamiliaris_gene_ensembl
## 20            chircus_gene_ensembl
## 21         choffmanni_gene_ensembl
## 22      cintestinalis_gene_ensembl
## 23           cjacchus_gene_ensembl
## 24          clanigera_gene_ensembl
## 25         cpalliatus_gene_ensembl
## 26         cporcellus_gene_ensembl
```

Exercise: (1 point)

Select dataset and get only genes from chromosome 2, and only chromosome, start, end, geneid/name and descrioption and hgnc_symbol. Use getBM() function.

The getBM() function is the main query function in biomaRt. It has four main arguments:

attributes: is a vector of attributes that one wants to retrieve (= the output of the query). filters: is a vector of filters that one wil use as input to the query. values: a vector of values for the filters. In case multple filters are in use, the values argument requires a list of values where each position in the list corresponds to the position of the filters in the filters argument (see examples below). mart: is an object of class Mart, which is created by the useMart() function.

Solution:

Select dataset and get only genes from chromosome 2, and only chromosome, start, end, geneid/name and descrioption and hgnc_symbol. Use getBM() function.

```
hgenes <- useDataset("hsapiens_gene_ensembl",mart=ensembl)
listAttributes(hgenes)
```

```
##                                                                 name
## 1                                                  ensembl_gene_id
## 2                                          ensembl_gene_id_version
## 3                                            ensembl_transcript_id
## 4                                   ensembl_transcript_id_version
## 5                                               ensembl_peptide_id
## 6                                      ensembl_peptide_id_version
## 7                                               ensembl_exon_id
## 8                                                   description
## 9                                               chromosome_name
## 10                                               start_position
## 11                                                 end_position
## 12                                                       strand
## 13                                                         band
## 14                                              transcript_start
## 15                                                transcript_end
## 16                                     transcription_start_site
## 17                                             transcript_length
## 18                                                transcript_tsl
## 19                                     transcript_gencode_basic
## 20                                             transcript_appris
## 21                                           external_gene_name
## 22                                       external_gene_source
## 23                                     external_transcript_name
```

Solution:

The end!

to be continued ...

The end!

to be continued ...