ggbio: visualization toolkits for genomic data

Tengfei Yin

January 21, 2013

Contents

Chapter 1

An Introduction to *ggbio*

1.1 Introduction

The *ggbio* package extends and specializes the grammar of graphics for biological data. The graphics are designed to answer common scientific questions, in particular those often asked of high throughput genomics data. All core Bioconductor data structures are supported, where appropriate. The package supports detailed views of particular genomic regions, as well as genome-wide overviews. Supported overviews include ideograms and grand linear views. High-level plots include sequence fragment length, edge-linked interval to data view, mismatch pileup, and several splicing summaries.

A mature graphic eco-system always has a well-developed data model, a grammar and a powerful computing platform. Grammar of graphics[1] is the essential part to help people understand the underlying data by using a general visualization framework. What's more, object-oriented graphics is especially useful for a well-developed infrastructure system that have carefully defined data model to store specific data sets for special purpose. Let's say, given a `GRanges` we know it represents annotated genetic intervals, given `TranscriptDb` we know it represents transcripts-centric annotation data, given `matrix`, in biology, we probably expect a heatmap.

Let's scrutinize what we have in R:

- **data model**: Bioconductor tries hard to define and generalize infrastructure for storing particular biological data. For example, we have *ExpressionSet* to store microarray data, we have *GappedAlignments* to store NGS alignments, and *IRanges* to represent numeric intervals. This is especially useful, which make object-oriented programming for specific biological questions much easier, and make object-oriented visualization possible in Bioconductor too.
- **Powerful computing platform**: R is a modern statistical computing environment, provides plenty of models and computing method for multivariate data analysis, at the same time, Bioconductor has numerous data mining tools in genetic analysis and other fields. These well-developed and tested tool kits make processing and analysis easier than before. And we have to pay attention to that many useful graphics are just statistical summary of raw data, so statistical transformation exists could be implemented as part of the visualization procedure.

[1] "The grammar of graphics" by Leland Wilkinson

- **The grammar of graphics**: This conceptual framework is proposed by Leland Wilkinson[2]. Hadley Wickham extended the grammar and also first implemented it in R in his package *ggplot2* with great success. *ggbio* is built on *ggplot2*2 and extends the grammar to genomic data with new features and extended components.

1.2 Documentation

From Bioconductor 2.11, I have two documentation:

- One is like all other bioconductor package, one single vignettes knited from sweave file. Yes, it's the one you are reading now. This vignette is trying to make a general tutorial for this graphical framework, with plenty of examples and case studies, following the logical order.
- The other source is under *ggbio* official websites, `http://tengfei.github.com/ggbio`, under *documentation* tab, I will use *knitr* to knit the Rd manual and put it under manual section(`http://tengfei.github.com/ggbio/docs/man`), so all the help manual with examples code hybrided with graphics is shown there only. It's a very good companion for this pdf based vignette, or R help, because you won't see vivid graphics in your help manual. Also more complete examples are present in the on-line help documentations too.

These two documentation are reproducible with the version of packages specified in `sessionInfo`, *knitr* is the key to make them reproducible. For more information about how those documentation generated, please visit *knitr*'s websites[3].

1.3 Support

As described on-line (`http://tengfei.github.com/ggbio/support.html`).

For issue/bug report and questions about usage, you could

- File a issue/bug report at `https://github.com/tengfei/ggbio/issues`, this will make sure I don't really forget to fix it later. *ggbio* is a huge and flexible package, combination of components are not all tested, you probably could hit a bug or issue the future, I will appreciate it if you could let me know it and help me improve and fix the problem.
- Send me email at yintengfei at gmail dot com directly.
- or ask question about *ggbio* on biocondcutor.

1.4 Installation

As described on-line (`http://tengfei.github.com/ggbio/download.html`).

[2] Please check Wilkinson's book "The grammar of graphics" for more detail.

[3] `http://yihui.name/knitr/`

Tips: **github** is only used for issue/bugs report and homepage build purpose, developemnt has been stopped and removed from there already. I only use bioconductor to maintain and develop my package.

After R 2.15, R release cycle falls into annual release instead of semi-annual release cycle, at the same time, Bioconductor project still follows semi-annual release cycle. So now you can install both released and developmental version for the same version of R.

In your R session, please run following code to install released version of ggbio, but if you are using developmental version of R, you will get developmental version of ggbio automatically. Because what you get depends on the bioconductor installer, which is implemented in package BiocInstaller and its version decides which version of Bioconductor you got.

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

After you run the code above, next time if you wish to install something new from Bioconductor, you can simply run

```
library("BiocInstaller")
biocLite("ggbio")
```

Or you can check all released bioc packages here.

To install developmental version, run

```
library("BiocInstaller")
useDevel(TRUE)
biocLite("ggbio")
```

For developers, please you can find latest source code in bioc svn, username and password are all ”readonly”(without quotes).

1.5 Citation

```
citation("ggbio")


##
## To cite package 'ggbio' in publications use:
##
##   Tengfei Yin, Dianne Cook and Michael Lawrence (2012): ggbio:
##   an R package for extending the grammar of graphics for
##   genomic data Genome Biology 13:R77
##
## A BibTeX entry for LaTeX users is
```

```
##
##   @Article{,
##     title = {ggbio: an R package for extending the grammar of graphics for genomic data},
##     author = {Tengfei Yin and Dianne Cook and Michael Lawrence},
##     journal = {Genome Biology},
##     volume = {13},
##     number = {8},
##     pages = {R77},
##     year = {2012},
##     publisher = {BioMed Central Ltd},
##   }
```

Chapter 2

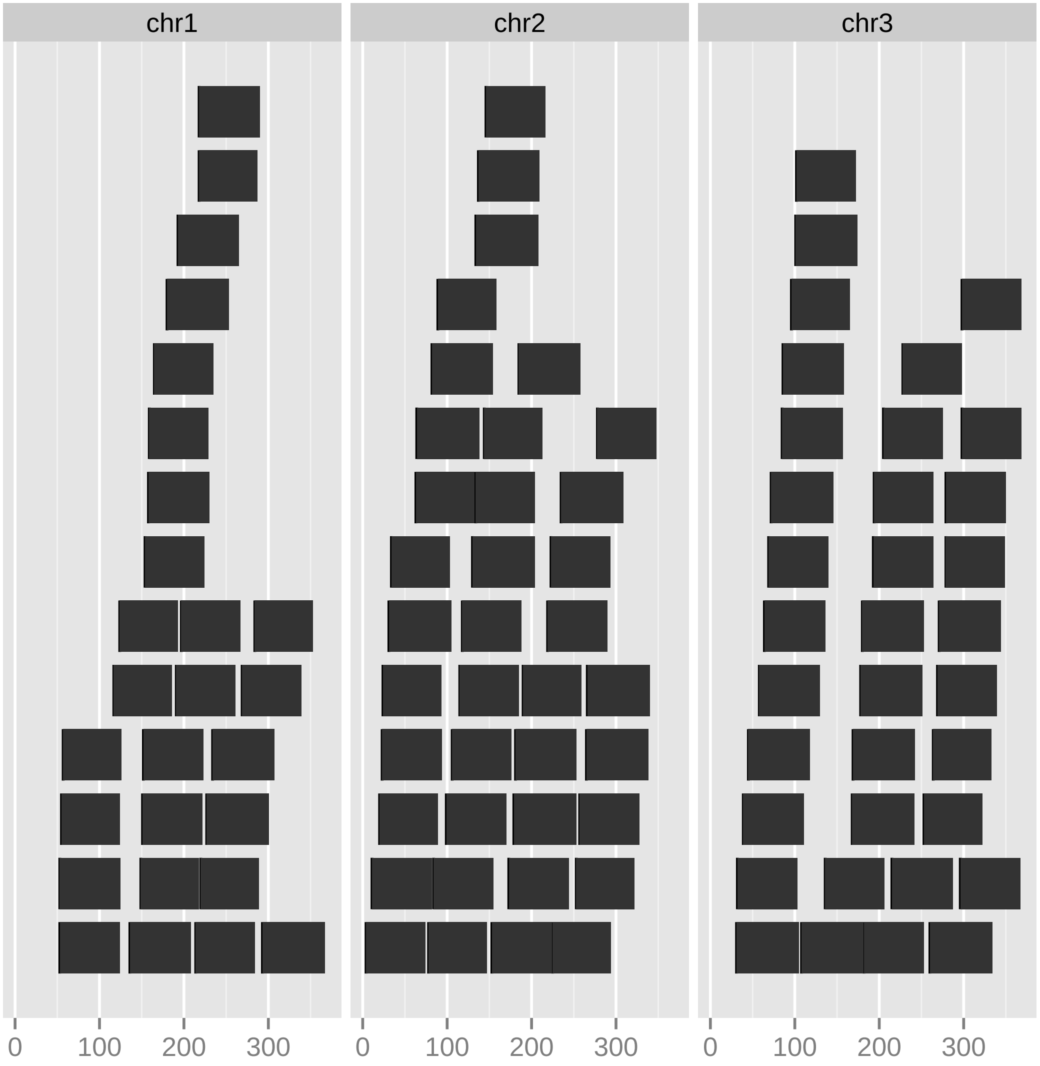
Quick start

This chapter gives your a very rough overview about the usage of *ggbio*, but not a complete coverage for all contents.

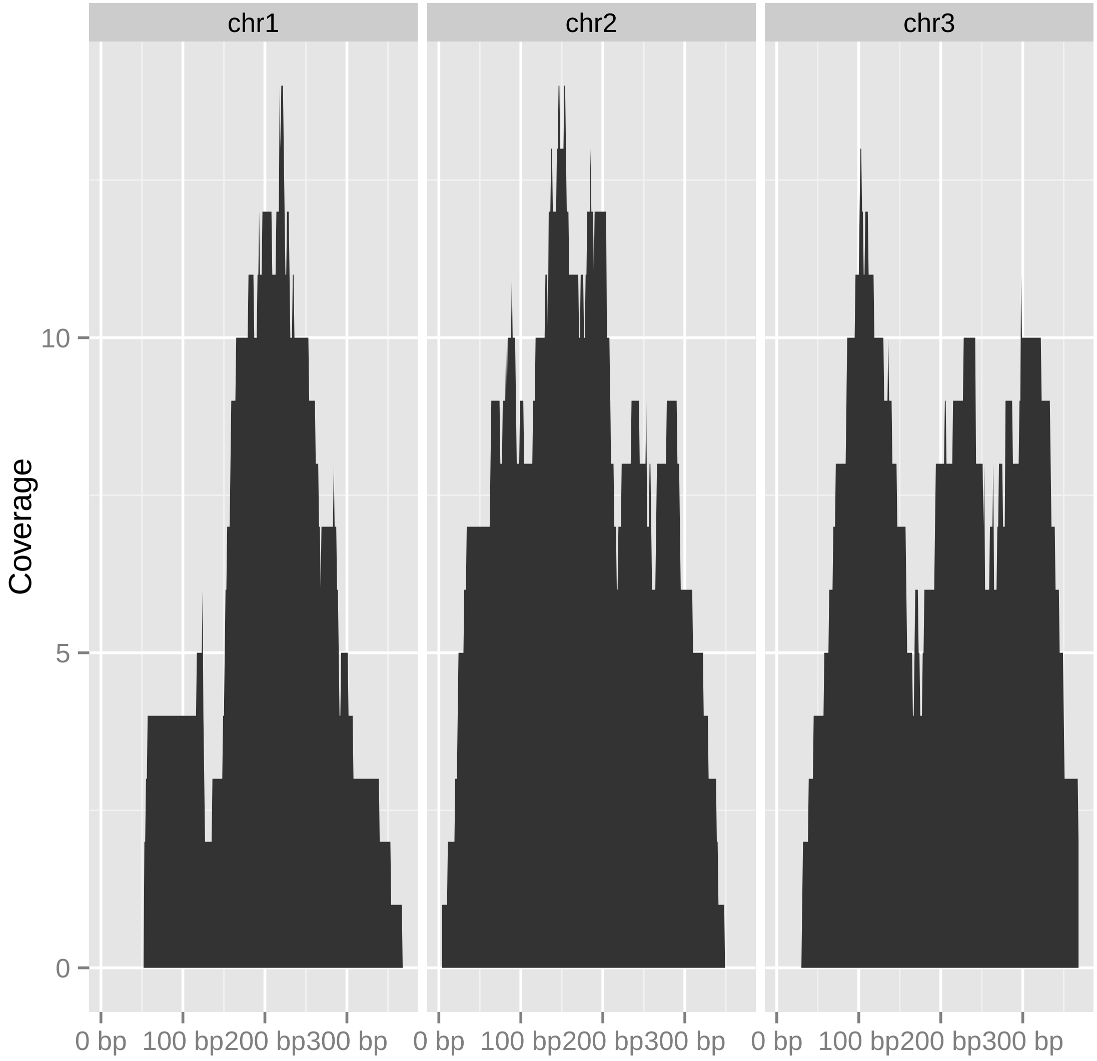
`autoplot` is the generic function which support most core Bioconductor objects, try to make different types of graphics for specific object. Please check Chapter 6 and manual for `autoplot` for more information.

```
library(ggbio)


## Loading required package:  ggplot2

## Need specific help about ggbio?  try mailing
##  the maintainer or visit http://tengfei.github.com/ggbio/

##
## Attaching package:  'ggbio'

## The following object(s) are masked from 'package:ggplot2':
##
##    geom_bar, geom_rect, geom_segment, stat_bin, stat_identity,
##    xlim


library(GenomicRanges)


## Loading required package:  BiocGenerics

##
## Attaching package:  'BiocGenerics'

## The following object(s) are masked from 'package:stats':
##
##    xtabs

## The following object(s) are masked from 'package:base':
##
##    Filter, Find, Map, Position, Reduce, anyDuplicated, cbind,
##    colnames, duplicated, eval, get, intersect, lapply, mapply,
##    mget, order, paste, pmax, pmax.int, pmin, pmin.int, rbind,
##    rep.int, rownames, sapply, setdiff, table, tapply, union,
##    unique

## Loading required package:  IRanges
```

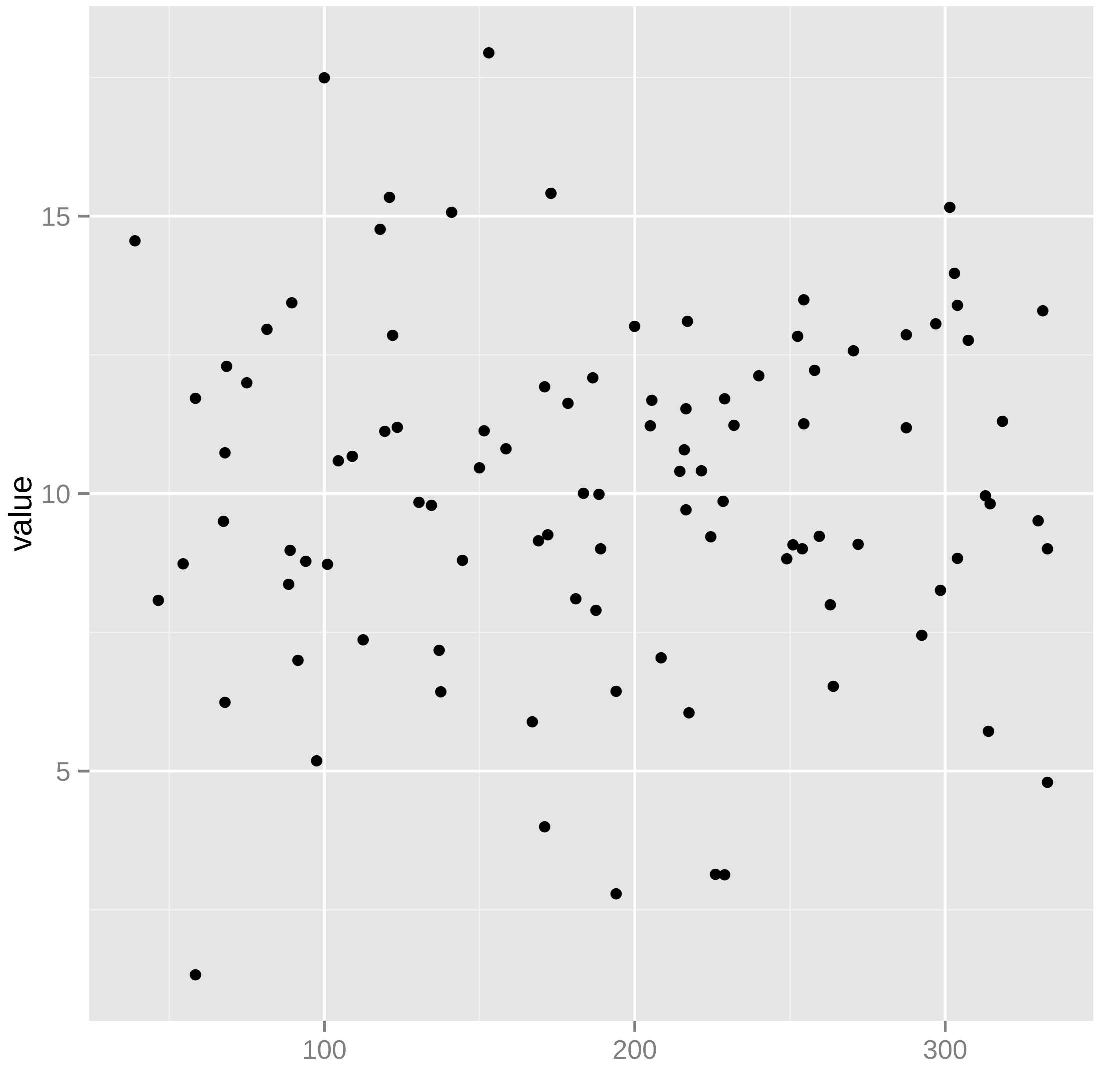
```
set.seed


      GRanges            sample c "chr1"  "chr2"  "chr3"
    IRanges         sample                                           sample
                                             sample c "+"  "-"  "*"
                                 rnorm                    rnorm
             sample c "Normal"  "Tumor"                                     sample

autoplot
```



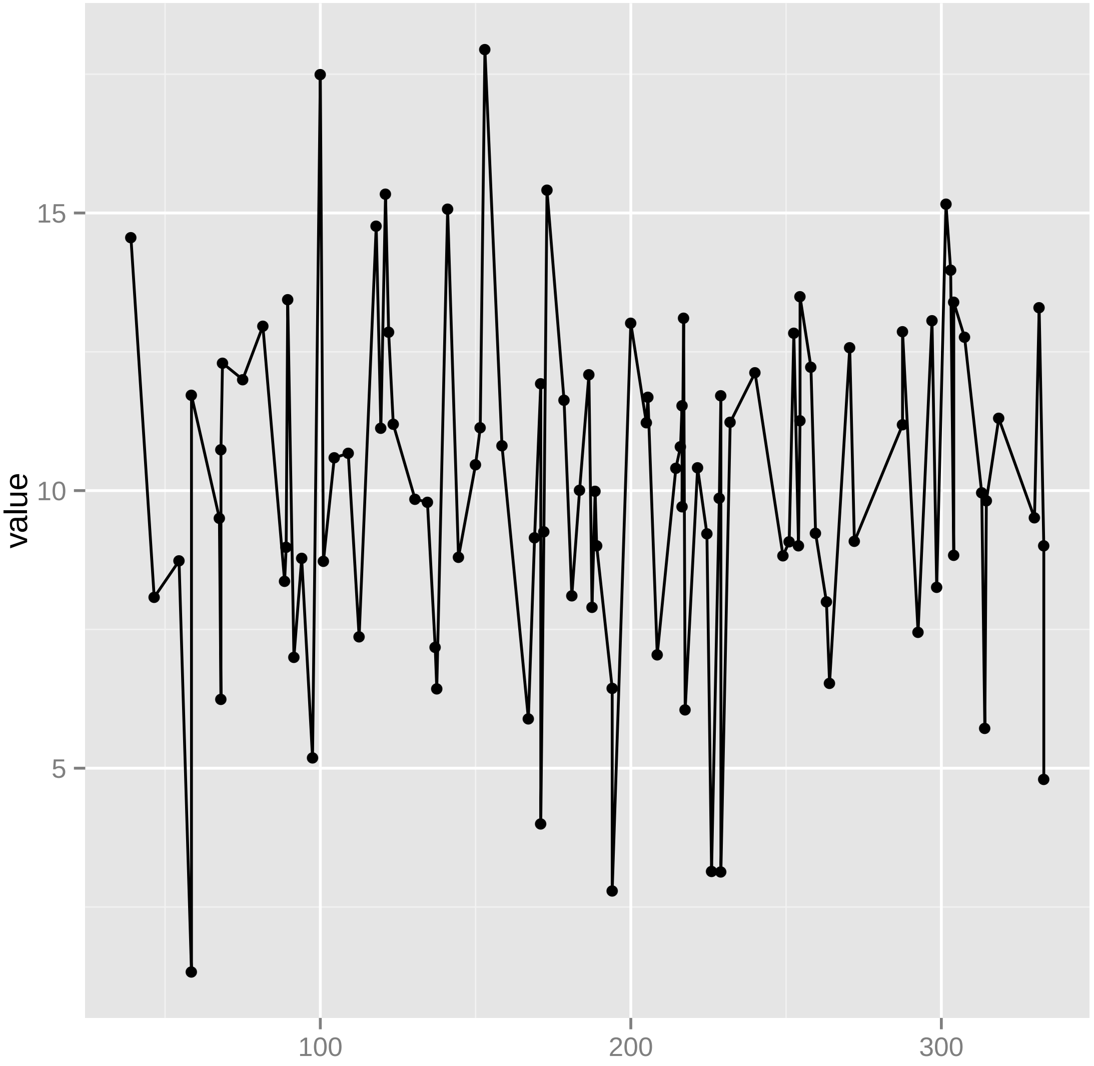
```
## NULL

autoplot(gr, stat = "coverage", geom = "area")

## Object of class "ggbio"
```



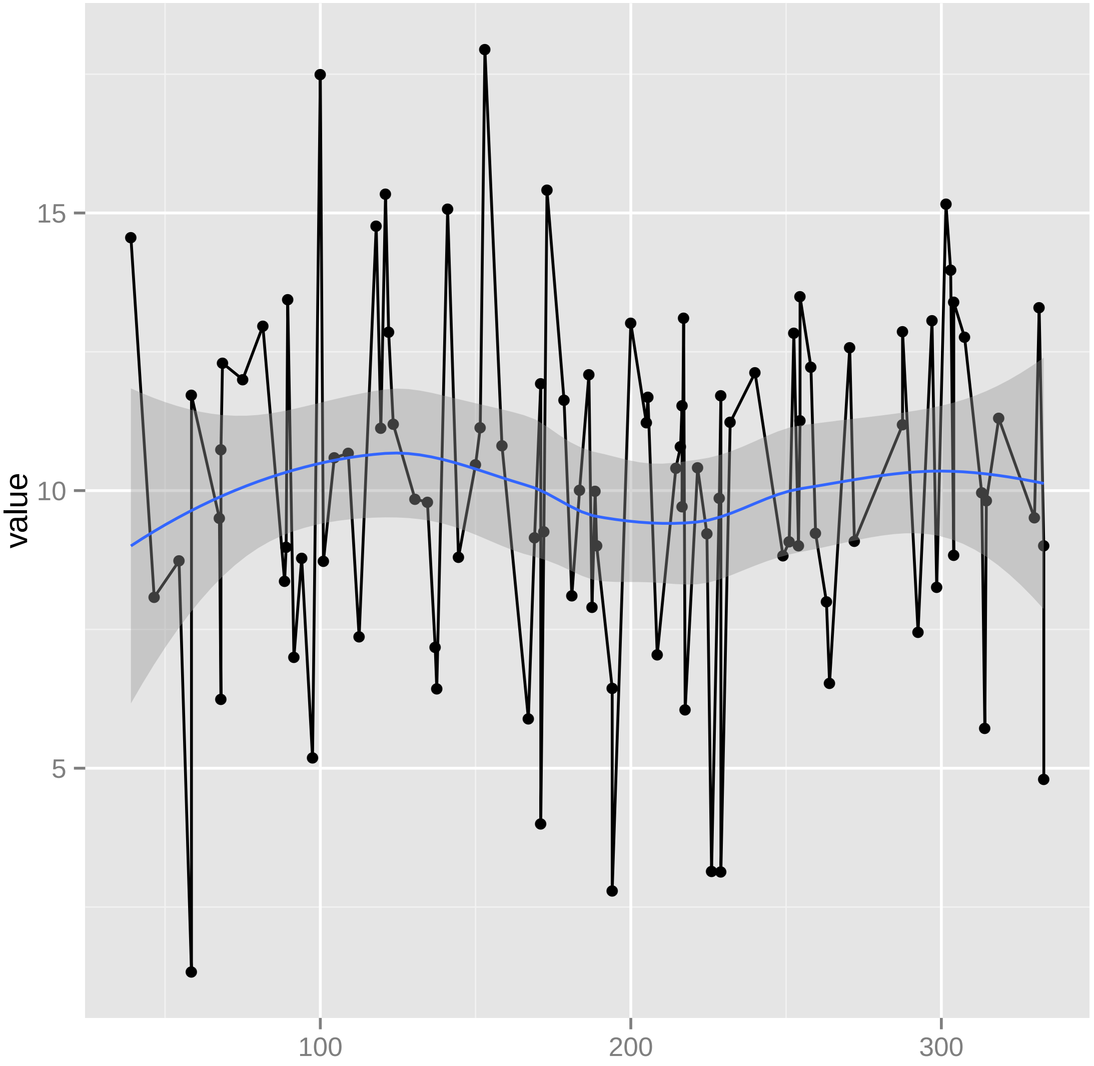
```
## NULL

autoplot(gr, aes(y = value), geom = "point")

## Object of class "ggbio"
```



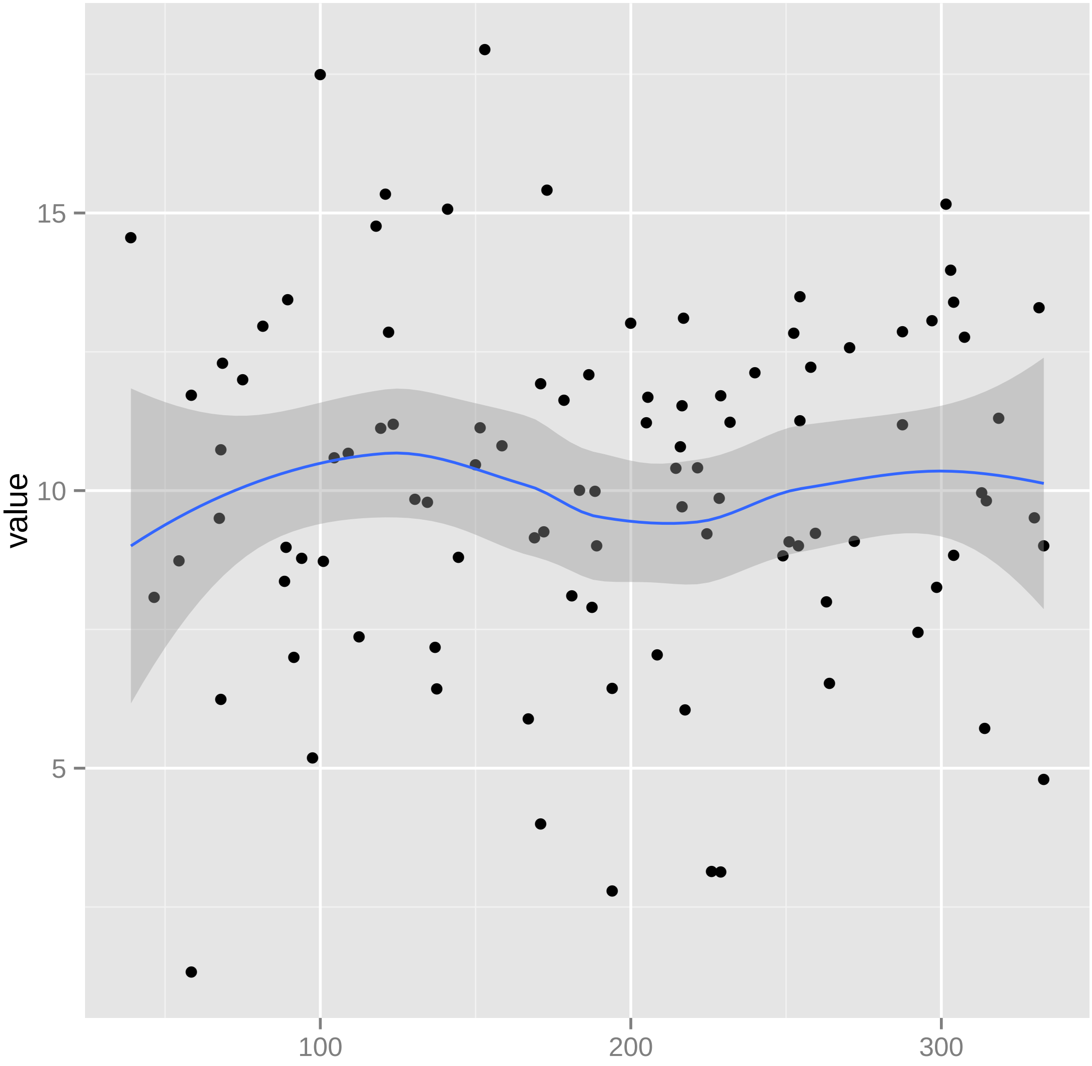
```
## NULL

autoplot(gr, aes(y = value), geom = "point") + geom_line()

## Object of class "ggbio"
```



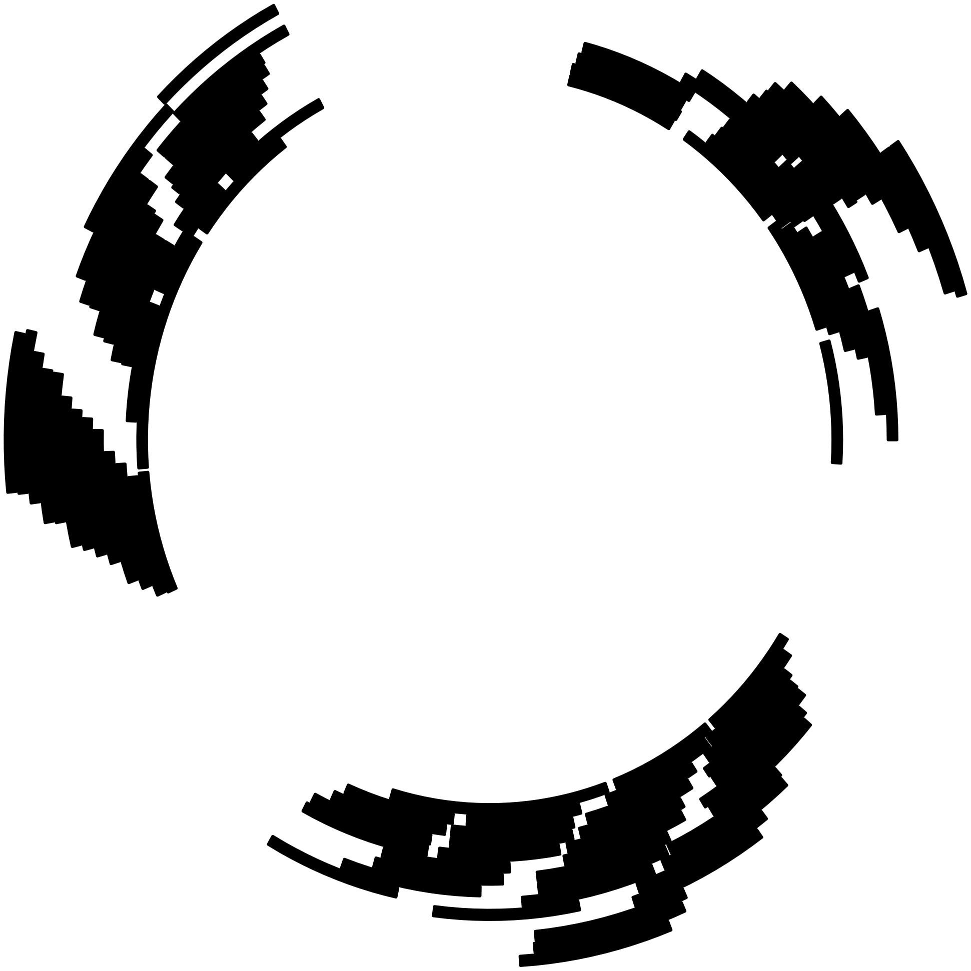
```
## NULL

autoplot(gr, aes(y = value), geom = "point") + geom_line() + stat_smooth()

## Object of class "ggbio"

## geom_smooth:  method="auto" and size of largest group is <1000, so using loess.  Use 'method
= x' to change the smoothing method.
```



```
## NULL

autoplot(gr, aes(y = value), geom = "point") + stat_smooth()

## Object of class "ggbio"

## geom_smooth:  method="auto" and size of largest group is <1000, so using loess.  Use 'method
= x' to change the smoothing method.
```



```
## NULL

autoplot(gr, layout = "circle")

## Object of class "ggbio"
```



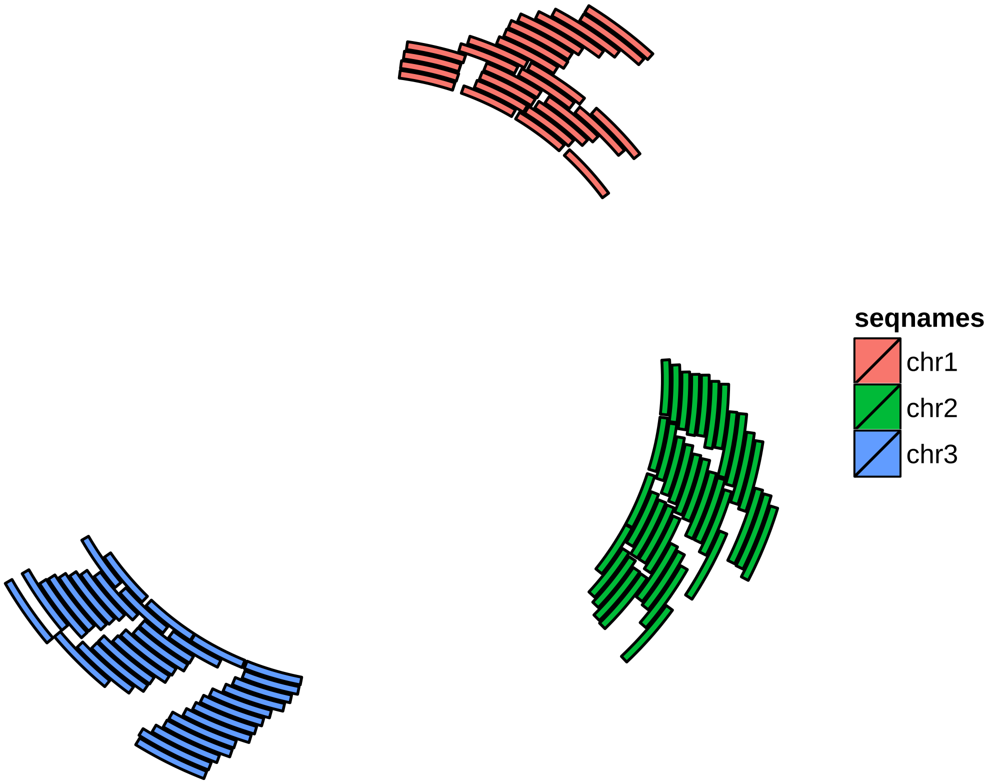
```
## NULL
```

```
seqlengths(gr)
```

```
## chr1 chr2 chr3
##   NA   NA   NA
```

```
seqlengths(gr) <- c(400, 500, 1000)
autoplot(gr, layout = "circle", aes(fill = seqnames))
```

```
## Object of class "ggbio"
```

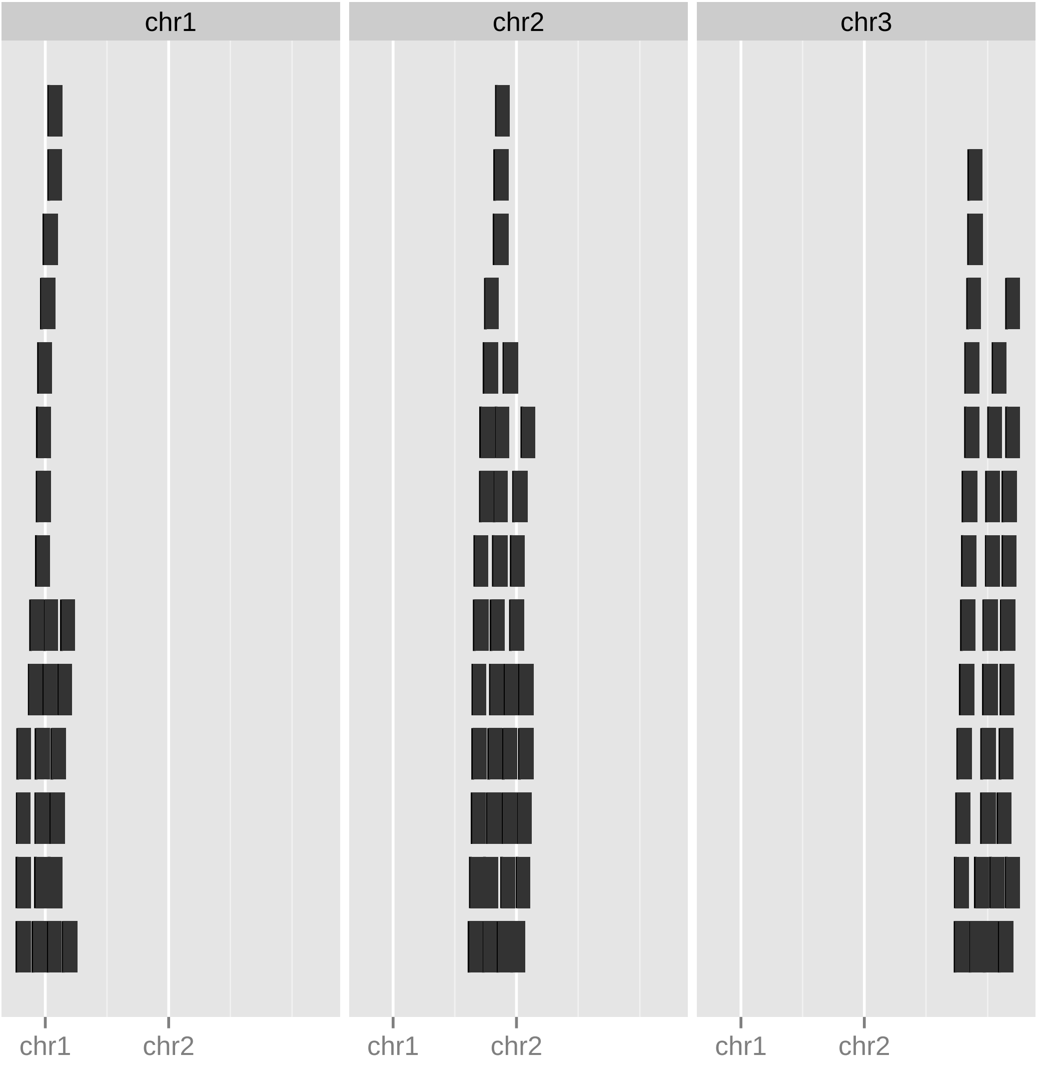


```
## NULL
```

```
autoplot(gr, coord = "genome")
```

```
## using coord:genome to parse x scale
```

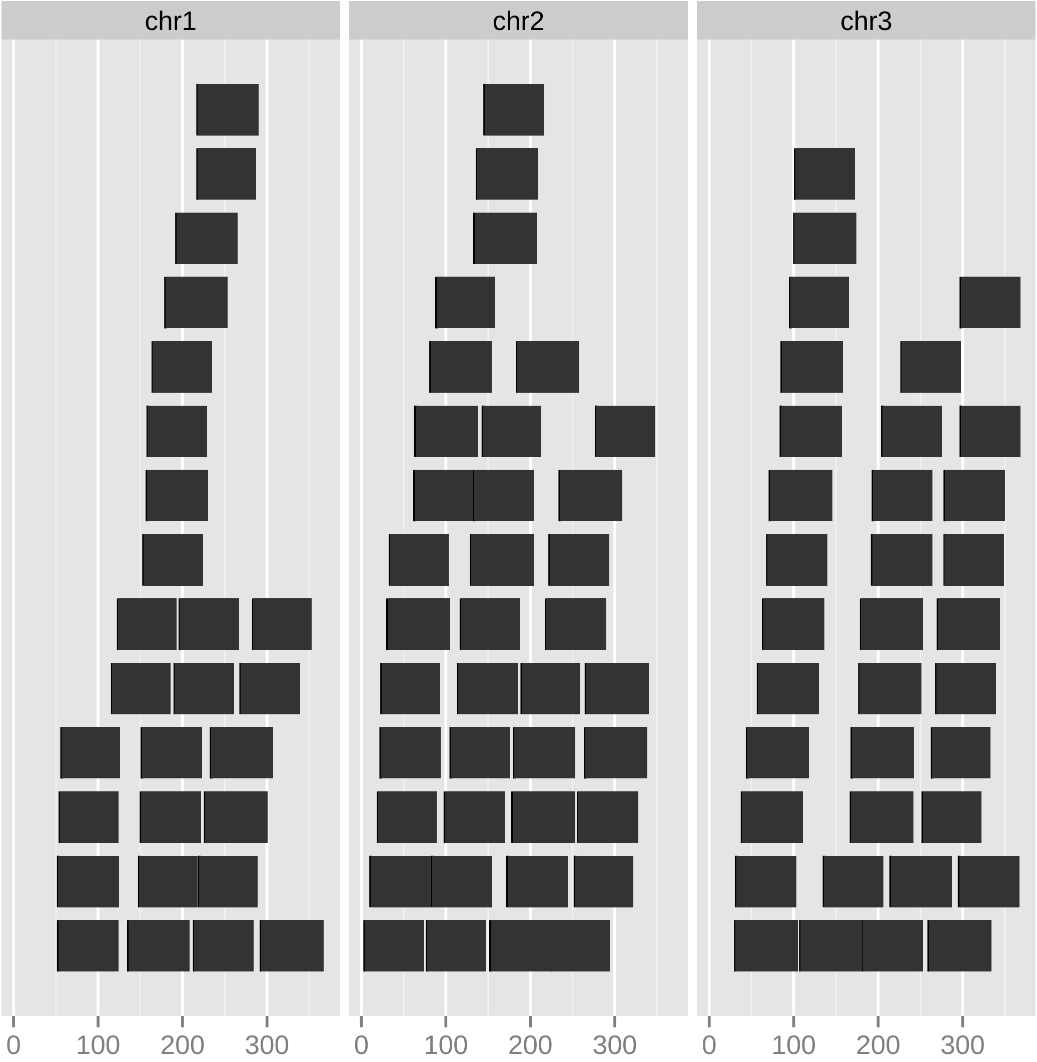
```
## Object of class "ggbio"
```



```
## NULL
```

`ggplot` generic method provides flexible API for constructing graphics layer by layer following the grammar of graphics. Actually `autoplot` method use `ggplot` and other low level utilities to construct customized graphics. Please check Chapter 5 and manual for more information.

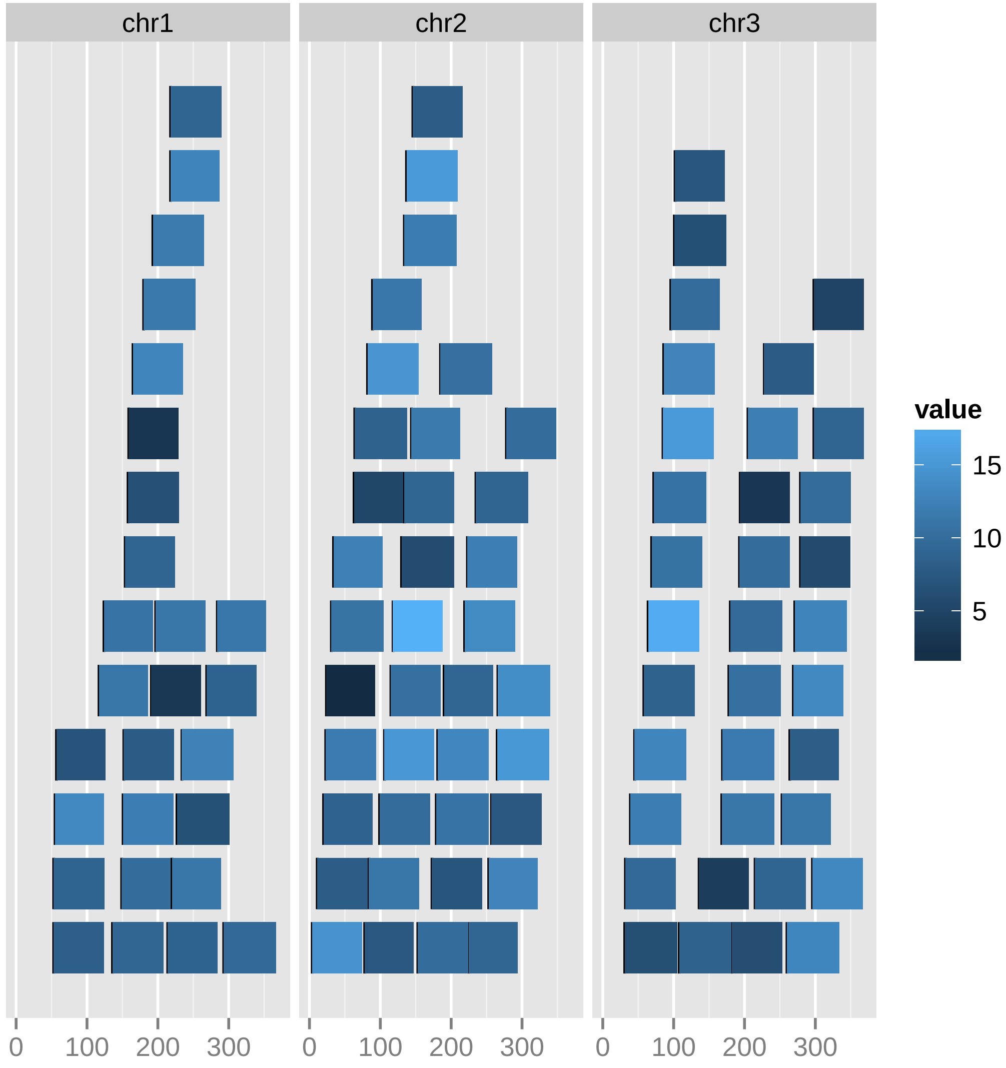
```
ggplot(gr) + geom_rect()


## Object of class "ggbio"
```



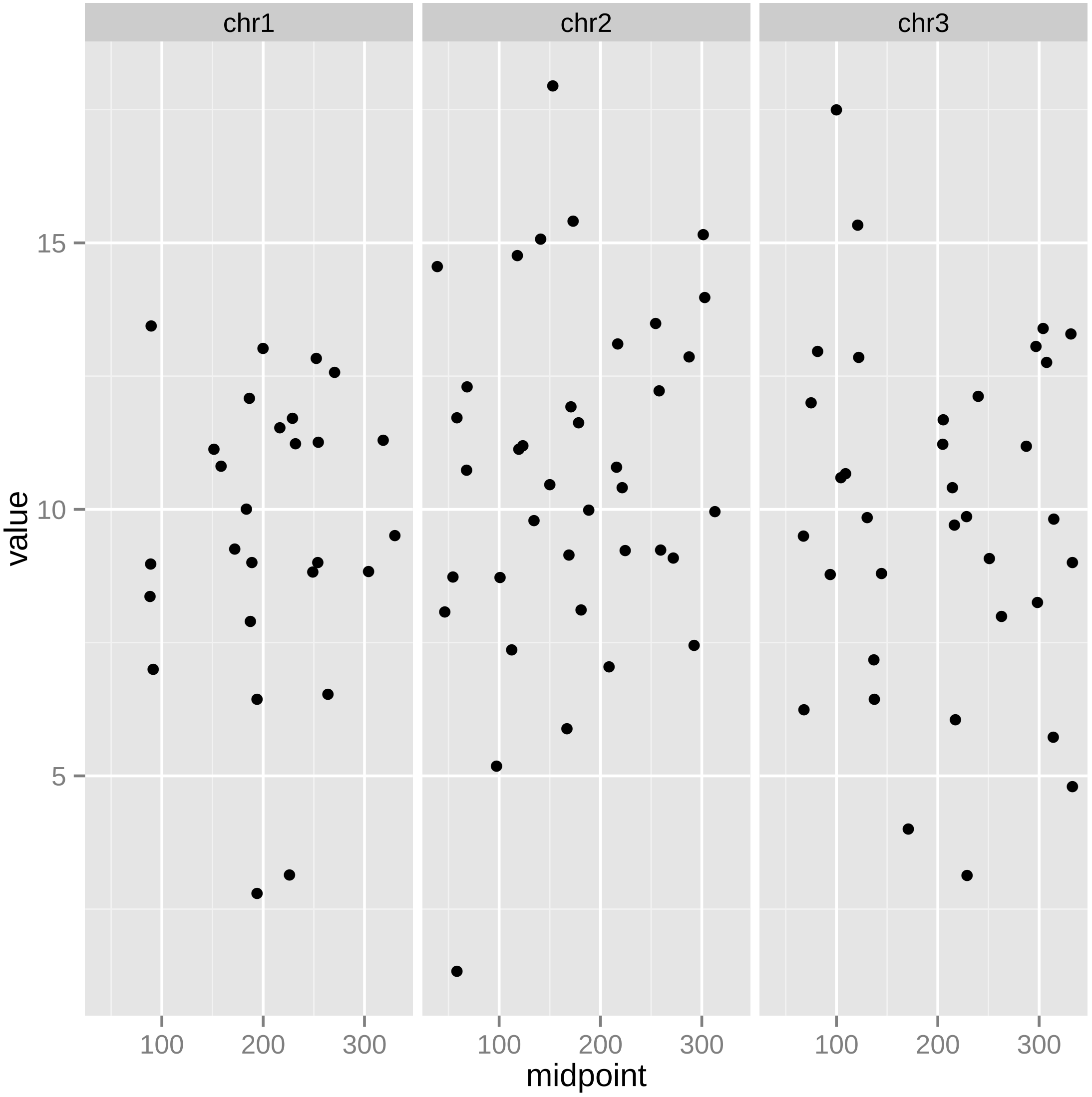
```
## NULL
```

```
ggplot(gr) + geom_rect(aes(fill = value))
```

```
## Object of class "ggbio"
```



```
## NULL

## for primitive geom from ggplot2, add facet manually for now
ggplot(gr, aes(x = midpoint, y = value)) + geom_point() + facet_grid(. ~
    seqnames)

## Object of class "ggbio"
```



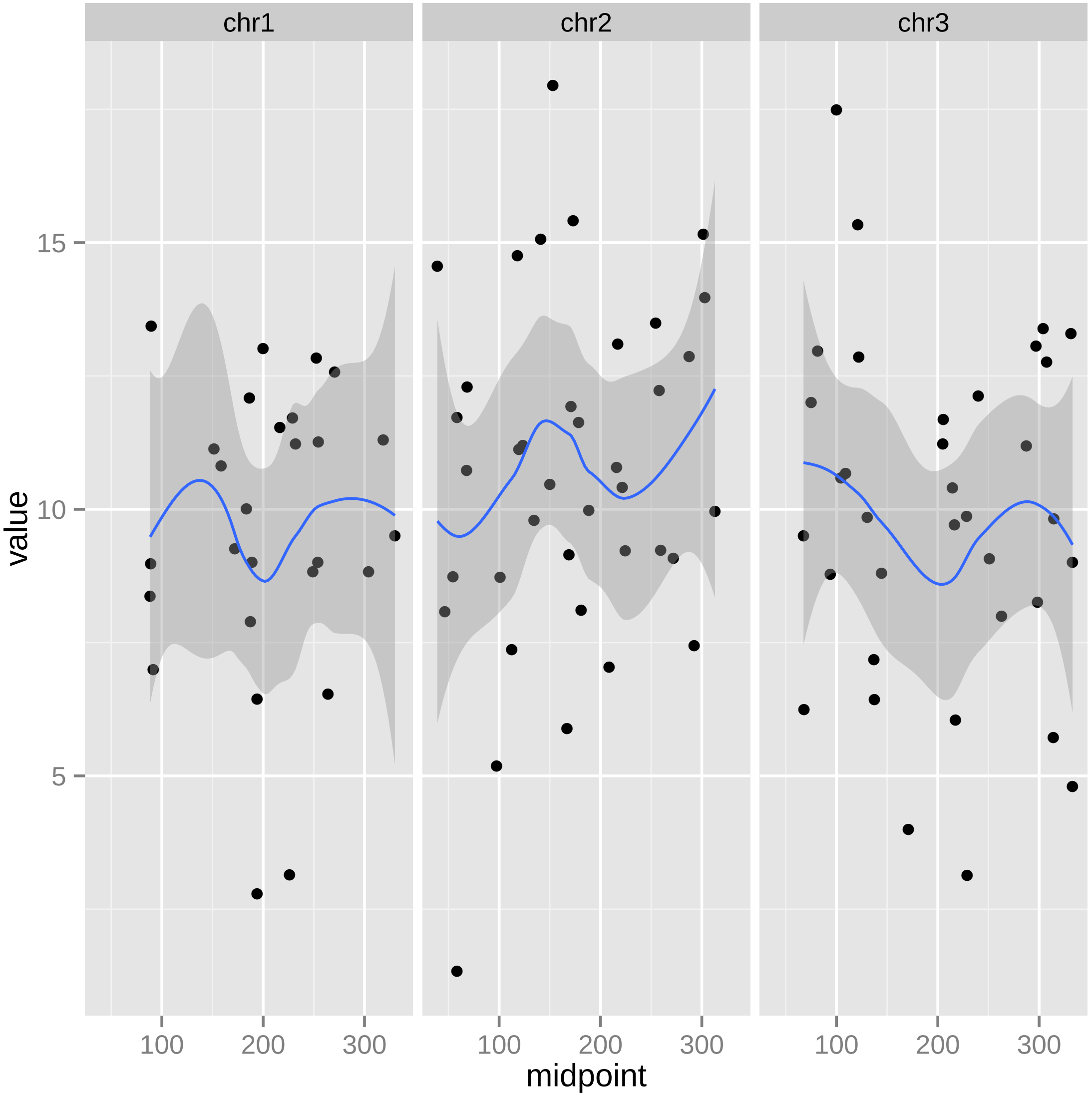
```
## NULL
```

```
ggplot(gr, aes(x = midpoint, y = value)) + facet_grid(. ~ seqnames) + geom_point() +
    stat_smooth()
```

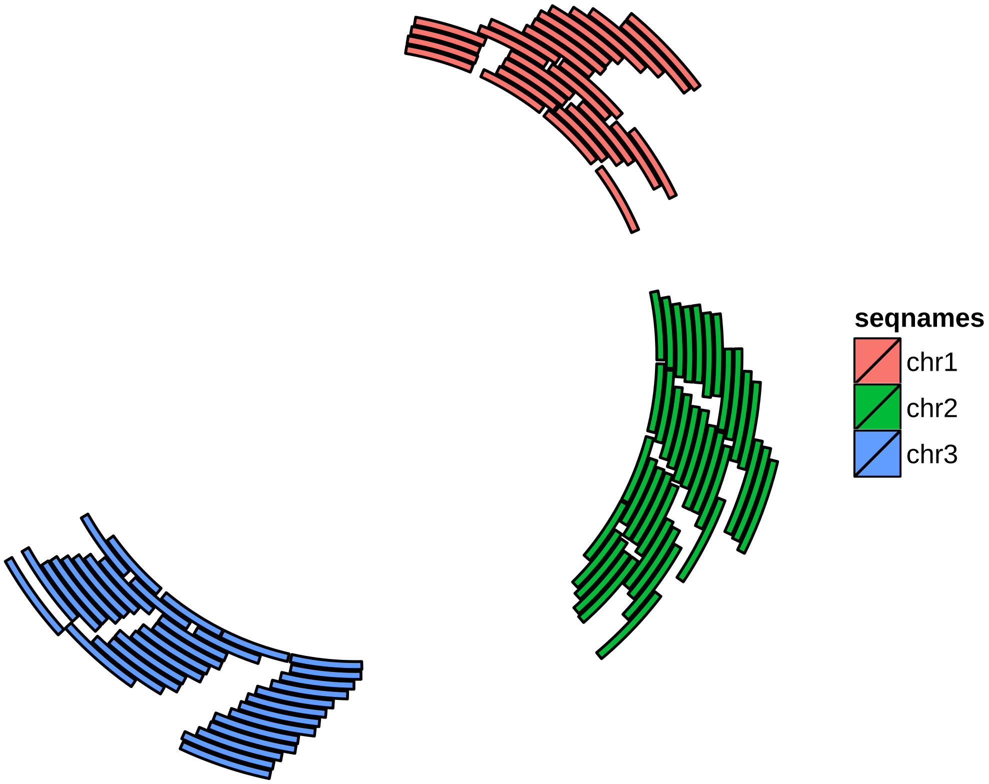
```
## Object of class "ggbio"
```

```
## geom_smooth:  method="auto" and size of largest group is <1000, so using loess.  Use 'method
= x' to change the smoothing method.

## geom_smooth:  method="auto" and size of largest group is <1000, so using loess.  Use 'method
= x' to change the smoothing method.
```

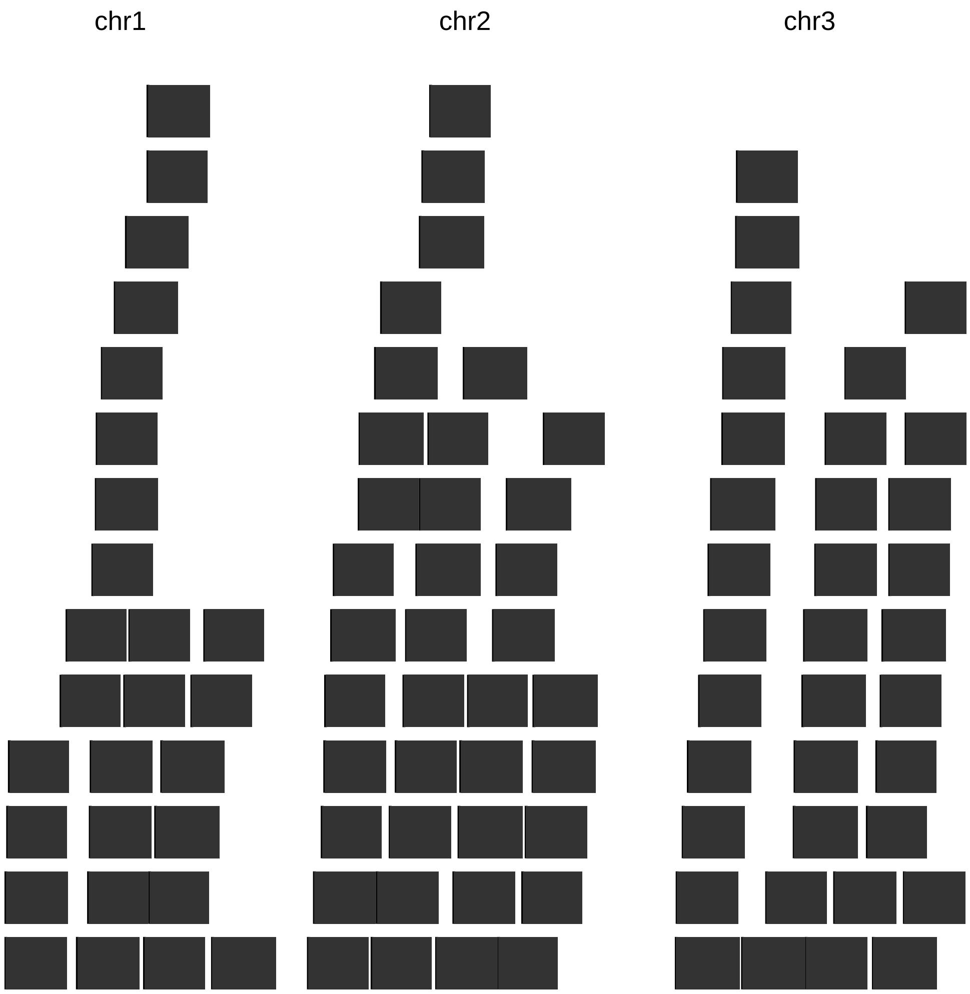
```
## geom_smooth:  method="auto" and size of largest group is <1000, so using loess.  Use 'method
= x' to change the smoothing method.
```



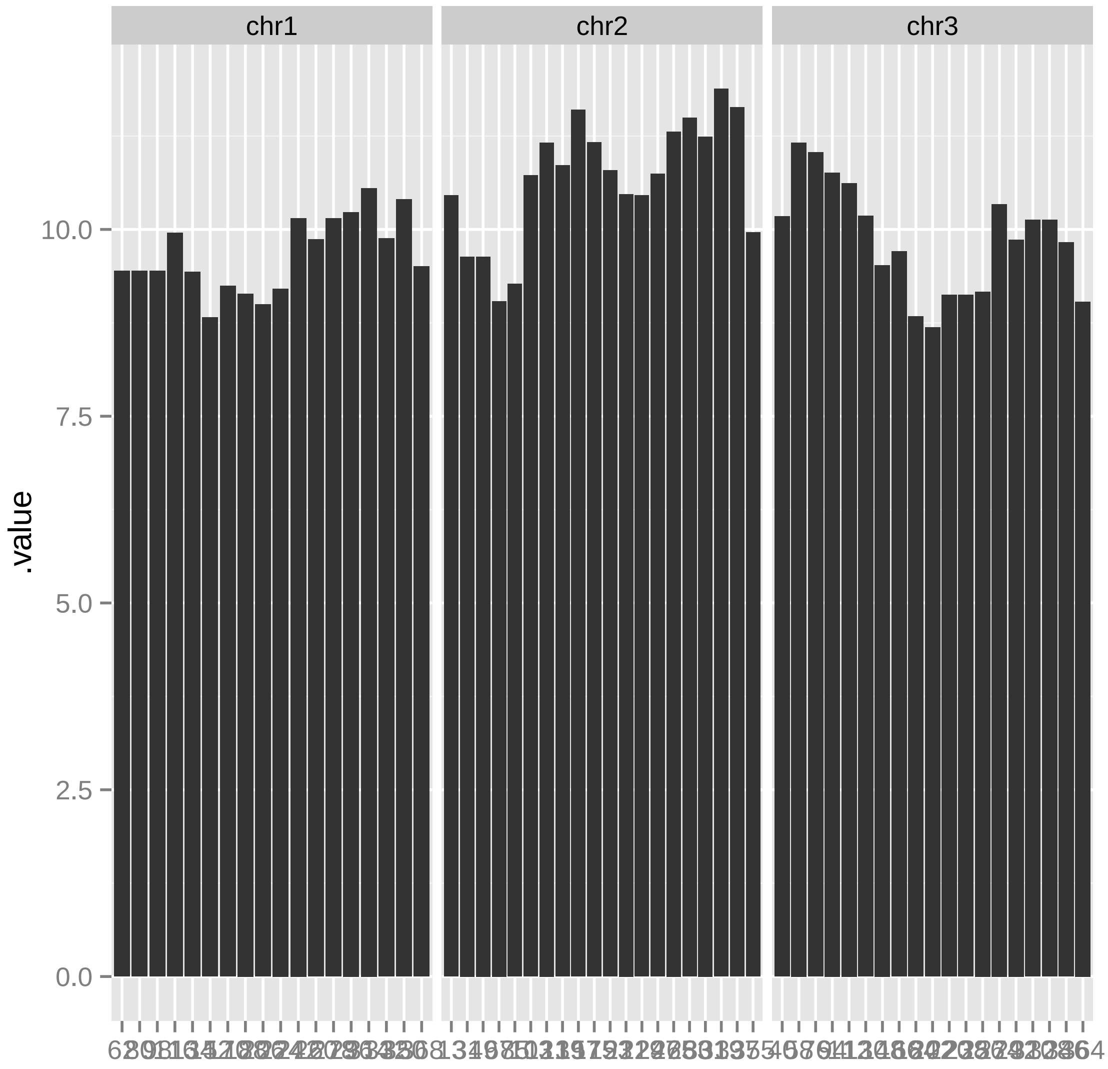
```
## NULL

ggplot(gr) + layout_circle(aes(fill = seqnames), geom = "rect")

## Object of class "ggbio"
```



```
## NULL

## slightly different with autoplot api
ggplot(gr) + geom_rect() + coord_genome()

## Scale for 'y' is already present.  Adding another scale for 'y', which will replace the
existing scale.

## Object of class "ggbio"
```



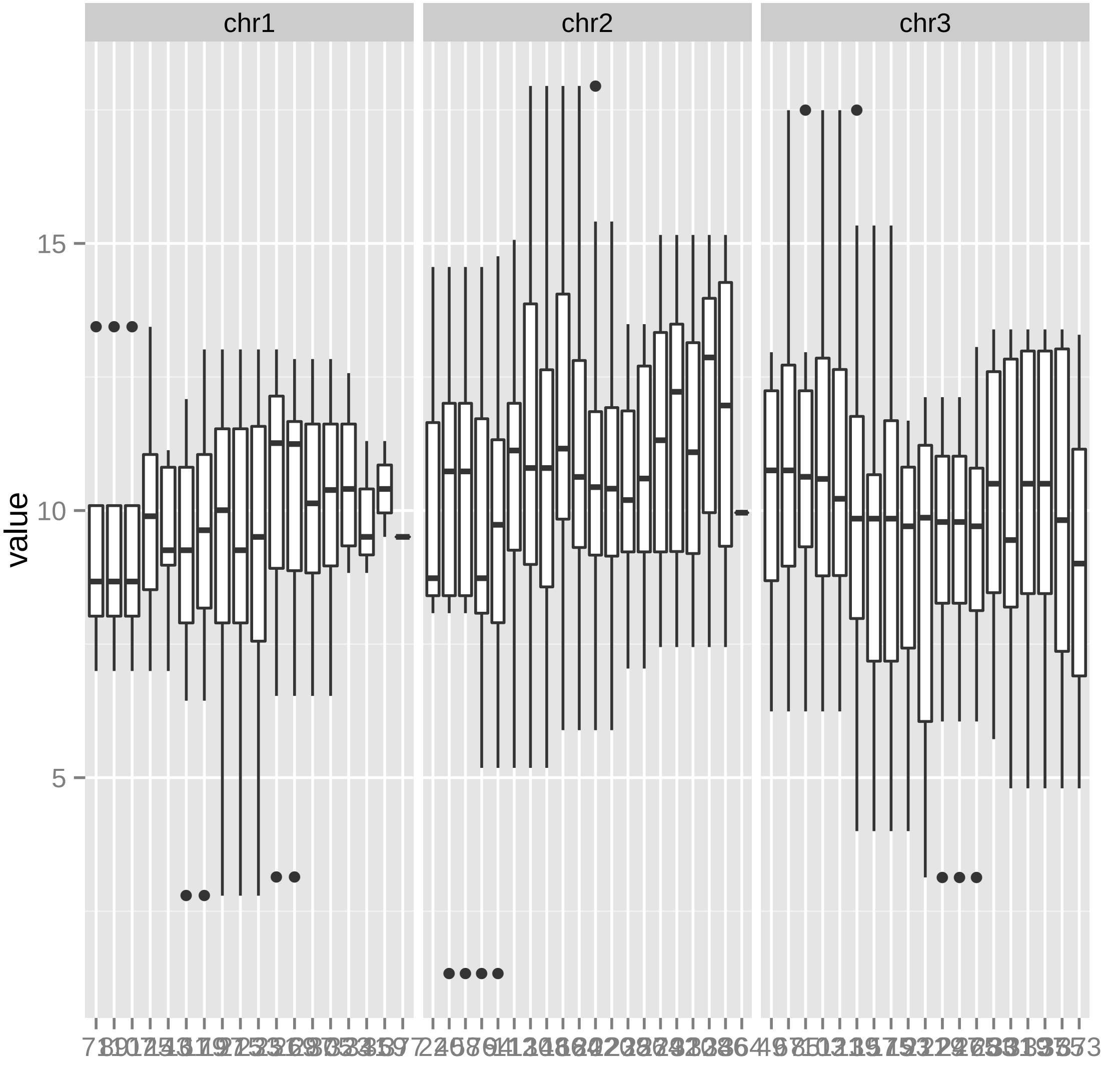
```
## NULL

ggplot(gr) + stat_aggregate(aes(y = value))

## Object of class "ggbio"
```



```
## NULL
```

```
ggplot(gr) + stat_aggregate(aes(y = value), geom = "boxplot")
```

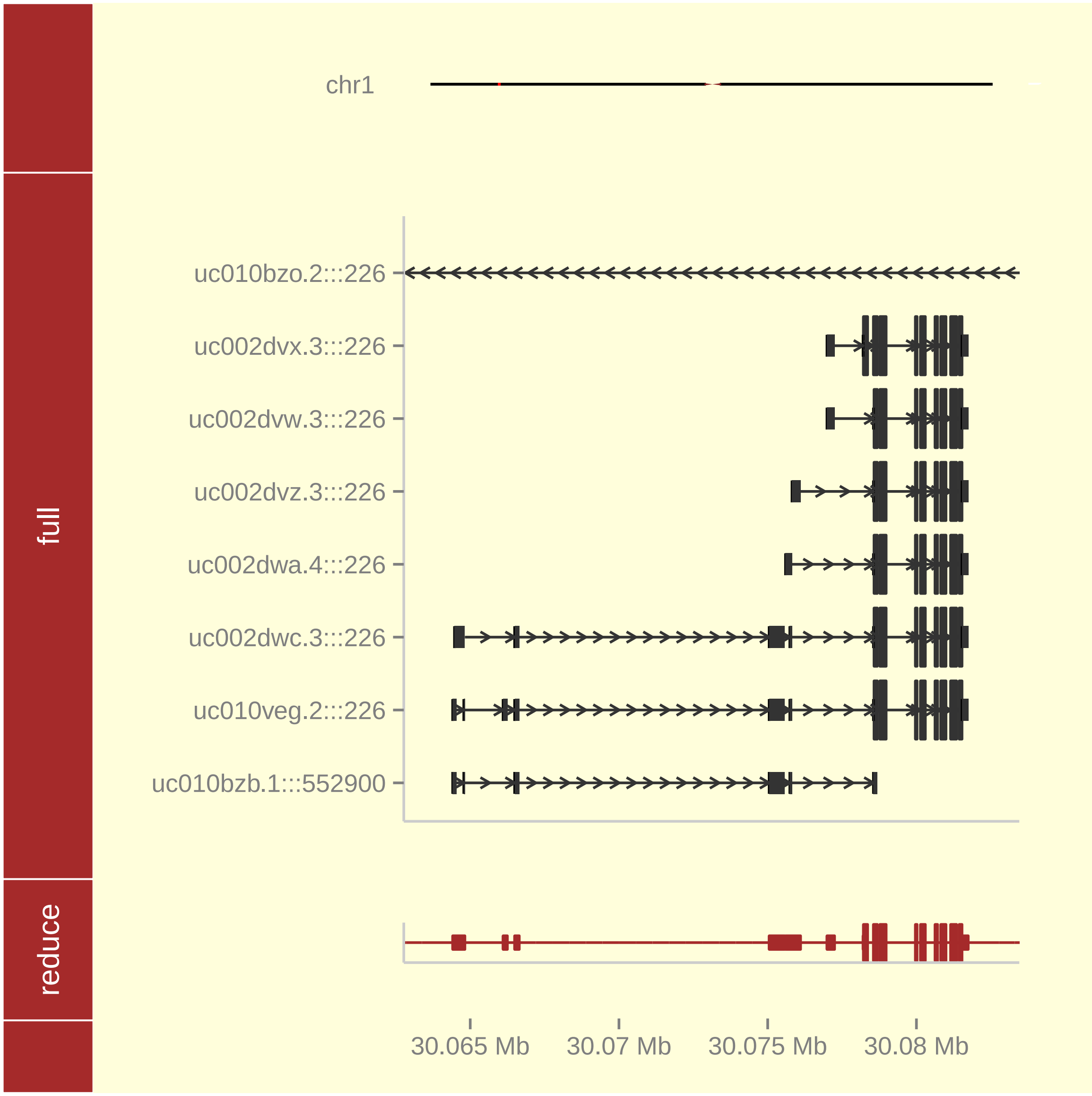
```
## Object of class "ggbio"
```



```
## NULL
```

`plotSingleChrom` and `plotIdeogram` provides functionality to construct ideogram and you could download it on the fly or save it and use it later, `tracks` function provides convenient control to bind your individual graphics as tracks. Please check Chapter 3 about tracks and Chapter7 about ideogram and manual for more information.

```
library(ggbio)
## require internet connection
p.ideo <- plotIdeogram(genome = "hg19")

## Loading required package:  rtracklayer

## Loading...
## Done

## use chr1 automatically

library(TxDb.Hsapiens.UCSC.hg19.knownGene)

## Loading required package:  GenomicFeatures

## Loading required package:  AnnotationDbi

## Loading required package:  Biobase

## Welcome to Bioconductor
##
##     Vignettes contain introductory material; view with
##     'browseVignettes()'.  To cite Bioconductor, see
##     'citation("Biobase")', and for packages
##     'citation("pkgname")'.

txdb <- TxDb.Hsapiens.UCSC.hg19.knownGene
wh <- GRanges("chr16", IRanges(30064491, 30081734))
p1 <- autoplot(txdb, which = wh, names.expr = "tx_name:::gene_id")

## Aggregating TranscriptDb...

## Parsing exons...

## Parsing cds...

## Parsing transcripts...

## Aggregating...

## Done

## Constructing graphics...

p2 <- autoplot(txdb, which = wh, stat = "reduce", color = "brown", fill = "brown")

## Aggregating TranscriptDb...

## Parsing exons...

## Parsing cds...

## Parsing transcripts...

## Aggregating...

## Done

## Constructing graphics...

tracks(p.ideo, full = p1, reduce = p2, heights = c(1.2, 5, 1)) + ylab("") +
    theme_tracks_sunset()
```



`plotGrandLinear` to plot the whole genome Manhattan plot. Please check Chapter 10 and manual for more information.

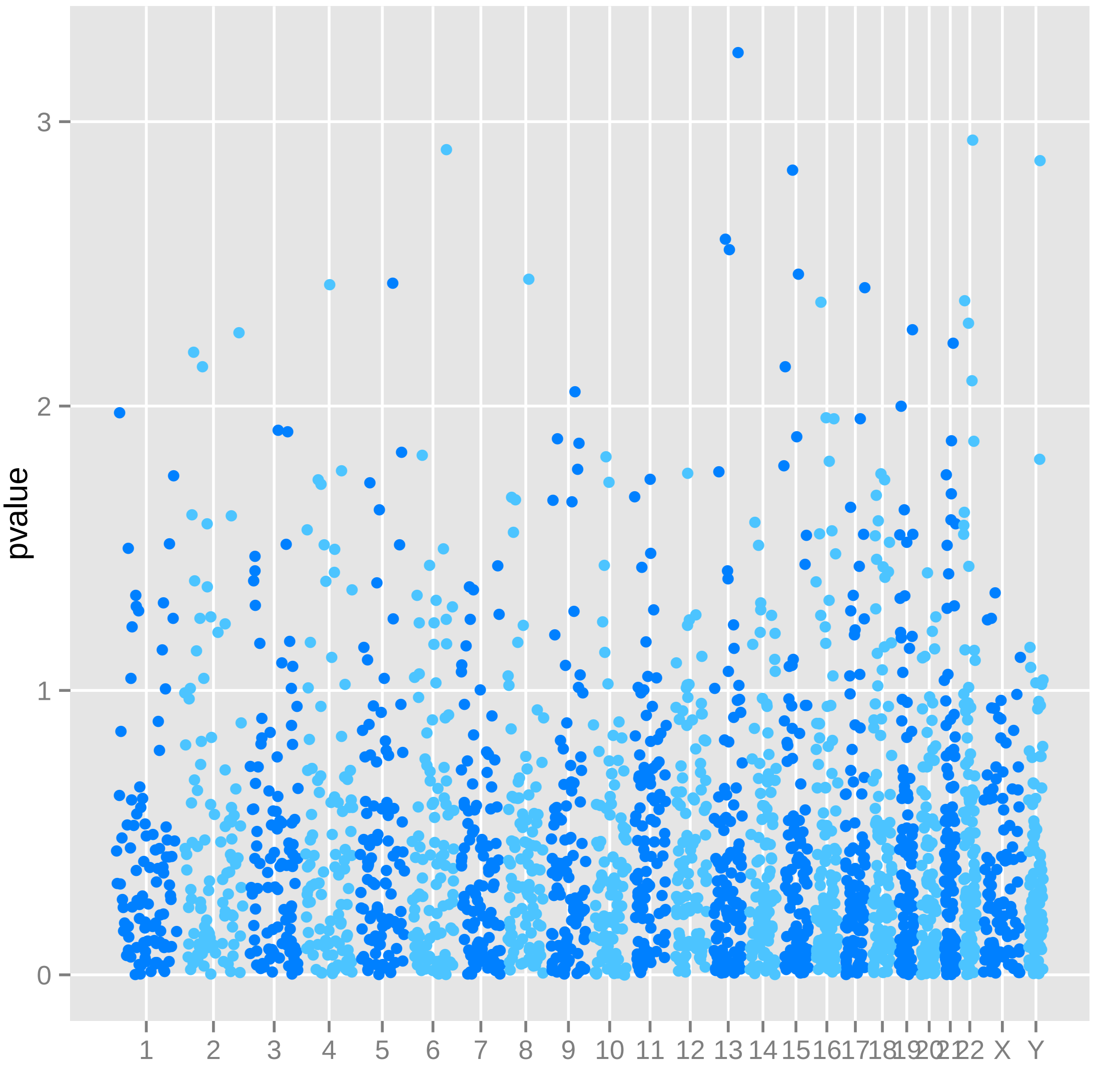
```
data(hg19IdeogramCyto, package = "biovizBase")
data(hg19Ideogram, package = "biovizBase")
chrs <- as.character(levels(seqnames(hg19IdeogramCyto)))
seqlths <- seqlengths(hg19Ideogram)[chrs]
set.seed(1)
nchr <- length(chrs)
nsnps <- 100
gr.snp <- GRanges(rep(chrs, each = nsnps), IRanges(start = do.call(c, lapply(chrs,
    function(chr) {
        N <- seqlths[chr]
        runif(nsnps, 1, N)
    })), width = 1), SNP = sapply(1:(nchr * nsnps), function(x) paste("rs",
    x, sep = "")), pvalue = -log10(runif(nchr * nsnps)), group = sample(c("Normal",
    "Tumor"), size = nchr * nsnps, replace = TRUE))
genome(gr.snp) <- "hg19"
nms <- seqnames(seqinfo(gr.snp))
nms.new <- gsub("chr", "", nms)
names(nms.new) <- nms
gr.snp <- renameSeqlevels(gr.snp, nms.new)
gr.snp <- keepSeqlevels(gr.snp, c(1:22, "X", "Y"))
gr.snp
```

```
## GRanges with 2400 ranges and 3 metadata columns:
##          seqnames                 ranges strand   |         SNP
##             <Rle>              <IRanges>  <Rle>   | <character>
##      [1]        1 [ 66178199,  66178199]      *   |         rs1
##      [2]        1 [ 92752113,  92752113]      *   |         rs2
##      [3]        1 [142784056, 142784056]      *   |         rs3
##      [4]        1 [226371355, 226371355]      *   |         rs4
##      [5]        1 [ 50269347,  50269347]      *   |         rs5
##      [6]        1 [223924186, 223924186]      *   |         rs6
##      [7]        1 [235460897, 235460897]      *   |         rs7
##      [8]        1 [164704260, 164704260]      *   |         rs8
##      [9]        1 [156807066, 156807066]      *   |         rs9
##      ...      ...                    ...    ... ...         ...
##   [2392]        Y   [36501485, 36501485]      *   |      rs2392
##   [2393]        Y   [30054272, 30054272]      *   |      rs2393
##   [2394]        Y   [20065602, 20065602]      *   |      rs2394
##   [2395]        Y   [19541601, 19541601]      *   |      rs2395
##   [2396]        Y   [34038689, 34038689]      *   |      rs2396
##   [2397]        Y   [ 3010837,  3010837]      *   |      rs2397
##   [2398]        Y   [23806602, 23806602]      *   |      rs2398
##   [2399]        Y   [15474595, 15474595]      *   |      rs2399
##   [2400]        Y   [10016302, 10016302]      *   |      rs2400
##              pvalue       group
##           <numeric> <character>
##      [1]   1.22380      Normal
##      [2]   1.27916      Normal
##      [3]   0.01199       Tumor
##      [4]   0.09985      Normal
##      [5]   1.49938       Tumor
##      [6]   0.26497       Tumor
##      [7]   1.75456       Tumor
##      [8]   0.10976       Tumor
##      [9]   0.12073       Tumor
##      ...       ...         ...
##   [2392]   0.93515      Normal
##   [2393]   0.08353       Tumor
##   [2394]   0.05148      Normal
##   [2395]   0.01483      Normal
##   [2396]   0.17601      Normal
##   [2397]   0.78685       Tumor
##   [2398]   0.48952      Normal
##   [2399]   0.60000      Normal
##   [2400]   0.03967      Normal
##   ---
##   seqlengths:
##      1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22  X  Y
##     NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA
```

```
plotGrandLinear(gr.snp, aes(y = pvalue))
```

```
## using coord:genome to parse x scale
```

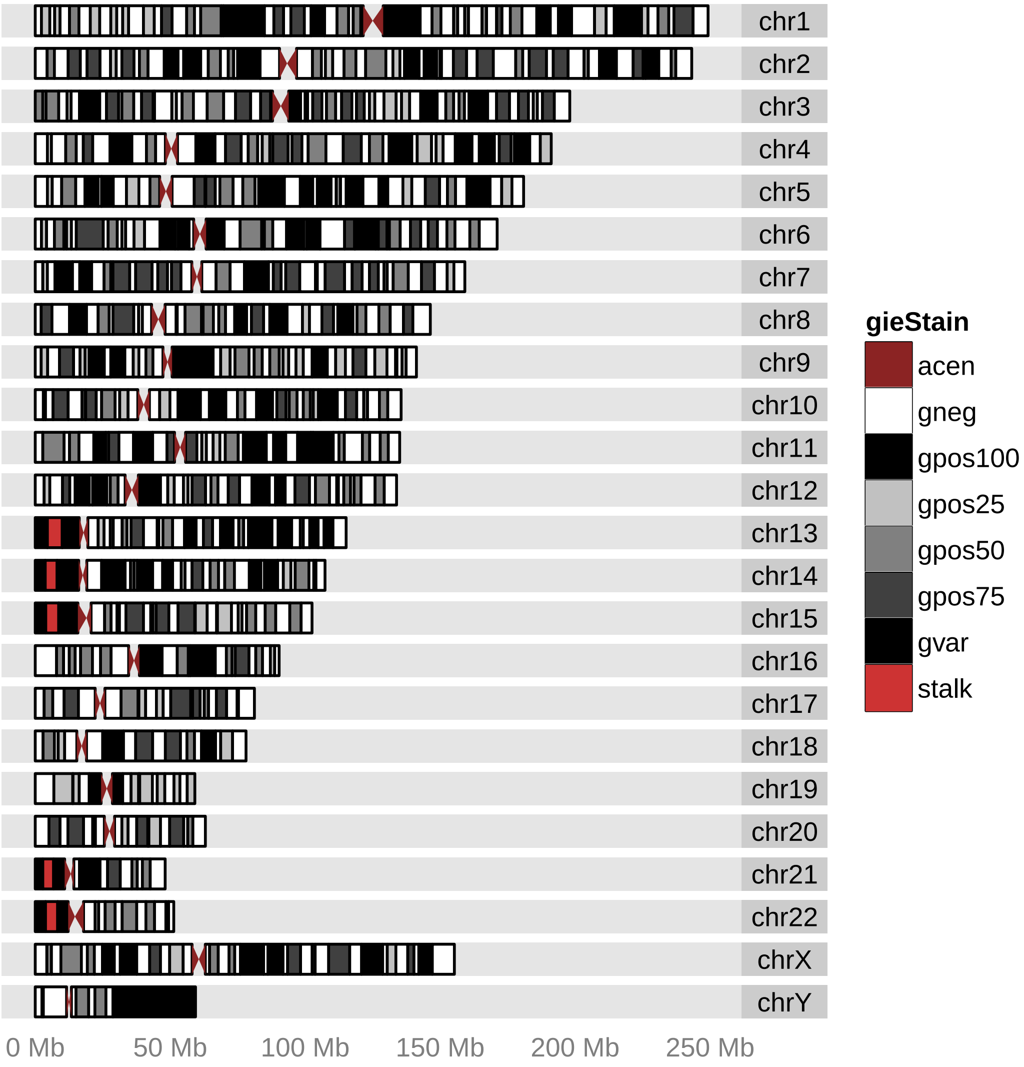
```
## Object of class "ggbio"
```



```
## NULL
```

`layout_karyogram` to plot the karyogram overview. Please check Chapter 11 and manual for more information.

```
hg19 <- keepSeqlevels(hg19IdeogramCyto, paste0("chr", c(1:22, "X", "Y")))
autoplot(hg19, layout = "karyogram", cytoband = TRUE)


## Object of class "ggbio"
```


chr1
chr2
chr3
chr4
chr5
chr6
chr7
chr8
chr9
chr10
chr11
chr12
chr13
chr14
chr15
chr16
chr17
chr18
chr19
chr20
chr21
chr22
chrX
chrY
0 Mb
50 Mb
100 Mb
150 Mb
200 Mb
250 Mb
gieStain
acen
gneg
gpos100
gpos25
gpos50
gpos75
gvar
stalk

```
## NULL
```

Chapter 3

Tracks: bind and align plots

Tips: To read this chapter, you don't need any background about biology. Basic knowledge about *ggplot2* is prefered.

3.1 Objective

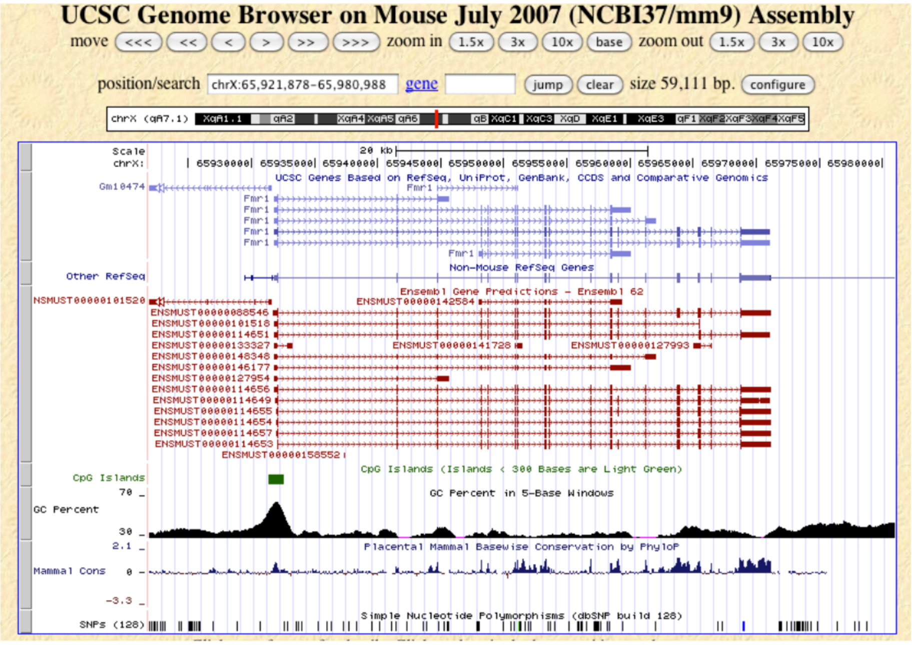
1. Get yourself familiar with basic *ggplot2* functions.
2. Get yourself familiar with basic grammar of graphics.
3. Get yourself familiar with *ggbio*'s `track` usage.

3.2 Motivation

It might be surprising that the first chapter we are going to talk about is about alignments of plots and especially for tracks. This is fundamental components used almost everywhere in the documentations, what's more important, this function could be used independently with **any other ggplot2 graphics**, not just for graphics produced by *ggbio*, well, this is the right time to tell you that, *ggbio* depends on *ggplot2* and extends it to genomic world, **so every graphics produced by *ggbio* is essentially a *ggplot2* object or a combination of them, so you can use any tricks works for *ggplot2* on *ggbio* graphics.**, but of course, we bring more features which doesn't exists in *ggplot2* at all.

Tips: If you want to manipulate graphics from *ggbio* more freely, I strongly recommend you to read documentation about *ggplot2*, most time the edit you want could be achieved by some basic functionality already in *ggplot2*, so enjoy those handy tools and don't reinvent the wheel! What's more, if you want to be an expert, knowledge about *grid, gtable* are necessary. Tracks relies on the new *gtable* package heavily, it has several convenient ways to manipulate the graphic objects.

Track-based view are widely used in almost all genome viewers, it usually stacks multiple plots row by row and align them on exactly the same coordinate, which in most cases, the genomic coordinates. In this way, we could be able to align various annotation data against each other to make an efficient comparison. UCSC genome browser[1] is one of the most widely used track-based genome browser, as shown in Figure **??**. There are some other packages in R, that support track-based view like UCSC genome browser, such as *Gviz*. General tracks for viewing genomic data should probably have following features:


UCSC Genome Browser on Mouse July 2007 (NCBI37/mm9) Assembly
move <<< << < > >> >>> zoom in 1.5x 3x 10x base zoom out 1.5x 3x 10x
position/search chrX:65,921,878-65,980,988 gene jump clear size 59,111 bp. configure
chrX (qA7.1) XqA1.1 qA2 XqA4 XqA5 qA6 qB XqC1 XqC3 XqD XqE1 XqE3 qF1 XqF2 XqF3 XqF4 XqF5
Scale
20 kb
chrX:
65930000| 65935000| 65940000| 65945000| 65950000| 65955000| 65960000| 65965000| 65970000| 65975000| 65980000|
UCSC Genes Based on RefSeq, UniProt, GenBank, CCDS and Comparative Genomics
Gm10474
Fmr1
Other RefSeq
Non-Mouse RefSeq Genes
Ensembl Gene Predictions - Ensembl 62
NSMUST00000101520
ENSMUST00000142584
ENSMUST00000088546
ENSMUST00000101518
ENSMUST00000114651
ENSMUST00000133327
ENSMUST00000141728
ENSMUST00000127993
ENSMUST00000148348
ENSMUST00000146177
ENSMUST00000127954
ENSMUST00000114656
ENSMUST00000114649
ENSMUST00000114655
ENSMUST00000114654
ENSMUST00000114657
ENSMUST00000114653
ENSMUST00000158552
CpG Islands (Islands < 300 Bases are Light Green)
CpG Islands
GC Percent in 5-Base Windows
70
GC Percent
30
Placental Mammal Basewise Conservation by PhyloP
2.1
Mammal Cons
0
-3.3
Simple Nucleotide Polymorphisms (dbSNP build 128)
SNPs (128)

- Align each plot in exactly the same X coordinate(genomic coordinate).
- Naming ability for each track, this is different from Y-label, which is used to illustrate variable used as y.
- Shared “scale” track.
- Multiple ways to visualize the data, as points, line, bar chart or density.etc.

As comparison, *ggbio* is trying to be even more general in terms of building tracks, and offer more features.

- You can bind any graphics produced by *ggplot2*, not necessarily produced by *ggbio*, in that way, *ggplot2* users will find it pretty conventient that they can construct plots independently, and `tracks` will align them for you. So you can use `tracks` to align your own data, e.g. time series data.
- Easy-to-use utilities for zooming, backup, restore a view. This is useful when you tweak around with your best snapshot, so you can always go back.
- A extended ”+” method. If you are familiar with *ggplot2*’s ”+” method to edit an existing plot, this is the way it works, if tracks is ”+” with anything behind, it will be applied to each track. This make it easy to tweak with theme and update all the plots.

[1] http://genome.ucsc.edu/cgi-bin/hgGateway

- You could specify whether you want to label a plot or not by using labeled, labeled<-, and to specify whether you what the plot x-axis synchronized with other tracks or not by using function fixed, fixed<-.
- Creating your own customized themes for not only single plot but also tracks! We will show an example how to create a theme called theme_tracks_subset in the following sections.
- Support not only vertical alignments, but also horizontal alignments.

> Tips: tracks function only support graphic objects produced by either *ggplot2* or *ggbio*. If you want to align plots, produced by other grid based system, like lattice, users need to tweak in grid level, to insert a lattice grob to a layout.

3.3 Usage

Function tracks is a constructor for an object with class *Tracks*. This object is a container for each plot you are going to align, and all the graphic attributes controlling the appearance of tracks.

3.3.1 A minimal example for *ggplot2* graphics

Instead of showing you the genomic examples for constructing tracks, let me first do a small favor for *ggplot2* users, suppose you don't have any background about biology, all you want to do is to align two time series data. We can construct any graphics **independently** without worrying about aligning them. Just use your knowlegge about *ggplot2* to create any simple or fancy graphics, only one thing you need to make sure about is that **the x-axis you are going to align must have the same meanings**, in this minimal example, it's *time*.

I am going to introduce some basic usage about *ggplot2* all the way through this vignette every now and then, to make it easier for people who first use *ggplot2* or *ggbio* and not quite familiar with its grammar.

```
## load ggbio automatically load ggplot2
library(ggbio)
## make a simulated time series data set
df1 <- data.frame(time = 1:100, score = sin((1:100)/20) * 10)
p1 <- qplot(data = df1, x = time, y = score, geom = "line")
df2 <- data.frame(time = 30:120, score = sin((30:120)/20) * 10, value = rnorm(120 -
    30 + 1))
p2 <- ggplot(data = df2, aes(x = time, y = score)) + geom_line() + geom_point(size = 4,
    aes(color = value))
```

In *ggplot2*, most working object are *data.frame*, in comparison, we support many other core data structure in Bioconductor, which we will introduce later mainly in Section6 and Section5, when we introduce function generic method such as autoplot and ggplot.

When you see qplot function, you have to know it's *ggplot2*'s function(means 'quick plot'), since Bioconductor 2.10, *ggbio* stop using a confusing generic qplot function, instead, we are using a new generic method introduced in *ggplot2*, called autoplot, we heavily override this function in *ggbio* to support more data structure.

To introduce `qplot` function, we need to first get an idea about grammar of graphics(GoG), it's basically composed of following components:

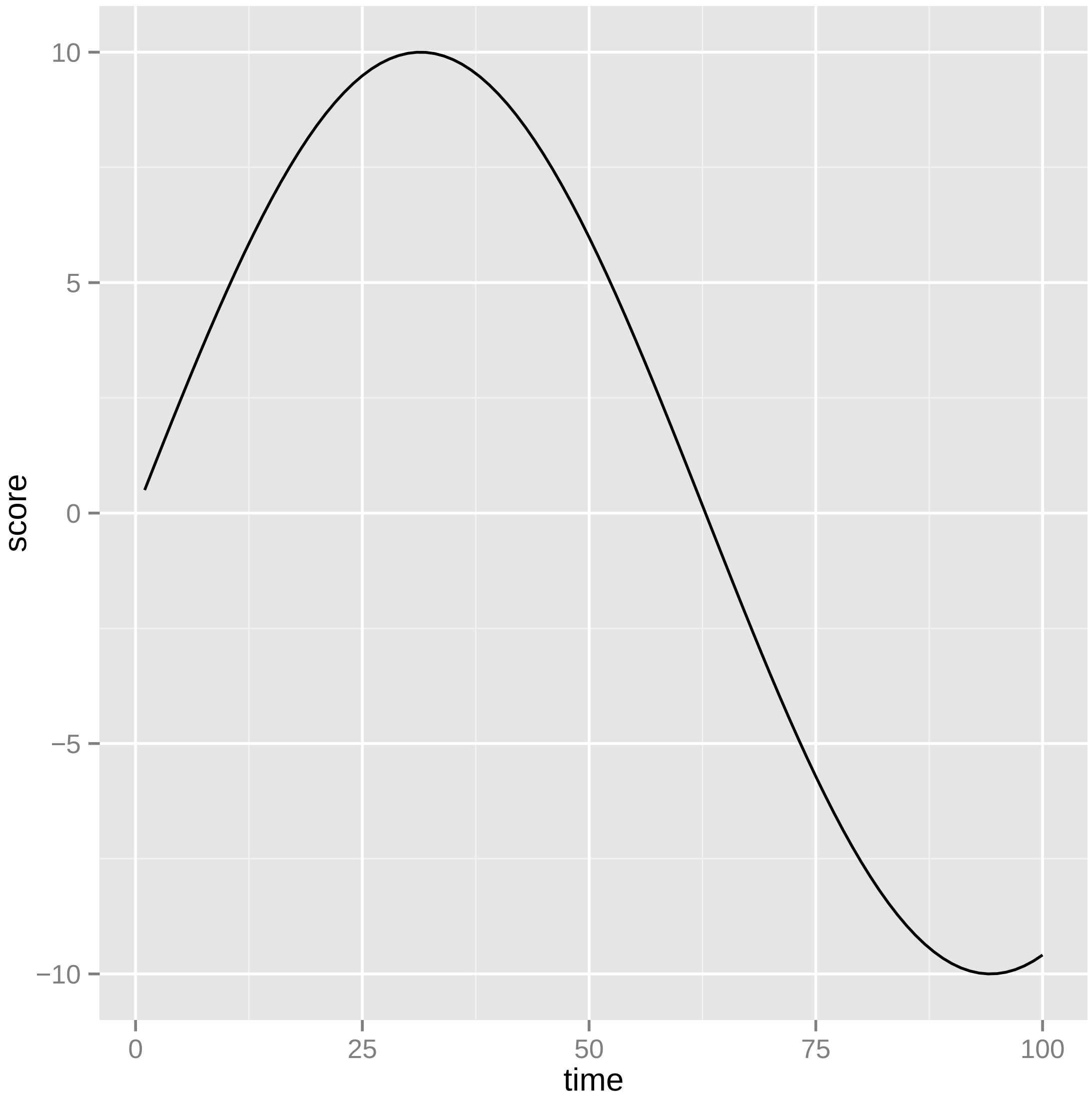
- *Data*: Data you are going to visualize with a set of variables, it's usually the first argument passed in function `autoplot`.
- *Statistical transformation*: Statistical methods performed on the variables of raw data and generate more informative summary. It's usually controlled by the parameters **stat**.
- *Geometric object*: e.g arrow, rectangle. It's usually controlled by the parameters **geom**.
- *Coordinate system*: eg Cartesian. It's usually controlled by the parameters **coord**.
- *Scales*:Transformation of scales, such as logarithm. It's usually controlled by the parameters **scale**.
- *Facetting*:Subset the data by factors and create small panels for each subset of data with same representation of graphics. It's usually controlled by the parameters **facets**.

So basically speaking we have two API or usage here for constructing graphics in *ggplot2*, it's similar in *ggbio*.

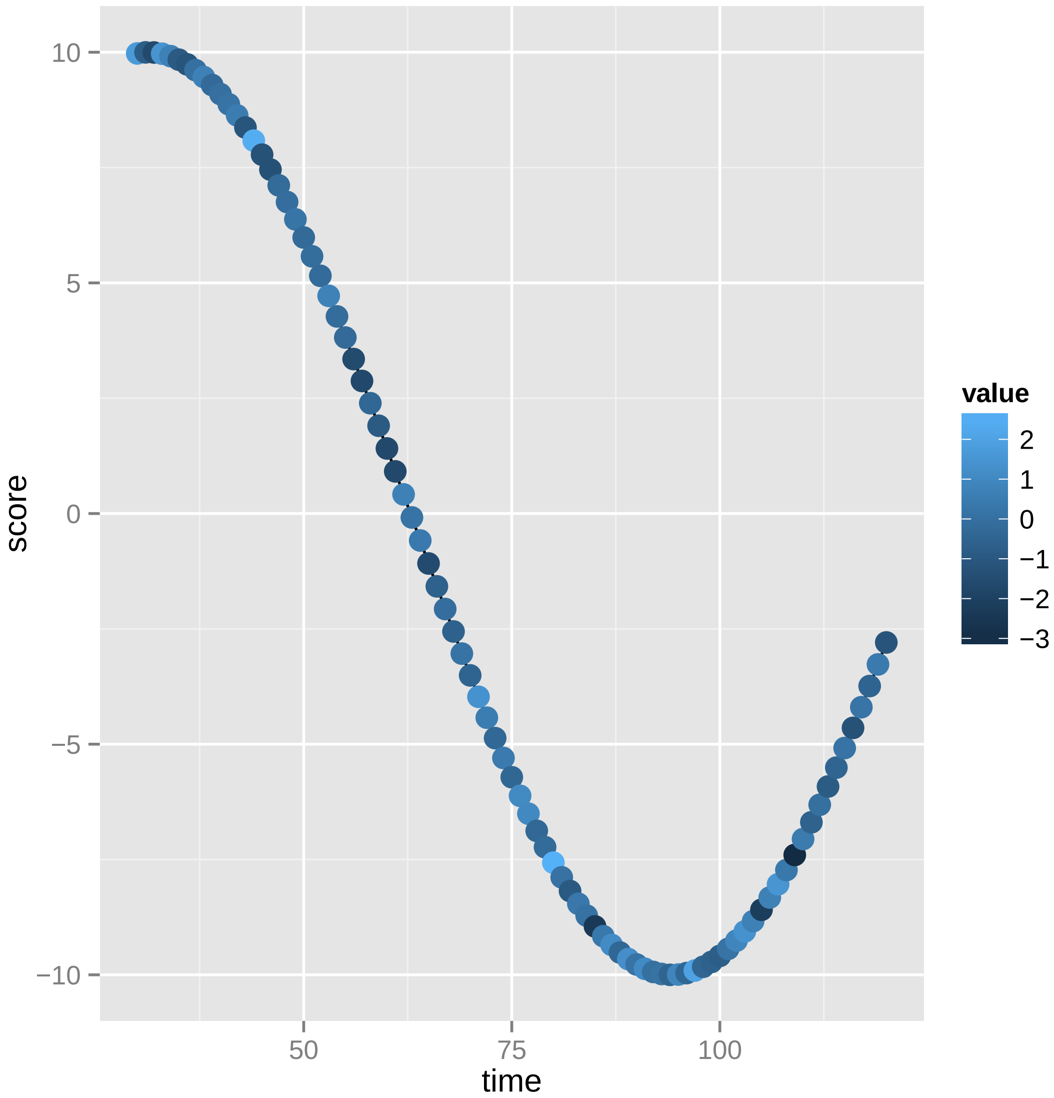
- The first method is called quick plot in *ggplot2*, implemented in function `qplot`, it's one general wrapper for quick mapping and constructing the grammar's components. Similar in *ggbio*, we have `autoplot` for this purpose, what's more, `autoplot` is more object-oriented visualization methods, which will be introduced in other chapter6. so in the `qplot` API, we specify data to be the data frame and map x to *time* variable and map y to *score* varialbe, `geom` arguments means *Geometric objects*, we could use multiple *geom* in `qplot` function. To print the graphic object on the screen, simply call `print` on it or just type the name and show it.
- The second method is very flexible or more close to the grammar itself, the way it is constructed is like the way it is described in the grammar or like in plain human languange. Let's say we want to "use data df2, and generally use *time* as x and use *score* as y, then we add a line to the plot, next we add points to the plots, for those points, we want to map color to *value* varialbe and use arbitrary value to set size for points". See, it is exactly what we described compared to the actuall code! That's what *ggplot2* bring to us, the implmentation of grammar of graphics in R. Notice, function `aes` used for mapping *aesthetics* to variables in the data.

> Tips: If you don't know how many existing components you could use in pure *ggplot2* package, please check Hadley's online documentation. Websites is here `http://had.co.nz/ggplot2/`, For *ggbio* based components, please read relevant part in this vignettes and visit `http://tengfei.github.com/ggbio/docs` to check documentation. There are plenty of examples with graphics there.

```
print(p1)
```



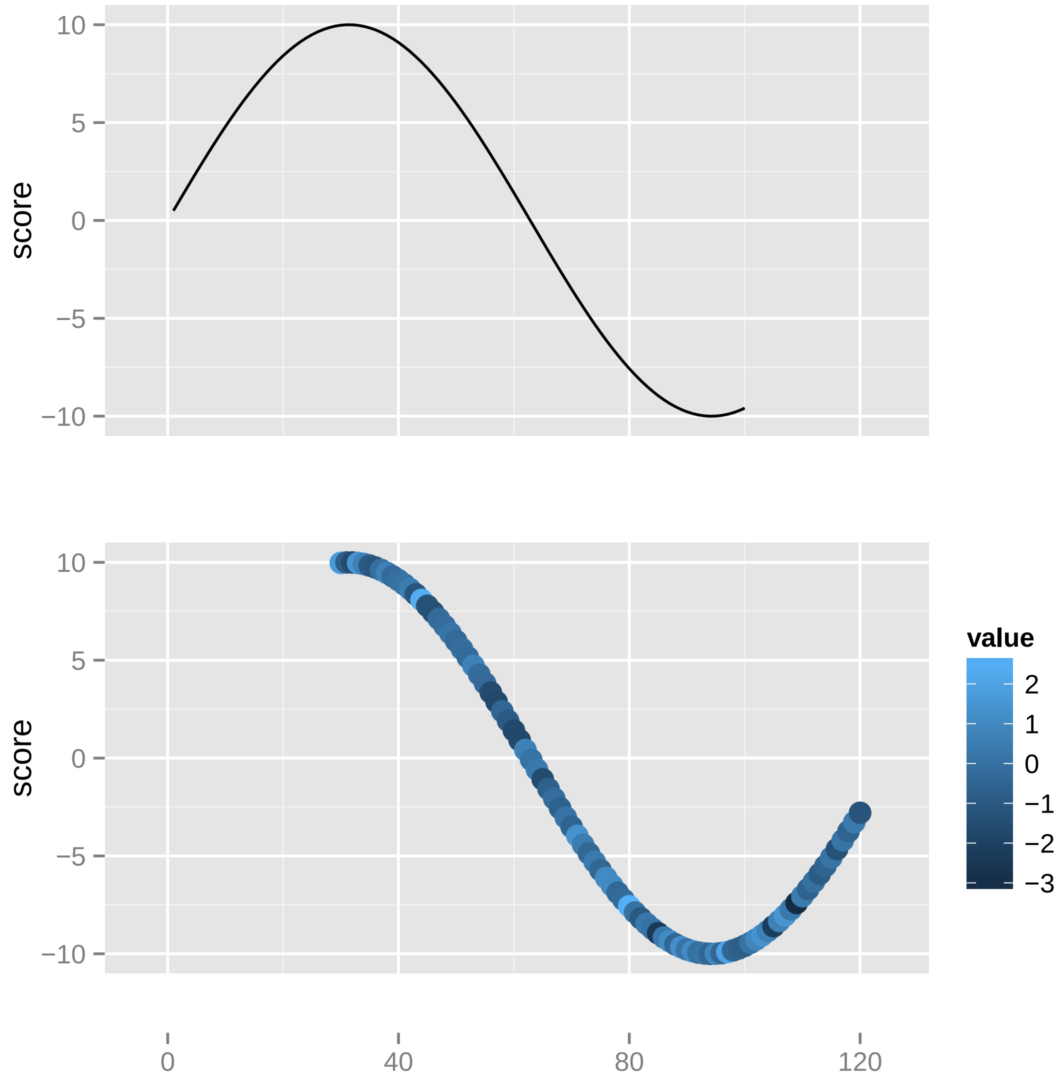
```
p2
```



As shown in Figure ??, we can see these two plots have different scale on x-axis, but we want to compare those two plots and hope to align them on exactly the same x-axis scale, then we could make vertical comaprison easily. Now we introduce the `tracks` function, we can pass the multiple plots we want to align into it, and it will do some obvious modification including :

- squeeze the plots together
- remove x-axis and make a shared scale.
- do the alignments automatically.
- construct and return a *tracks* object.

```
tracks(p1, p2)
```



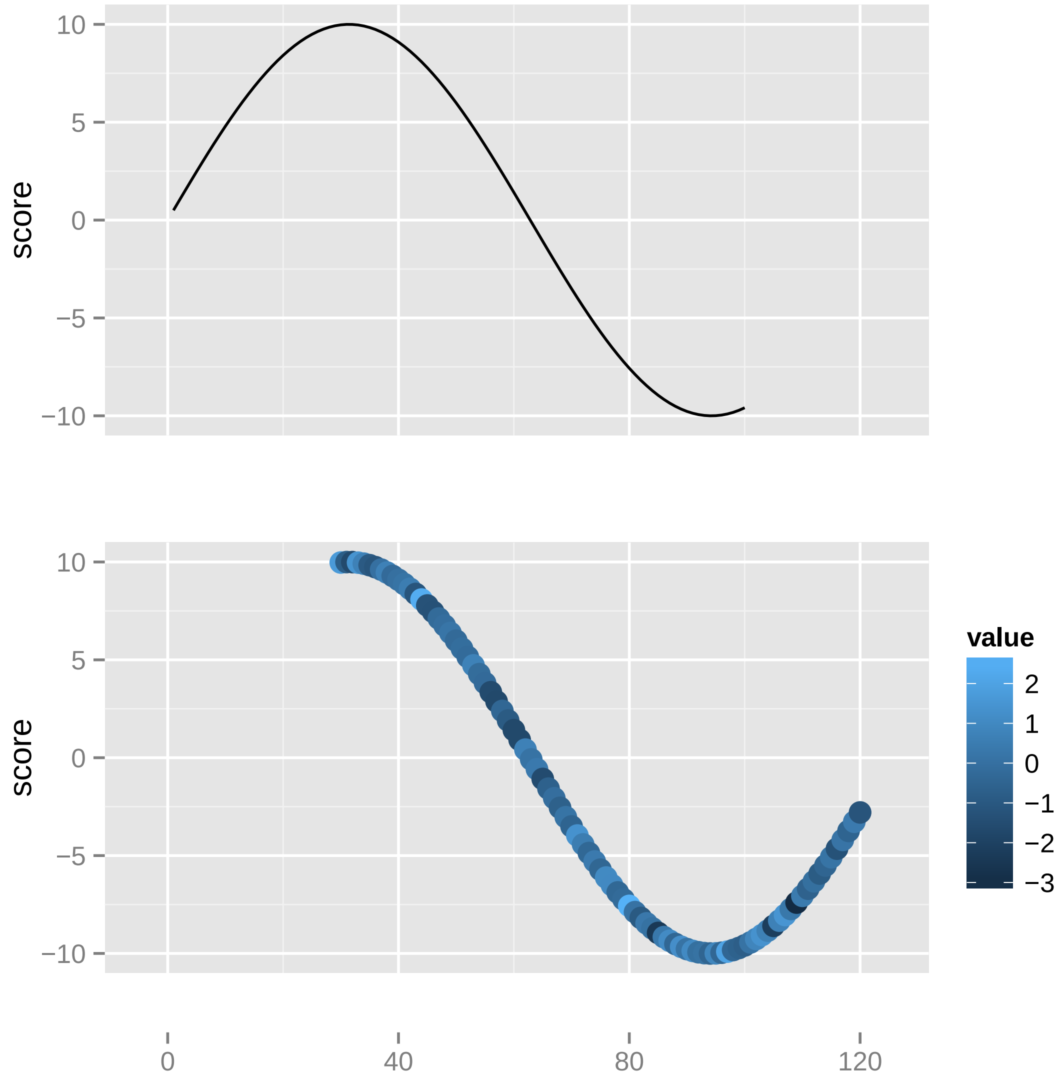
As shown in Figure **??**, those two plots are aligned well on the x-axis, so here it is, our first track. You could also assign the tracks to an object, this will avoid printing on the screen immediately.

```
tks <- tracks(p1, p2)
tks
```

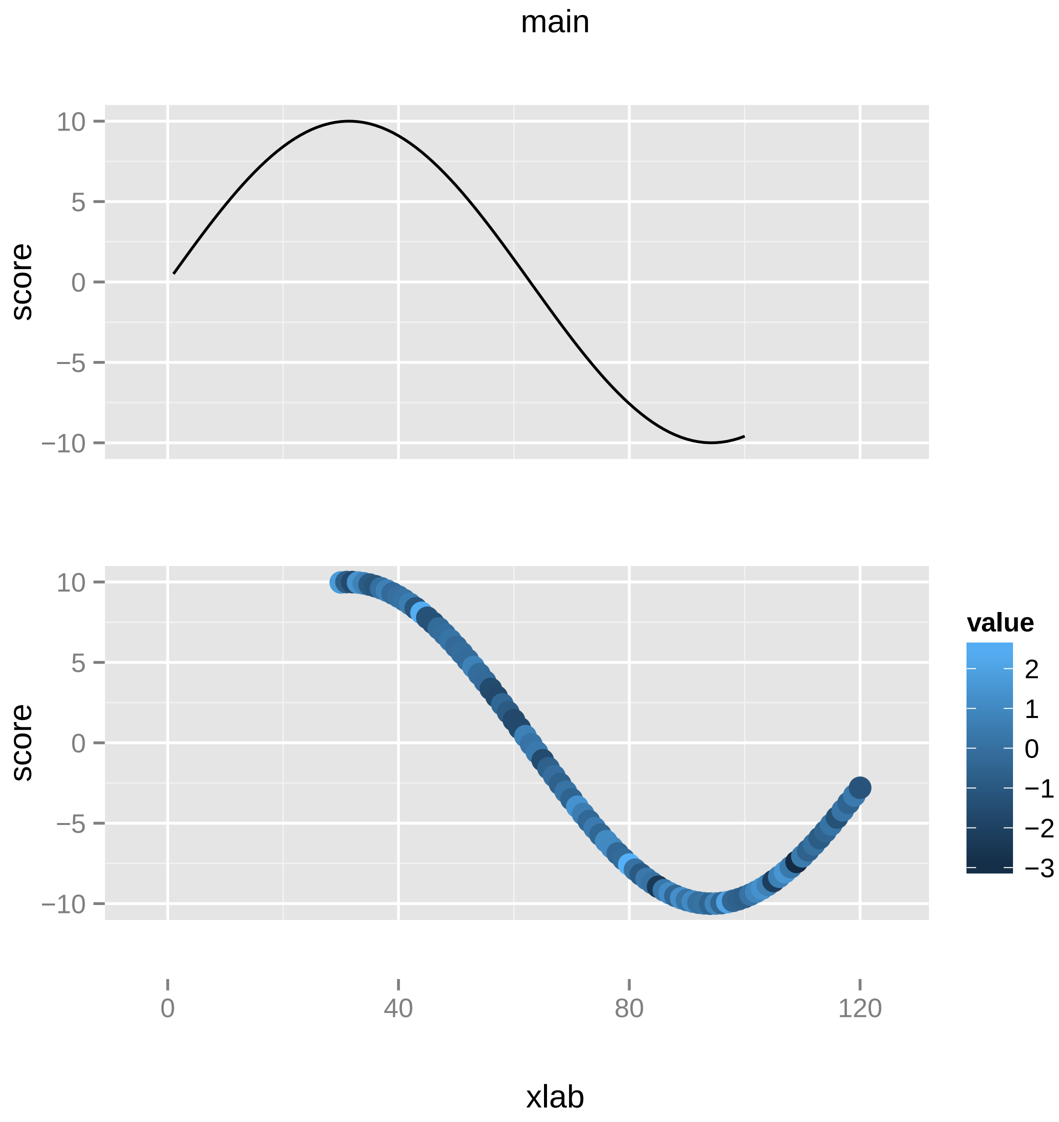
3.3.2 Labeling and naming a track

Y labels are kept for each track plot, but in general, you may want to annotate the plot for title or x label, just specify arguments in `tracks` function.

```
tracks(p1, p2)
```



```
tracks(p1, p2, xlab = "xlab", main = "main")
```

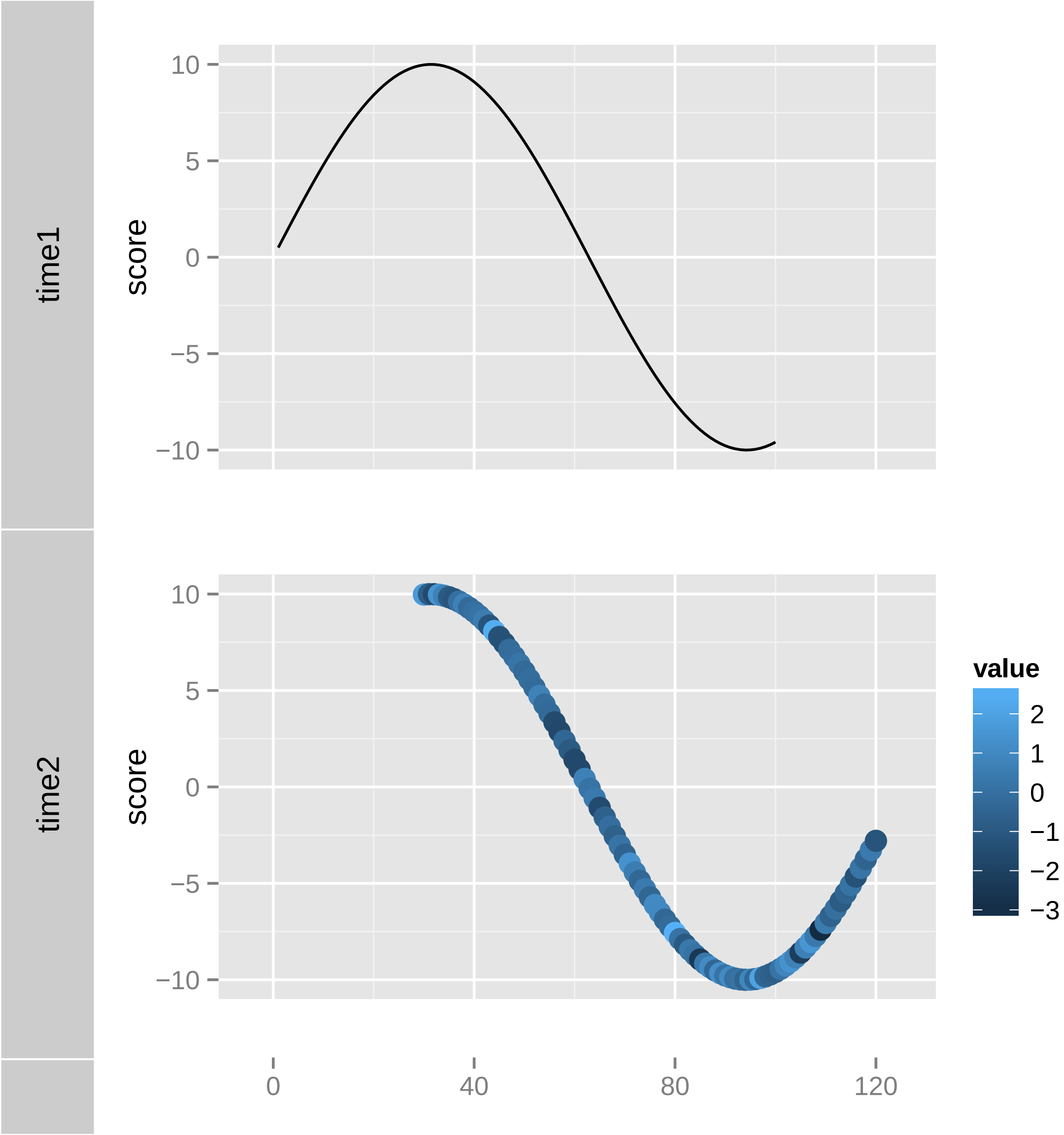


To add label for each track, simply naming the plots, there are several ways to label it.

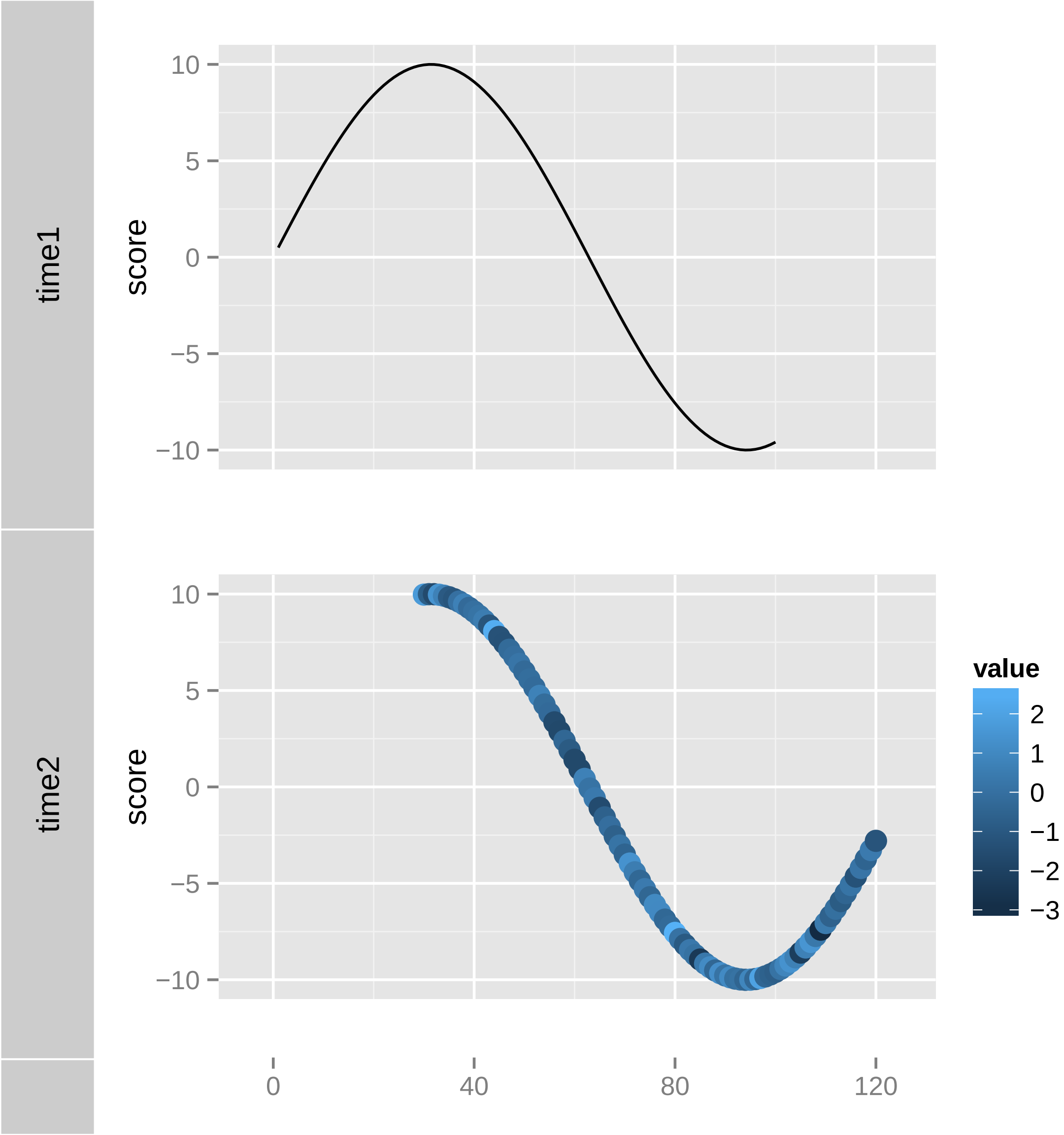
- pass names togther with graphics.
- for complicated name, use quotes.
- use named list of graphics

As shown below in following examples.

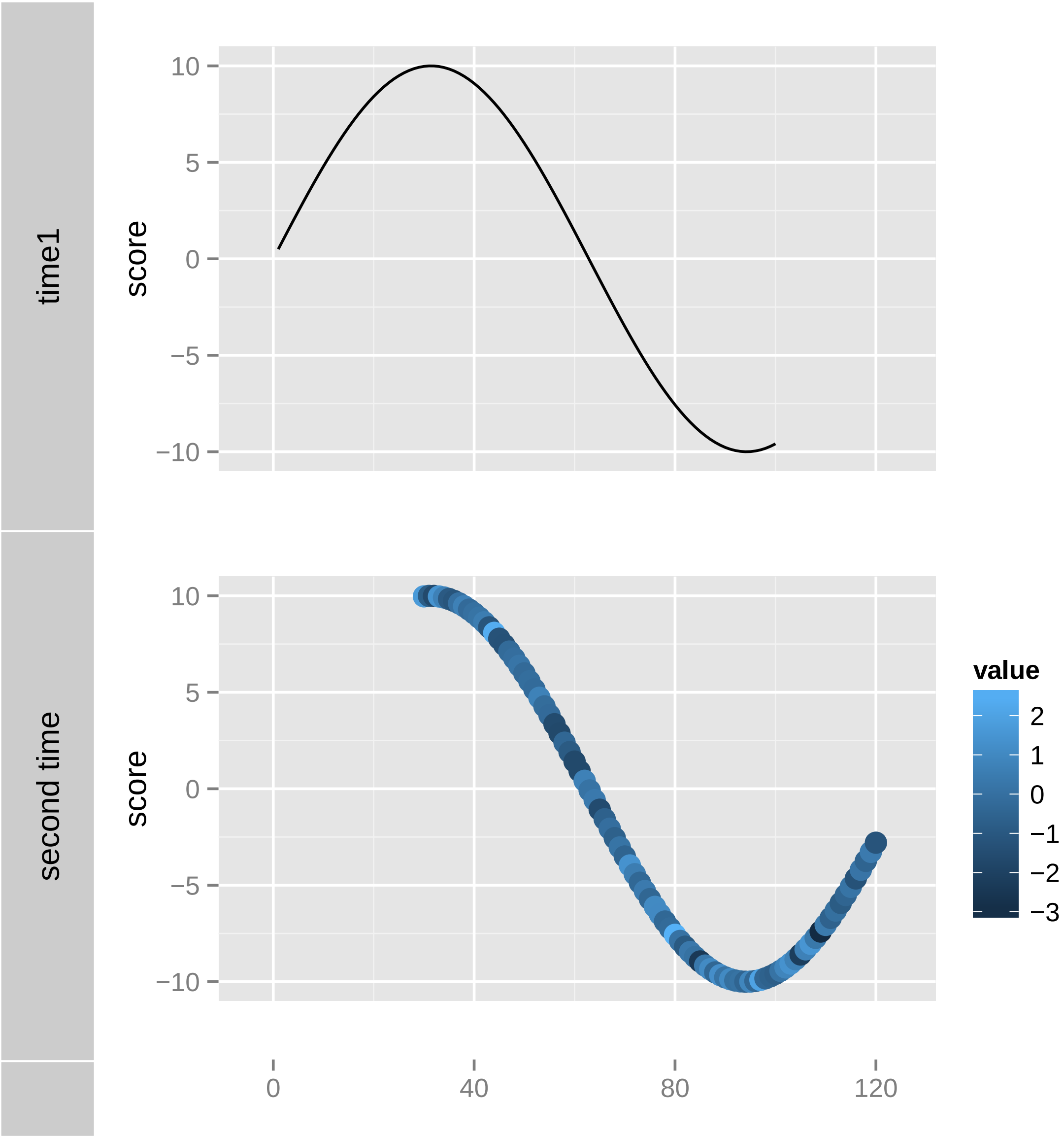
```
## labeling: default labeling a named graphic simply pass a name with
## it
tracks(time1 = p1, time2 = p2)
```



```
## or pass a named list with it
lst <- list(time1 = p1, time2 = p2)
tracks(lst)
```

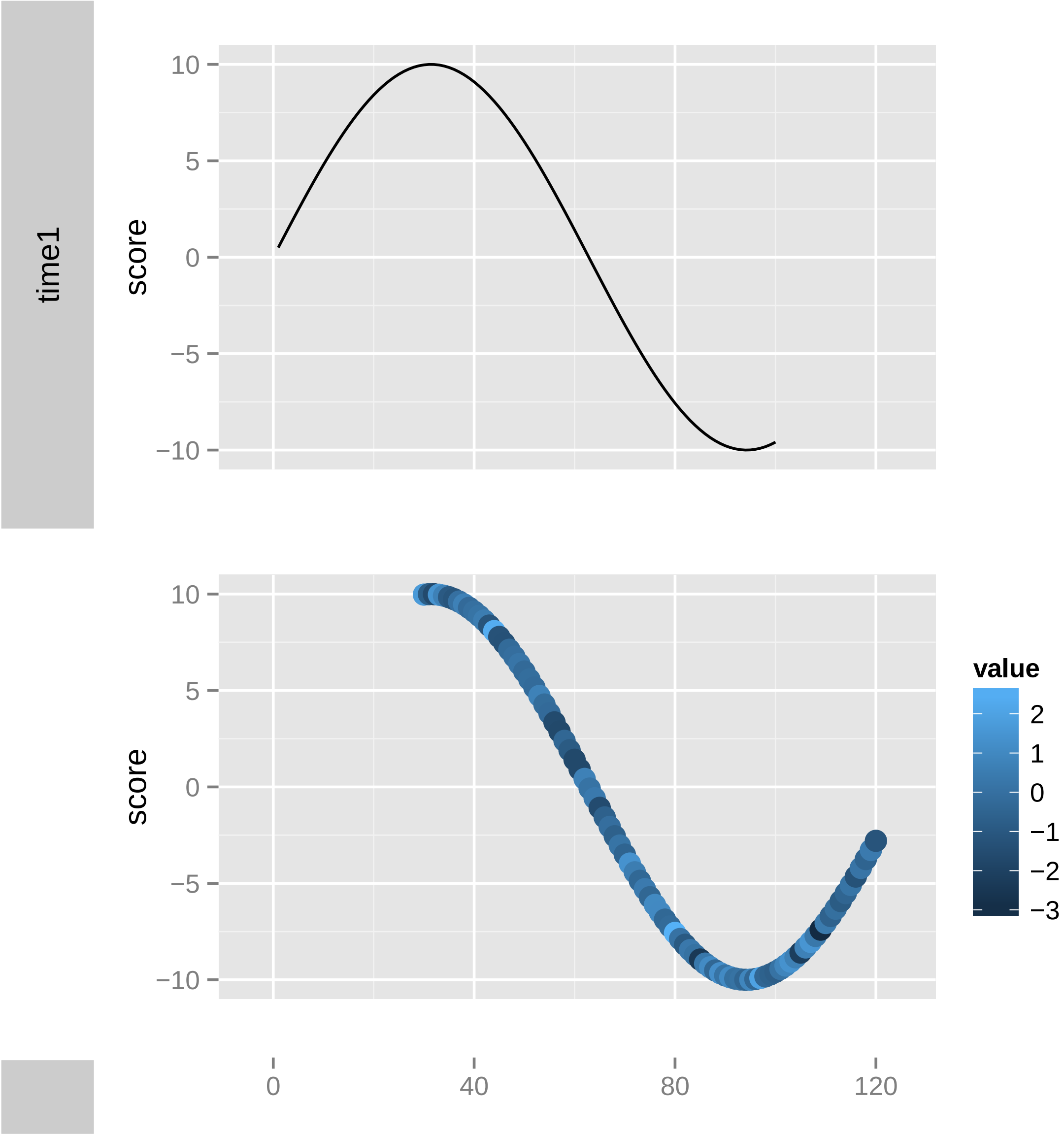


```
## more complicated case please use quotes
tracks(time1 = p1, `second time` = p2)
```



Sometimes, even though you passed a named or no-name graphics, but you still don't the gracks to show the label background for that plot, you can simply set the labeled attribute of plot to be `FALSE`.

```
labeled(p2)

## [1] TRUE

labeled(p2) <- FALSE
## set labeled to FALSE, remove label even the plot has a name
tracks(time1 = p1, time2 = p2)
```



```
labeled(p2) <- TRUE
```

3.3.3 Arith method +

Before we move on to more modificatino method for *Tracks* object. We are going to introduce the flexible + method first. Arith + method is very powerful and heavily used in *ggplot2* and *ggbio*, for constructing a plot layer by layer, + is used to connect and add components one by one, and could also be used for updating and editing an existing plot. Many people hates this, this maybe raise the learning curve, but mean wile, more people love it, at least, I am one of those people. This is a good way to learn grammar of graphics, after you master it, you will definitely benefit from it.

I have to mention the improvements in ggbio, + is extended to work on not only the single plot but also the

Tracks object.

- For single plot, + apply or add the change on the right hand side to the left hand side object.
- For Tracks object, + apply or add the change on the right hand side to every plot stored in the tracks, it's like a 'batch' mode, so you don't have to edit plots before passing into the tracks, unless you want to edit them respectively.

remember, you can always get the plot you passed from a Tracks object, by accessing grobs slot, such as tks@grobs[[i]], grobs is a list of plots.

Please read following section for more examples.

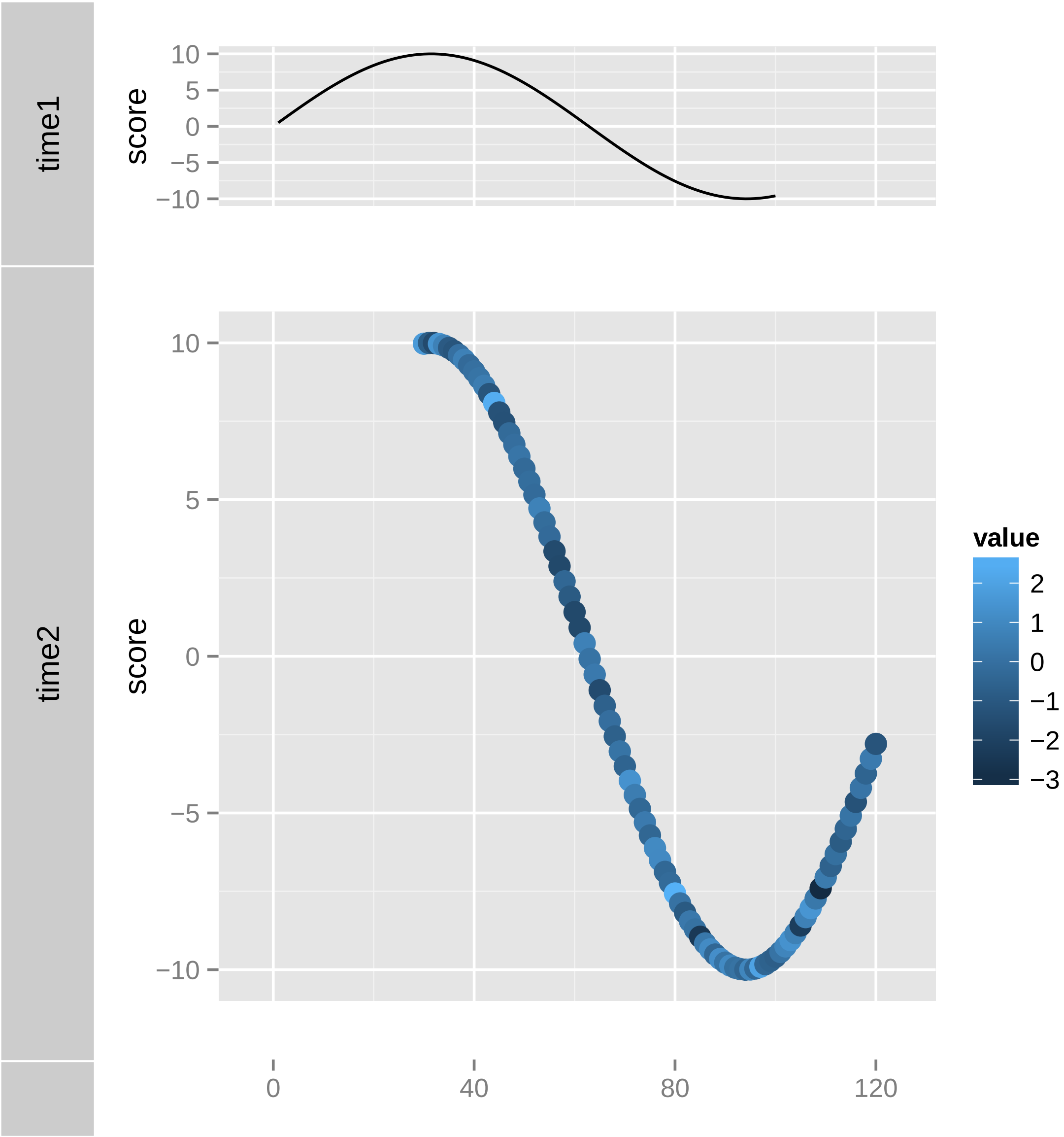
3.3.4 Modification

We provide some attributes associated with plot, they won't affect the single plot, those attributes will take effect when they are embeded into tracks. Those attributes include

- height: defeault height for this plot.
- bgColor: background color for this plot.
- labeled: if you want to show label(and backgrond) for the plot or not, even through the plot is named.
- fixed: control if scale of plot is fixed or not.
- mutable:control if plot is affected by + method on tracks or not.

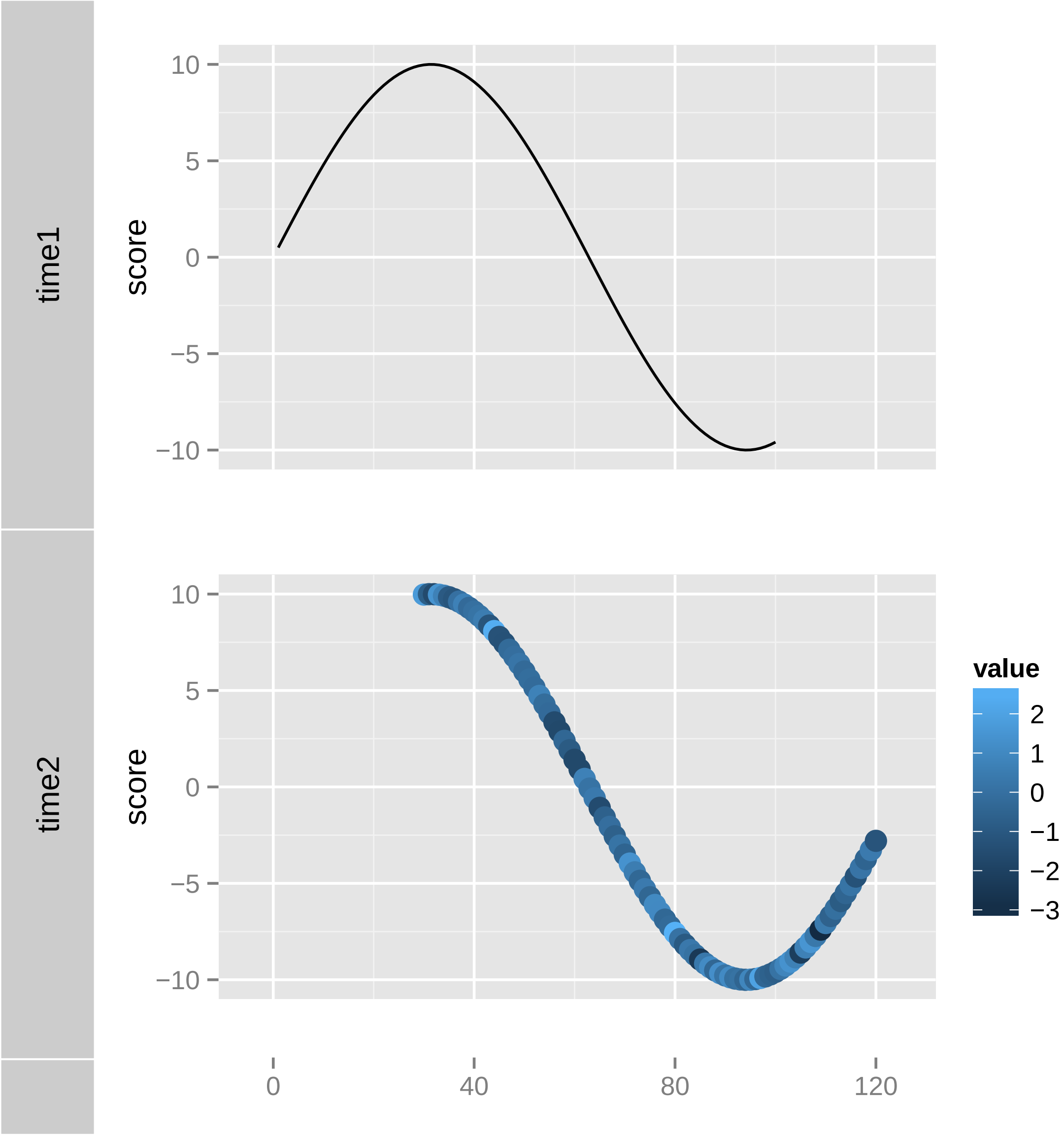
To modify the heights for each track, simply pass the heights argument with ratio.

```
## set heights
tracks(time1 = p1, time2 = p2, heights = c(1, 3))
```

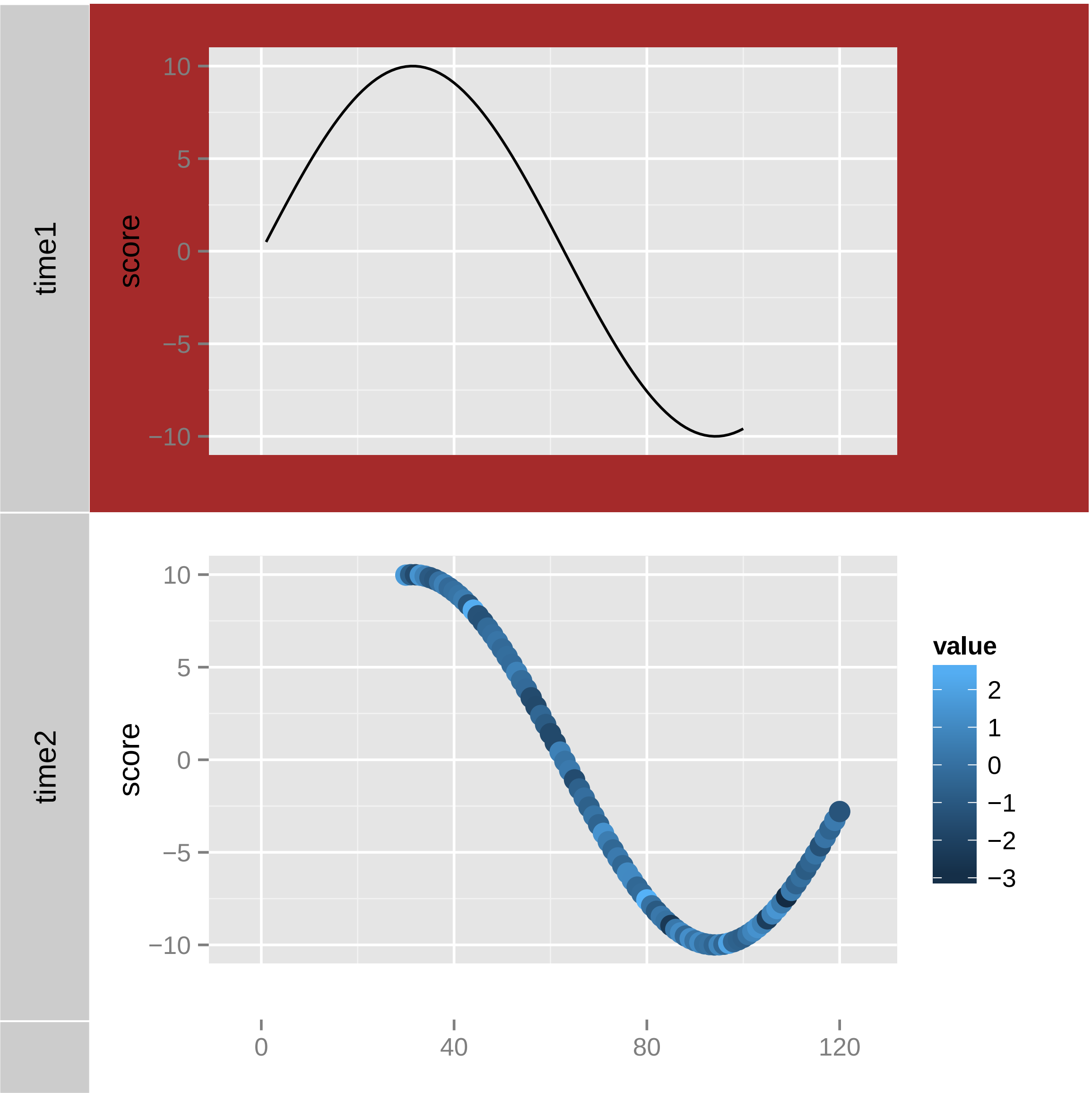


To change background for each plot, you could set **bgColor** attribute or use argument `track.plot.color`.

```
## bgColor
bgColor(p1)


## [1] "white"


tracks(time1 = p1, time2 = p2)
```

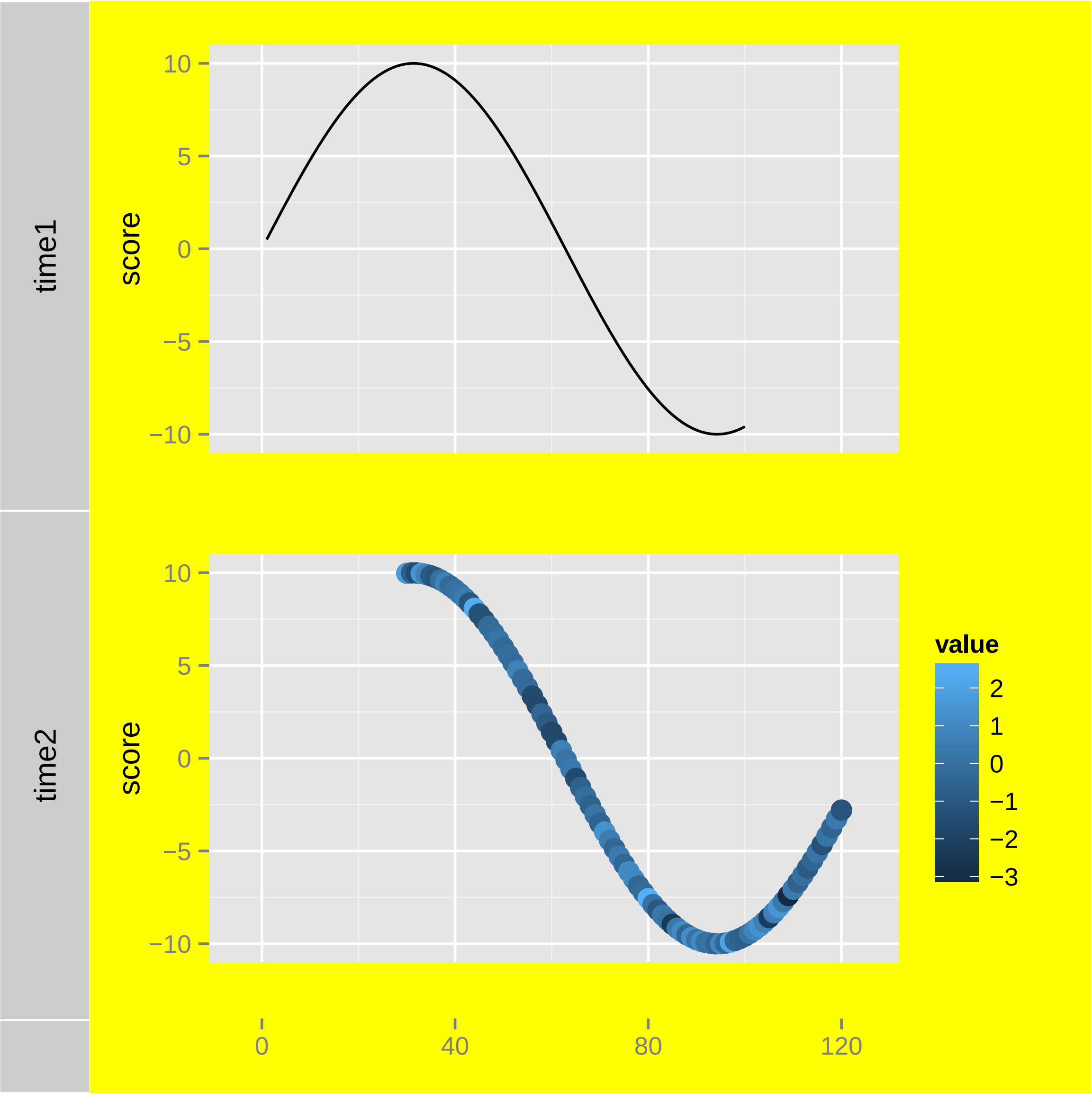


```r
bgColor(p1) <- "brown"
# mutable for '+' method
tracks(time1 = p1, time2 = p2)
```

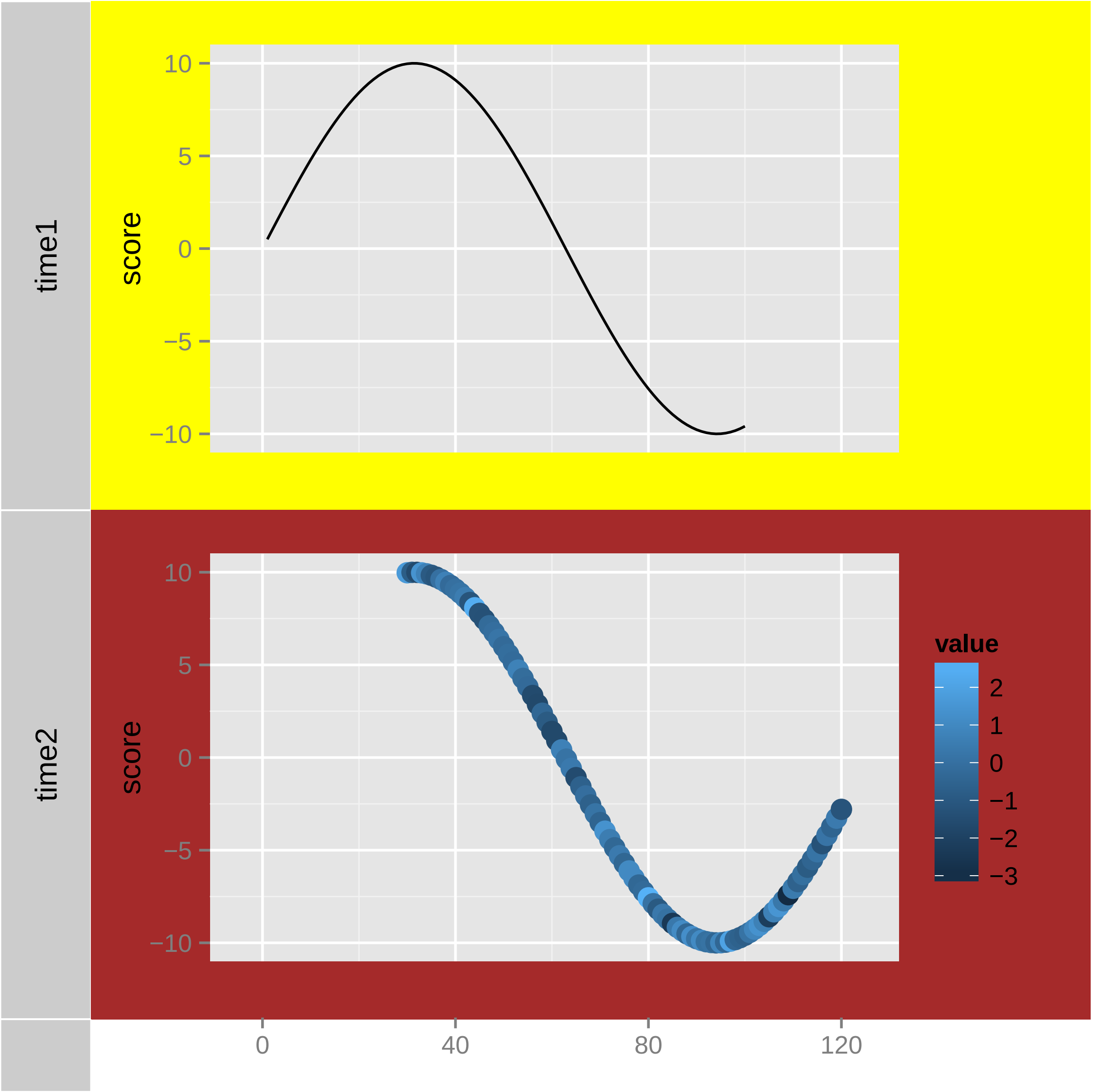


```
# set it back
bgColor(p1) <- "white"
```

```
## track color
tracks(time1 = p1, time2 = p2, track.bg.color = "yellow")
```

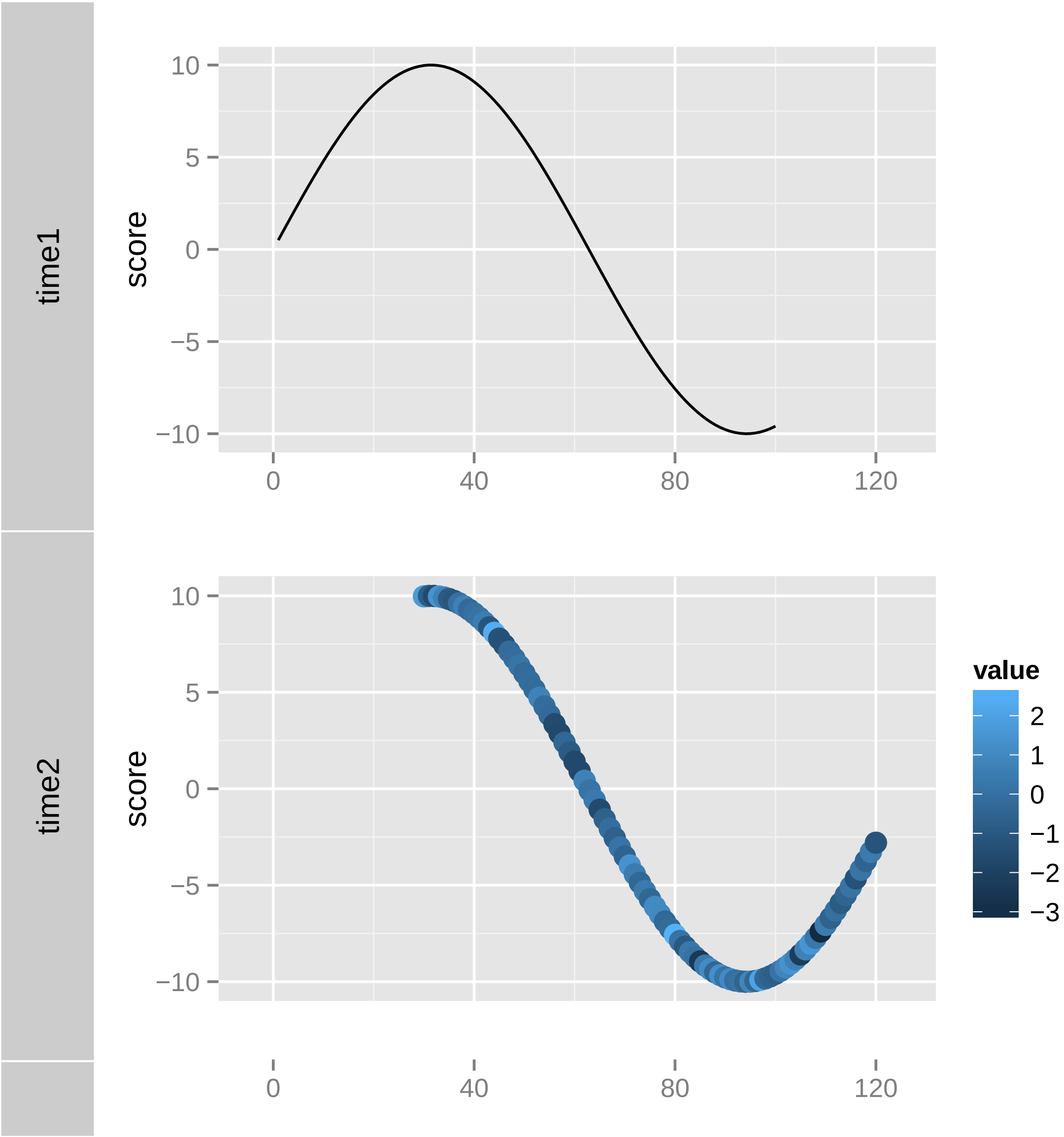


```
tracks(time1 = p1, time2 = p2, track.plot.color = c("yellow", "brown"))
```



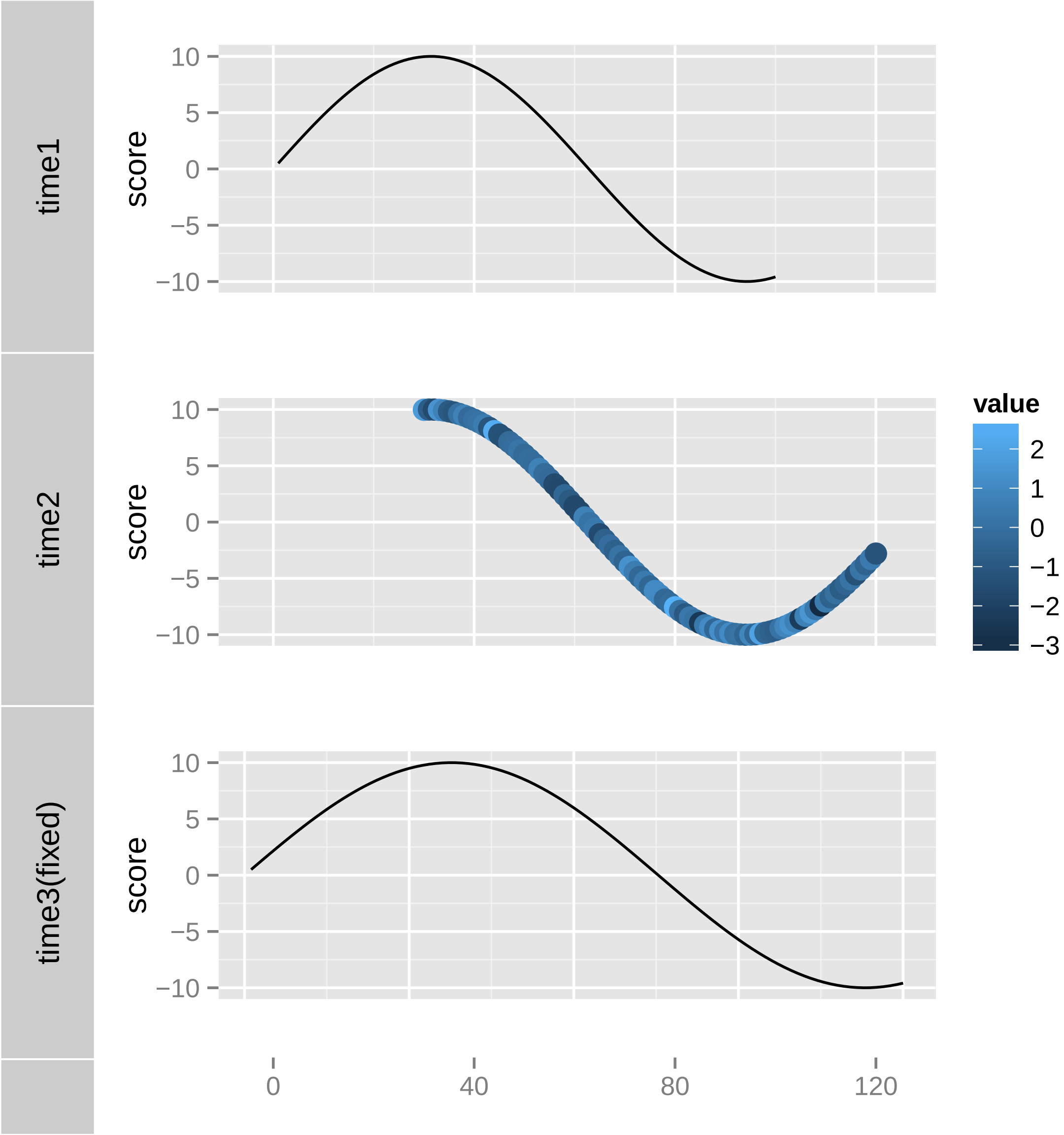
To control axis, you could set attribute **hasAxis**.

```
hasAxis(p1)

## [1] FALSE

hasAxis(p1) <- TRUE
# ready for weird looking
tracks(time1 = p1, time2 = p2)
```

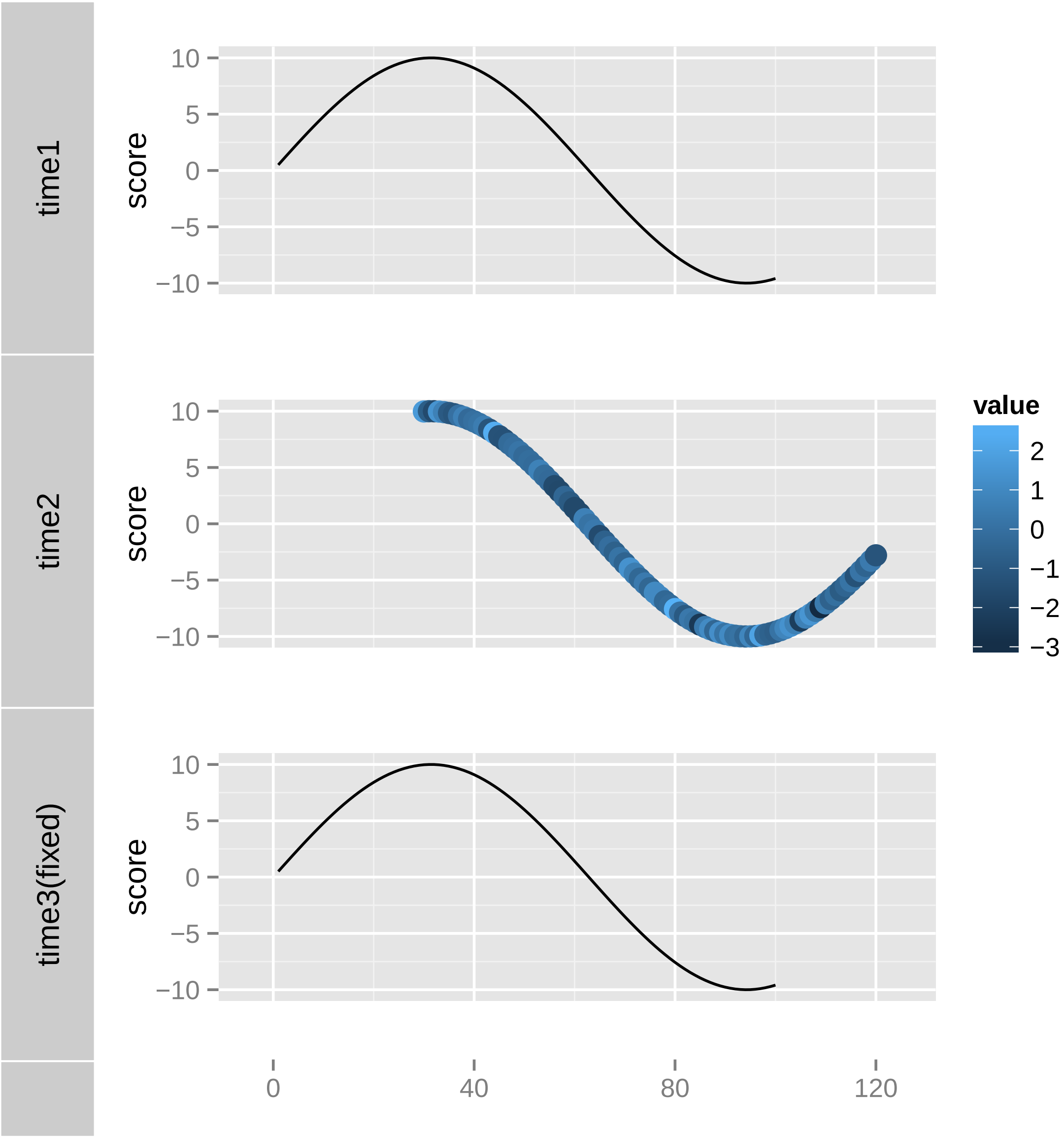


```
hasAxis(p1) <- FALSE
```

```
## fix a plot, not synchronize with other plots
p3 <- p1
## default is always FALSE
fixed(p3)

## [1] FALSE

## set to TRUE
fixed(p3) <- TRUE
tracks(time1 = p1, time2 = p2, `time3(fixed)` = p3)
```



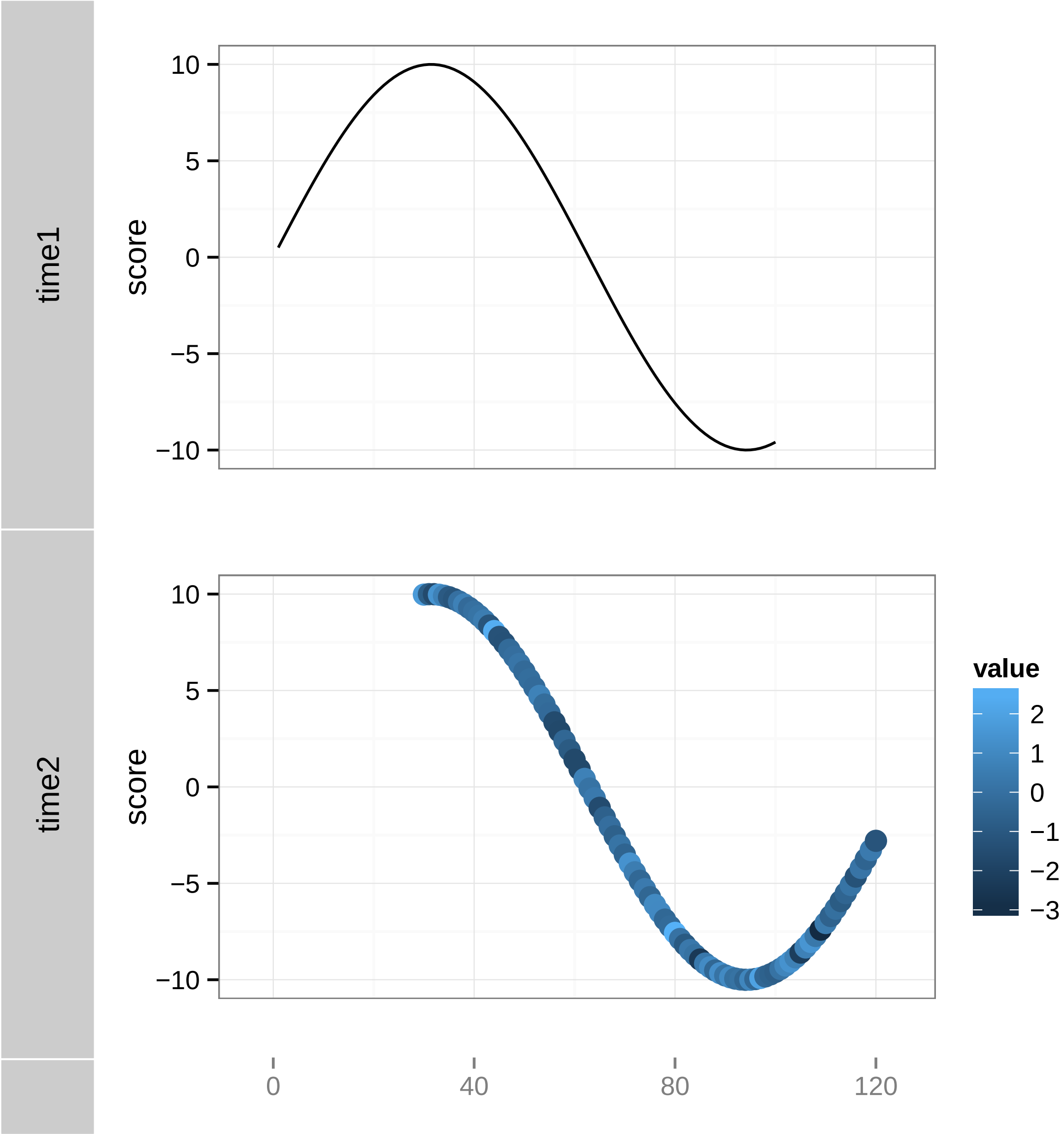
```
fixed(p3) <- FALSE
tracks(time1 = p1, time2 = p2, `time3(fixed)` = p3)
```



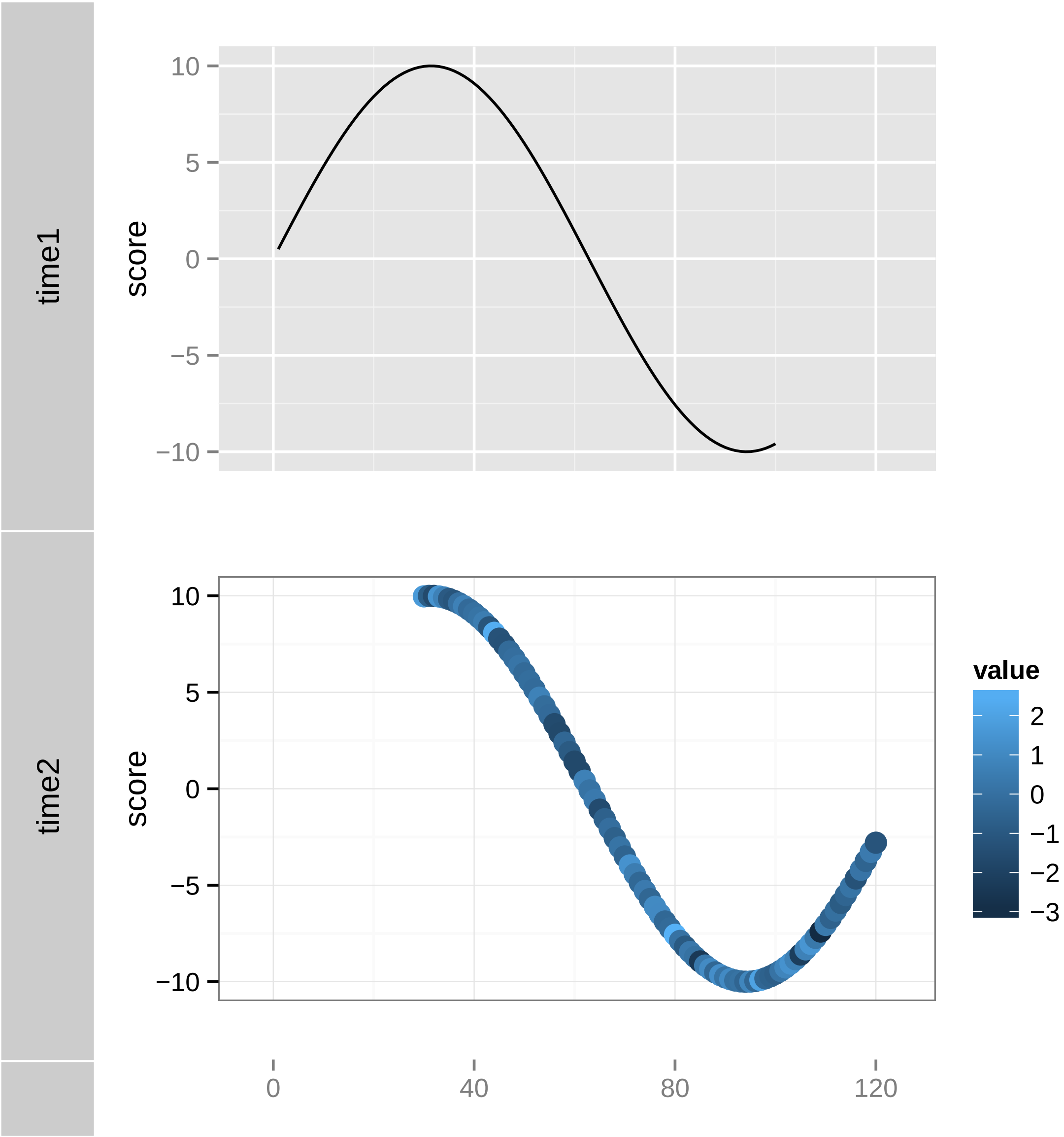
```
## otherwise you could run tracks(time1 = p1, time2 = p2,
## 'time3(fixed)' = p3, fixed = c(FALSE, FALSE, TRUE))
```

`mutable` only control whether 'mutable' to themes or not, **NOT** control x,y limmits changing, the `fixed` control it's response to x limits change.

```
## mutable
mutable(p1)


## [1] TRUE


tracks(time1 = p1, time2 = p2) + theme_bw()
```



```
mutable(p1) <- FALSE
# mutable for '+' method
tracks(time1 = p1, time2 = p2) + theme_bw()
```



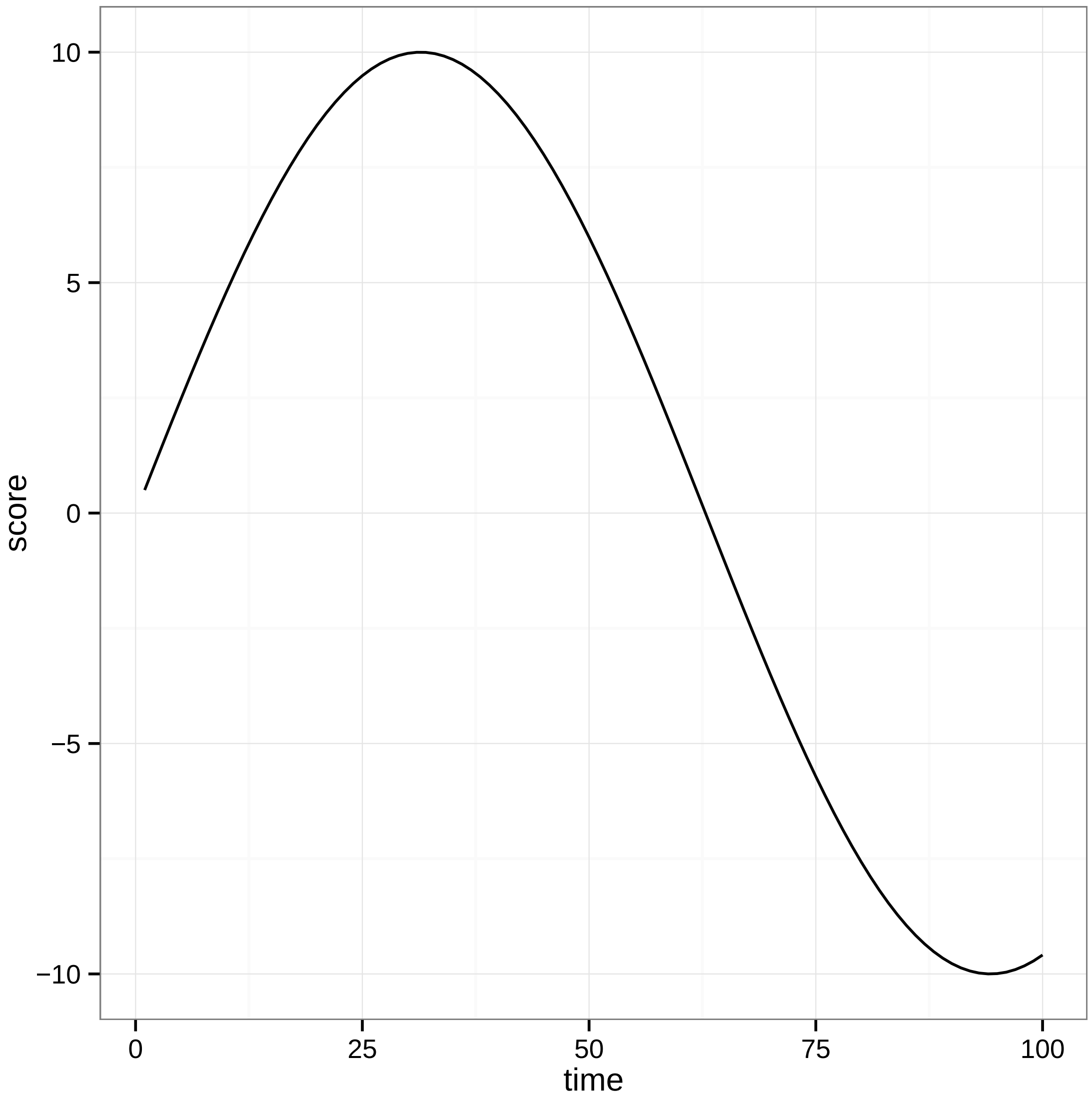
```
mutable(p1) <- TRUE
```

3.3.5 Customized themes for plots and tracks

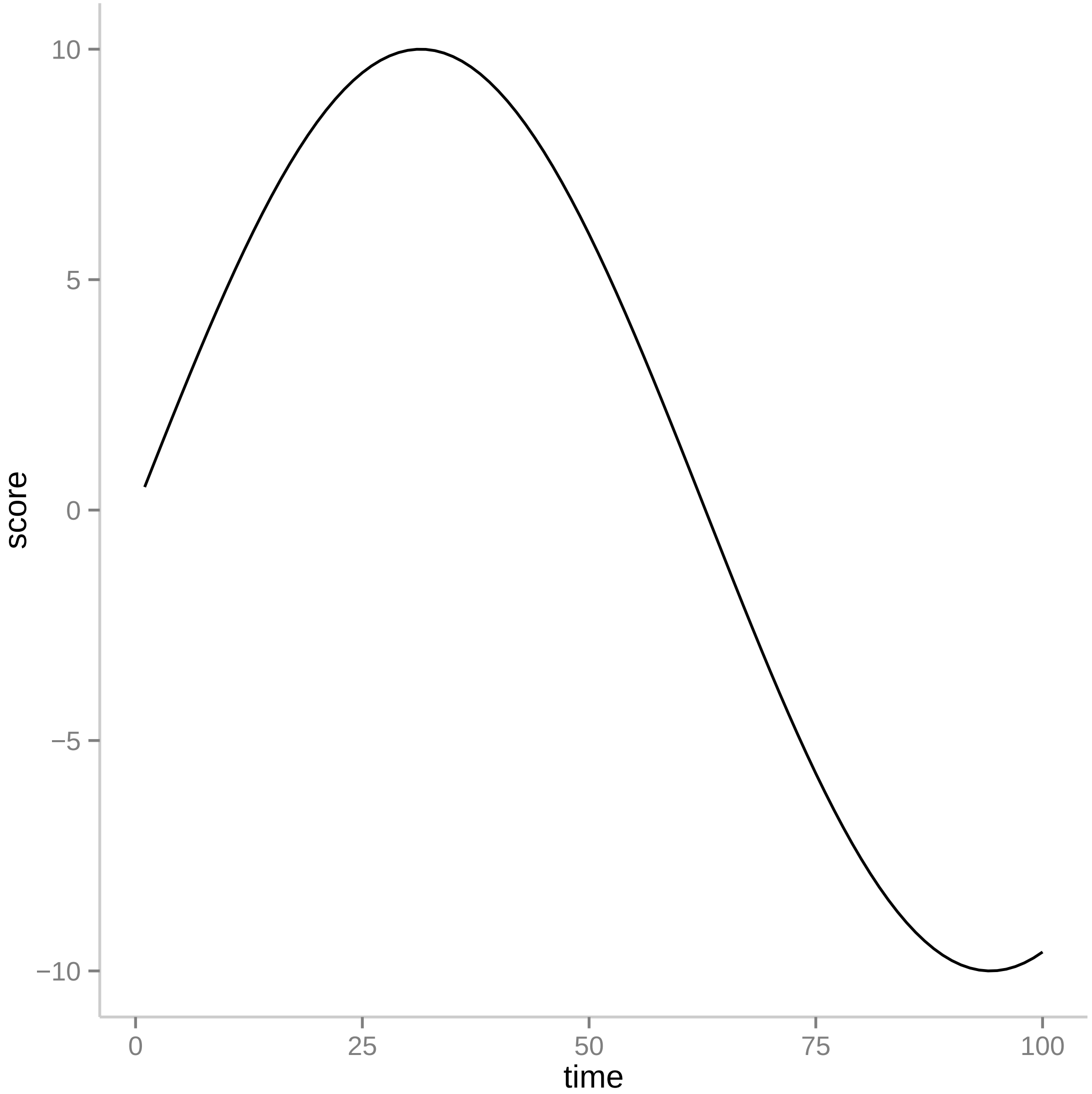
ggplot2 has a theme system, please run `?theme` to find out how to merge, update theme for your own usage.

For example, we simply change the theme for a single plot.

```
## try theme_bw() defined in ggplot2
p1 + theme_bw()
```



```
## try theme_clear() define in ggbio
p1 + theme_clear()
```



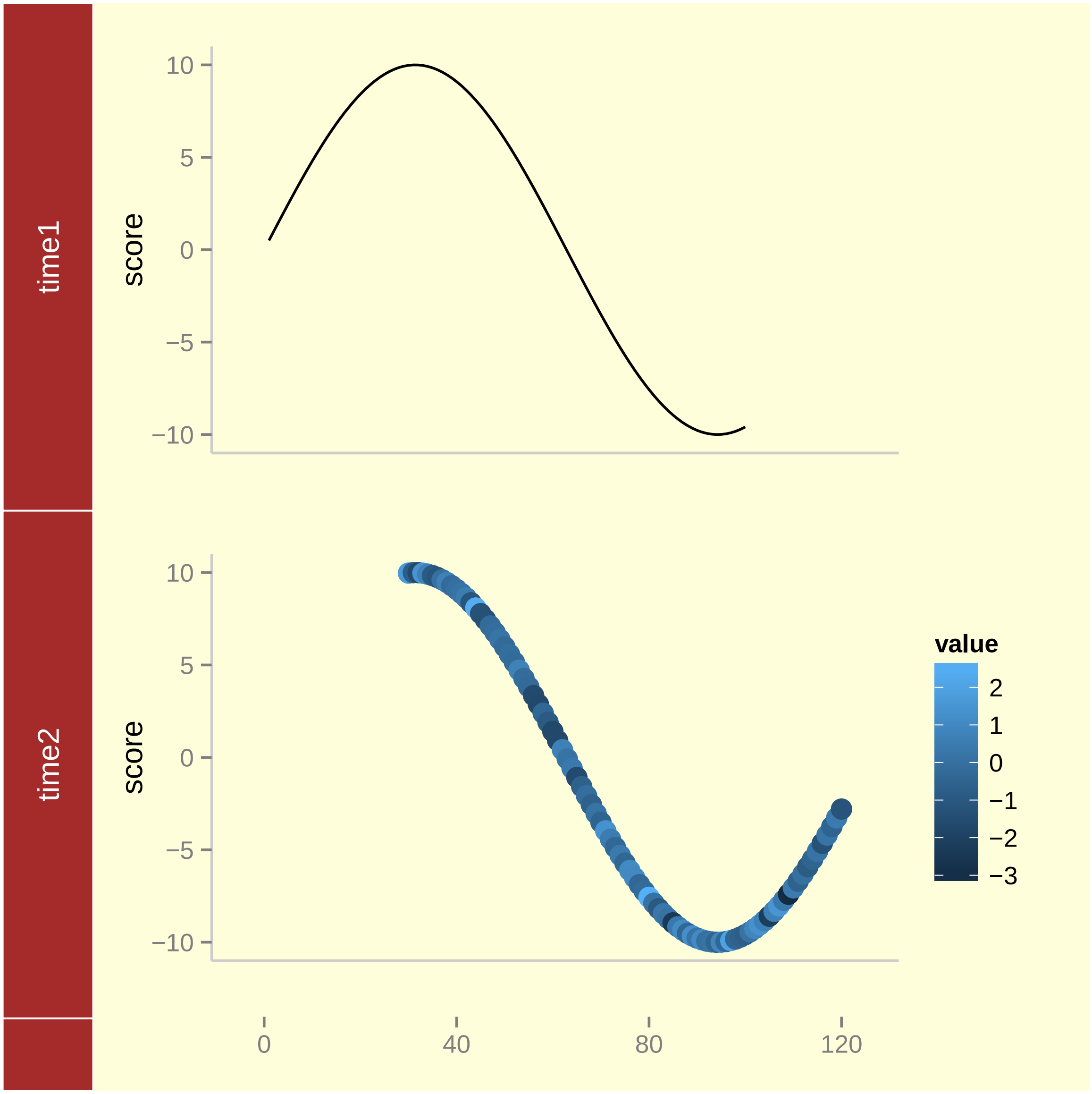
ggbio has a way to define a theme to even affect the `Tracks` object, the strategy is to store attributes with single plots and parsing and apply those attributes to tracks in the construction time of tracks. Below we show an example how to define a 'sunset' theme for tracks, please make sure you name your theme in different way, to indicate if it affect tracks or not.

- theme_* for theme apply to any graphics.
- theme_tracks_* for theme apply to also tracks.

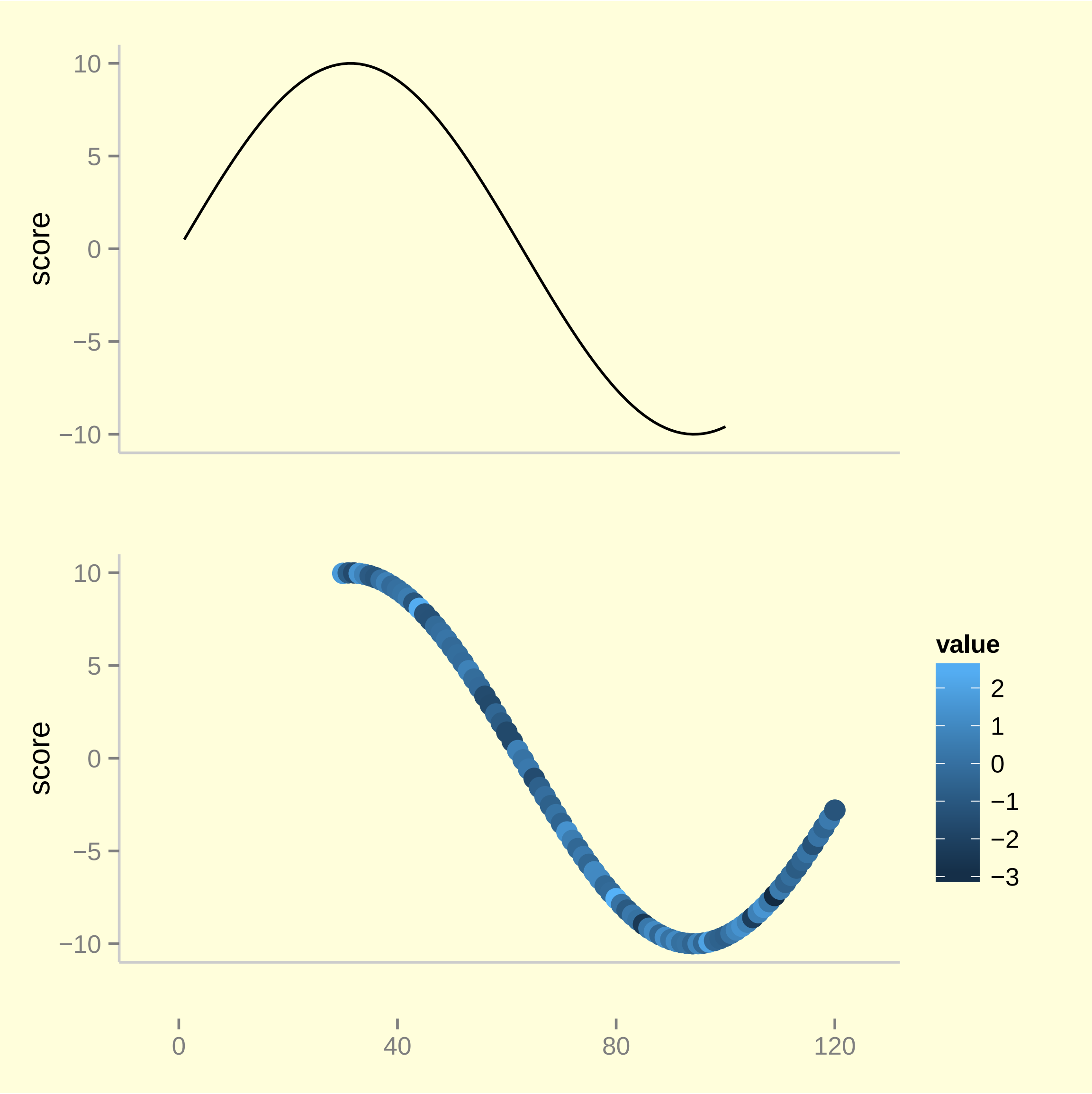
```
theme_tracks_sunset <- function(bg = "#fffedb", alpha = 1, ...) {
    res <- theme_clear(grid.x.major = FALSE, ...)
    attr(res, "track.plot.color") <- sapply(bg, scales::alpha, alpha)
    attr(res, "track.bg.color") <- bg
    attr(res, "label.text.color") <- "white"
    attr(res, "label.bg.fill") <- "#a52a2a"
    res
}
```

This theme is defined in *ggbio*, you can use it directly after loading *ggbio*.

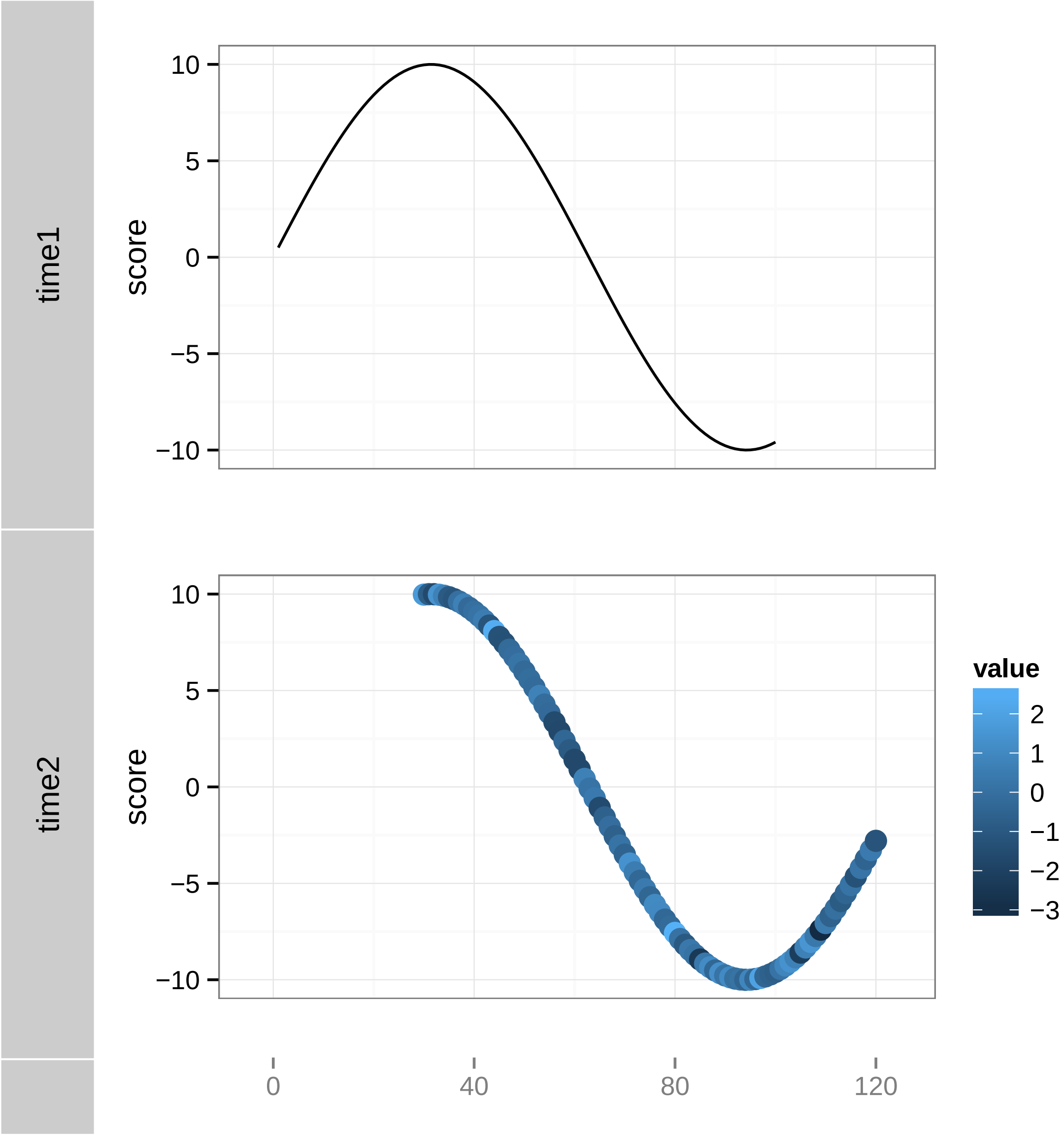
```
## apply a pre-defiend theme for tracks!
tracks(time1 = p1, time2 = p2) + theme_tracks_sunset()
```



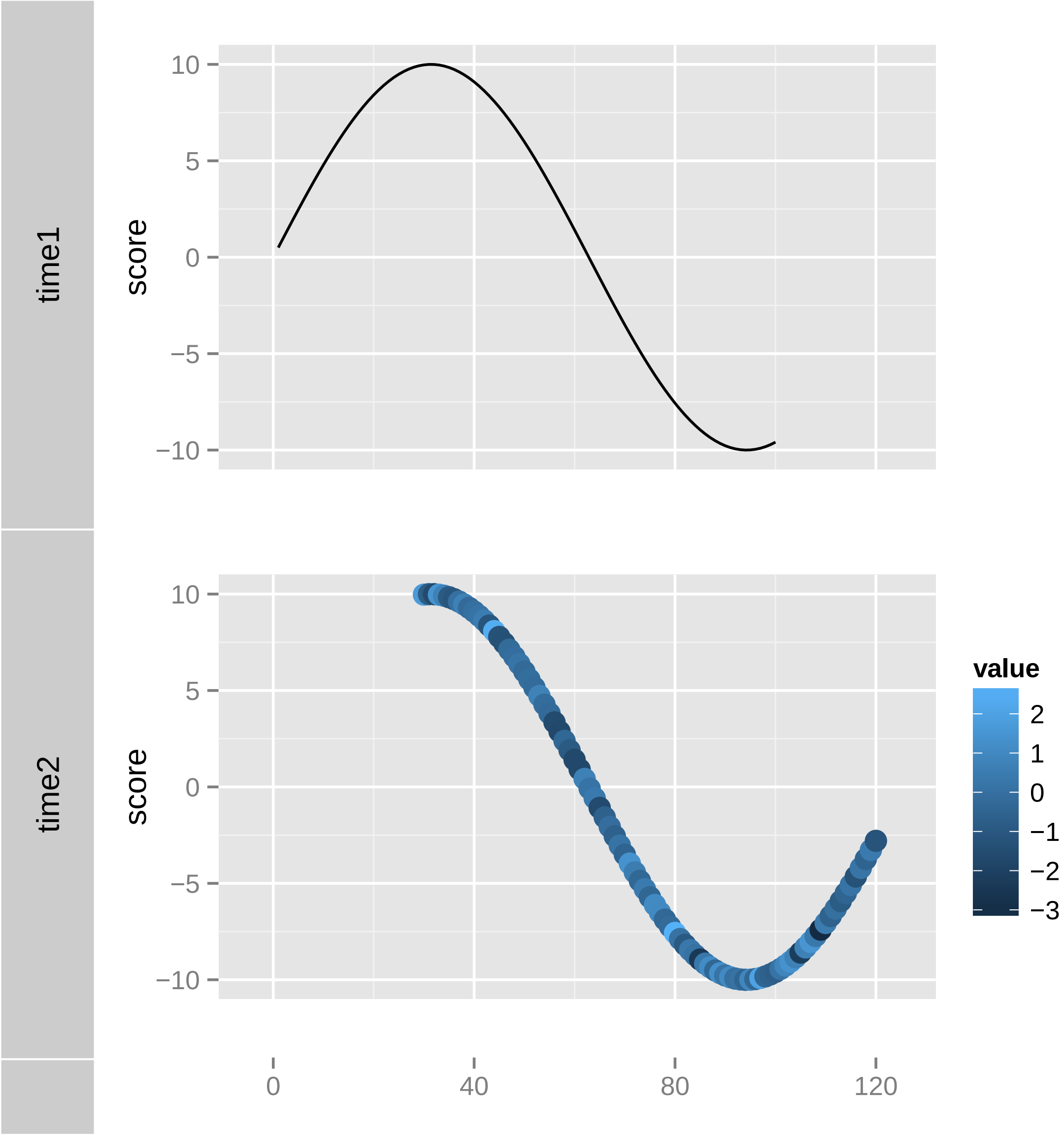
```
tracks(p1, p2) + theme_tracks_sunset()
```



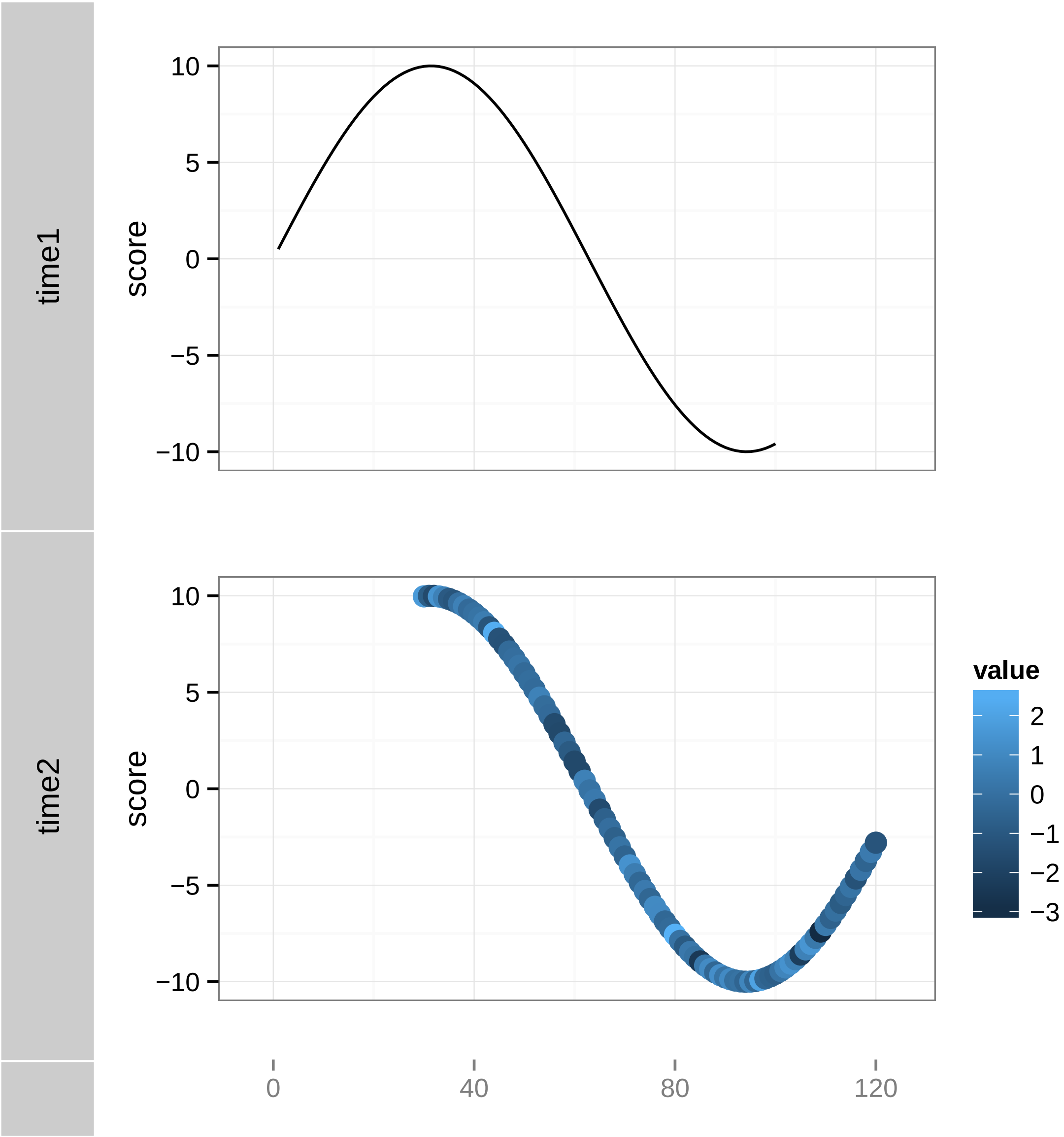
```r
## apply a theme to each track
tks <- tracks(time1 = p1, time2 = p2) + theme_bw()
tks
```



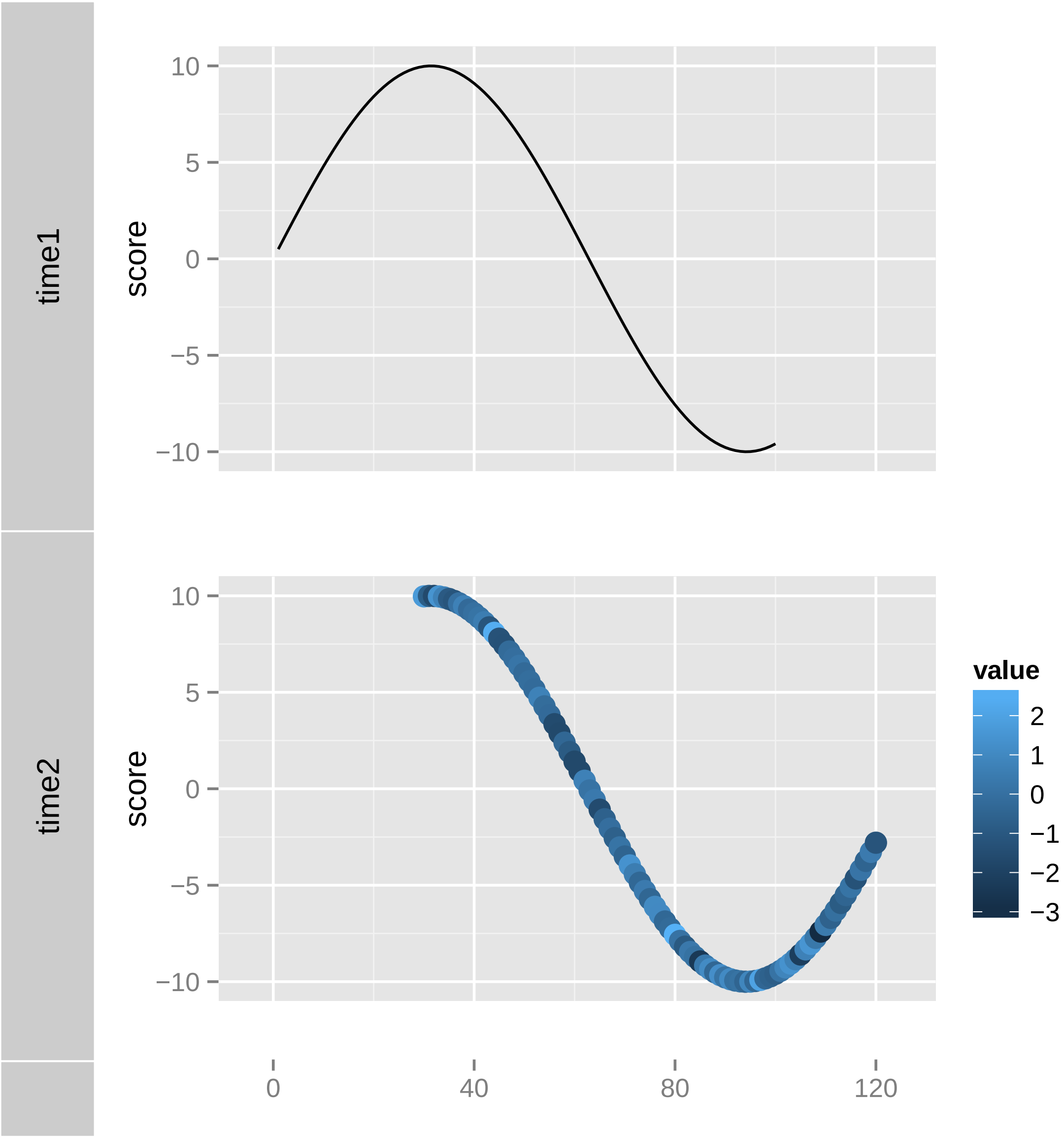
```
## will introduce
reset(tks)
```



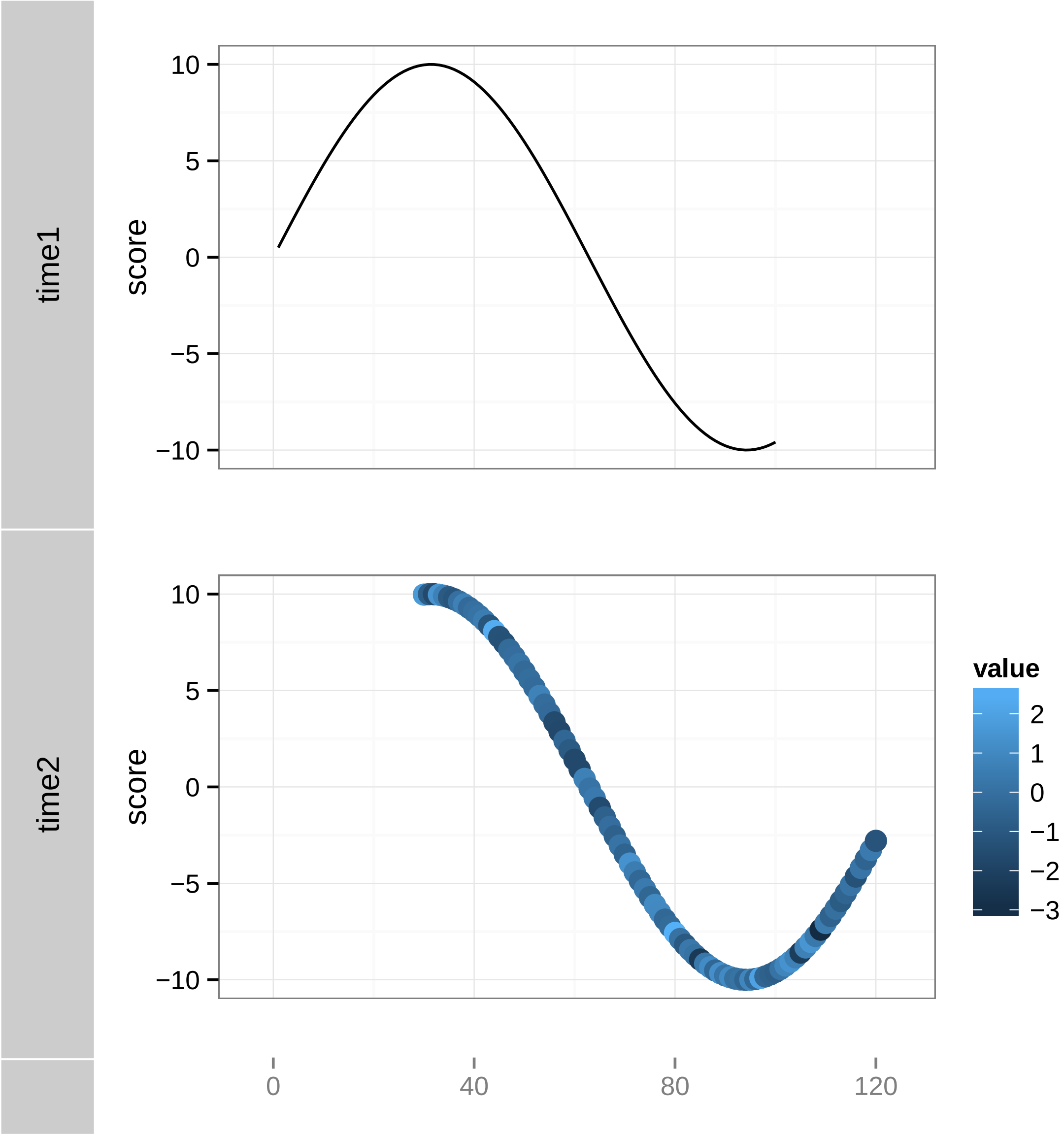
```
## store it with tracks
tks <- tracks(time1 = p1, time2 = p2, theme = theme_bw())
tks
```



```
tks <- tks + theme_gray()
tks
```

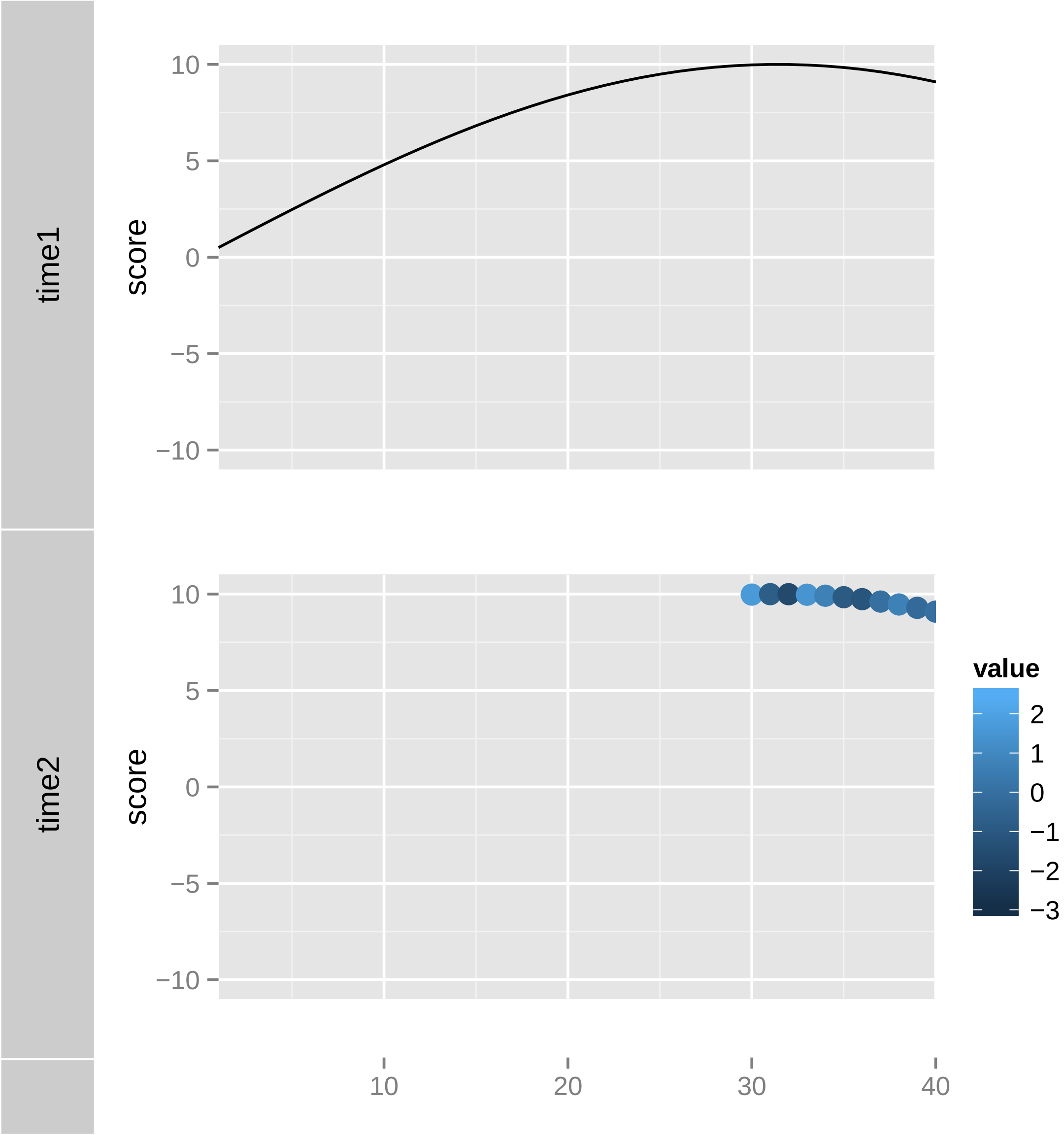


```
## reset will be introduced later
reset(tks)
```

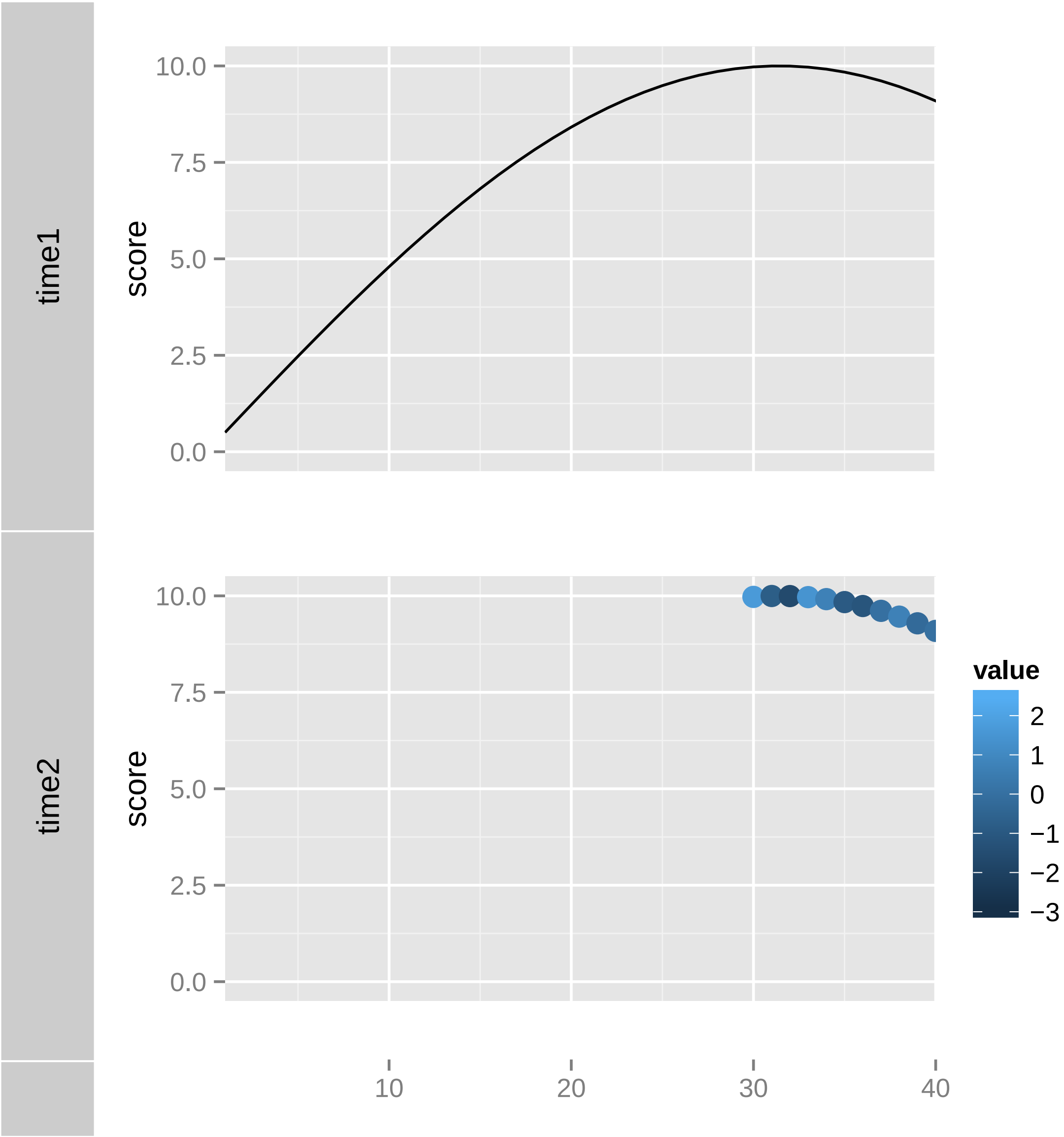


3.3.6 Zoom in/out

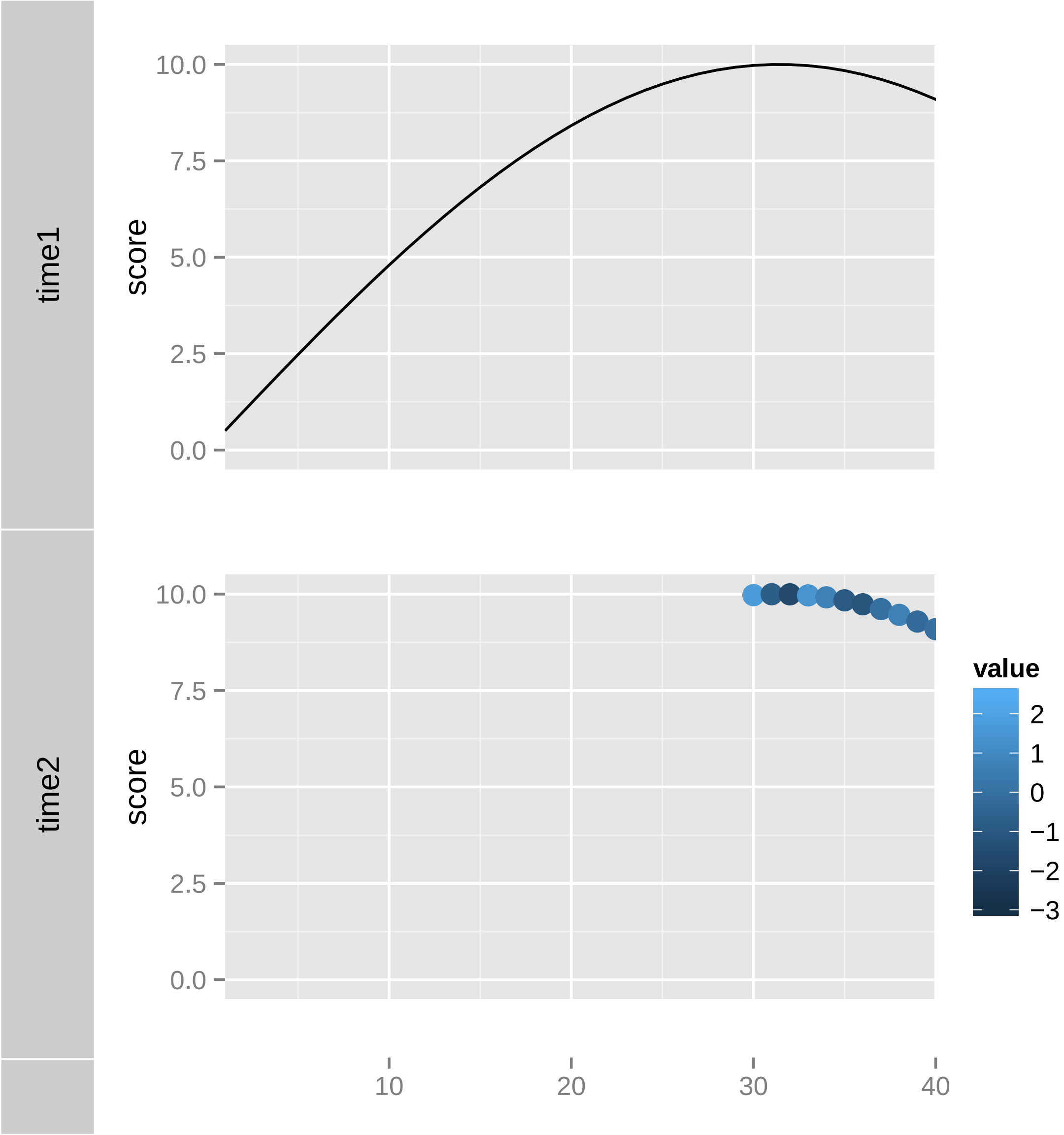
```
tracks(time1 = p1, time2 = p2) + xlim(1, 40)
```



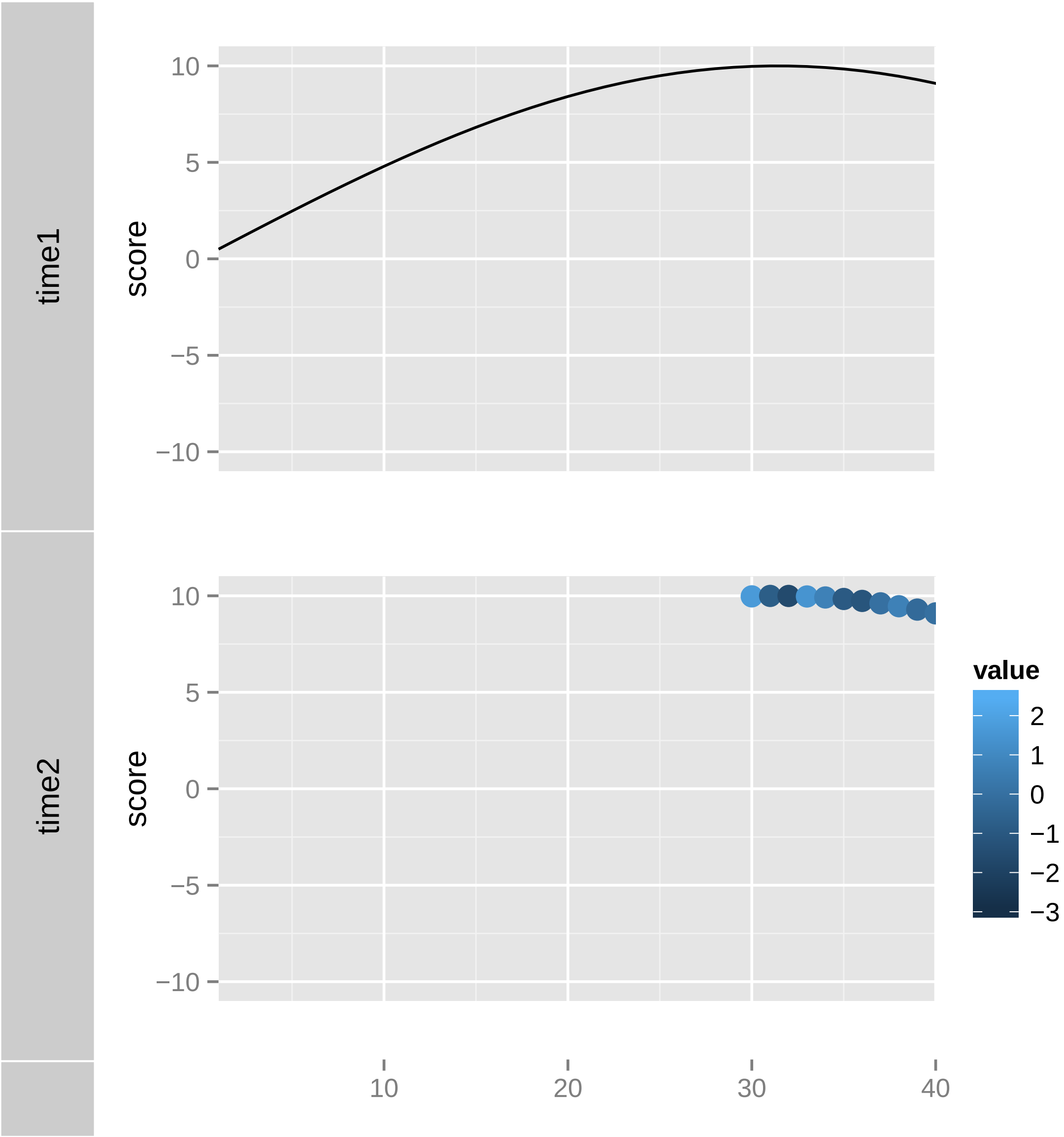
```
tracks(time1 = p1, time2 = p2) + xlim(1, 40) + ylim(0, 10)

## Warning:  Removed 38 rows containing missing values (geom_path).

## Warning:  Removed 58 rows containing missing values (geom_path).

## Warning:  Removed 58 rows containing missing values (geom_point).

## Warning:  Removed 38 rows containing missing values (geom_path).
```



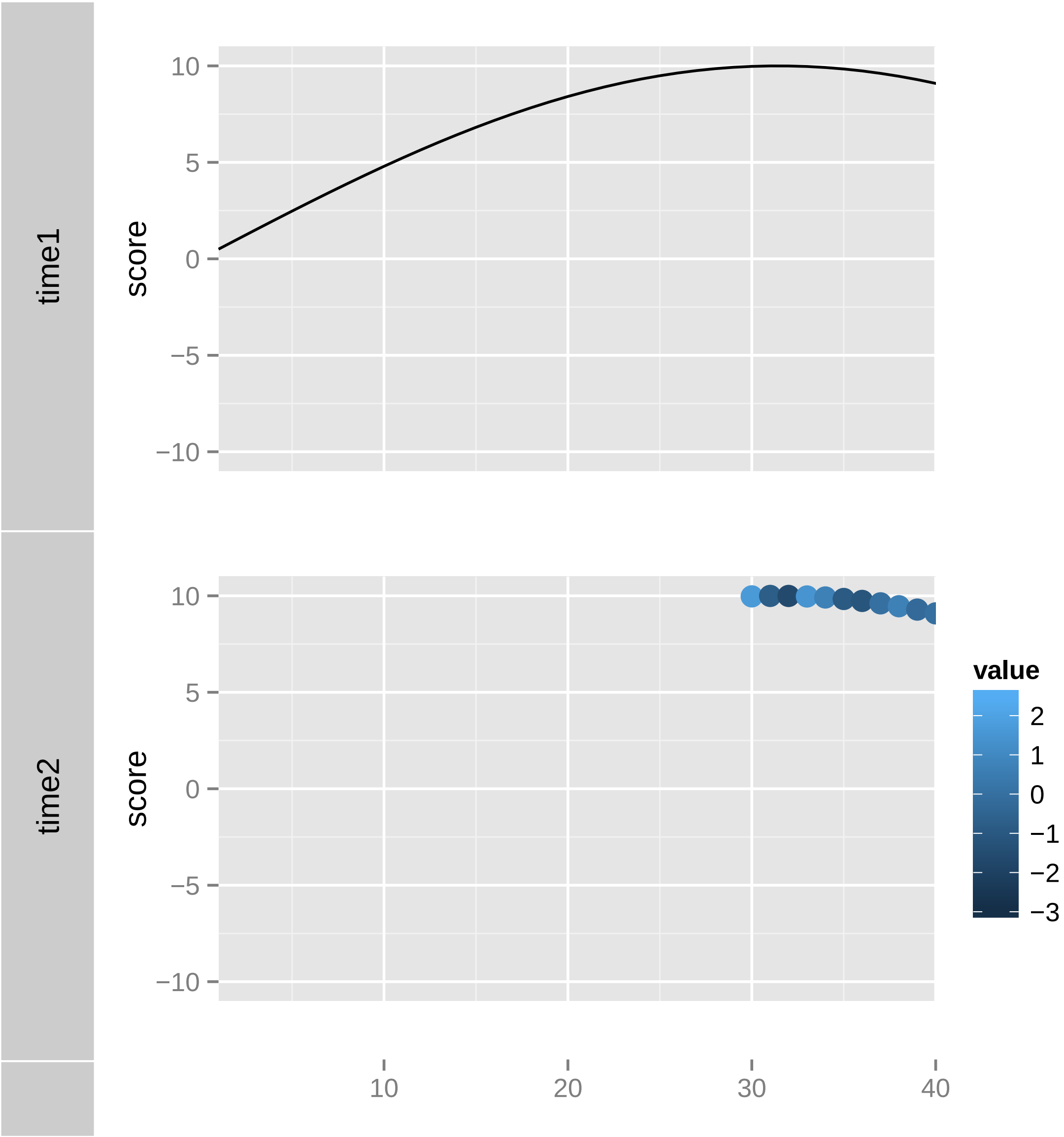
```
tracks(time1 = p1, time2 = p2) + xlim(1, 40) + ylim(0, 10)

## Warning:  Removed 38 rows containing missing values (geom_path).

## Warning:  Removed 58 rows containing missing values (geom_path).

## Warning:  Removed 58 rows containing missing values (geom_point).

## Warning:  Removed 38 rows containing missing values (geom_path).
```



```
library(GenomicRanges)
gr <- GRanges("chr", IRanges(1, 40))
# GRanges
tracks(time1 = p1, time2 = p2) + xlim(gr)
```



```
# IRanges
tracks(time1 = p1, time2 = p2) + xlim(ranges(gr))
```



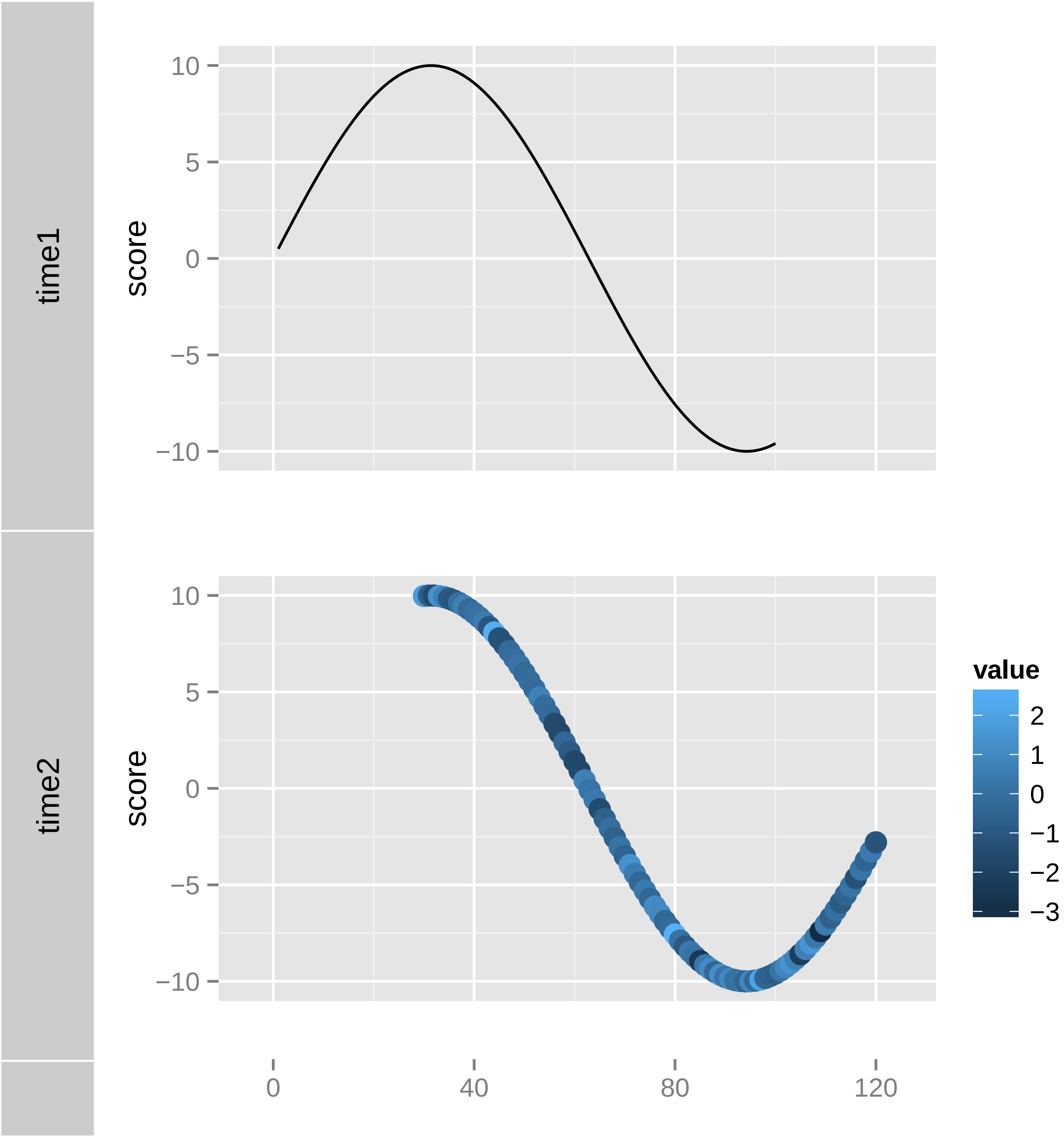
```
tks <- tracks(time1 = p1, time2 = p2)
xlim(tks)


## [1] -10.9 131.9


xlim(tks) <- c(1, 35)
xlim(tks) <- gr
xlim(tks) <- ranges(gr)
```

3.3.7 Backup/restore utilities

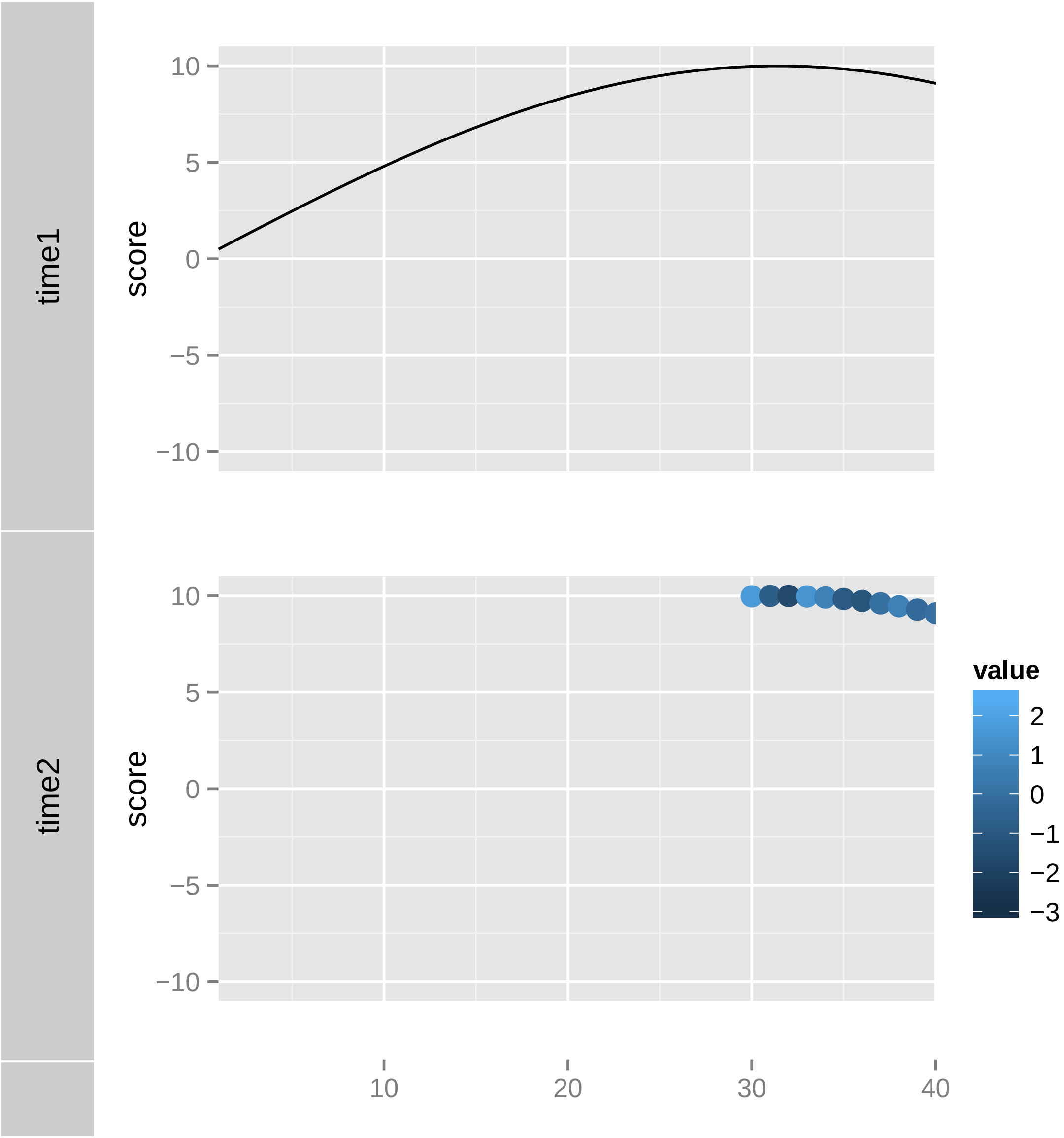
3.3.8 Reset and backup

- reset restore a backup tracks.
- backup clear previous backup and save and backup current tracks.

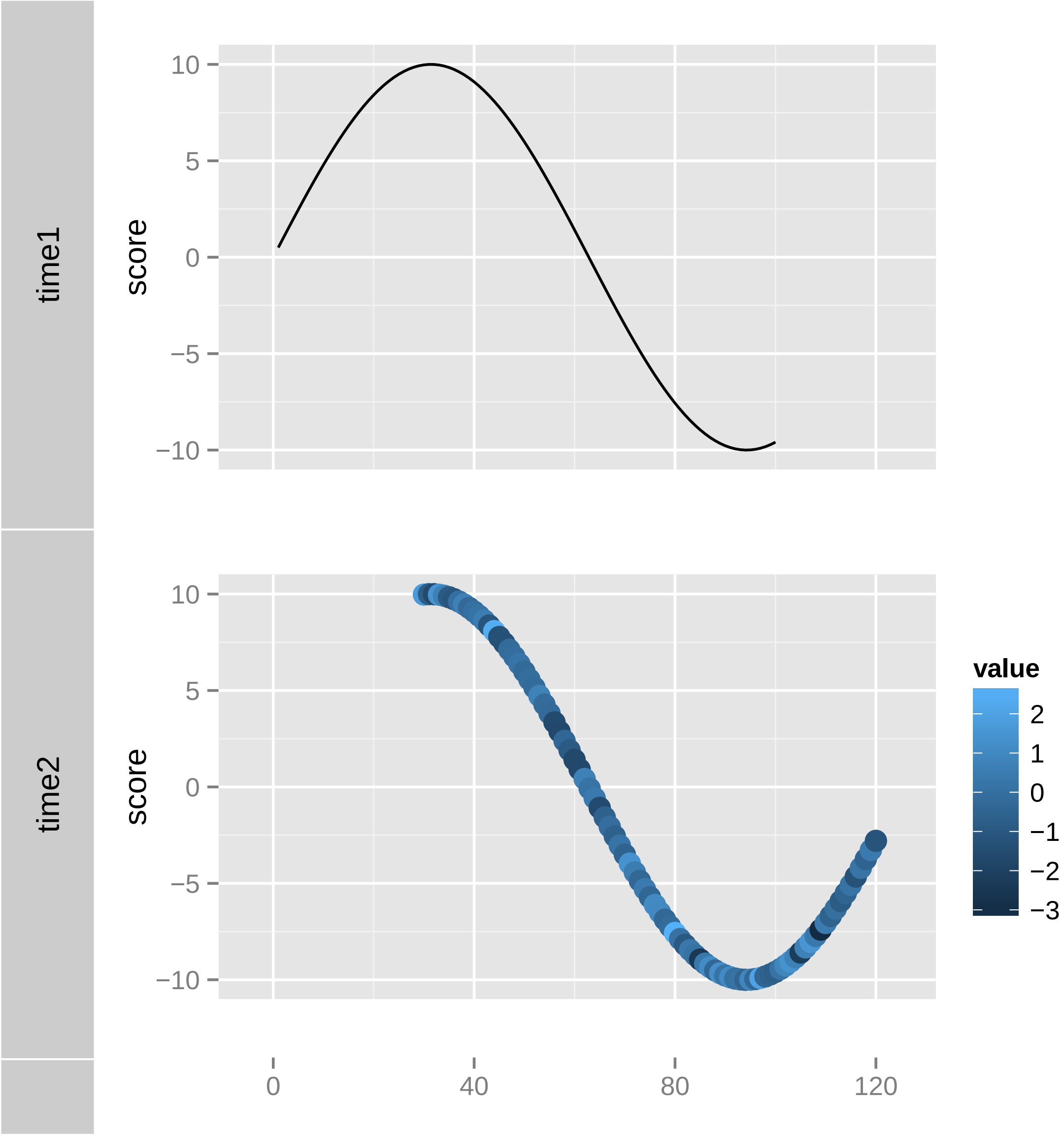
```
tks <- tracks(time1 = p1, time2 = p2)
tks
```



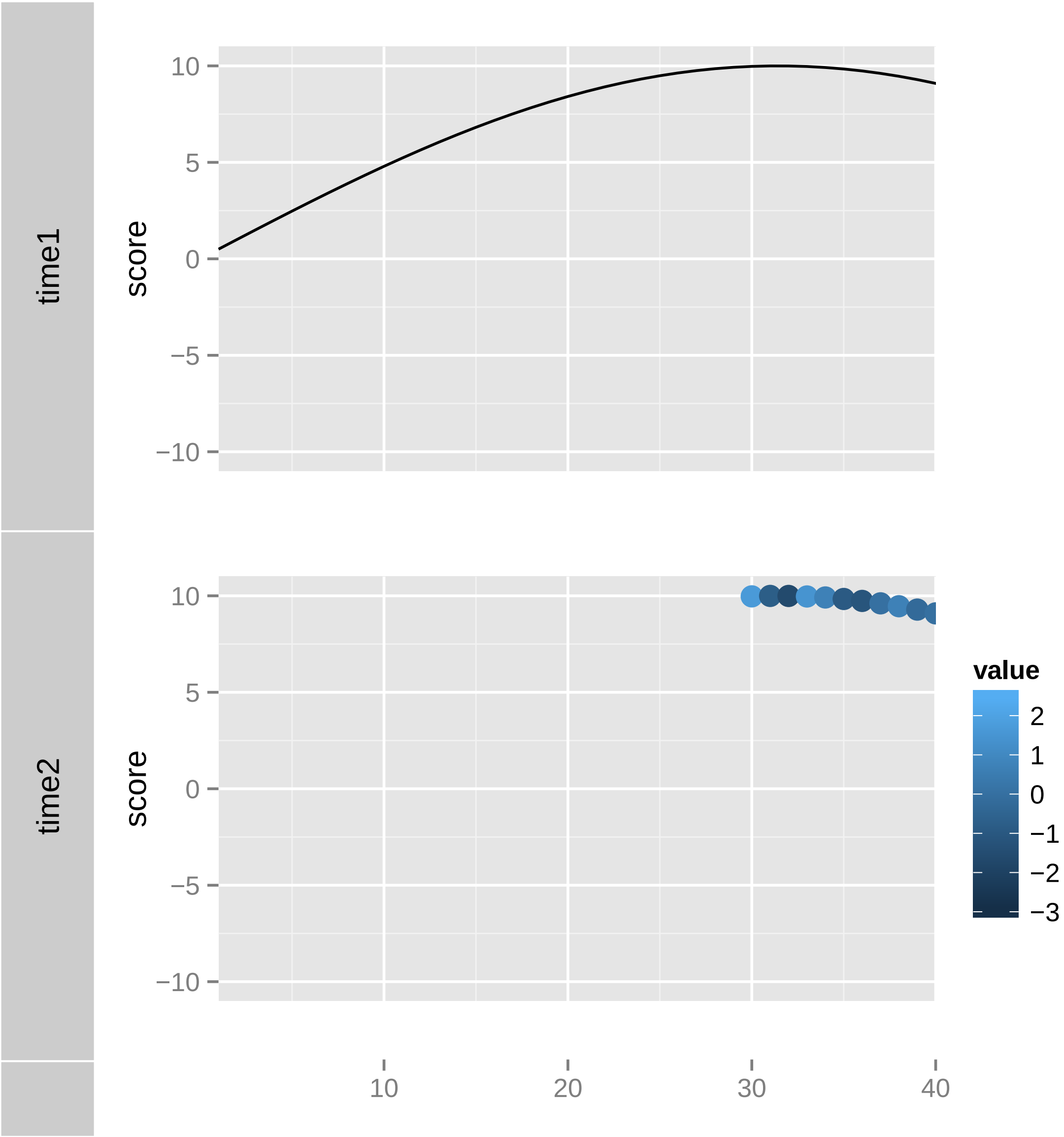
```
tks <- tks + xlim(1, 40)
tks
```



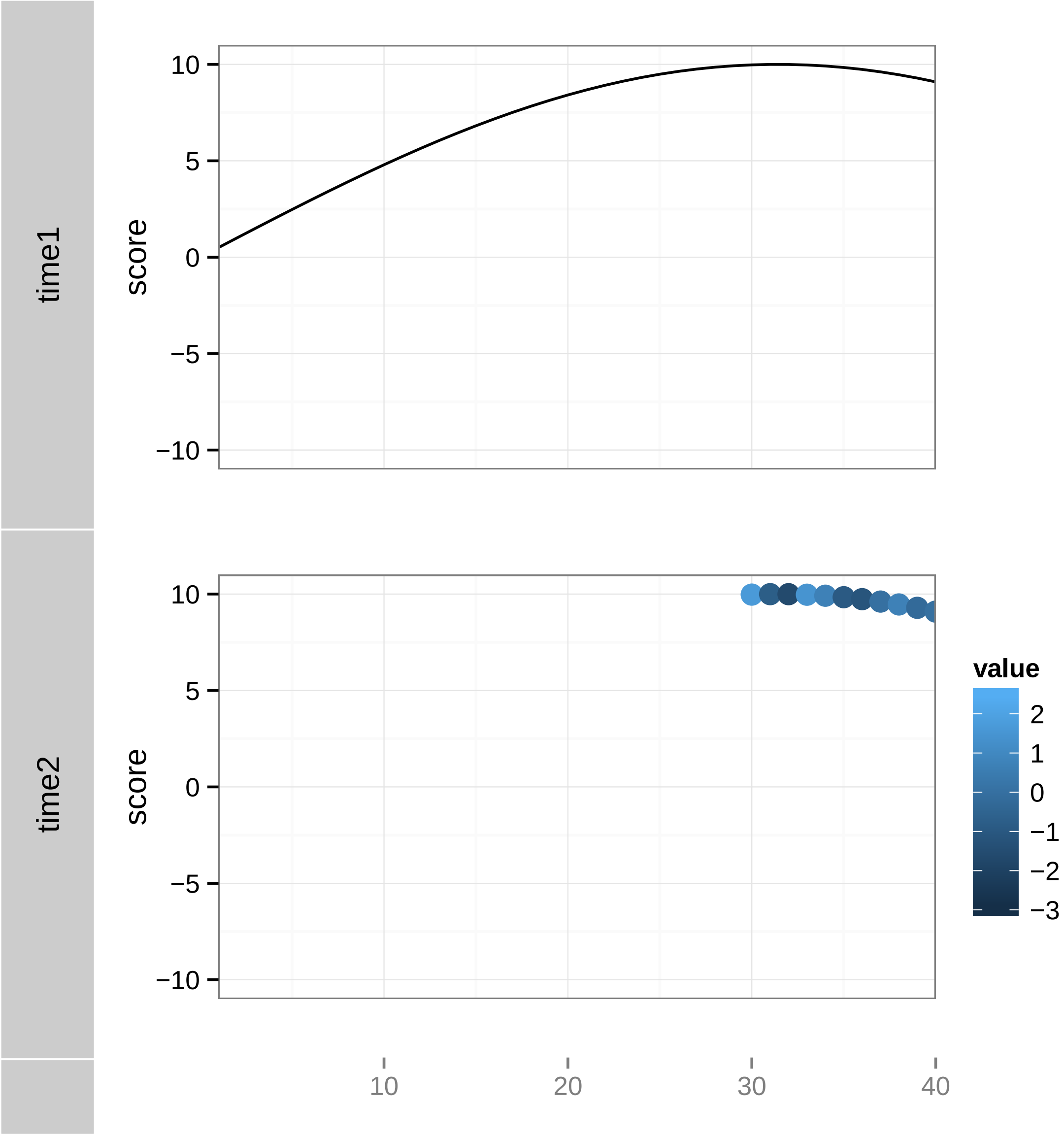
```
reset(tks)
```



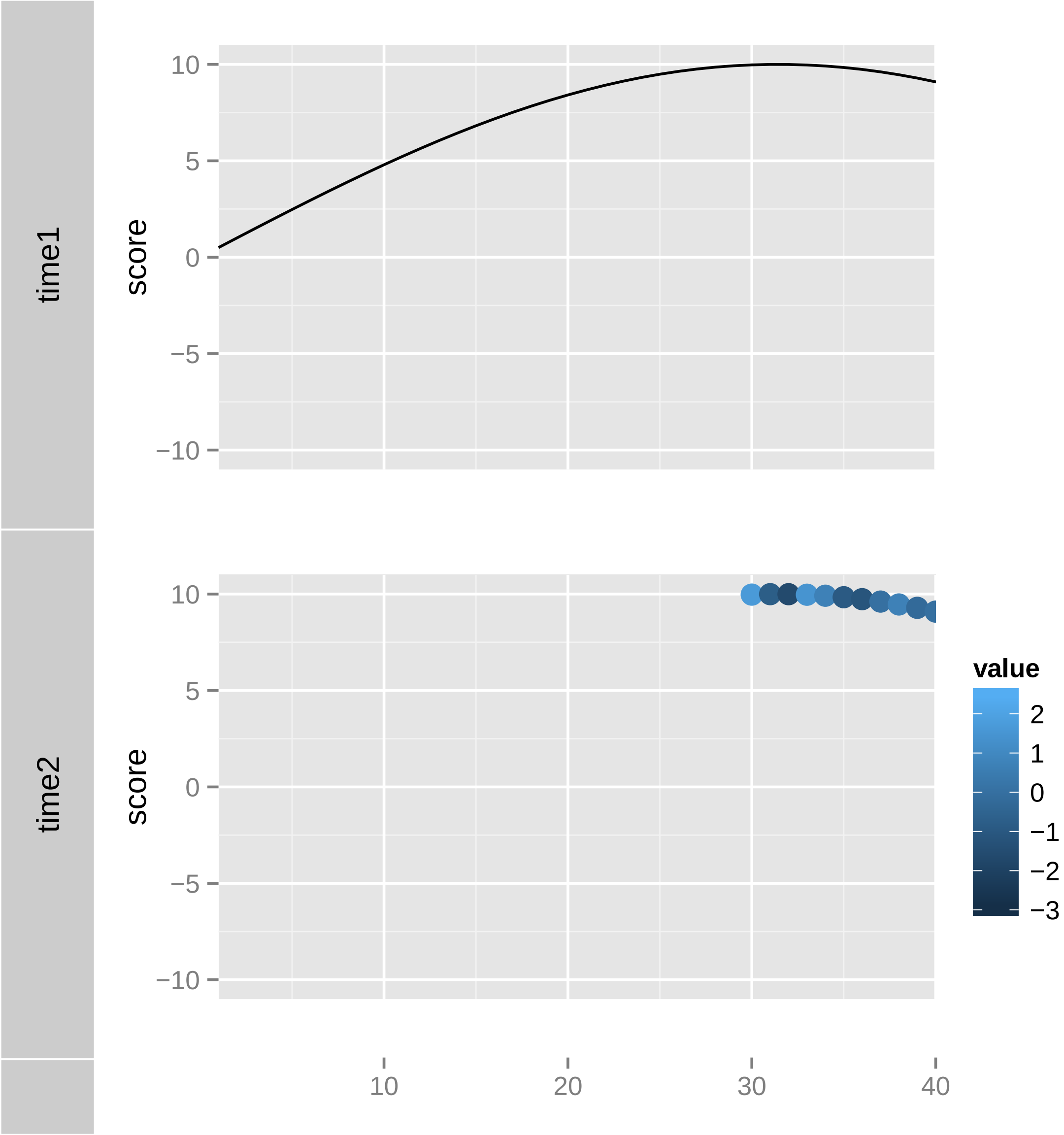
```
tks <- tks + xlim(1, 40)
tks
```



```
tks <- backup(tks)
tks <- tks + theme_bw()
tks
```


time1
score
10
5
0
−5
−10
time2
score
10
5
0
−5
−10
value
2
1
0
−1
−2
−3
10
20
30
40

```
reset(tks)
```



3.4 Discussion

Chapter 4

`mold` method

We already know *ggbio* depends heavily on *ggplot2*, and *data.frame* is the data format *ggplot2* support, everything happens from this point. Since biological data is much more complicated and more specific format are introduced in Bioconductor, to utilize existing components and to make a smooth working pipeline, the first step we do is almost always to convert an object into a *data.frame*. `mold` is added into *ggbio* after Bioconductor 2.11, used for this purpose, before that we use `fortify` generic method, this usually take original data as second argument, and take model as first argument, so we developed our own new generic function here which accept original data as its first argument for dispatching.

Tips: alternatively, *GRanges* is a core data structure we supported in *ggbio*, most components knows how to work for it directly, so coercion from other object to a *GRanges* object is also doable. Actually internally, it is exactly what we did most time.

You may be aware of that most object already support coercion to a *data.frame* object by using function `as.data.frame` or `as` method. So what's the deal here?

- More information will be coerced into *data.frame*, for example, column names and row names of the matrix, or phenotype data for eSet-like object.
- Create more variable statistics, for example, 'midpoint' is added when 'start' and 'end' provided.
- Some object may doesn't have one coercion defined, here is the working point.

eSet, GRanges, IRanges, GRangesList, Seqinfo, matrix, Views, ExpressionSet, SummarizedExperiment, Rle, RleList are currently supported. Please check manual for more information about which column created.

Chapter 5

ggplot generic method and low level utilities

5.1 Objective

- Learn how to construct the graphics by using low level utilities.

To start this chapter, it's recommended to take look at current supported components in *ggplot2*'s websitehttp://docs.ggplot2.org/current/. Just walk around, you will see basic components you could utilize already with pure *ggplot2*.

5.2 ggplot

autoplot 6 is indeed the most conventient way to plot something in *ggbio*, but to create customized graphics, what happened inside autoplot? or sometimes later you may want to create your own graphics layer by layer, you may want to learn the trick more in this chapter.

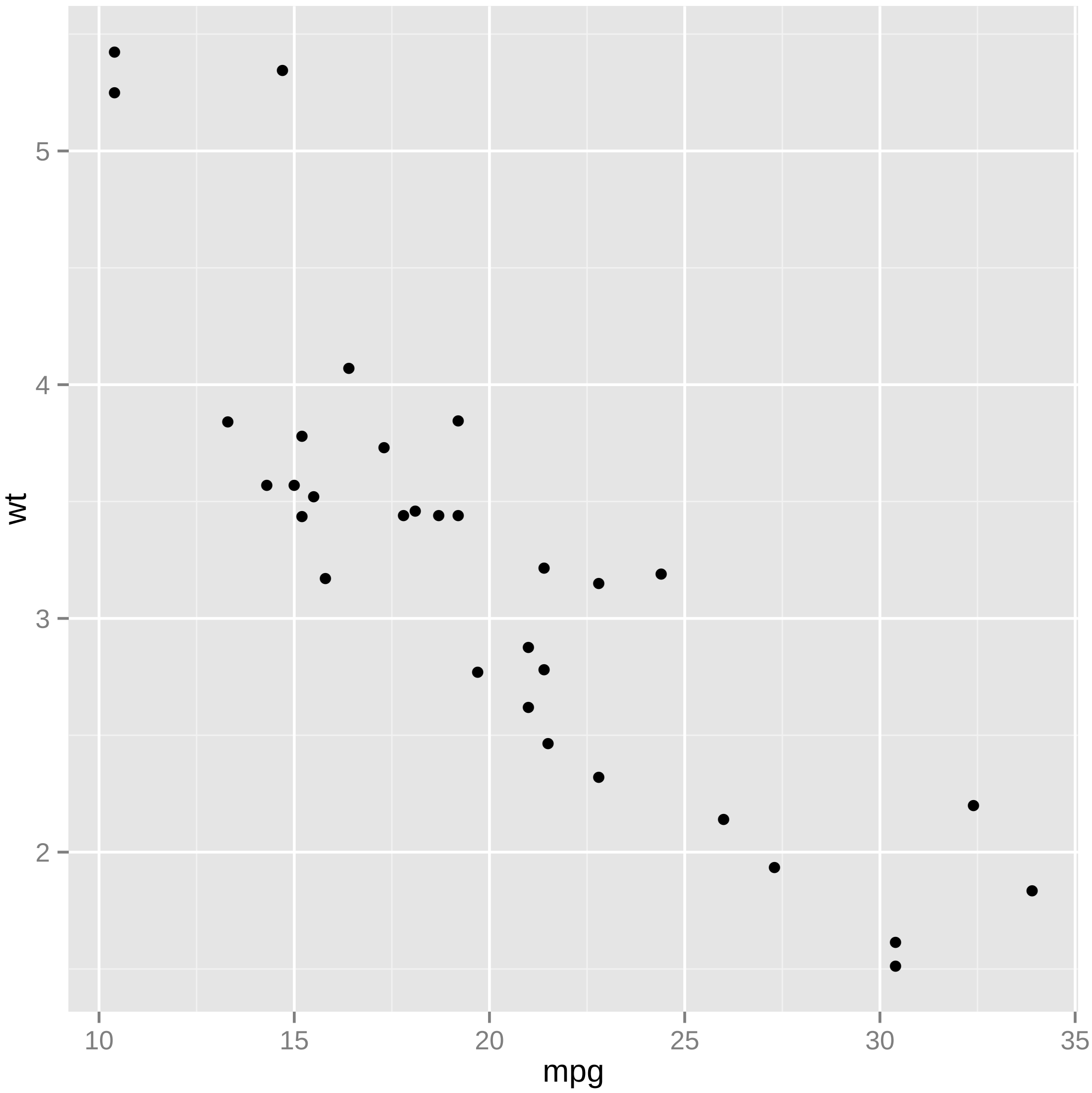
In *ggbio*, ggplot support many core data object in Bioconductor, it take in the original data, and save the original data in .data element of the object, you can use obj$.data to get the original data, and a *data.frame* is stored as any other ggplot in *ggplot2*. The data frame is coerced by running mold method 4 in *ggbio*.

Running ggplot is just creating the **data** layer only, no plot will be generated at all. You have to specify statistics and geometry by adding components later.

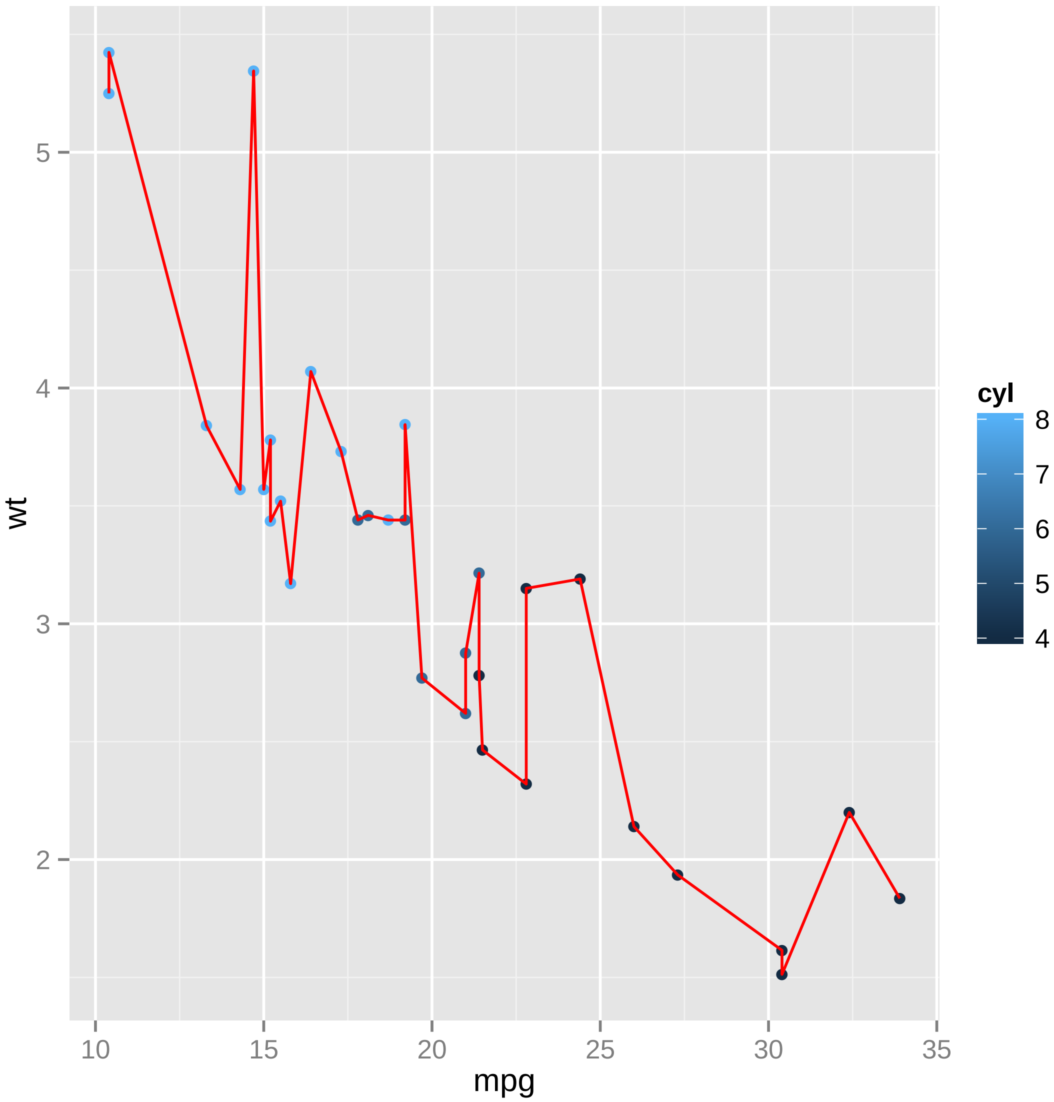
Keep in mind, all varaibles in your molded data.frame could be used to map to graphics.

Let's see a *ggplot2* style construction first.

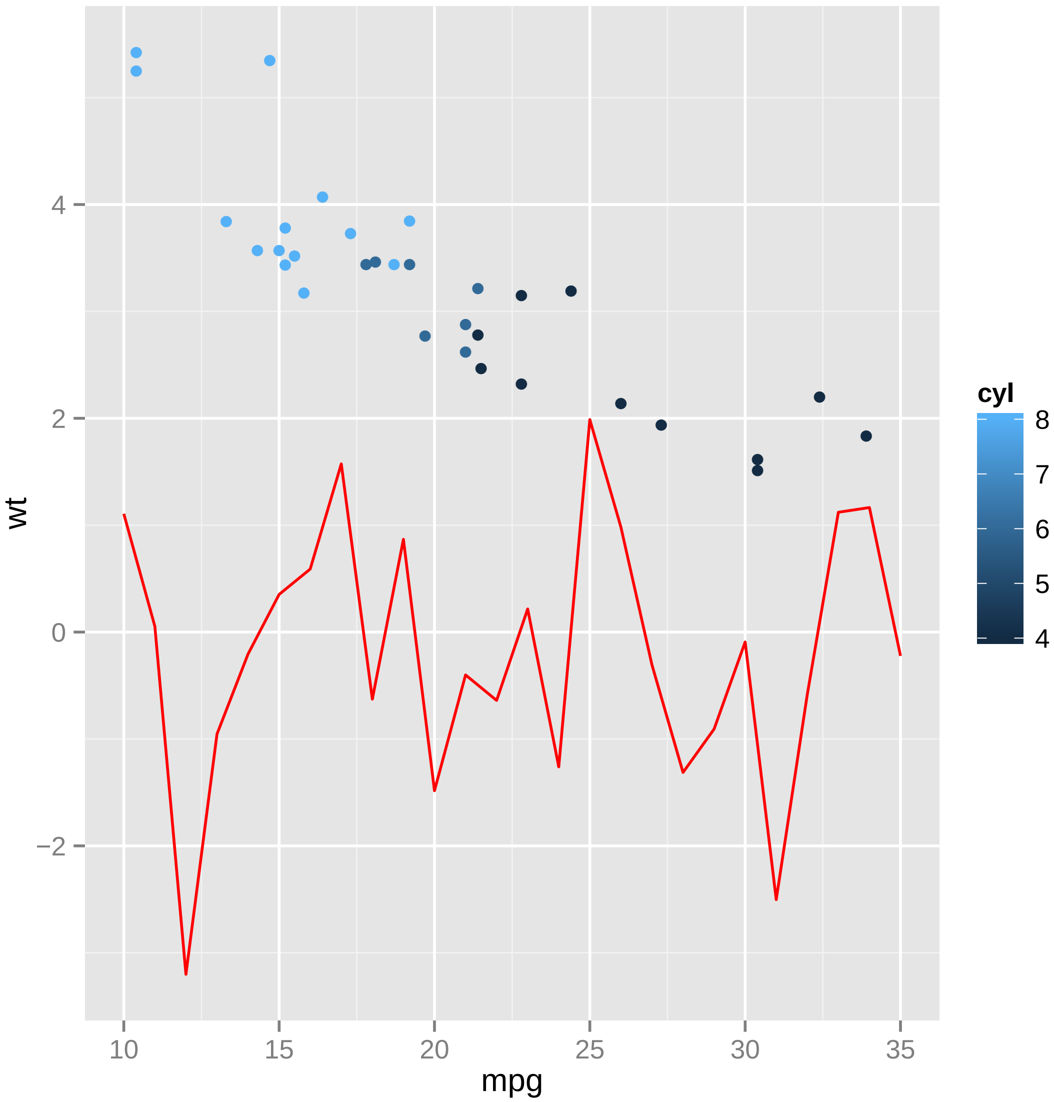
```r
library(ggbio)
p <- ggplot(mtcars, aes(x = mpg, y = wt))
p + geom_point()
```



```
p + geom_point(aes(color = cyl)) + geom_line(color = "red")
```



```
## adding a new data layer
p + geom_point(aes(color = cyl)) + geom_line(data = data.frame(x = 10:35,
    y = rnorm(26)), aes(x = x, y = y), color = "red")
```



If you don't pass any new data in later additive component, the default data would be used, otherwise if you want to show a different data on another layer, just pass it with the components.

Then let's take a look at *ggbio*'s API, following the same style.

```
library(GenomicRanges)
set.seed(1)
N <- 100
gr <- GRanges(seqnames = sample(c("chr1", "chr2", "chr3"), size = N, replace = TRUE),
    IRanges(start = sample(1:300, size = N, replace = TRUE), width = sample(70:75,
        size = N, replace = TRUE)), strand = sample(c("+", "-", "*"), size = N,
        replace = TRUE), value = rnorm(N, 10, 3), score = rnorm(N, 100, 30),
    sample = sample(c("Normal", "Tumor"), size = N, replace = TRUE), pair = sample(letters,
        size = N, replace = TRUE))
```

```
class(gr)
```

```
## [1] "GRanges"
## attr(,"package")
## [1] "GenomicRanges"
```

```
str(gr)
```

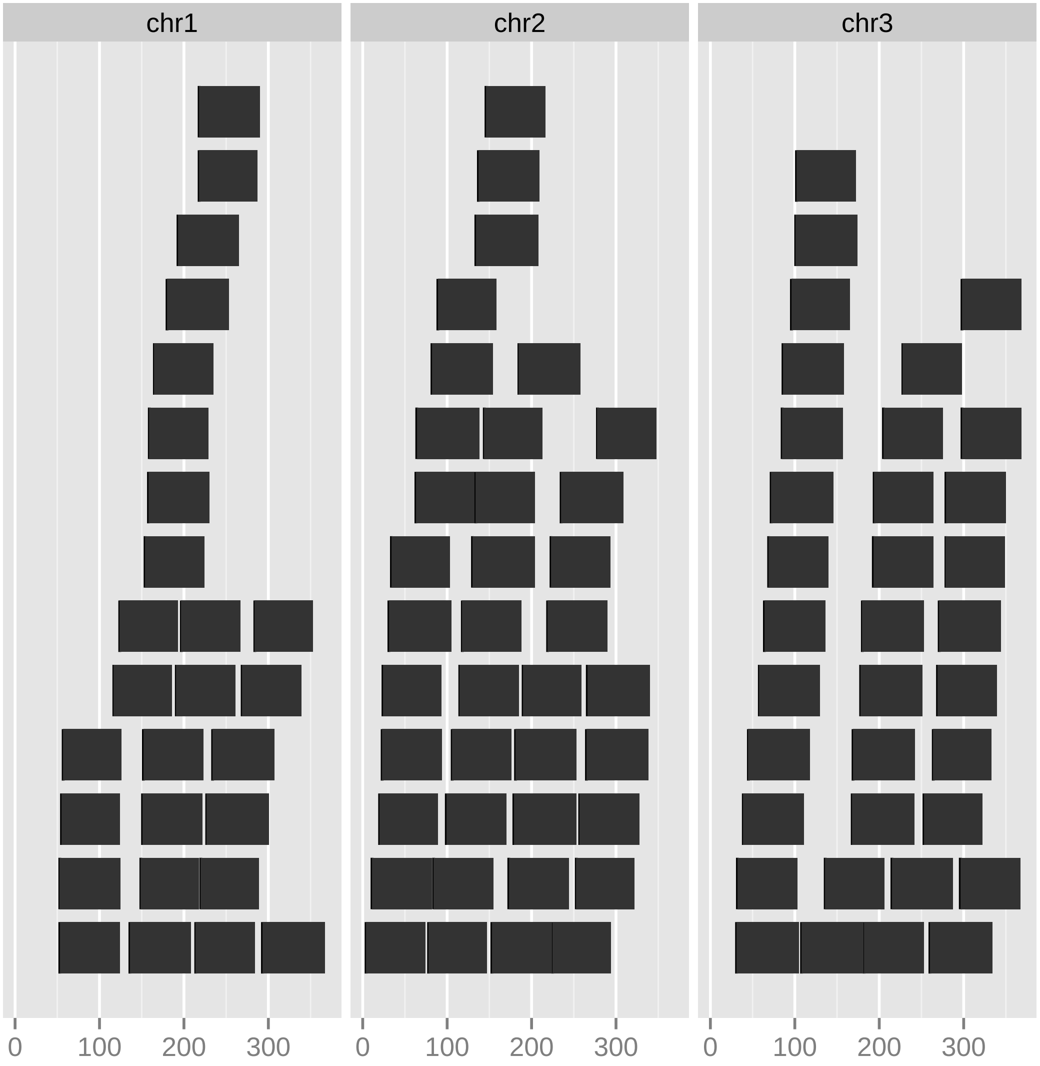
```
## Formal class 'GRanges' [package "GenomicRanges"] with 6 slots
##   ..@ seqnames       :Formal class 'Rle' [package "IRanges"] with 4 slots
##   .. .. ..@ values         : Factor w/ 3 levels "chr1","chr2",..: 1 2 3 1 3 2 1 3 2 3 ...
##   .. .. ..@ lengths        : int [1:69] 1 2 1 1 2 2 3 1 1 1 ...
##   .. .. ..@ elementMetadata: NULL
##   .. .. ..@ metadata       : list()
##   ..@ ranges         :Formal class 'IRanges' [package "IRanges"] with 6 slots
##   .. .. ..@ start          : int [1:100] 197 106 82 298 191 64 39 144 278 180 ...
##   .. .. ..@ width          : int [1:100] 71 71 73 71 71 73 73 70 71 74 ...
##   .. .. ..@ NAMES          : NULL
##   .. .. ..@ elementType    : chr "integer"
##   .. .. ..@ elementMetadata: NULL
##   .. .. ..@ metadata       : list()
##   ..@ strand         :Formal class 'Rle' [package "IRanges"] with 4 slots
##   .. .. ..@ values         : Factor w/ 3 levels "+","-","*": 3 1 2 3 1 3 1 2 1 3 ...
##   .. .. ..@ lengths        : int [1:74] 1 1 3 1 1 1 1 1 2 2 ...
##   .. .. ..@ elementMetadata: NULL
##   .. .. ..@ metadata       : list()
##   ..@ elementMetadata:Formal class 'DataFrame' [package "IRanges"] with 6 slots
##   .. .. ..@ rownames       : NULL
##   .. .. ..@ nrows          : int 100
##   .. .. ..@ listData       :List of 4
##   .. .. .. ..$ value : num [1:100] 11.23 15.07 14.76 9.01 3.14 ...
##   .. .. .. ..$ score : num [1:100] 126.8 68.6 159.1 88.5 149.6 ...
##   .. .. .. ..$ sample: chr [1:100] "Tumor" "Normal" "Tumor" "Tumor" ...
##   .. .. .. ..$ pair  : chr [1:100] "v" "t" "h" "e" ...
##   .. .. ..@ elementType    : chr "ANY"
##   .. .. ..@ elementMetadata: NULL
##   .. .. ..@ metadata       : list()
##   ..@ seqinfo        :Formal class 'Seqinfo' [package "GenomicRanges"] with 4 slots
##   .. .. ..@ seqnames   : chr [1:3] "chr1" "chr2" "chr3"
##   .. .. ..@ seqlengths : int [1:3] NA NA NA
##   .. .. ..@ is_circular: logi [1:3] NA NA NA
##   .. .. ..@ genome     : chr [1:3] NA NA NA
##   ..@ metadata       : list()
```

```
head(gr)
```

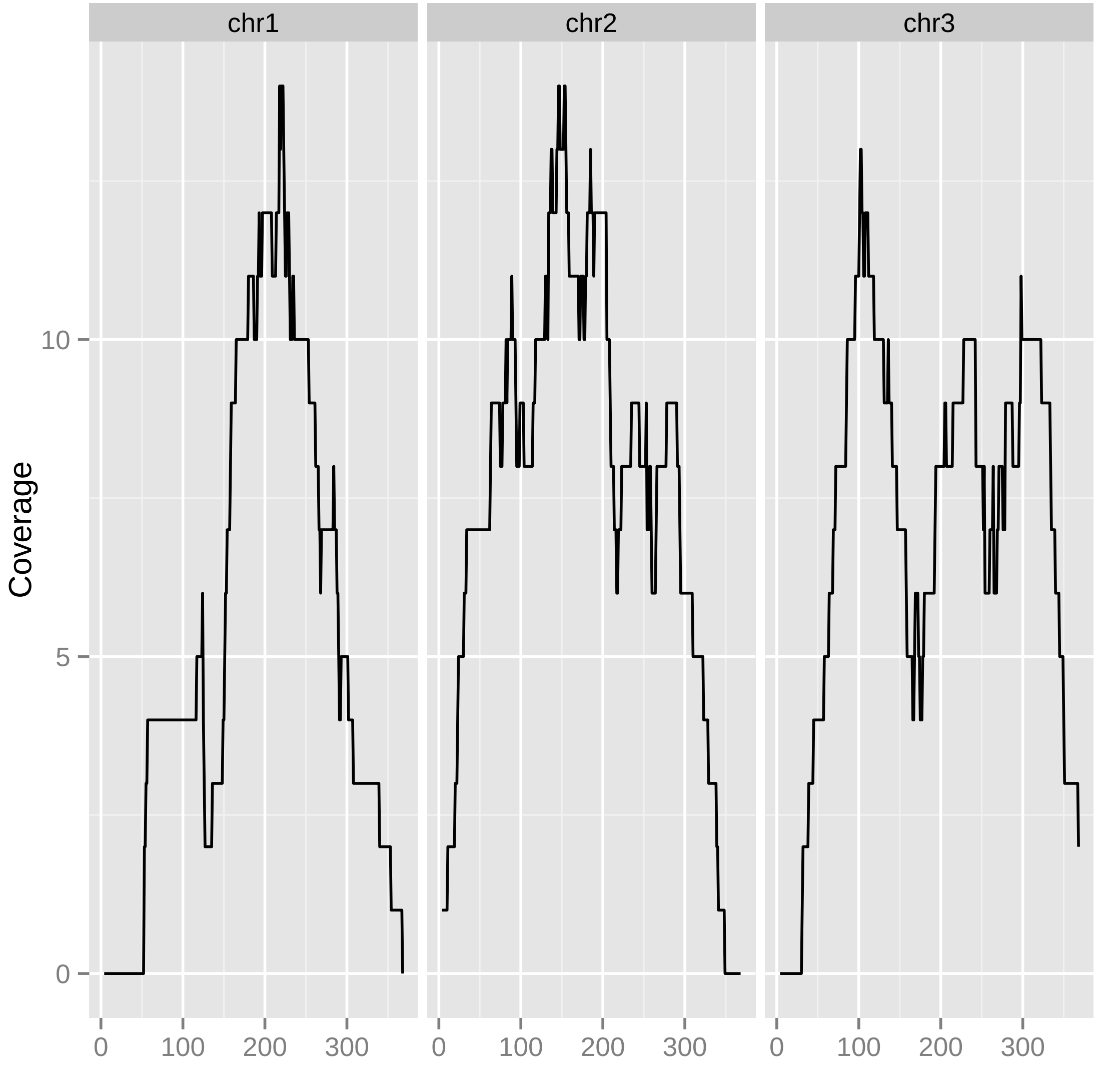
```
## GRanges with 6 ranges and 4 metadata columns:
##       seqnames     ranges strand |     value     score      sample
##          <Rle>  <IRanges>  <Rle> | <numeric> <numeric> <character>
##   [1]     chr1 [197, 267]      * |    11.228    126.81       Tumor
##   [2]     chr2 [106, 176]      + |    15.067     68.58      Normal
##   [3]     chr2 [ 82, 154]      - |    14.760    159.14       Tumor
##   [4]     chr3 [298, 368]      - |     9.007     88.49       Tumor
##   [5]     chr1 [191, 261]      - |     3.144    149.62      Normal
##   [6]     chr3 [ 64, 136]      * |    17.493    145.37       Tumor
##              pair
##       <character>
##   [1]           v
##   [2]           t
##   [3]           h
##   [4]           e
##   [5]           f
##   [6]           b
##   ---
##   seqlengths:
##    chr1 chr2 chr3
##      NA   NA   NA
```

```
p <- ggplot(gr)
p + geom_rect()
```

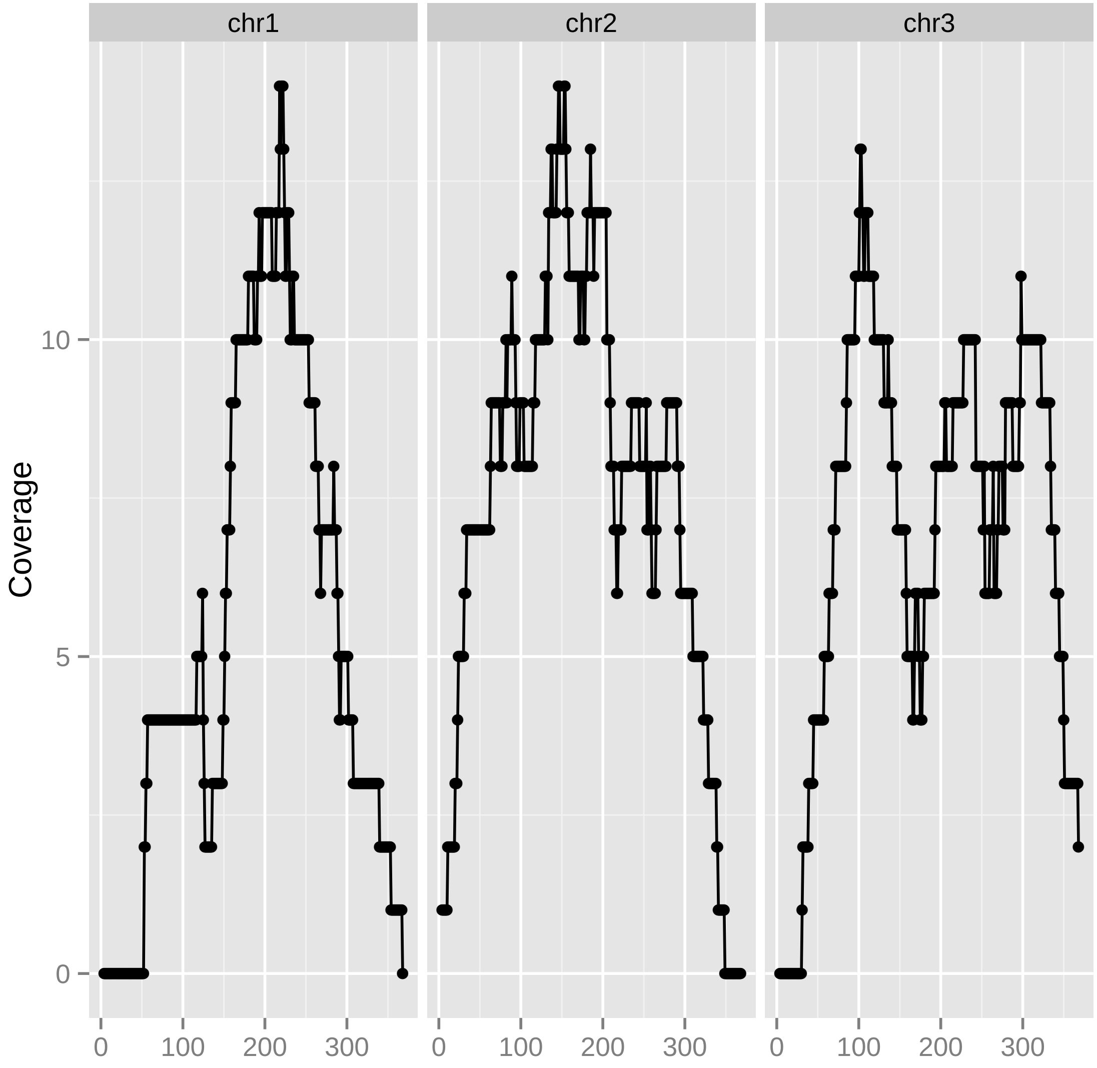
```
## Object of class "ggbio"
```



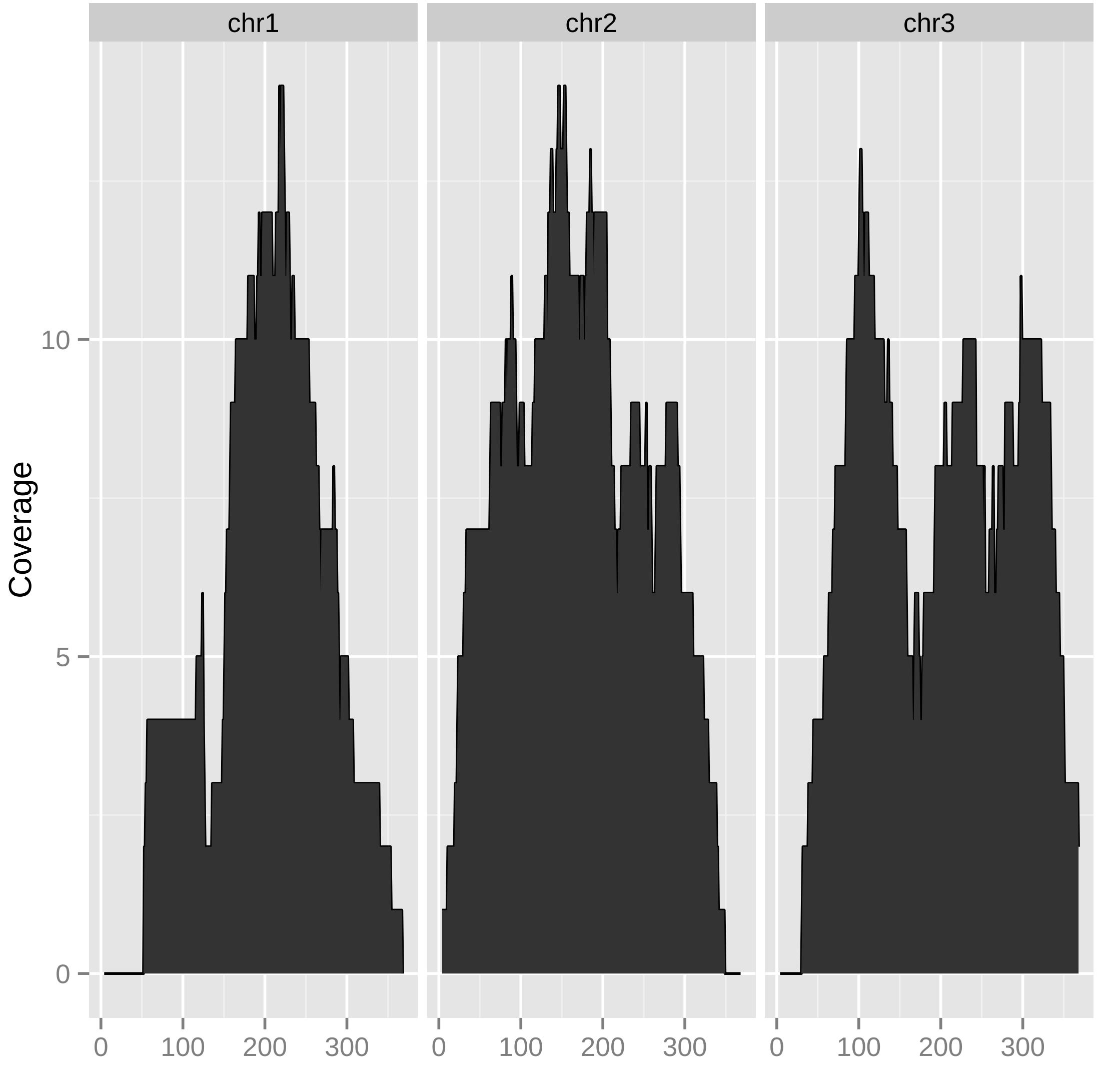
```
## NULL

p + stat_coverage()

## Object of class "ggbio"
```



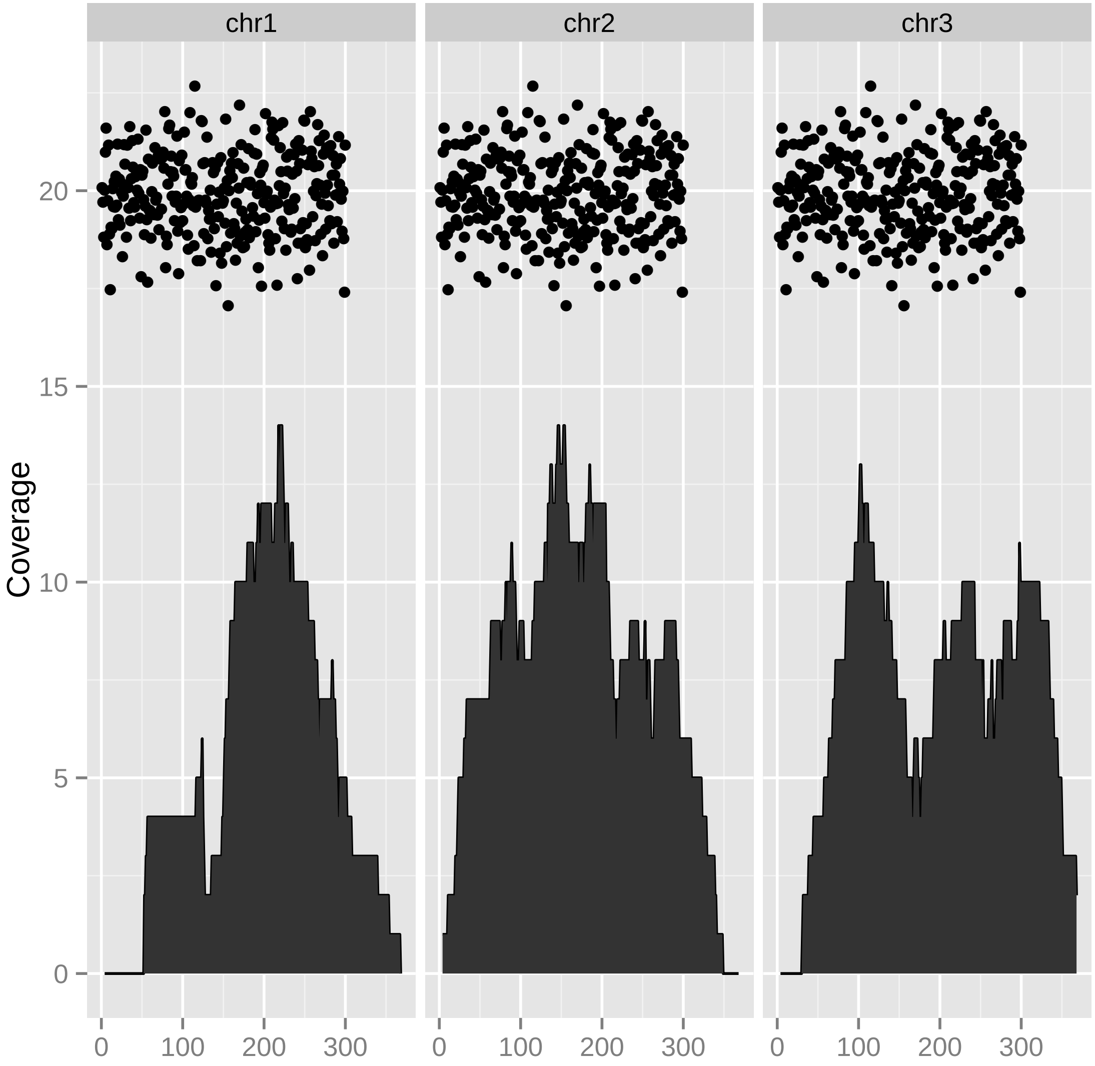
```
## NULL

p + stat_coverage() + geom_point()

## Object of class "ggbio"
```



```
## NULL

p + stat_coverage() + geom_area()

## Object of class "ggbio"
```



```
## NULL


## new data
p + stat_coverage() + geom_area() + geom_point(data = data.frame(x = 1:300,
    y = rnorm(300, 20)), aes(x = x, y = y))


## Object of class "ggbio"
```



```
## NULL
```

In next section, we introduce more extended components.

5.3 Components

Let's first take a look at a general table about stat/geom/layout/coord/scale, Please notice the difference between *ggplot2* and *ggbio*, in *ggbio*, those components are also generic method. So many of them works for not only *GRanges* object, but some other objects too. But don't be supprized when you autoplot on objects other than *GRanges, GRangesList, IRanges*, and some components or transformation doesn't work. Support will be fullfiled gradually.

Comp	name	usage	icon
geom	geom_rect	rectangle	
	geom_segment	segment	
	geom_chevron	chevron	
	geom_arrow	arrow	
	geom_arch	arches	
	geom_bar	bar	
	geom_alignment	alignment (gene)	
stat	stat_coverage	coverage (of reads)	
	stat_mismatch	mismatch pileup for alignments	
	stat_aggregate	aggregate in sliding window	
	stat_stepping	avoid overplotting	
	stat_gene	consider gene structure	
	stat_table	tabulate ranges	
	stat_identity	no change	
coord	linear	ggplot2 linear but facet by chromosome	
	genome	put everything on genomic coordinates	
	truncate gaps	compact view by shrinking gaps	
layout	track	stacked tracks	
	karyogram	karyogram display	
	circle	circular	
faceting	formula	facet by formula	
	ranges	facet by ranges	
scale	scale_x_sequnit	change x unit:Mb, kb, bp	
	scale_fill_giemsa	ideogram color	
	scale_fill_fold_change	around 0 scaling, for heatmap.	

Table 5.1: Components of the basic grammar of graphics, with the extensions available in *ggbio*.

If you want't to get some instance about using those components, please check the on-line manual(`http://tengfei.github.com/ggbio/docs/man/index.html`), it is provided with graphics help you to understand.

Chapter 6

Autoplot method

6.1 API

API about is autoplot is kind of like a wrapper around the grammar. If you are familiar with API of *ggplot2*, it's very similar.

```
autoplot(object = , geom = , stat = , coord = , facets = , scale = , ...)
```

Most time only object parameters are required, and we have default for all other components, *ggbio* have default for each object trying to make smart guess for user's purpose for particular data. ... means for particular data we accept or sometimes require extra arguments to control the graphics. For example, in some cases, like for a TranscriptDb object, user has to pass a which argument to tell *ggbio* which region you want to visualize, not the entire genome which make no sense here.

6.2 Usage

6.2.1 autoplot,GRanges

autoplot for GRanges object is designed to be most general plot API in *ggbio* package. GRanges is most suitable data structure for storing interval data with medata data, which could be used for representing a set of short reads or genomic features.

Supported geom designed specifically for GRanges, including "rect", "chevron", "alignment", "arrowrect", "arrow", "segment", "arch", and special statistical transformation contains "identity", "coverage", "stepping", "aggregate", "table", "gene", "mismatch". And they are implemented in lower API, such as geom_alignment and stat_coverage. If you pass other 'geom'and 'stat' other than those ones, it first use 'mold' method in *ggbio* to coerce a GRanges into a 'data.frame' object. And a new variable 'midpoint' is created and added to final 'data.frame' to be used to mapped as 'x'. So you can use it as other *ggplot2* API. For a full table supported, please check

Inside, autoplot will choose the best choice for your combination of 'geom' and 'stat'.

```
autoplot(data, color = score)
## this won't work, you have to use aes() around variable mapping.
autoplot(data, aes(color = score))
```

For arbitrary setting like you just want to use “red” for your fill color, don’t wrap it in your code.

```
autoplot(data, aes(color = score), fill = "red")
```

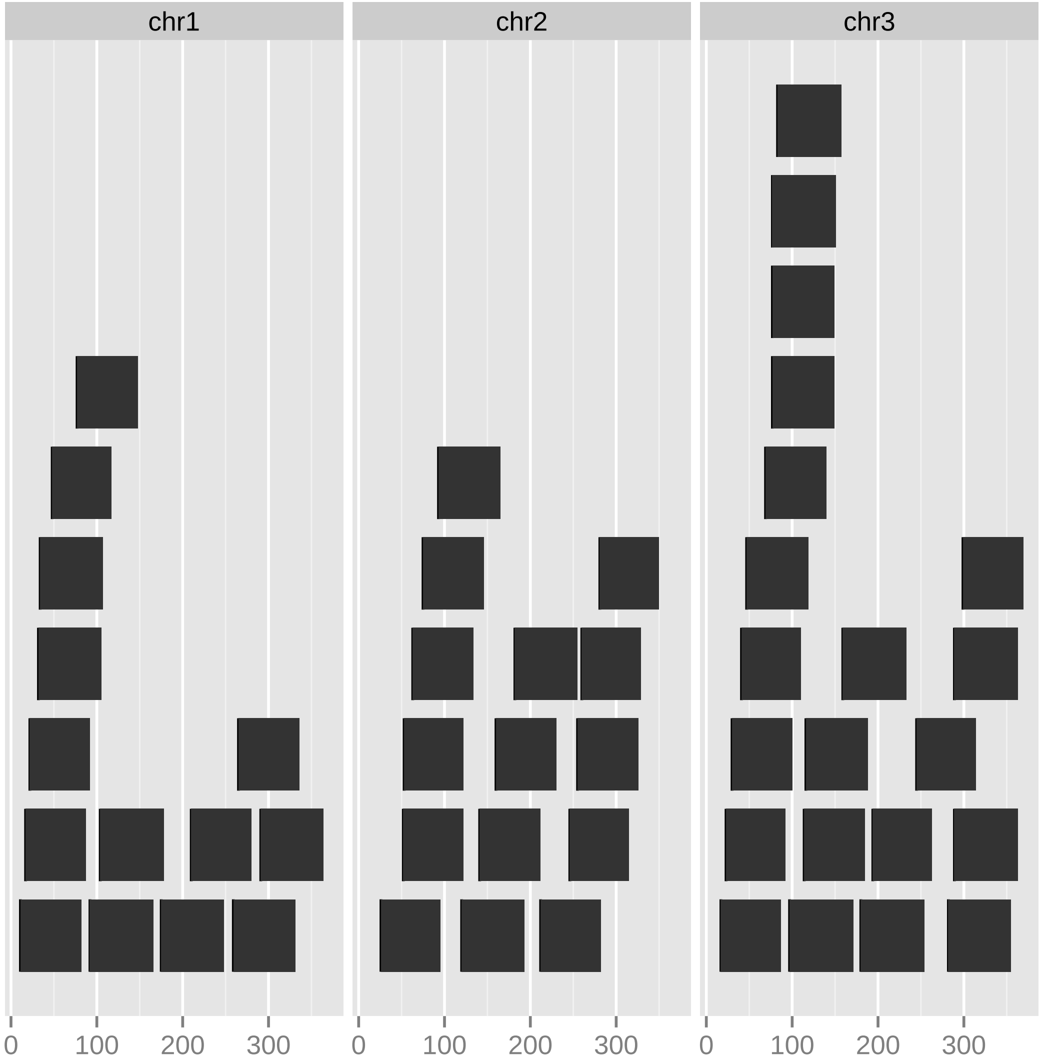
Tips: This is very different from design of `qplot` API in *ggplot2* package, I **force** users to pass the mapping in `aes`. This allow you to wrap the function into your own customized function and make sure the evaluation accurate.

Let’s generate some simulated interval data and store it as `GRanges` object.

```
set.seed(1)
N <- 1000
library(GenomicRanges)
gr <- GRanges(seqnames = sample(c("chr1", "chr2", "chr3"), size = N, replace = TRUE),
    IRanges(start = sample(1:300, size = N, replace = TRUE), width = sample(70:75,
        size = N, replace = TRUE)), strand = sample(c("+", "-", "*"), size = N,
        replace = TRUE), value = rnorm(N, 10, 3), score = rnorm(N, 100, 30),
    sample = sample(c("Normal", "Tumor"), size = N, replace = TRUE), pair = sample(letters,
        size = N, replace = TRUE))

idx <- sample(1:length(gr), size = 50)
```

Default is to use geom "rect" to represent those 'short reads', show overlaped intervals on different levels to help visualize the data, by defeault, the plot will be facetted automatically by chromosomes.

```
autoplot(gr[idx])

## Object of class "ggbio"
```



```
## NULL
```

Figure 6.1: gr-default

Geom ’bar’just show intervals’ region as they are and use a sepcified y in `aes()` to show as the height of bars, default is to use ’score’ in the data if exists, because in most genomic data format, such as BED format, the score are reserved column.

```
set.seed(123)
gr.b <- GRanges(seqnames = "chr1", IRanges(start = seq(1, 100, by = 10),
    width = sample(4:9, size = 10, replace = TRUE)), score = rnorm(10, 10,
    3), value = runif(10, 1, 100))
gr.b2 <- GRanges(seqnames = "chr2", IRanges(start = seq(1, 100, by = 10),
    width = sample(4:9, size = 10, replace = TRUE)), score = rnorm(10, 10,
    3), value = runif(10, 1, 100))
gr.b <- c(gr.b, gr.b2)


## Warning:  Each of the 2 combined objects has sequence levels not in the other:
##   - in ’x’:  chr1
##   - in ’y’:  chr2
##   Make sure to always combine/compare objects based on the same reference
##   genome (use suppressWarnings() to suppress this warning).


head(gr.b)


## GRanges with 6 ranges and 2 metadata columns:
##       seqnames    ranges strand |            score            value
##          <Rle> <IRanges>  <Rle> |        <numeric>        <numeric>
##   [1]     chr1  [ 1,  5]      * | 15.1451949606498 96.3393990211189
##   [2]     chr1  [11, 18]      * | 11.3827486179676  90.327605466824
##   [3]     chr1  [21, 26]      * |  6.2048162961804 69.3798225638457
##   [4]     chr1  [31, 39]      * | 7.93944144431942 79.7512743510306
##   [5]     chr1  [41, 49]      * | 8.66301408970013   3.436754766386
##   [6]     chr1  [51, 54]      * | 13.6722453923184 48.3018011380918
##   ---
##   seqlengths:
##    chr1 chr2
##      NA   NA
```

Facetting, some combination of geom/stat

```
autoplot(gr[idx], geom = "arch", aes(color = value), facets = sample ~ seqnames)


## Object of class "ggbio"
```

```
p1 <- autoplot(gr.b, geom = "bar")

                                    ## use score as y by default

## use value to fill the bar
p2 <- autoplot(gr.b, geom = "bar", aes(fill = value))

                                    ## use score as y by default

tracks(default = p1, fill = p2)
```

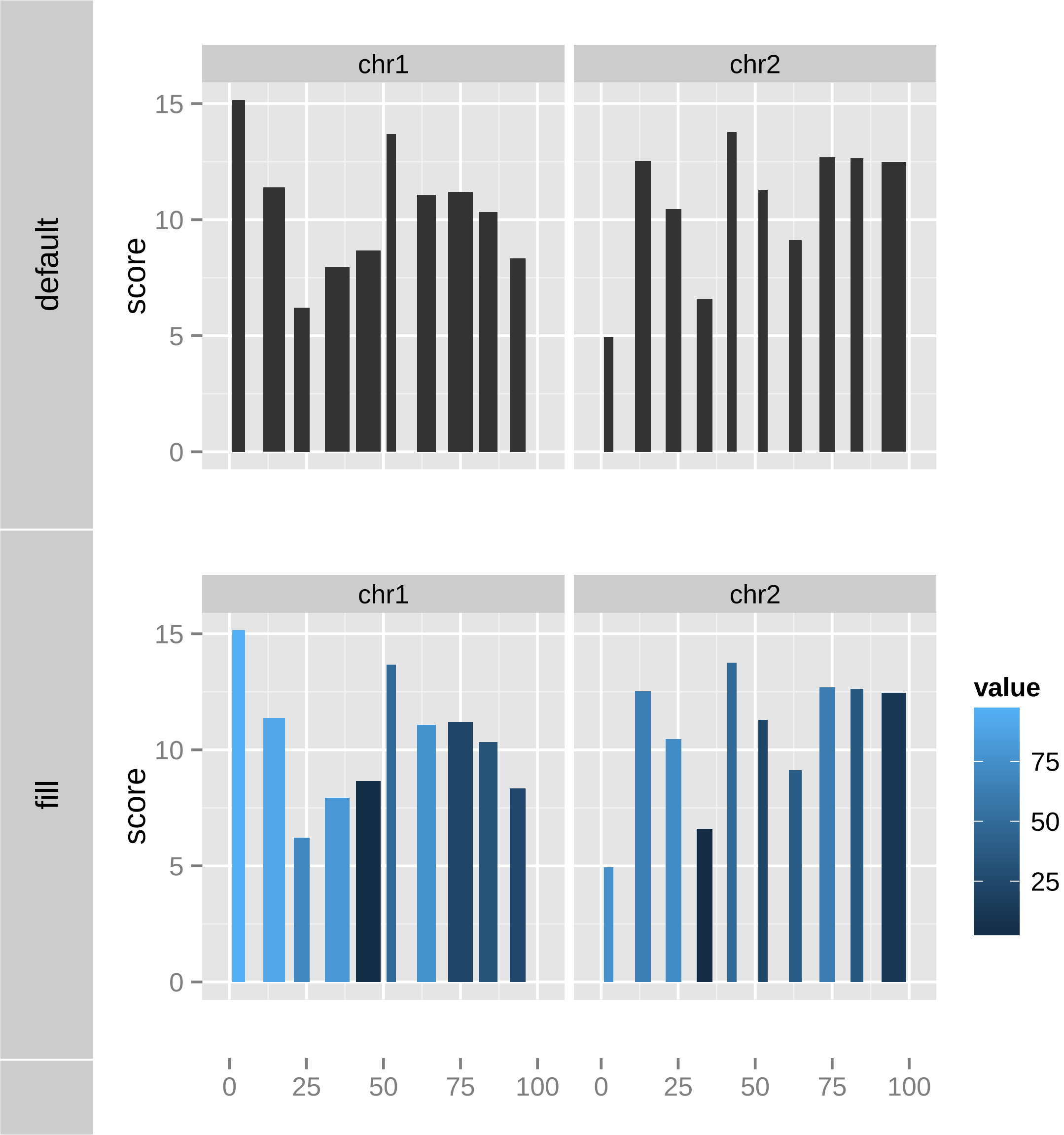
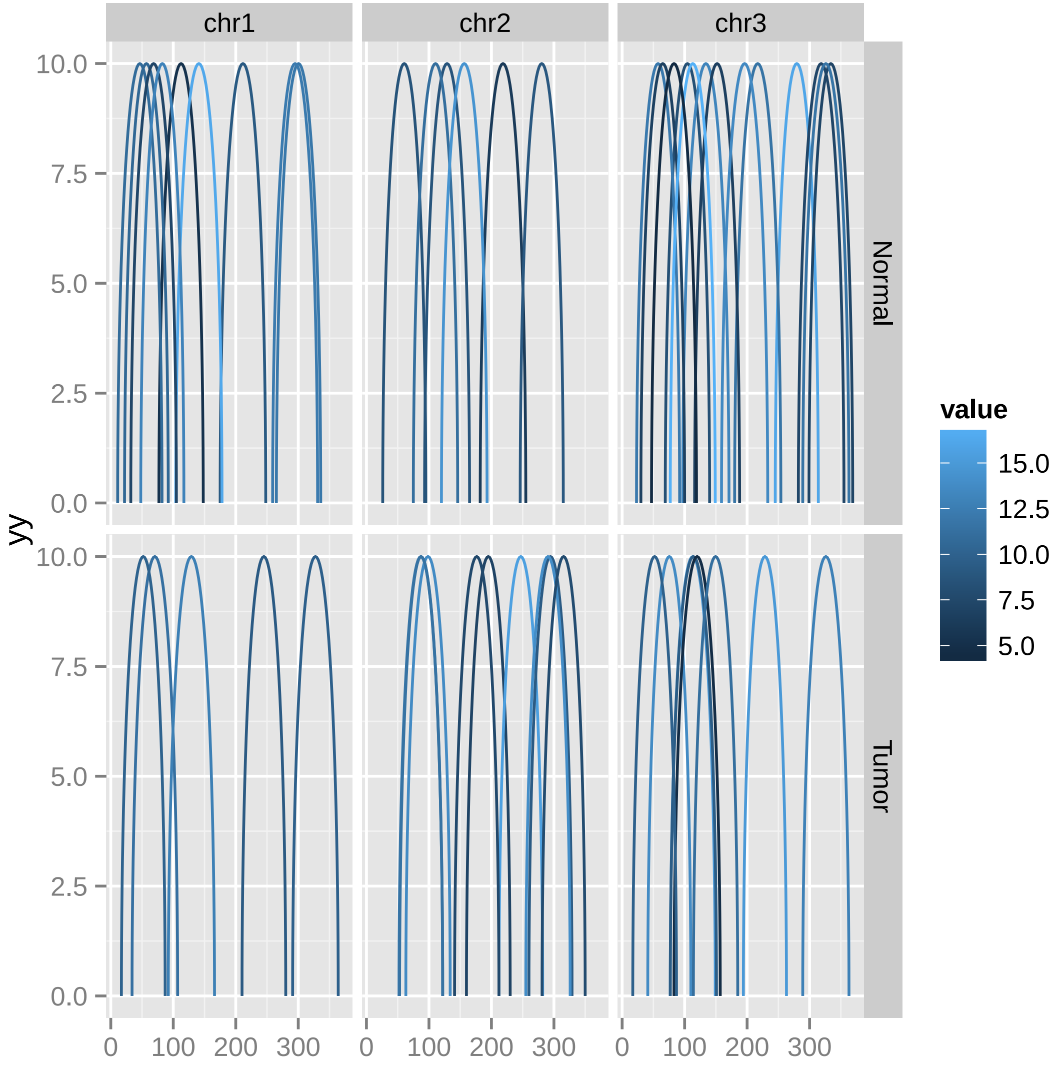


Figure 6.2: Bar geom for GRanges.



```
## NULL
```

Faceted by strand help you understand coverage from different sequencing direction.

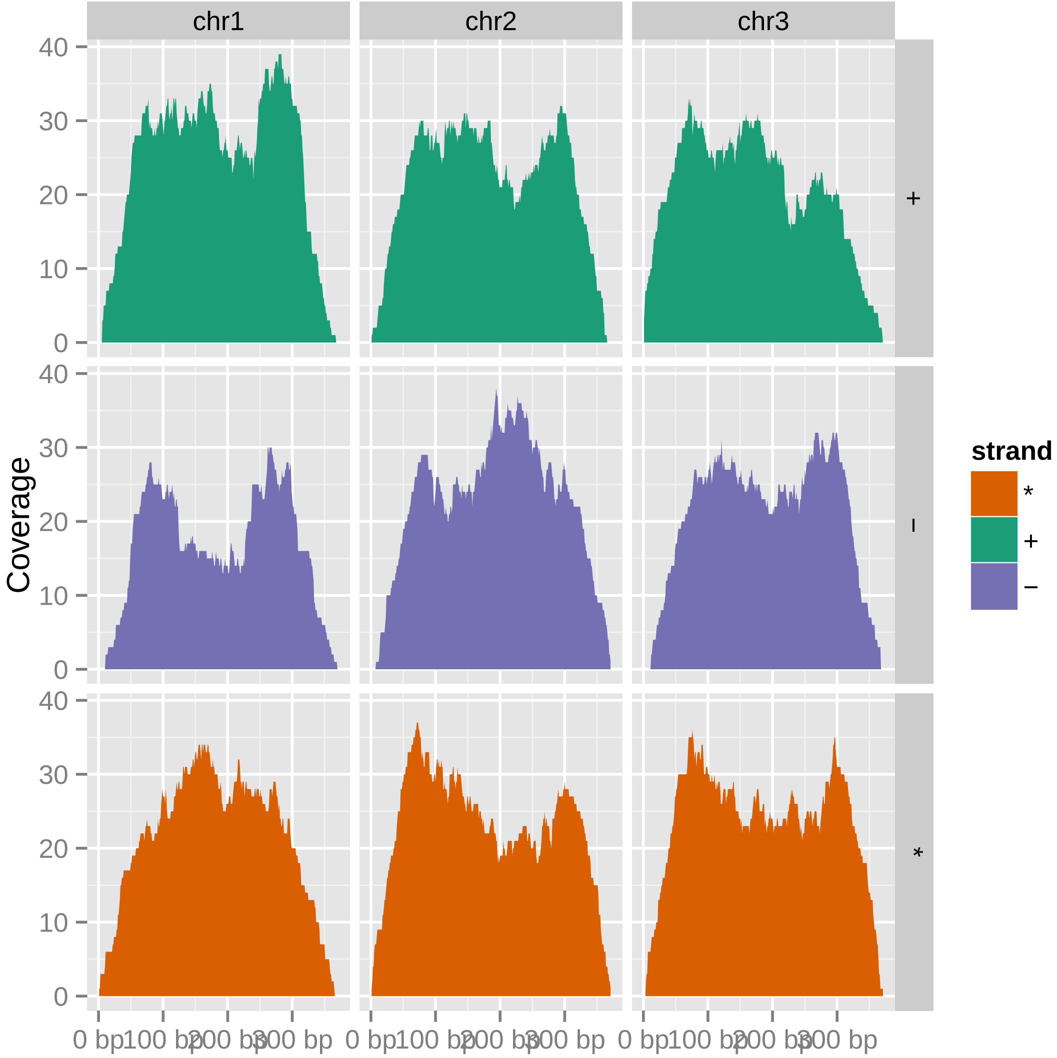
New coordinate transformation 'genome' will transform a `GRanges` object into a genome space, align them up based on 'seqlevel' orders. This transformation allows you to add 'seqlengths' to your `GRanges` object to produce a fixed width. and add buffer in between by specifying `space.skip`. This transformation is useful for grand linear view as Manhattan plot or circular view.

Please read another two vignette about how to plot Manhattan plot and generate circular view for detail.

A little more

You will find a more general tutorial for circular view in chapter 9

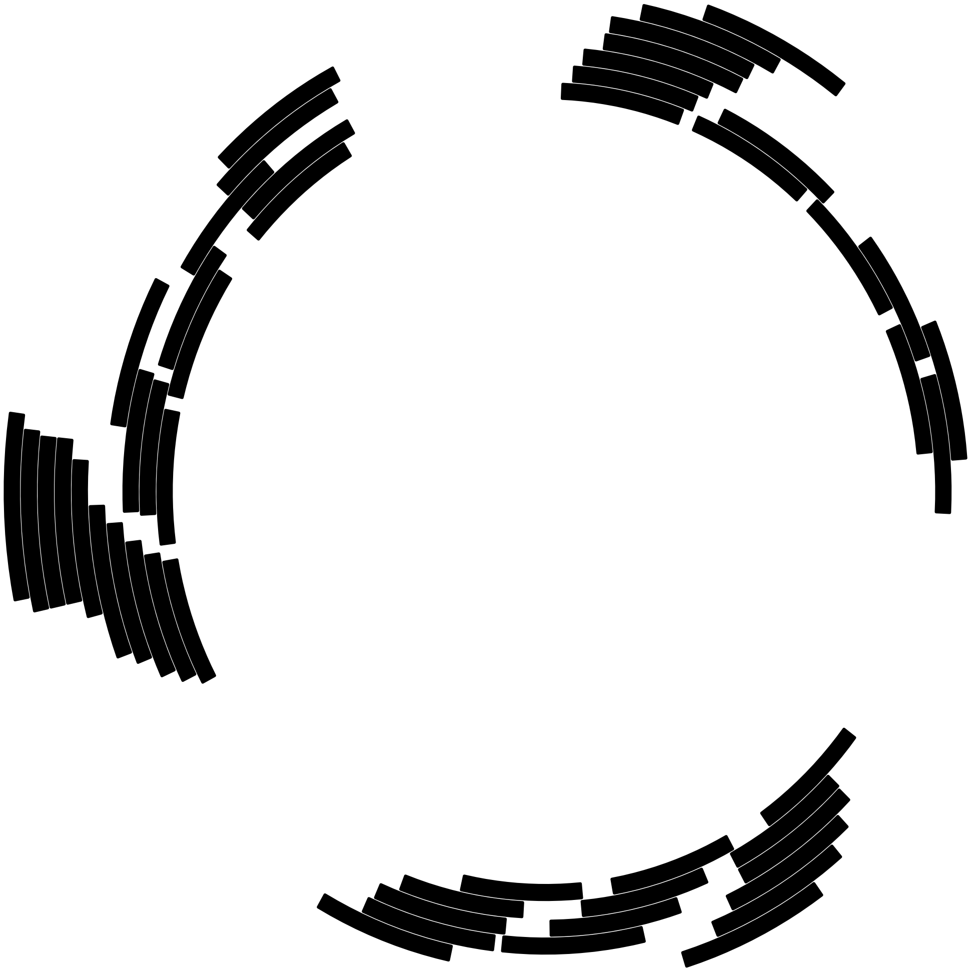
```
autoplot(gr, stat = "coverage", geom = "area", facets = strand ~ seqnames,
    aes(fill = strand))

## Object of class "ggbio"
```



```
## NULL
```

Figure 6.3: Facet by strand to show coverage.

```
autoplot(gr[idx], layout = "circle")

## Object of class "ggbio"
```



```
## NULL
```

Figure 6.4: minimal example for circular transformation.

```
seqlengths(gr) <- c(400, 500, 700)
values(gr)$to.gr <- gr[sample(1:length(gr), size = length(gr))]
idx <- sample(1:length(gr), size = 50)
gr <- gr[idx]
ggplot() + layout_circle(gr, geom = "ideo", fill = "gray70", radius = 7,
    trackWidth = 3) + layout_circle(gr, geom = "bar", radius = 10, trackWidth = 4,
    aes(fill = score, y = score)) + layout_circle(gr, geom = "point", color = "red",
    radius = 14, trackWidth = 3, grid = TRUE, aes(y = score)) + layout_circle(gr,
    geom = "link", linked.to = "to.gr", radius = 6, trackWidth = 1)
```

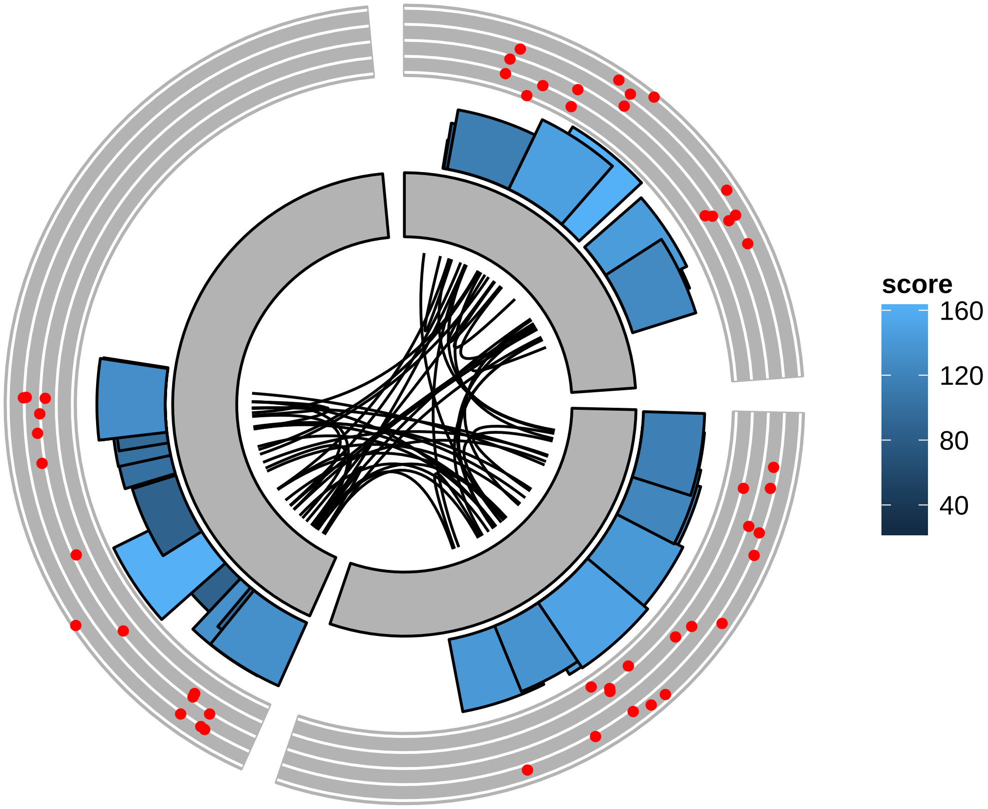


Figure 6.5: Circular layout minimal example.

6.2.2 autoplot,Seqinfo

When a GRanges has seqlengths information which defined chromomsome lengths, we have a way to quickly give a karyogram overview for adding data on later. Please read another vignette about karyogram overview.

You can easily subset/re-order the visualized the chromosomes by using '[' method.

```
data(hg19Ideogram, package = "biovizBase")
sq <- seqinfo(hg19Ideogram)
sq


## Seqinfo of length 93
## seqnames             seqlengths isCircular genome
## chr1                  249250621       <NA>   hg19
## chr1_gl000191_random     106433       <NA>   hg19
## chr1_gl000192_random     547496       <NA>   hg19
## chr2                  243199373       <NA>   hg19
## chr3                  198022430       <NA>   hg19
## chr4                  191154276       <NA>   hg19
## chr4_ctg9_hap1           590426       <NA>   hg19
## chr4_gl000193_random     189789       <NA>   hg19
## chr4_gl000194_random     191469       <NA>   hg19
## ...                         ...        ...    ...
## chrUn_gl000242            43523       <NA>   hg19
## chrUn_gl000243            43341       <NA>   hg19
## chrUn_gl000244            39929       <NA>   hg19
## chrUn_gl000245            36651       <NA>   hg19
## chrUn_gl000246            38154       <NA>   hg19
## chrUn_gl000247            36422       <NA>   hg19
## chrUn_gl000248            39786       <NA>   hg19
## chrUn_gl000249            38502       <NA>   hg19
## chrM                      16571       <NA>   hg19
```

6.2.3 autoplot,IRanges

autoplot for IRanges is used to visualize simple interval data with element data together, it's almost identical to API for *GRanges*, actually, everything works for *GRanges* should work for *IRanges*, we simply turn it to a fake *GRanges* inside.

Let's generate some simulated interval data and store it as *IRanges* object. and add some element meta data.

```
set.seed(1)
N <- 100
ir <- IRanges(start = sample(1:300, size = N, replace = TRUE), width = sample(70:75,
    size = N, replace = TRUE))
## add meta data
df <- DataFrame(value = rnorm(N, 10, 3), score = rnorm(N, 100, 30), sample = sample(c("Normal",
    "Tumor"), size = N, replace = TRUE), pair = sample(letters, size = N,
autoplot(sq[paste0("chr", c(1:22, "X"))])
```

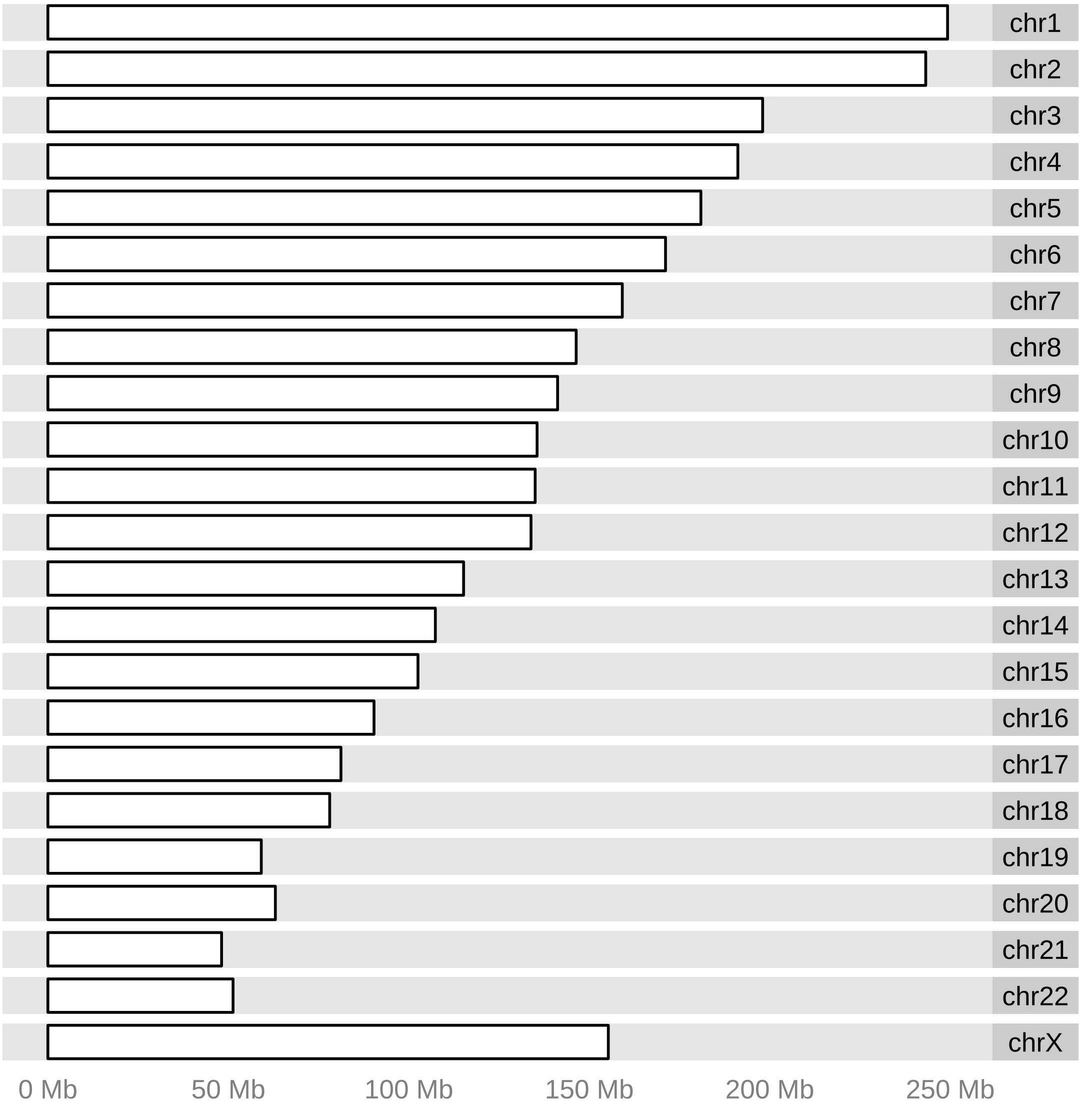


Figure 6.6: Seqinfo visualization for chromosomes 1 to 22 and X.

```
    replace = TRUE))
values(ir) <- df
ir


## IRanges of length 100
##       start end width
## [1]      80 152    73
## [2]     112 183    72
## [3]     172 242    71
## [4]     273 347    75
## [5]      61 133    73
## [6]     270 340    71
## [7]     284 353    70
## [8]     199 270    72
## [9]     189 263    75
## ...     ... ...   ...
## [92]     18  89    72
## [93]    193 262    70
## [94]    263 337    75
## [95]    234 304    71
## [96]    240 312    73
## [97]    137 206    70
## [98]    124 198    75
## [99]    244 314    71
## [100]   182 255    74
```

autoplot will coerce IRanges together with its element meta data, so aesthetics mapping works for those extra information too.

6.2.4 autoplot,GRangesList

GRangesList is most suitable data structure for storing a set of genomic features, for example, exons/utrs in a gene. ‘autoplot‘ is designed to consider the native grouping information in this structure and automatically showing gaps within group in ‘geom‘ *alignment* and make sure grouped items are shown together on the same level with nothing falling in between.

Argument range.geom and gap.geom control geometry for entities and gaps computed for them. group.selfish help you put grouped items in unique y levels and show the y labels for group names.

Let’s create a GRangesList object by splitting a GRanges object.

```
set.seed(1)
N <- 100
##
## ======================================================================
## simmulated GRanges
## ======================================================================
gr <- GRanges(seqnames = sample(c("chr1", "chr2", "chr3"), size = N, replace = TRUE),
    IRanges(start = sample(1:300, size = N, replace = TRUE), width = sample(30:40,
p1 <- autoplot(ir)
p2 <- autoplot(ir, aes(fill = pair)) + opts(legend.position = "none")

 ## 'opts' is deprecated.  Use 'theme' instead.  (Deprecated; last used in version 0.9.1)

p3 <- autoplot(ir, stat = "coverage", geom = "line", facets = sample ~ .)
p4 <- autoplot(ir, stat = "reduce")
tracks(p1, p2, p3, p4)
```

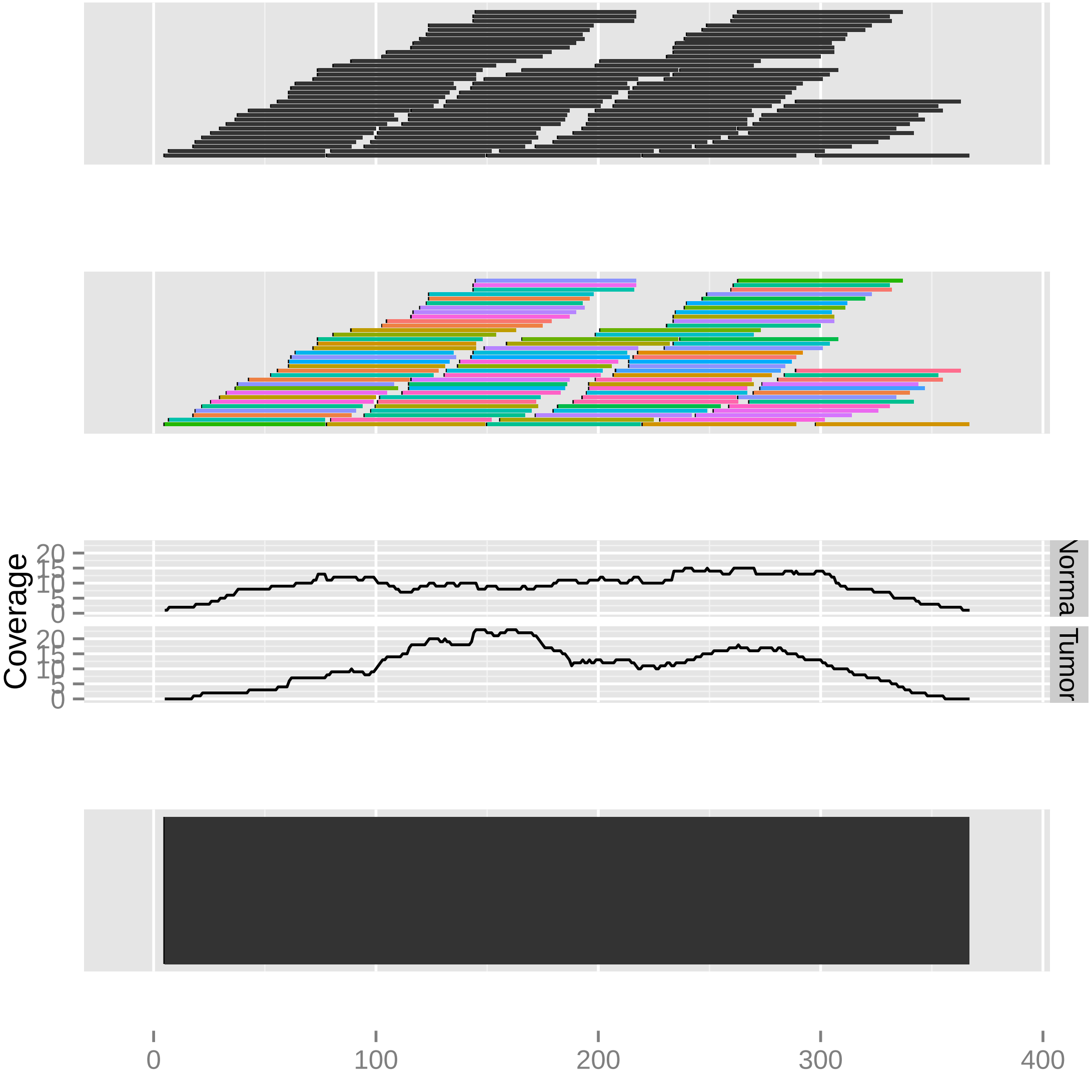


Figure 6.7: IRanges visualization.

```
        size = N, replace = TRUE)), strand = sample(c("+", "-", "*"), size = N,
        replace = TRUE), value = rnorm(N, 10, 3), score = rnorm(N, 100, 30),
    sample = sample(c("Normal", "Tumor"), size = N, replace = TRUE), pair = sample(letters,
        size = N, replace = TRUE))


grl <- split(gr, values(gr)$pair)
```

For GRangesList object, default is coerce it to GRanges and adding extra column to preserve the grouping information. main geoms and gaps geom are separately controlled.

Internal variable grl_name added to keep a track for grouping information, you could use it for faceting or other aesthetic mapping, the variables could be renamed by indName argument in autoplot, you could pass either ..grl_name.. or grl_name in the mapping, I prefer the first one, it tells that it's interval variables.

6.2.5 autoplot,Rle

Rle is a general container for storing atomic vector which is defined in package IRanges, data is stored in a run-length encoding format.

For Rle, we bring following method, three stat, two geom and four types.

Two geom

- **bar**: default, controlled by 'nbin'.
- **heatmap**: show Rle as heatmap, use color to indicate values, controlled by 'nbin'.

Three default statistical transformation

- **bin**: bin the object, default is 30 bins in the view, controlled by argument nbin. Then in each bin make summary against specified types.
- **identity**: transform data to raw vector, then you can use many other geom such as line or point. Default x and y is internally set to position and value.
- **slice**: use lower to slice the object to islands, then use bar or heatmap to represent the island.

Four types for compute the statistical summary.

- **viewSums**: sums in the sliced view or bins.
- **viewMins**: min values in the sliced view or bins.
- **viewMaxs**: max values in the sliced view or bins.
- **viewMeans**: mean values in the sliced view or bins.

Let's simulate some data first.

```
## default gap.geom is 'chevron'
p1 <- autoplot(grl, group.selfish = TRUE)
p2 <- autoplot(grl, group.selfish = TRUE, main.geom = "arrowrect", gap.geom = "segment")
tracks(p1, p2)
```

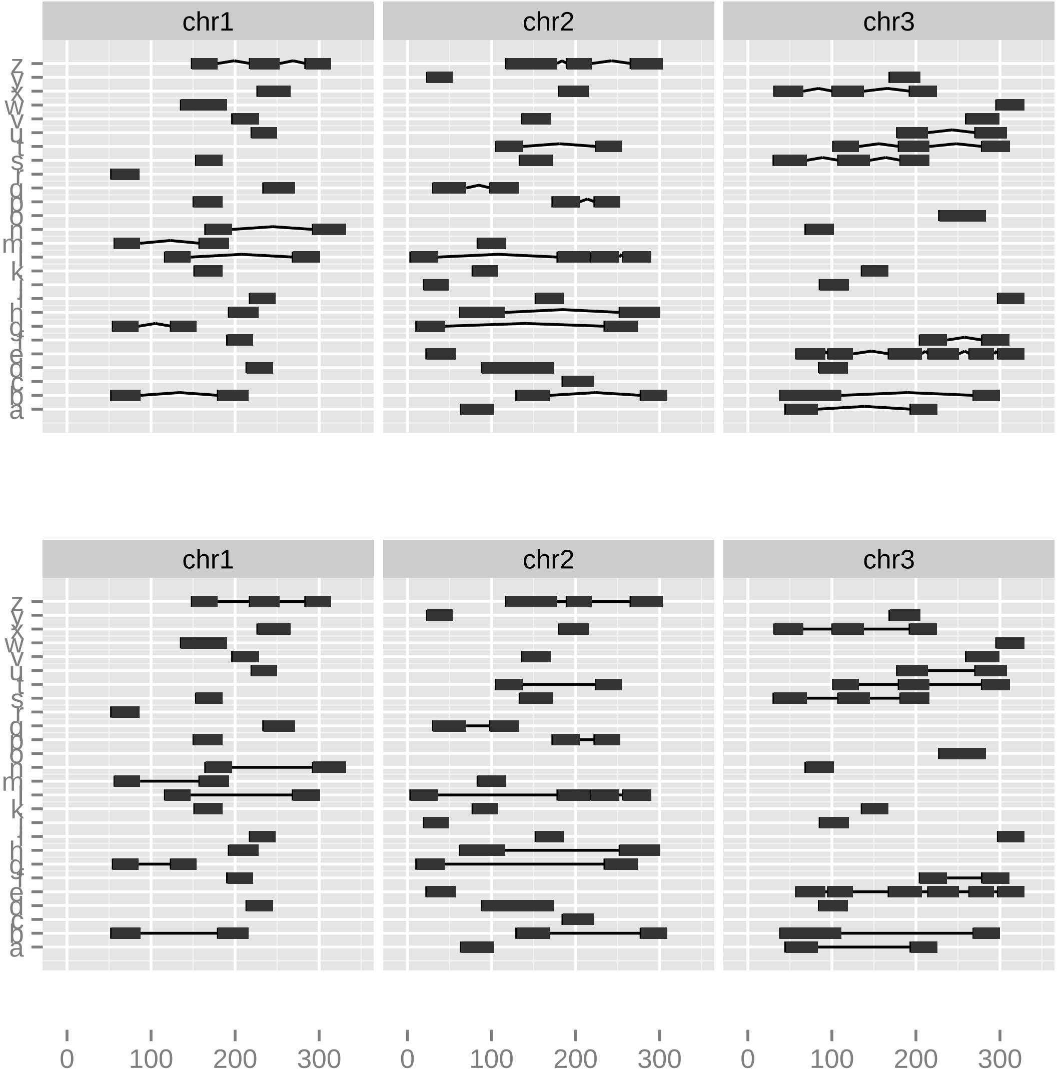
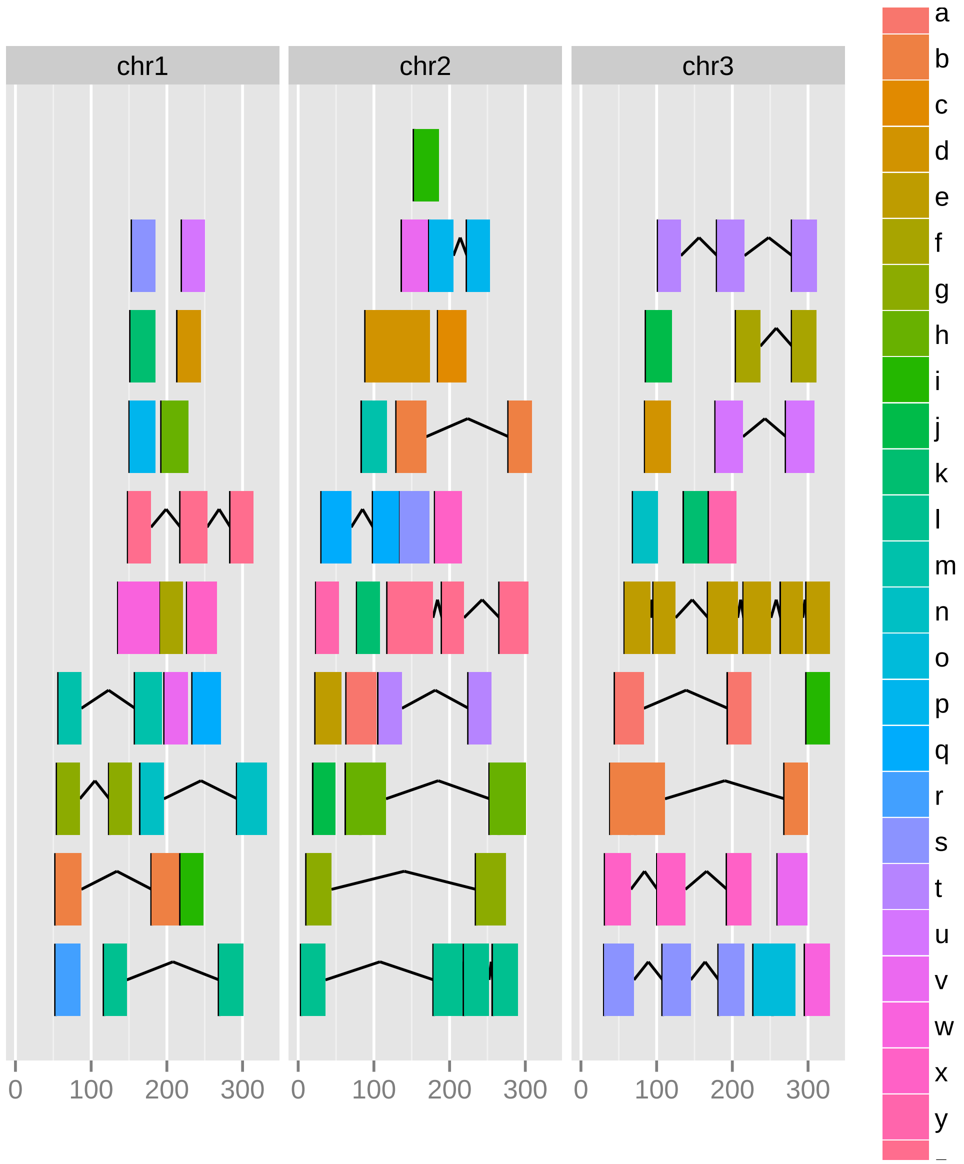


Figure 6.8: Some examples showing GRangesList

```
autoplot(grl, aes(fill = ..grl_name..))

## Object of class "ggbio"
```



```
## NULL

## equal to autoplot(grl, aes(fill = grl_name))
```

Figure 6.9: Tweak with name.

```
library(IRanges)
library(ggbio)
set.seed(1)
lambda <- c(rep(0.001, 4500), seq(0.001, 10, length = 500), seq(10, 0.001,
    length = 500))

## @knitr create
xVector <- rpois(10000, lambda)
xRle <- Rle(xVector)
xRle


## numeric-Rle of length 10000 with 823 runs
##   Lengths:  779    1  208    1 1599    1 ...    5    2    9    1 4507
##   Values :    0    1    0    1    0    1 ...    0    1    0    1    0
```

6.2.6 autoplot,RleList

All methods are the same for `RleList` as for `Rle`, it's just faceted by listed group automatically. Please read the autoplot,Rle section first.

Let's simulate some data first.

```
xRleList <- RleList(xRle, 2L * xRle)
xRleList


## SimpleRleList of length 2
## [[1]]
## numeric-Rle of length 10000 with 823 runs
##   Lengths:  779    1  208    1 1599    1 ...    5    2    9    1 4507
##   Values :    0    1    0    1    0    1 ...    0    1    0    1    0
##
## [[2]]
## numeric-Rle of length 10000 with 823 runs
##   Lengths:  779    1  208    1 1599    1 ...    5    2    9    1 4507
##   Values :    0    2    0    2    0    2 ...    0    2    0    2    0
```

6.2.7 autoplot,TranscriptDb

Some simple demonstration:

```
library(TxDb.Hsapiens.UCSC.hg19.knownGene)
data(genesymbol, package = "biovizBase")
txdb <- TxDb.Hsapiens.UCSC.hg19.knownGene
```

```
p1 <- autoplot(xRle)
                                ## Default use binwidth:  range/30
p2 <- autoplot(xRle, nbin = 80)
                                ## Default use binwidth:  range/80
p3 <- autoplot(xRle, geom = "heatmap", nbin = 200)
                                ## Default use binwidth:  range/200
tracks(`nbin = 30` = p1, `nbin = 80` = p2, `nbin = 200(heatmap)` = p3)
```

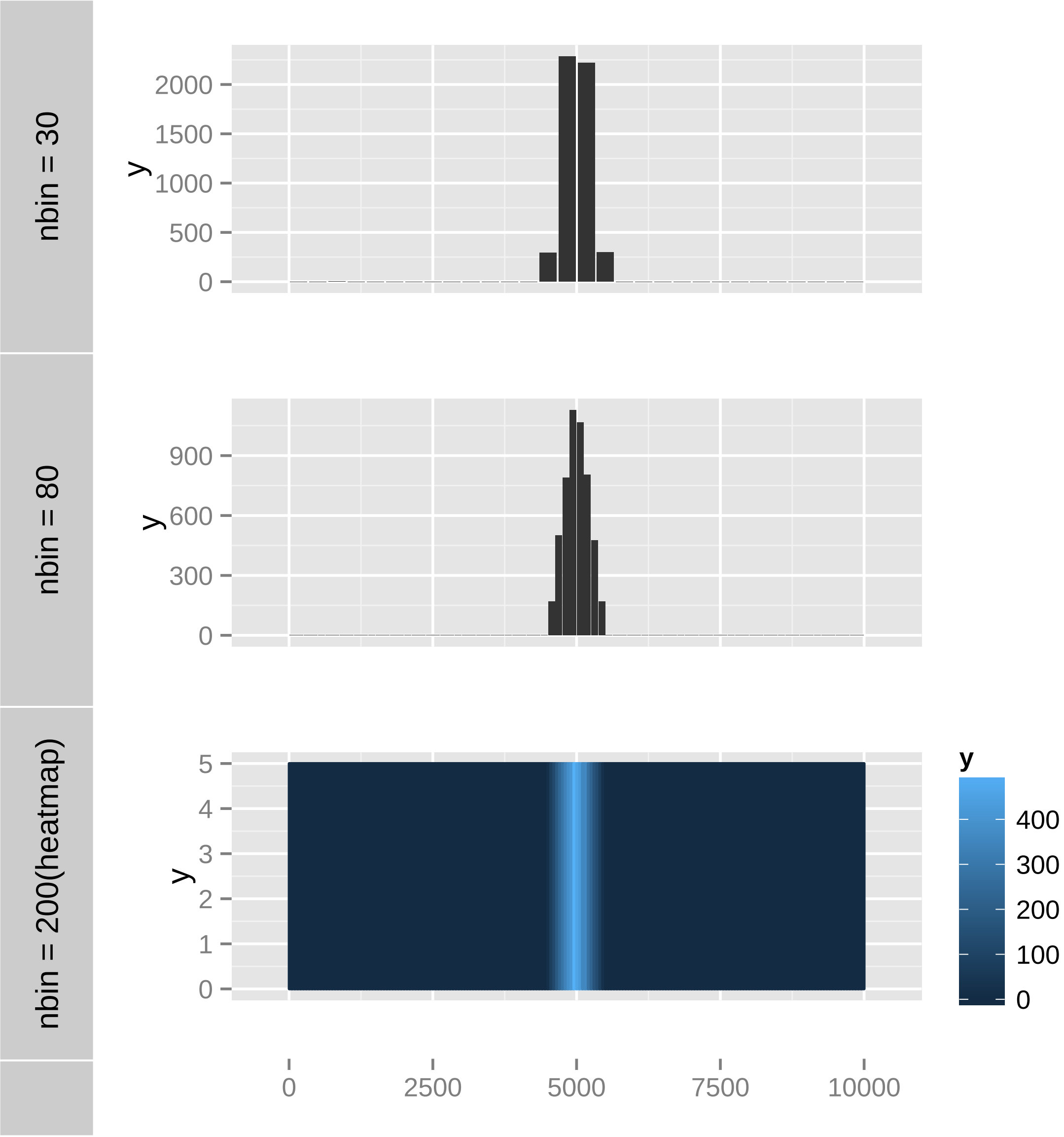


Figure 6.10: Compare different geom and nbin by using default bin stat.

```
p1 <- autoplot(xRle, stat = "identity")
p2 <- autoplot(xRle, stat = "identity", geom = "point", color = "red")
tracks(line = p1, point = p2)
```

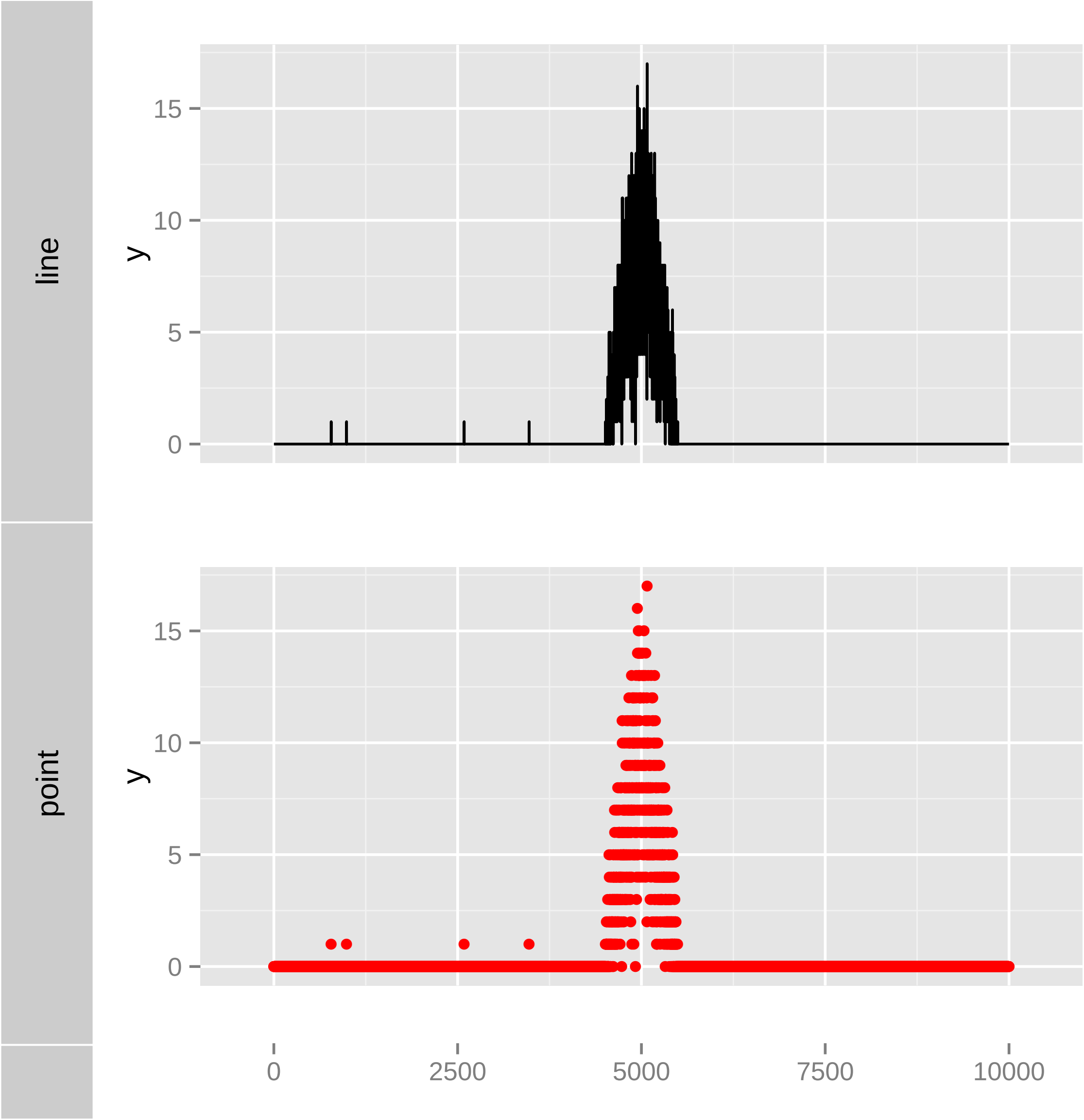


Figure 6.11: Compare different geom and nbin by using stat identity.

```
p1 <- autoplot(xRle, type = "viewMaxs", stat = "slice", lower = 5)
p2 <- autoplot(xRle, type = "viewMaxs", stat = "slice", lower = 5, geom = "heatmap")
tracks(bar = p1, heatmap = p2)
```

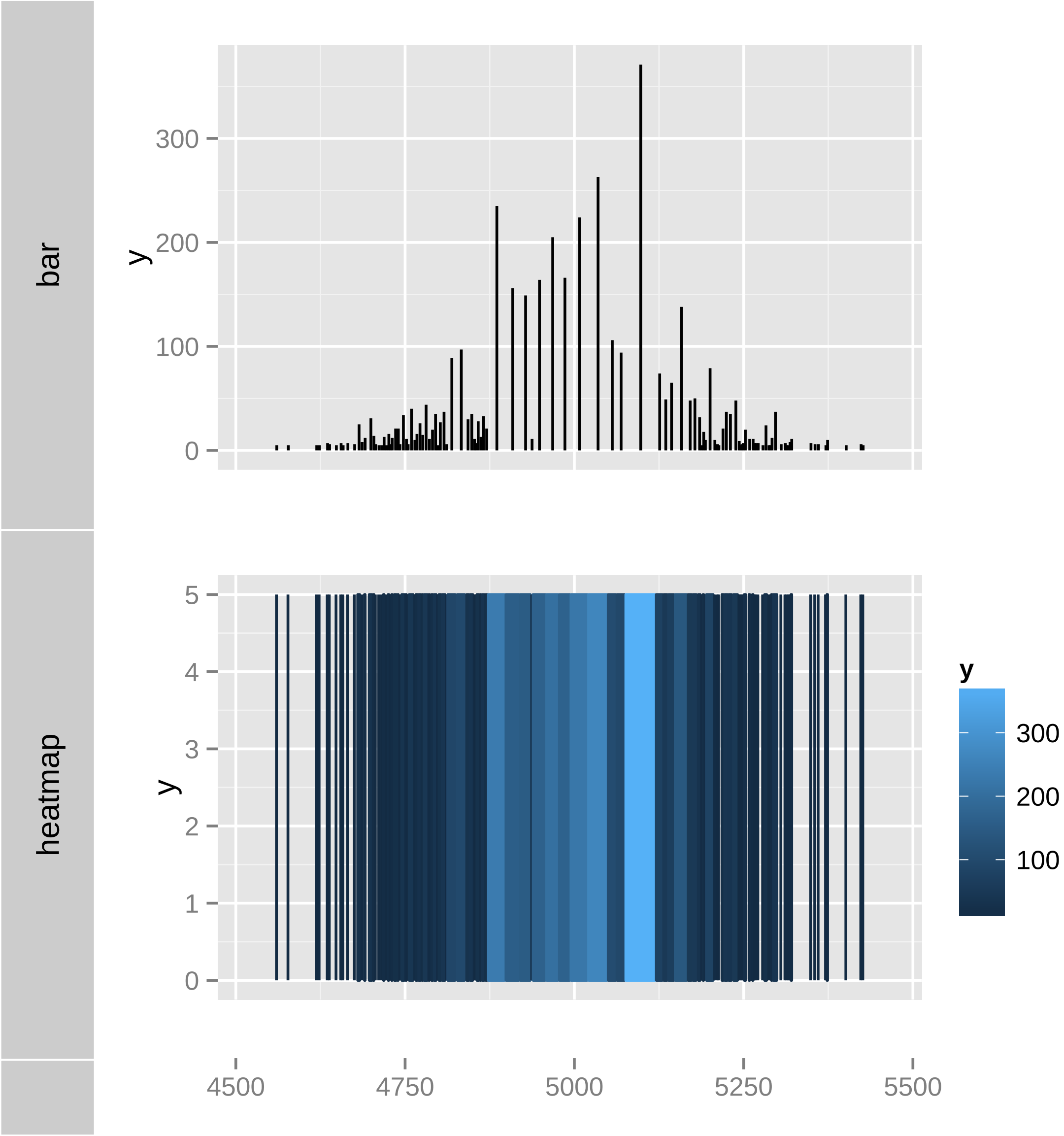


Figure 6.12: Compare different geom and nbin by using stat slice.

```
p1 <- autoplot(xRleList)

                              ## Default use binwidth:  range/30

p2 <- autoplot(xRleList, nbin = 80)

                              ## Default use binwidth:  range/80

p3 <- autoplot(xRleList, geom = "heatmap", nbin = 200)

                              ## Default use binwidth:  range/200

tracks(`nbin = 30` = p1, `nbin = 80` = p2, `nbin = 200(heatmap)` = p3)
```

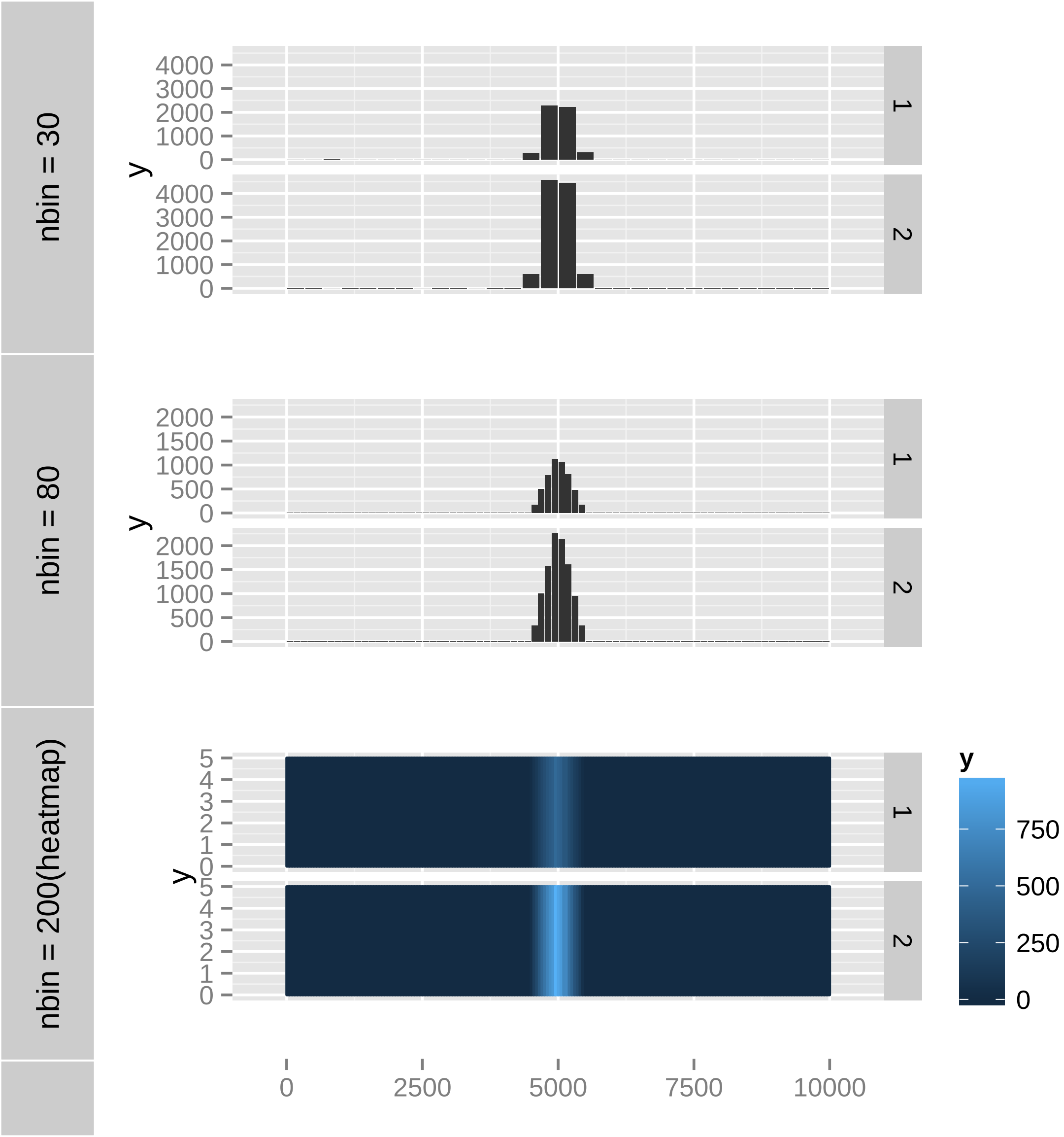


Figure 6.13: Compare different geom and nbin by using default bin stat.

```
p1 <- autoplot(xRleList, stat = "identity")
p2 <- autoplot(xRleList, stat = "identity", geom = "point", color = "red")
tracks(line = p1, point = p2)
```

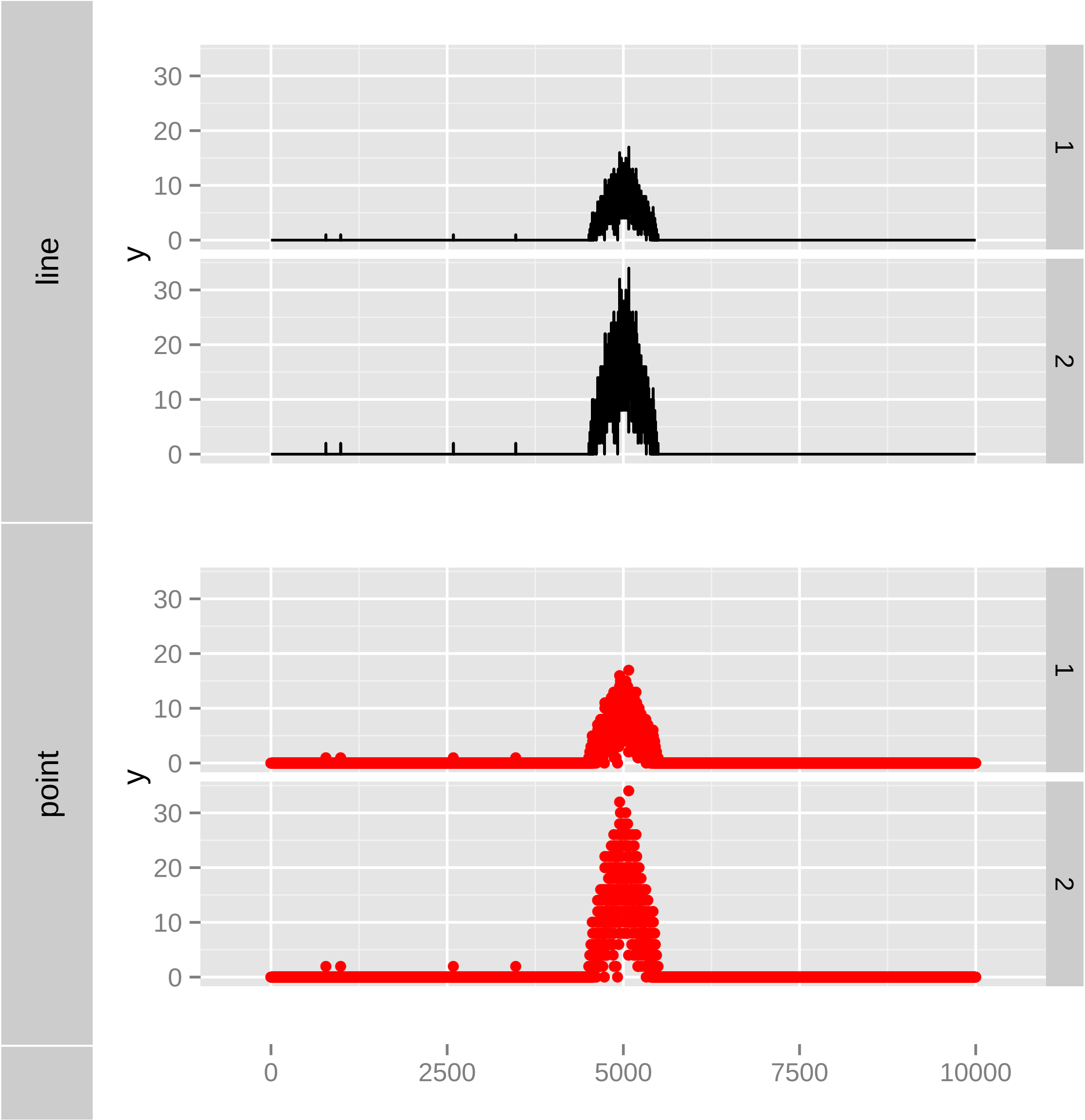


Figure 6.14: Compare different geom and nbin by using stat identity.

```
p1 <- autoplot(xRleList, type = "viewMaxs", stat = "slice", lower = 5)
p2 <- autoplot(xRleList, type = "viewMaxs", stat = "slice", lower = 5, geom = "heatmap")
tracks(bar = p1, heatmap = p2)
```

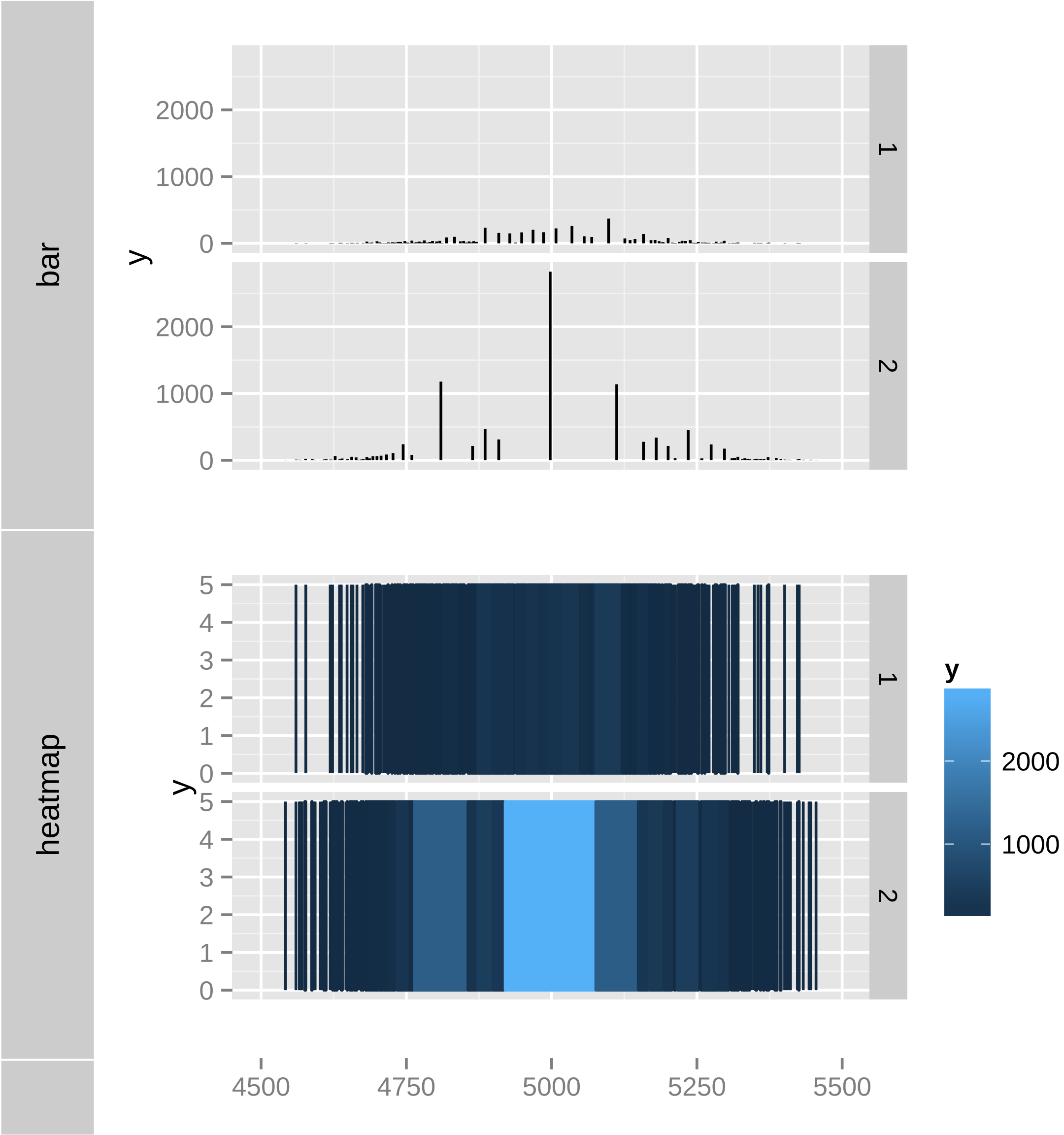


Figure 6.15: Compare different geom and nbin by using stat slice.

which argument accept a GRanges, list which is **required** to subset the data. names.expr accept string pattern or expression to parse the y tick labels. Otherwise it's not going to show all of them. We are trying to show the ALDOA gene in the following example.

6.2.8 autoplot,GappedAlignment

The GappedAlignments class is a container to store a set of alignments, which is defined in package *GenomicRanges*.

Let's load some data.

```
library(Rsamtools)

## Loading required package:  Biostrings

data("genesymbol", package = "biovizBase")
bamfile <- system.file("extdata", "SRR027894subRBM17.bam", package = "biovizBase")
## need to set use.names = TRUE
ga <- readBamGappedAlignments(bamfile, param = ScanBamParam(which = genesymbol["RBM17"]),
    use.names = TRUE)
```

Default is to show gapped line, we also could show them as simple short reads and coverage.

6.2.9 autoplot,BamFile

For BamFile, we bring a fast estimated method(implemented by Michael Lawrence), which is suitable for overview for particular chromosome and a much slower raw data view which could be used in visualizing a small region.

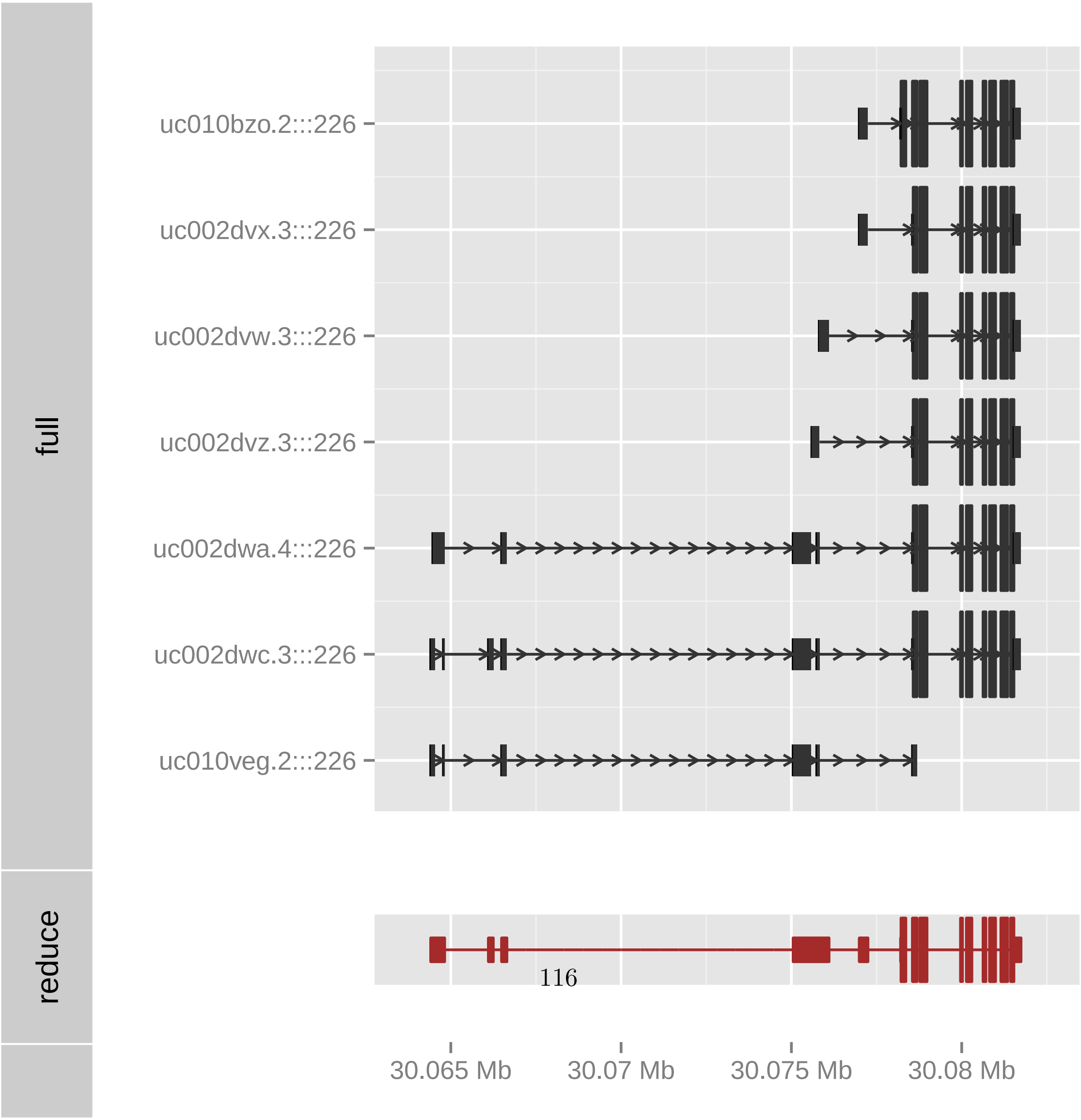
Load some raw data first, we didn't provide an attached data here, you can try to download a whole genome NGS seq file fro ENCODE or somewhere else.

```
library(Rsamtools)
bamfile <- "./wgEncodeCaltechRnaSeqK562R1x75dAlignsRep1V2.bam"
bf <- BamFile(bamfile)
```

A very efficient method called 'estimate', which argument accepted a chromsome names, will give you an overview about coverage. If multiple chromosome names are provided, it will be faceted by seqnames. If which is missing, it's going to use the first chromosomes appeared in the header.

```
autoplot(bamfile)
autoplot(bamfile, which = c("chr1", "chr2"))
autoplot(bf)
autoplot(bf, which = c("chr1", "chr2"))

data(genesymbol, package = "biovizBase")
autoplot(bamfile, method = "raw", which = genesymbol["ALDOA"])
```

```
p1 <- autoplot(txdb, which = genesymbol["ALDOA"], names.expr = "tx_name:::gene_id")

## Aggregating TranscriptDb...
## Parsing exons...
## Parsing cds...
## Parsing transcripts...
## Aggregating...
## Done
## Constructing graphics...

p2 <- autoplot(txdb, which = genesymbol["ALDOA"], stat = "reduce", color = "brown",
    fill = "brown")

## Aggregating TranscriptDb...
## Parsing exons...
## Parsing cds...
## Parsing transcripts...
## Aggregating...
## Done
## Constructing graphics...

tracks(full = p1, reduce = p2, heights = c(5, 1)) + ylab("")
```



```
p1 <- autoplot(ga)
p2 <- autoplot(ga, geom = "rect")

                                ## extracting information...

p3 <- autoplot(ga, geom = "line", stat = "coverage")

                                ## extracting information...

tracks(default = p1, rect = p2, coverage = p3)
```

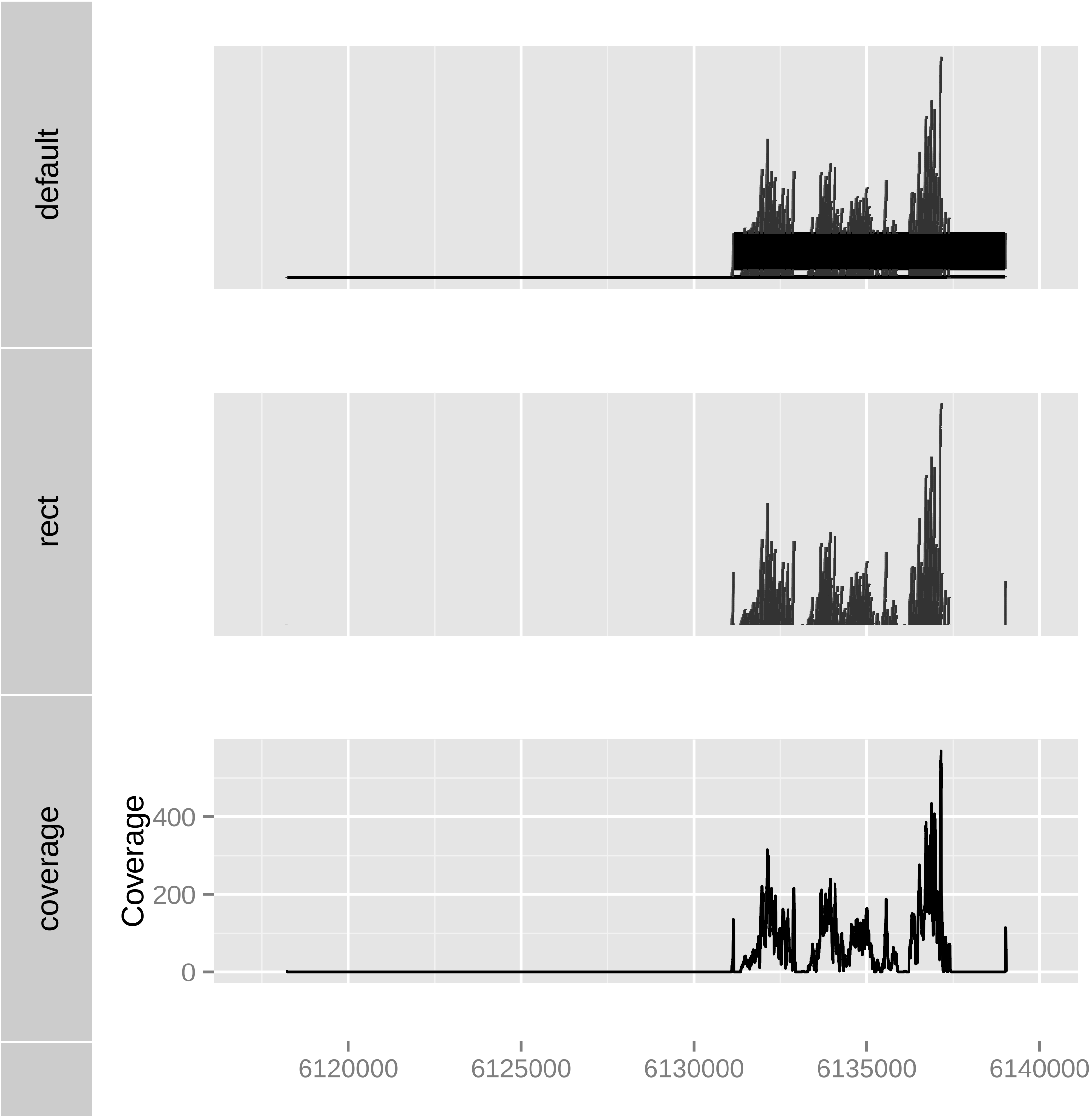


Figure 6.17: Visualization of GappedAlignemnt object

```r
library(BSgenome.Hsapiens.UCSC.hg19)
autoplot(bf, stat = "mismatch", which = genesymbol["ALDOA"], bsgenome = Hsapiens)
```

6.2.10 autoplot,character

When the object is character it accept a file with extensions *.bam* or any other extension names package *rtracklayer* supported, such as *.bed, .gif*. If the object could be imported by *rtracklayer*, it will be turned into a `GRanges` object, and 'score' column will be potentially used. So please read Section 6.2.1 Section 6.2.8 Section 6.2.9 for related topics.

For example, if you have a bam file

```r
bamfile <- "./wgEncodeCaltechRnaSeqK562R1x75dAlignsRep1V2.bam"
autoplot(bamfile)
```

Or for an example bed file, remember you can pass an argument `which` to subset the data.

6.2.11 autoplot,matrix

For object `matrix`, the default graphic would be heatmap, here we bring more controls over it.

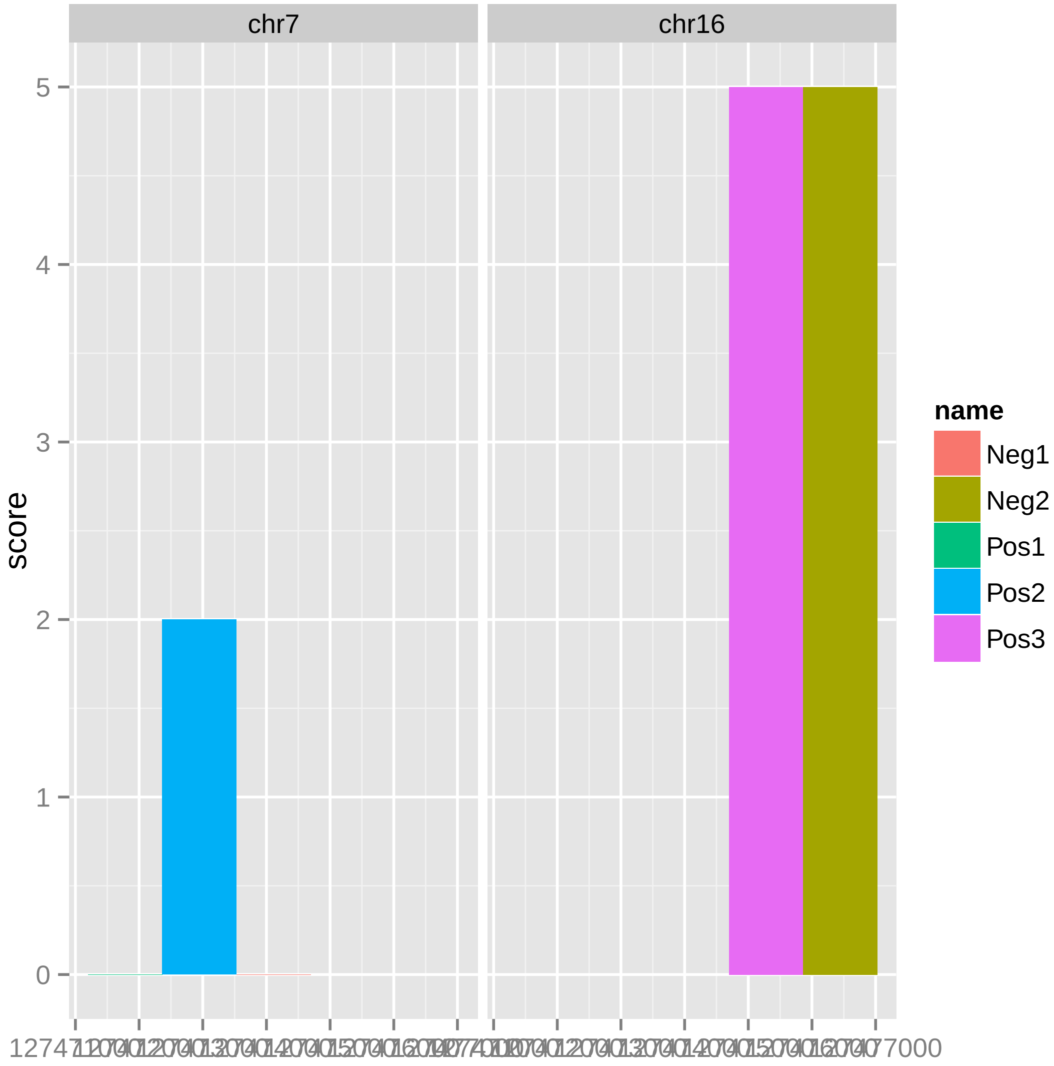
- Function `scale_fill_fold_change`(not default) will scale the heatmap due to a classic blue-white-red color scheme, where 0 is set to white color, negative value set to blue and positive value set to red.

This underlies fundamental heatmap for other object such as `ExpressionSet, SummarizedExperiment, VCF`, which we will introduce later.

We use `volcano` default data as an example, it's not a real microarray data, just demonstrate how to visualize a `matrix`.

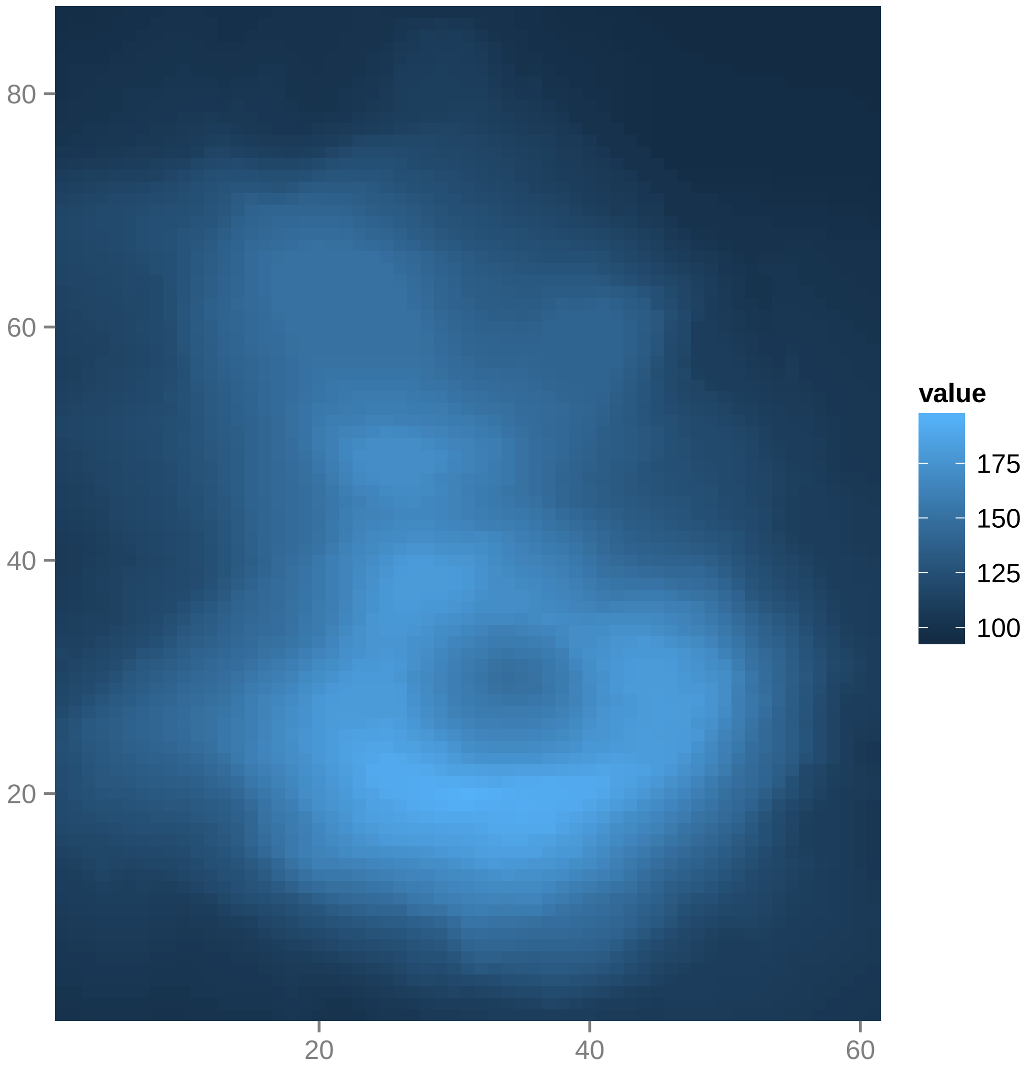
```r
autoplot(volcano)
```

```
library(rtracklayer)
test_path <- system.file("tests", package = "rtracklayer")
test_bed <- file.path(test_path, "test.bed")
autoplot(test_bed, aes(fill = name))

                                                    ## reading in
                                            ## use score as y by default

## Object of class "ggbio"
```



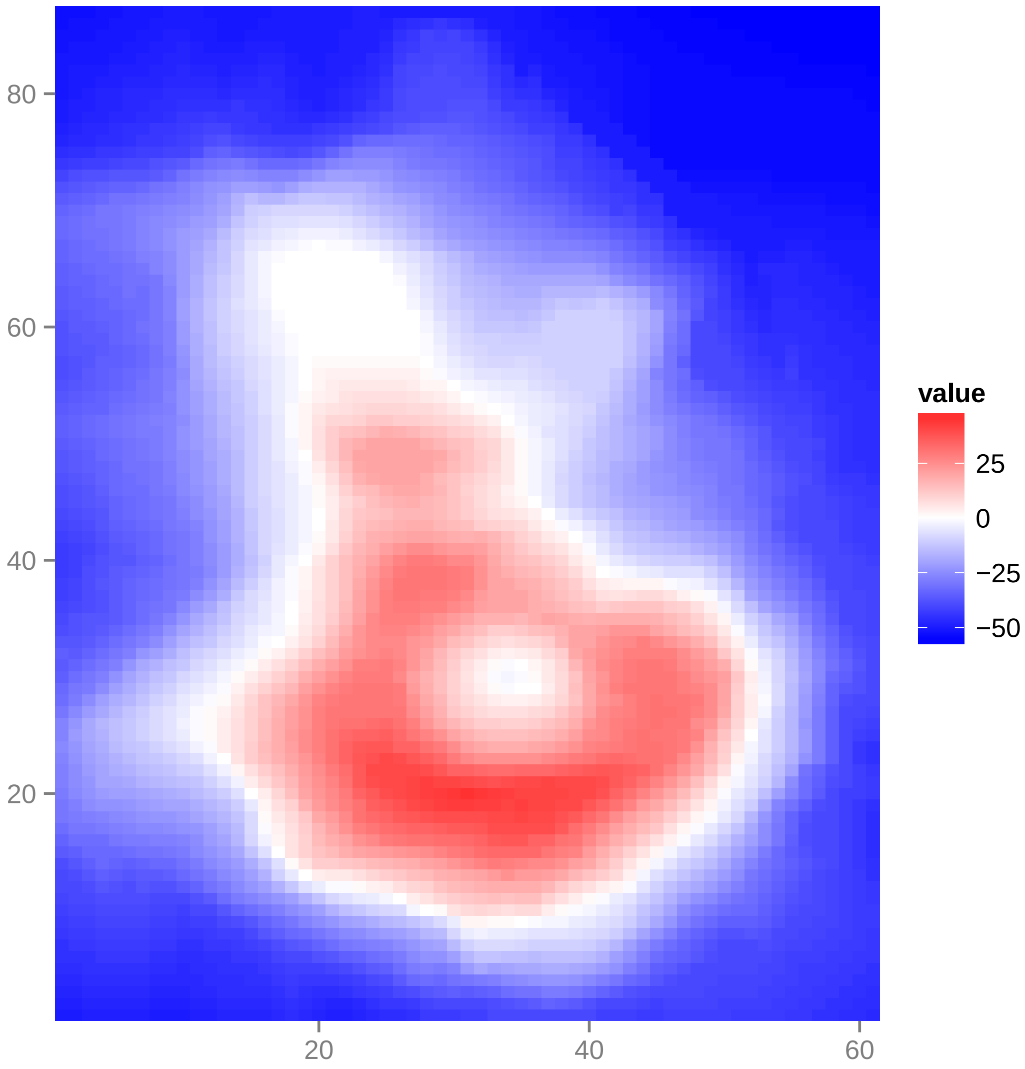
```
## NULL
```

Figure 6.18: autoplot for bed files



In biological papers, a blue-white-scale is commonly used for making heatmap.

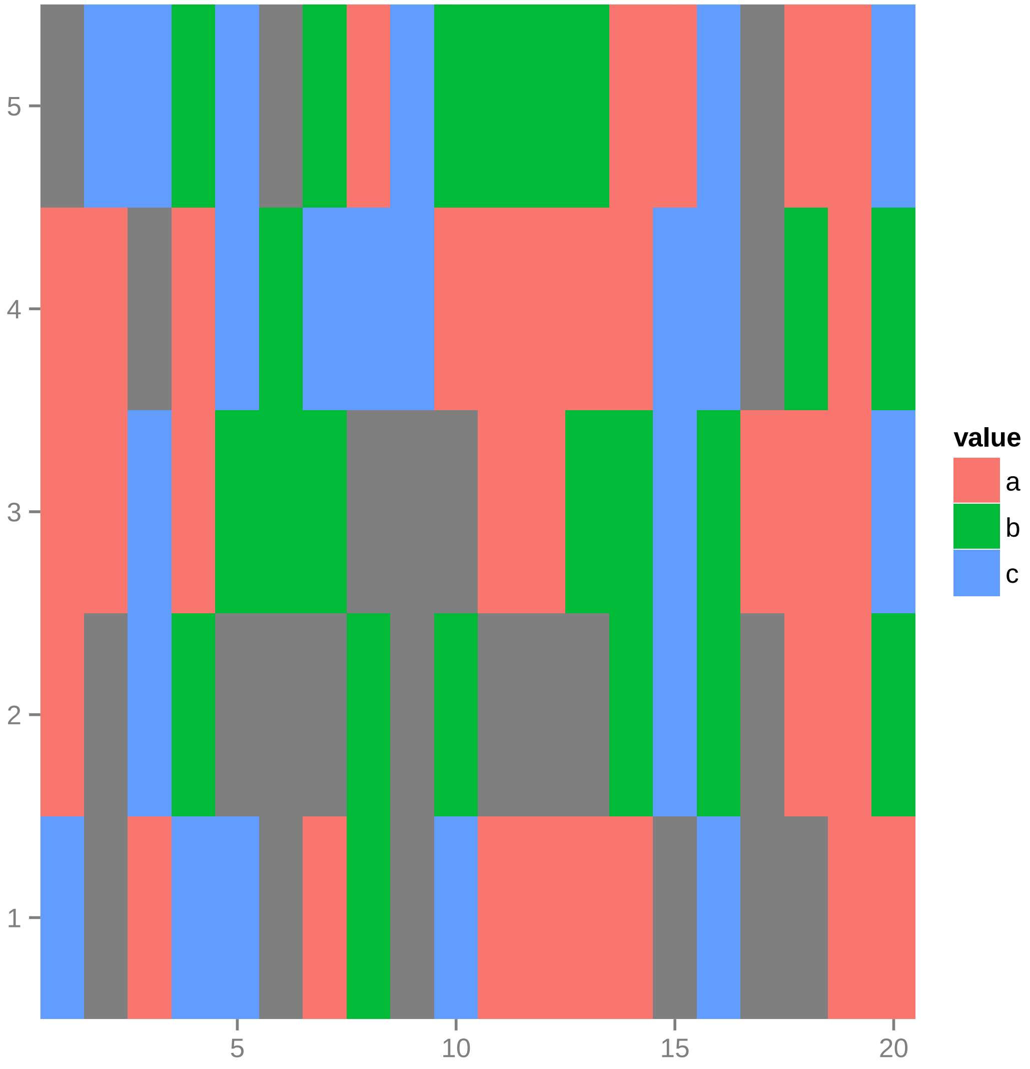
```
autoplot(volcano - 150) + scale_fill_fold_change()
```



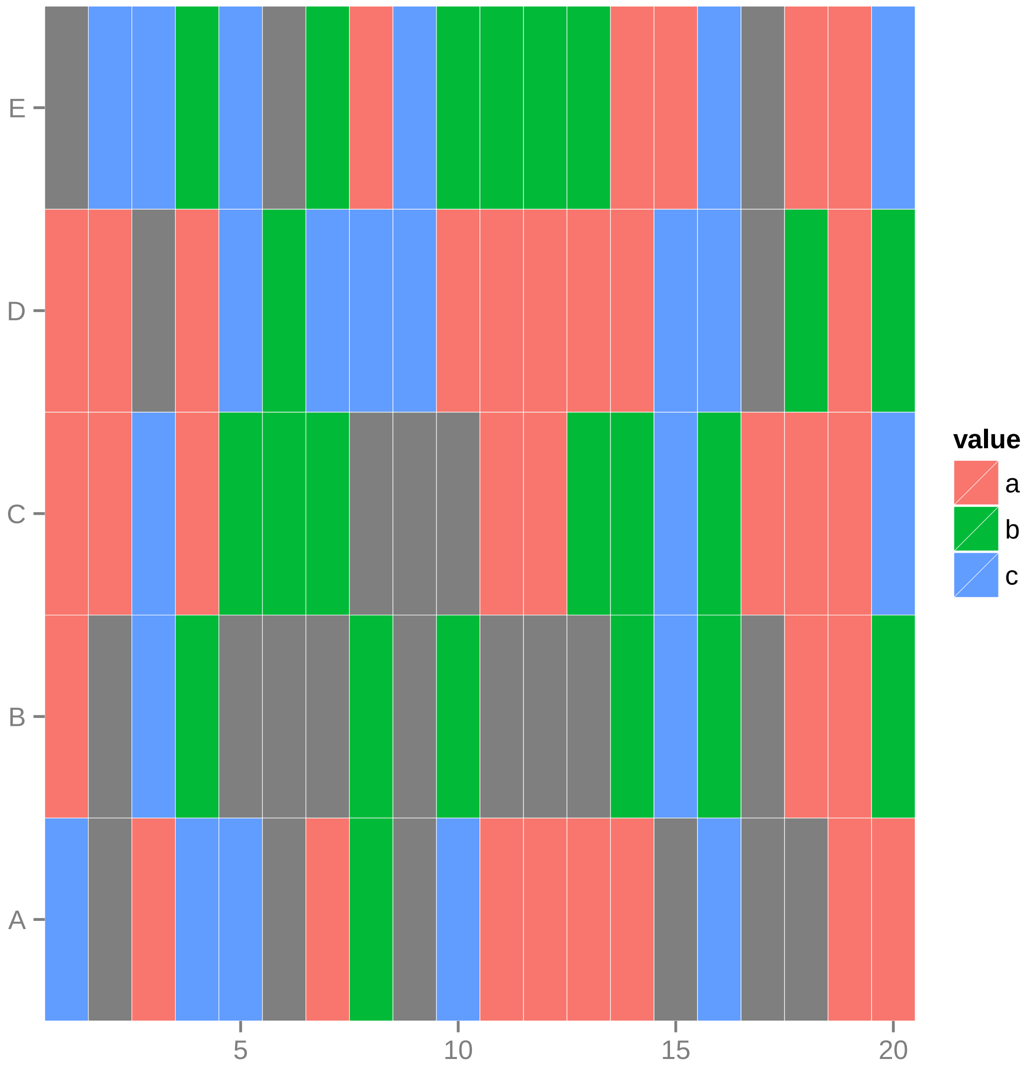
When column name or row name is associated with matrix, they will be labeld, but you can still force disable the label by using logical arguments `colnames.label, rownames.label`.

`autoplot` for *matrix* also support a matrix storing categorical data, even with `NA`, missing value will be shown as gray color by default, by your can explicitly set it to other colors. Default geom for this is 'tile' more flexible, you can specify the height and width for the unit.

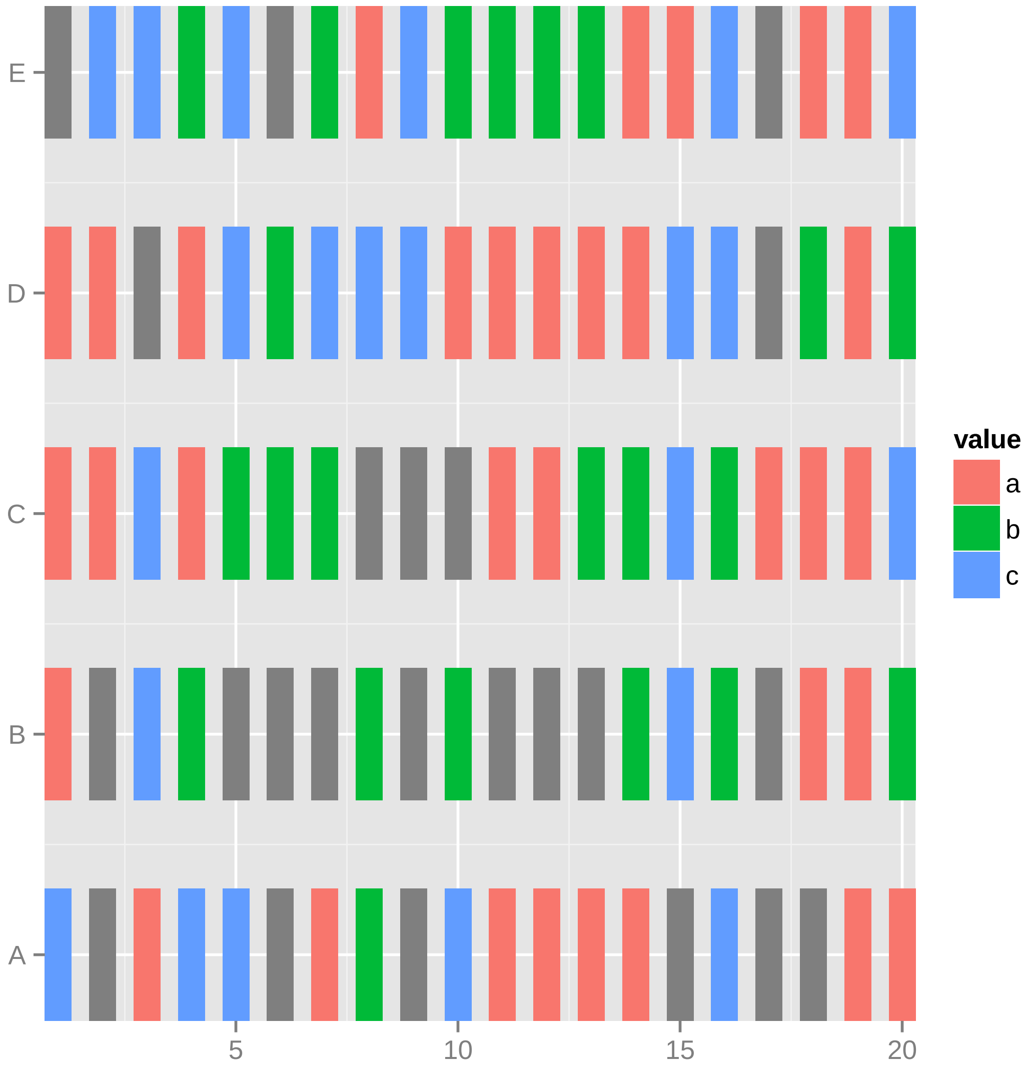
```
x <- sample(c(letters[1:3], NA), size = 100, replace = TRUE)
mx <- matrix(x, nrow = 5)
mx[1:5, 1:5]


##      [,1] [,2] [,3] [,4] [,5]
## [1,] "c"  NA   "a"  "c"  "c"
## [2,] "a"  NA   "c"  "b"  NA
## [3,] "a"  "a"  "c"  "a"  "b"
## [4,] "a"  "a"  NA   "a"  "c"
## [5,] NA   "c"  "c"  "b"  "c"

autoplot(mx)
```



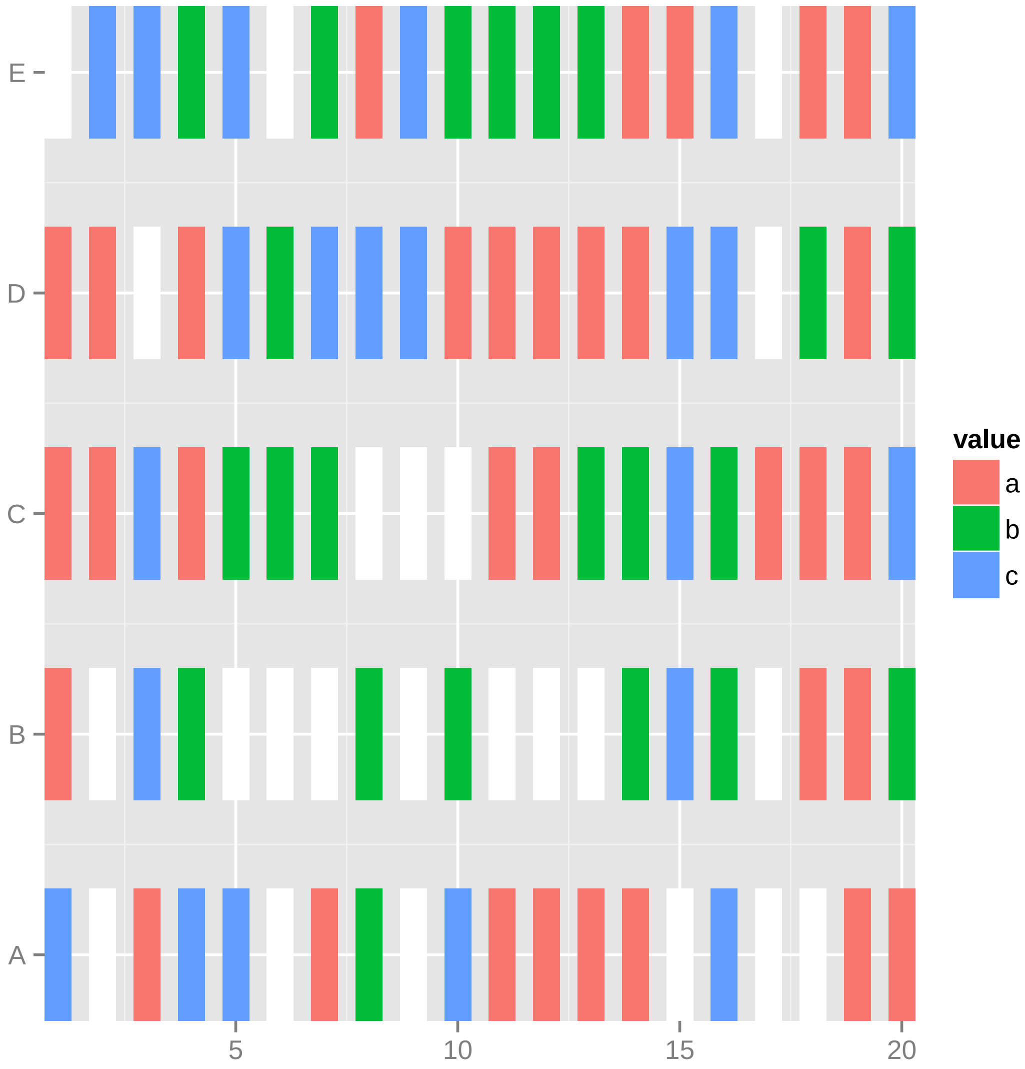
```
## tile gives you a white margin
rownames(mx) <- LETTERS[1:5]
autoplot(mx, color = "white")

## Scale for 'y' is already present.  Adding another scale for 'y', which will replace the
existing scale.
```



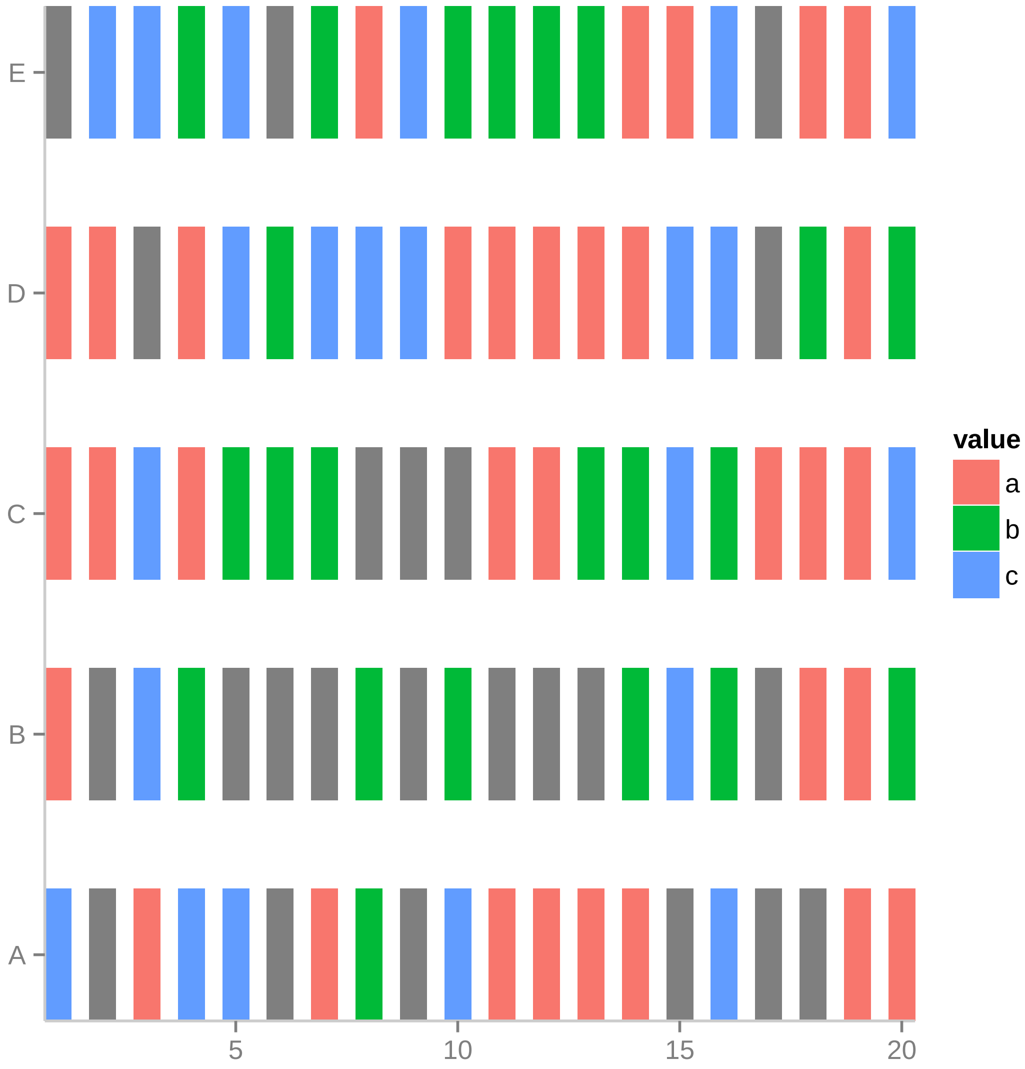
```
## default 'tile' is flexible
autoplot(mx, aes(width = 0.6, height = 0.6))


## Scale for 'y' is already present.  Adding another scale for 'y', which will replace the
existing scale.
```



```
## change missing value color
autoplot(mx, aes(width = 0.6, height = 0.6), na.value = "white")

## Scale for 'y' is already present.  Adding another scale for 'y', which will replace the
existing scale.
```



```
autoplot(mx, aes(width = 0.6, height = 0.6)) + theme_clear()

## Scale for 'y' is already present.  Adding another scale for 'y', which will replace the
existing scale.
```



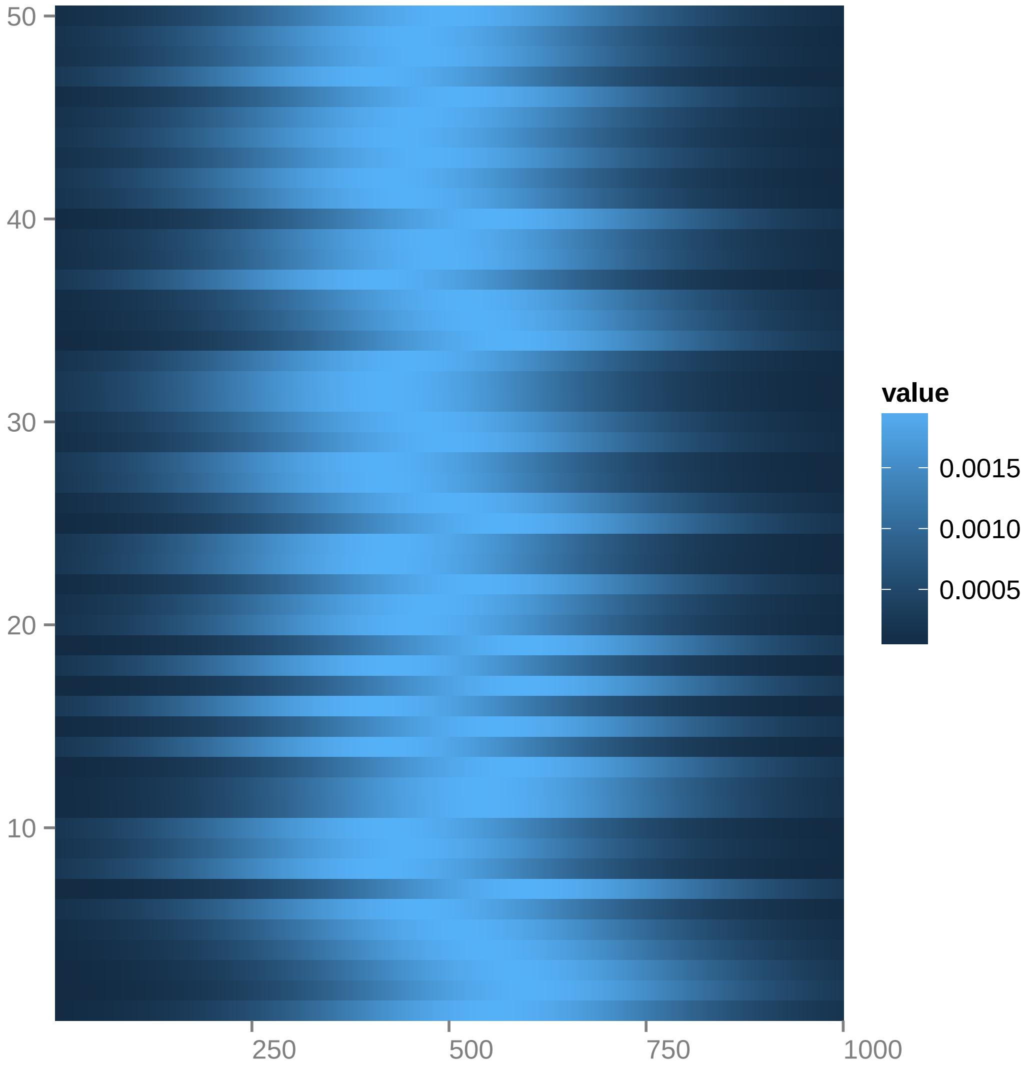
6.2.12 autoplot, Views

autoplot for *Views* object, first convert it to a matrix, and align it from left, so you can compare multiple region on the genome with scores all together, this is useful, when you are trying to compare multiple binding region around tss and make a summary plot.

Here is a simulated data.

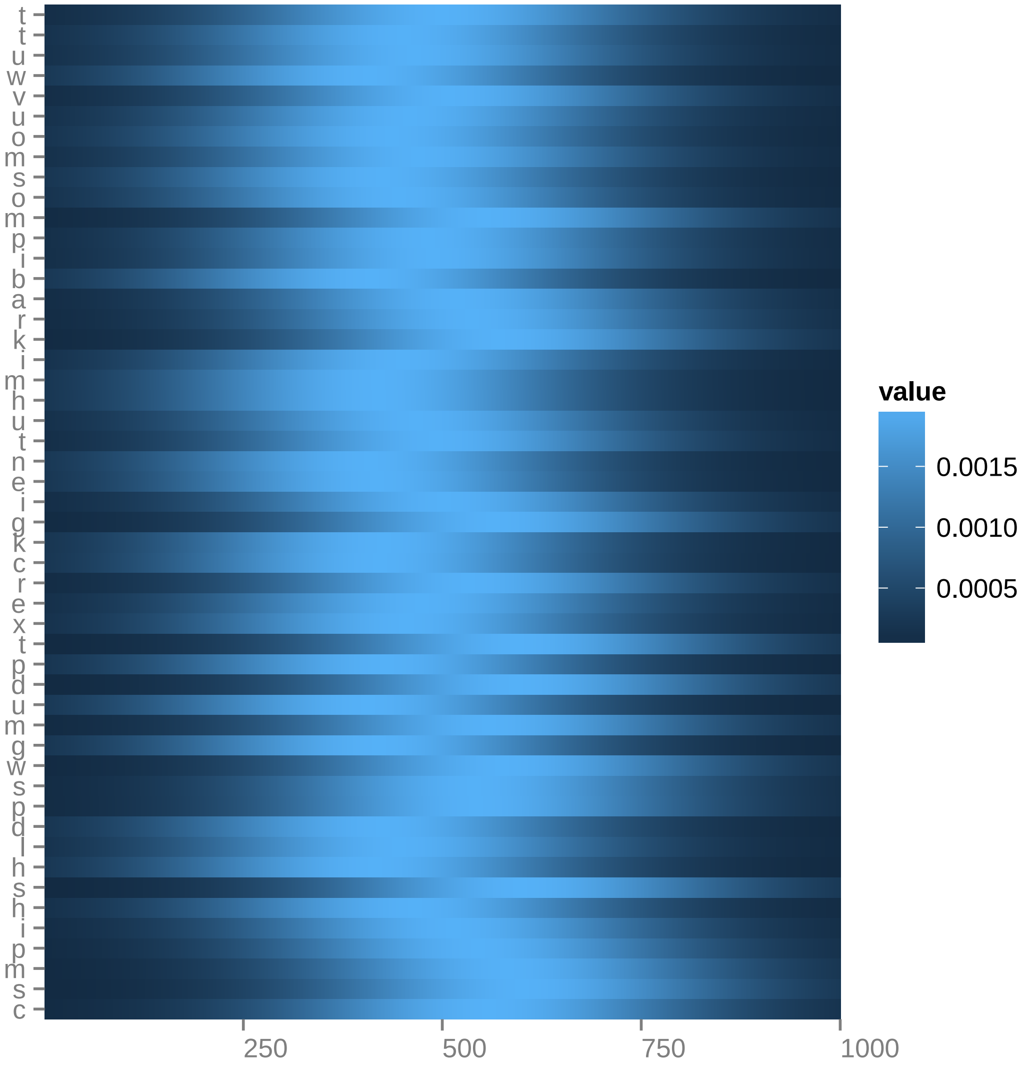
```
lambda <- c(rep(0.001, 4500), seq(0.001, 10, length = 500), seq(10, 0.001,
    length = 500))
xVector <- dnorm(1:5000, mean = 1000, sd = 200)
xRle <- Rle(xVector)
v1 <- Views(xRle, start = sample(400:600, size = 50, replace = FALSE), width = 1000)
```

```
autoplot(v1)

## Object of class "ggbio"
```

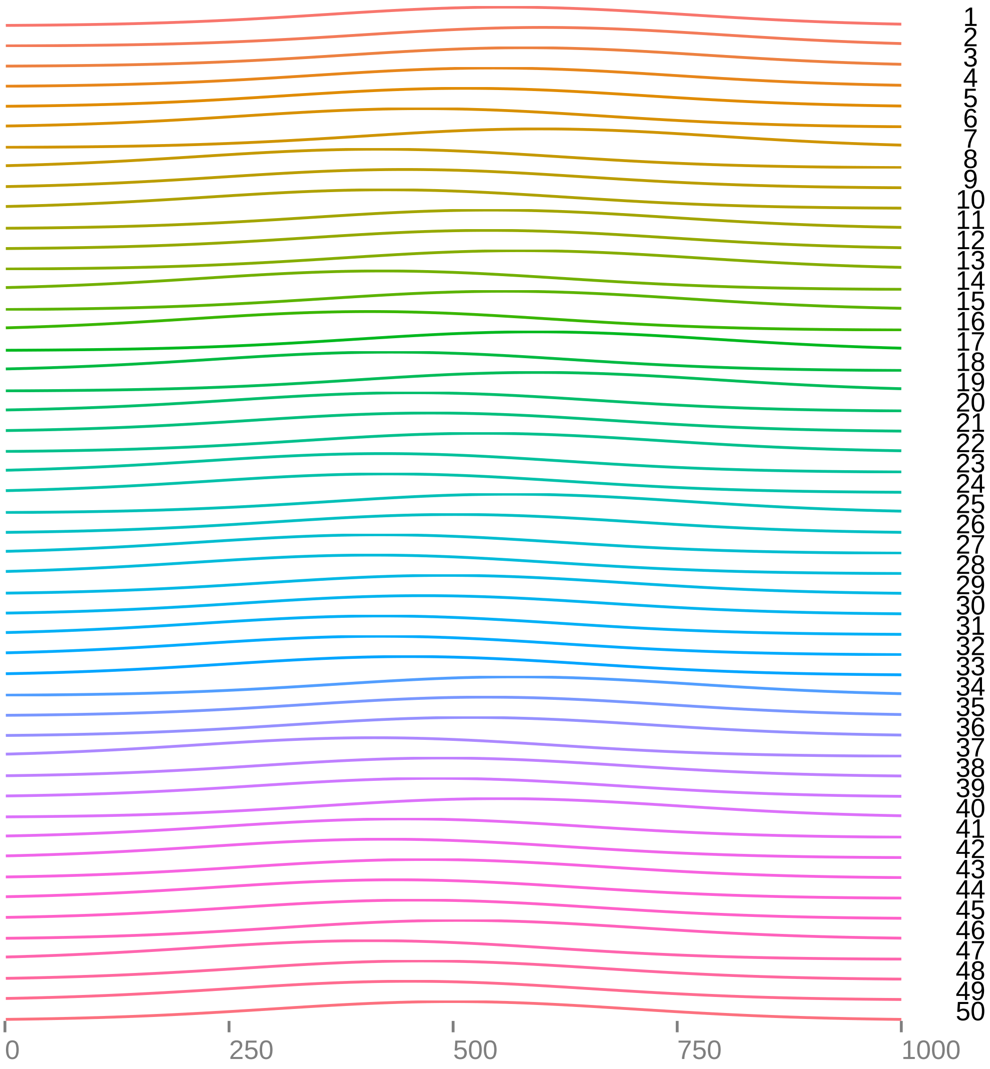


```
## NULL

names(v1) <- letters[sample(1:24, size = length(v1), replace = TRUE)]
autoplot(v1)

## Scale for 'y' is already present.  Adding another scale for 'y', which will replace the
existing scale.
```

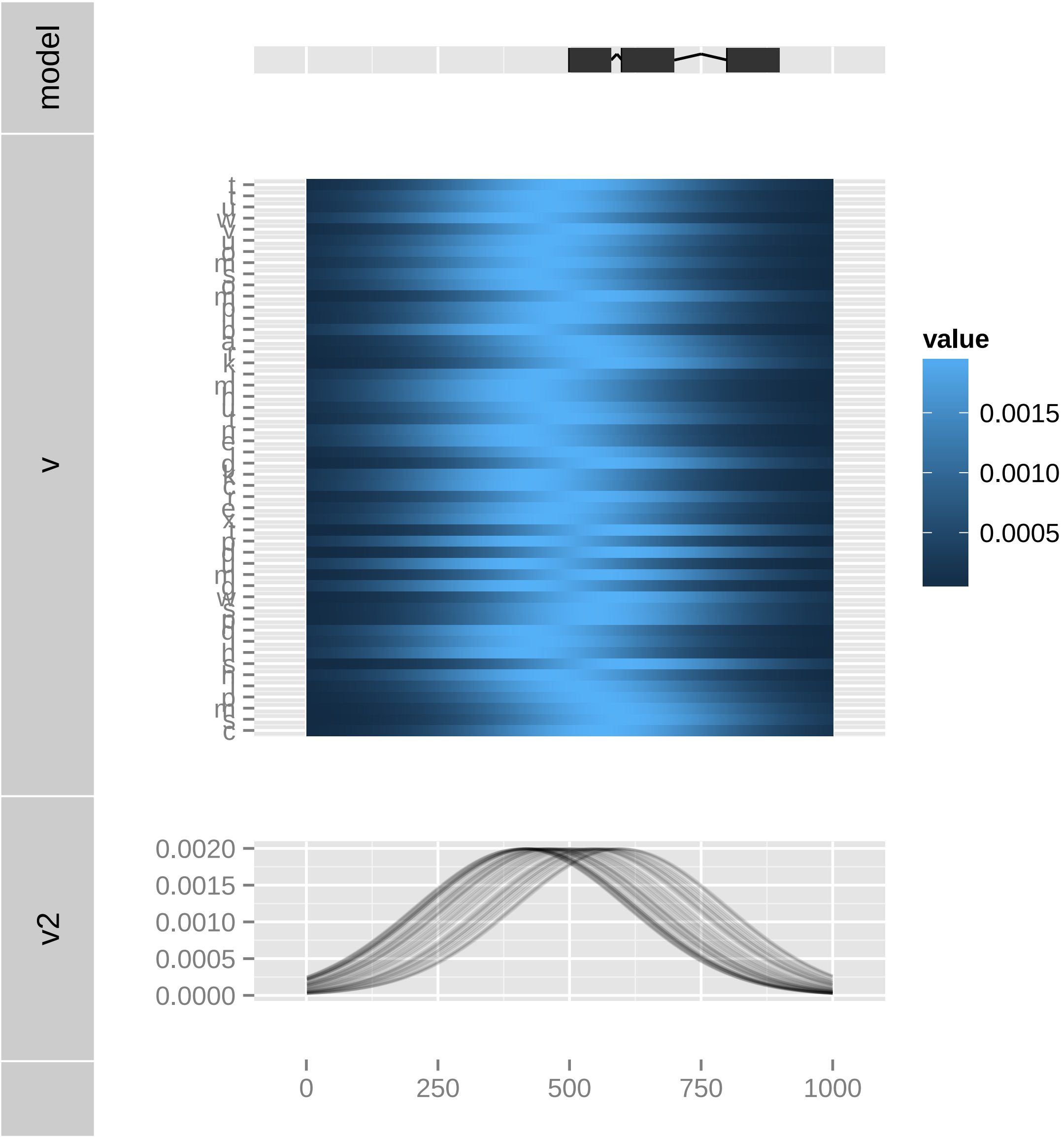
```
## Object of class "ggbio"
```



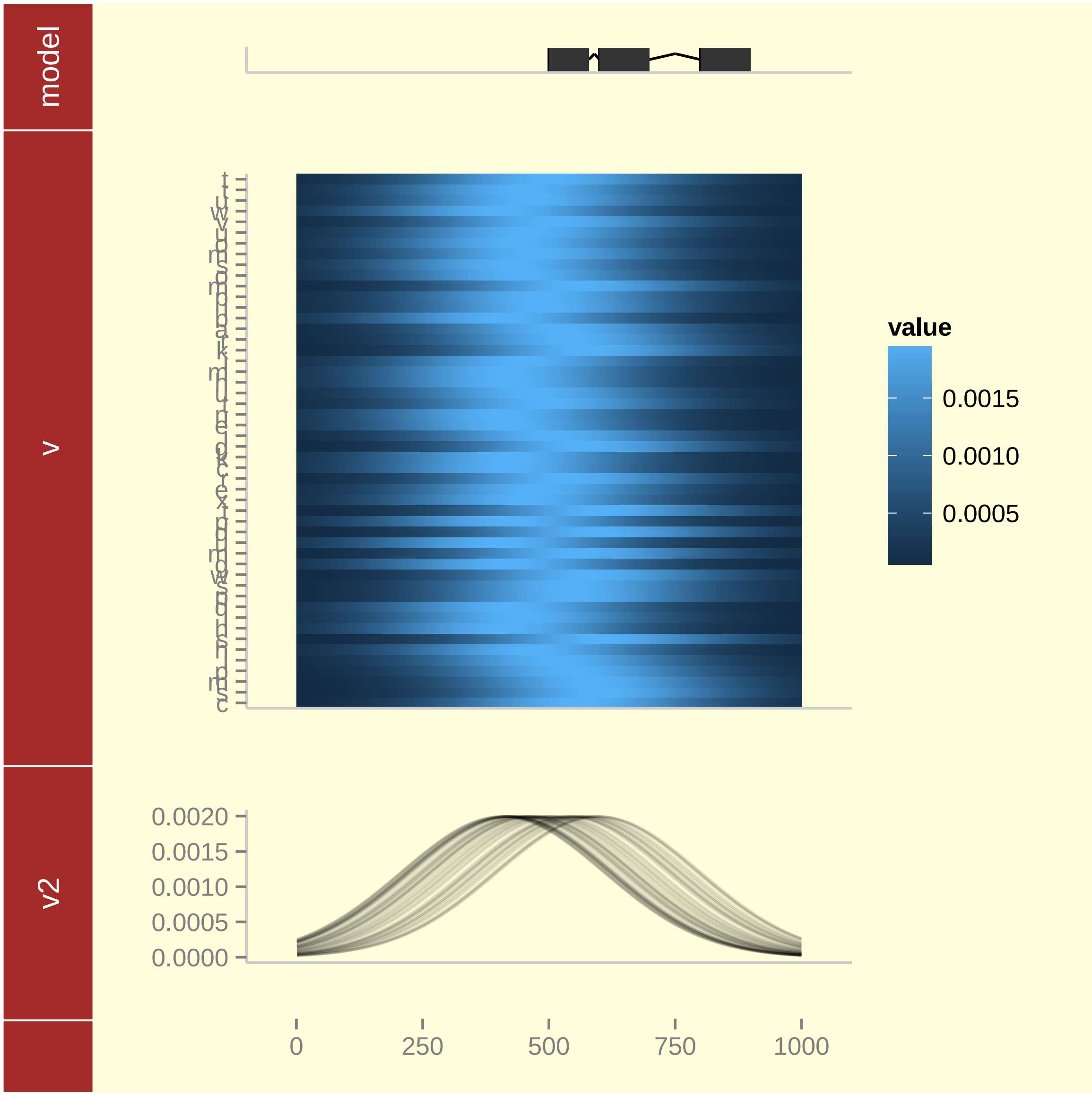
```
## NULL

autoplot(v1, geom = "line", aes(color = row)) + theme(legend.position = "none")

## Object of class "ggbio"
```



```
## NULL

## make a 1000 fake gene region,
gr <- GRanges("chr0", IRanges(start = c(500, 600, 800), width = c(80, 100,
    100)))
p.model <- autoplot(gr, geom = "alignment")
p.v <- autoplot(v1)

## Scale for 'y' is already present.  Adding another scale for 'y', which will replace the
existing scale.

p.v2 <- autoplot(v1, geom = "line", facets = NULL, alpha = 0.1)
tracks(model = p.model, v = p.v, v2 = p.v2, heights = c(1, 5, 2))
```



```
tracks(model = p.model, v = p.v, v2 = p.v2, heights = c(1, 5, 2)) + theme_tracks_sunset()
```



6.2.13 autoplot, ExpressionSet

`ExpressionSet` object is commonly used container for storing high-throughput assays and experimental metadata. it's defined in *Biobase*.

Graphics we bring for this type of data includes:

- 'heatmap': default.
- 'pcp': parallel coordinate plots, level change for particular gene(row) can be easily observed cross samples.
- 'boxplot': boxplot, summary over samples.
- 'scatterplot.matrix' pairwised comparison across samples, a quick way to observe correlation.

- other specific experimental types may require loading other packages, such as types 'mean-sd' and 'volcano'.

Let's have some examples.

```
library(Biobase)
data(sample.ExpressionSet)
sample.ExpressionSet


## ExpressionSet (storageMode: lockedEnvironment)
## assayData: 500 features, 26 samples
##   element names: exprs, se.exprs
## protocolData: none
## phenoData
##   sampleNames: A B ... Z (26 total)
##   varLabels: sex type score
##   varMetadata: labelDescription
## featureData: none
## experimentData: use 'experimentData(object)'
## Annotation: hgu95av2


set.seed(1)
idx <- sample(seq_len(dim(sample.ExpressionSet)[1]), size = 50)
eset <- sample.ExpressionSet[idx, ]
```

6.2.14 autoplot, SummarizedExperiment

SummarizedExperiment is a eSet-like container, where column represetns samples and rows represent ranges of interest, for example, a GRanges object, and it could contain one or more assays. It's defined in package *GenomicRanges*.

- 'heatmap': default.
- 'pcp': parallel coordinate plots, level change for particular gene(row) can be easily observed cross samples.
- 'boxplot': boxplot, summary over samples.
- 'scatterplot.matrix' pairwised comparison across samples, a quick way to observe correlation.

```
nrows <- 200
ncols <- 6
counts <- matrix(runif(nrows * ncols, 1, 10000), nrows)
rowData <- GRanges(rep(c("chr1", "chr2"), c(50, 150)), IRanges(floor(runif(200,
    1e+05, 1e+06)), width = 100), strand = sample(c("+", "-"), 200, TRUE))
colData <- DataFrame(Treatment = rep(c("ChIP", "Input"), 3), row.names = LETTERS[1:6])
sset <- SummarizedExperiment(assays = SimpleList(counts = counts), rowData = rowData,
    colData = colData)
```

```
p1 <- autoplot(eset)

## Scale for 'y' is already present.  Adding another scale for 'y', which will replace
                                    the existing scale.
## Scale for 'x' is already present.  Adding another scale for 'x', which will replace
                                    the existing scale.

p1
```

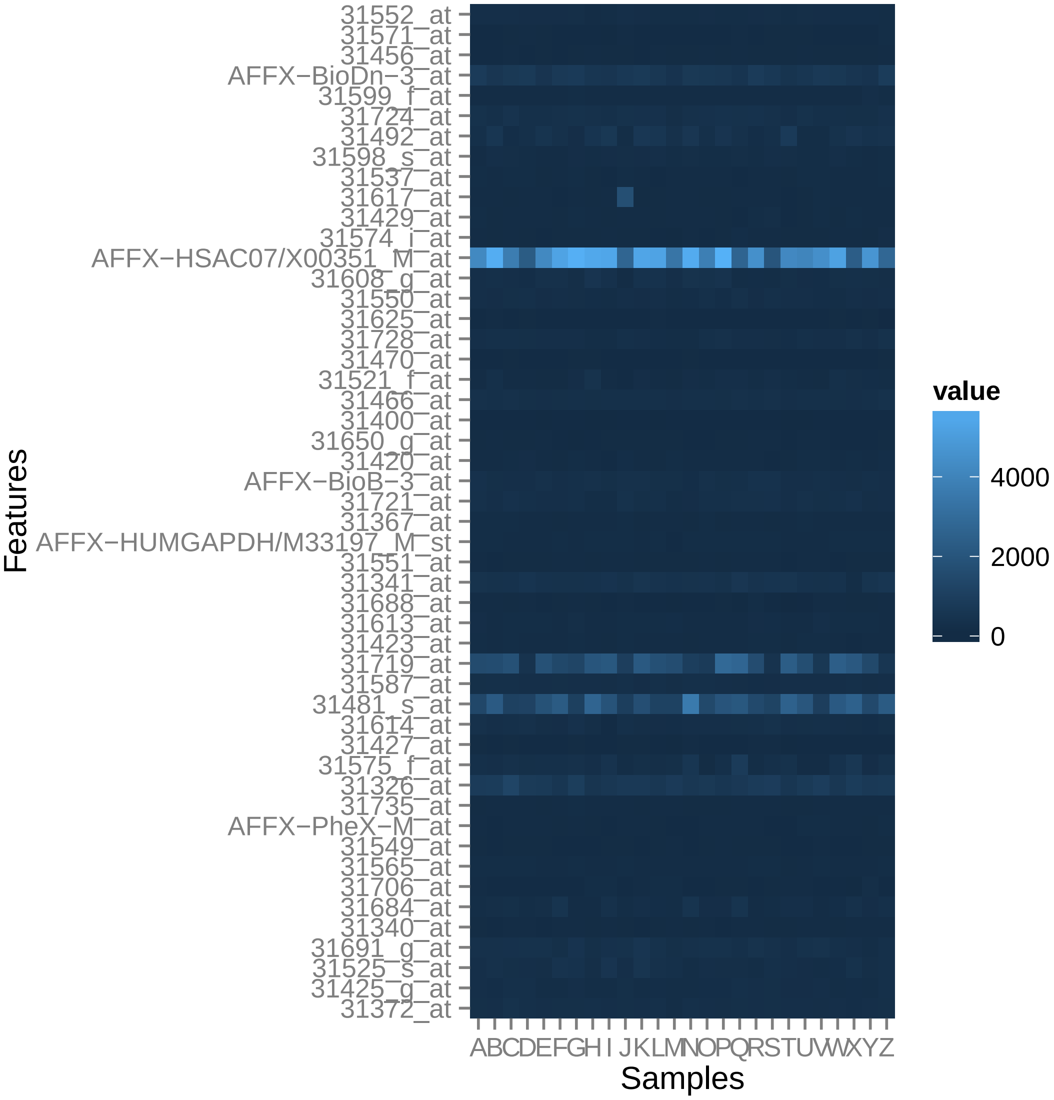


Figure 6.19: Heatmap default

```
p2 <- p1 + scale_fill_fold_change()
p2
```

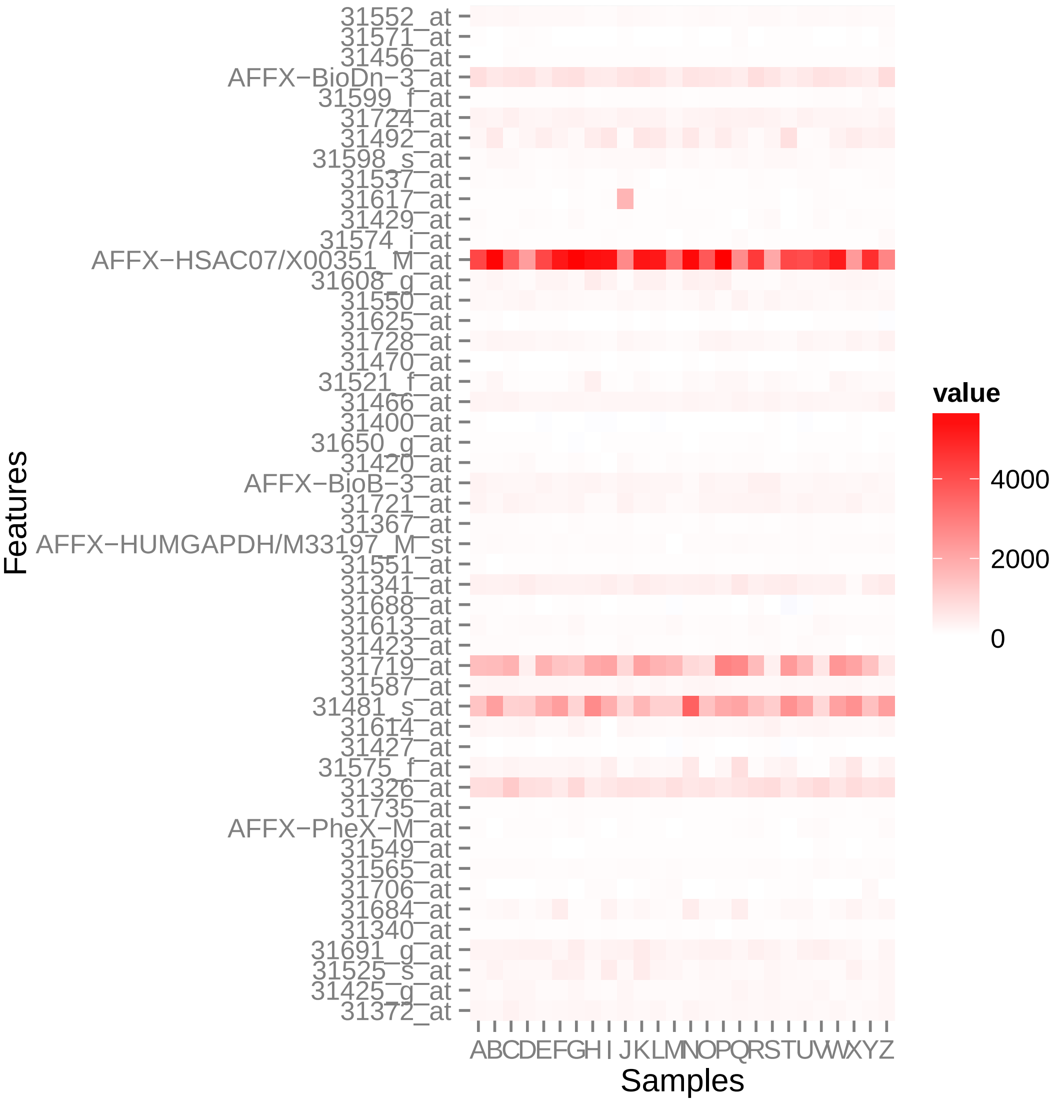


Figure 6.20: Heatmap default with blue-white-red scale

```
autoplot(eset, type = "pcp")
```

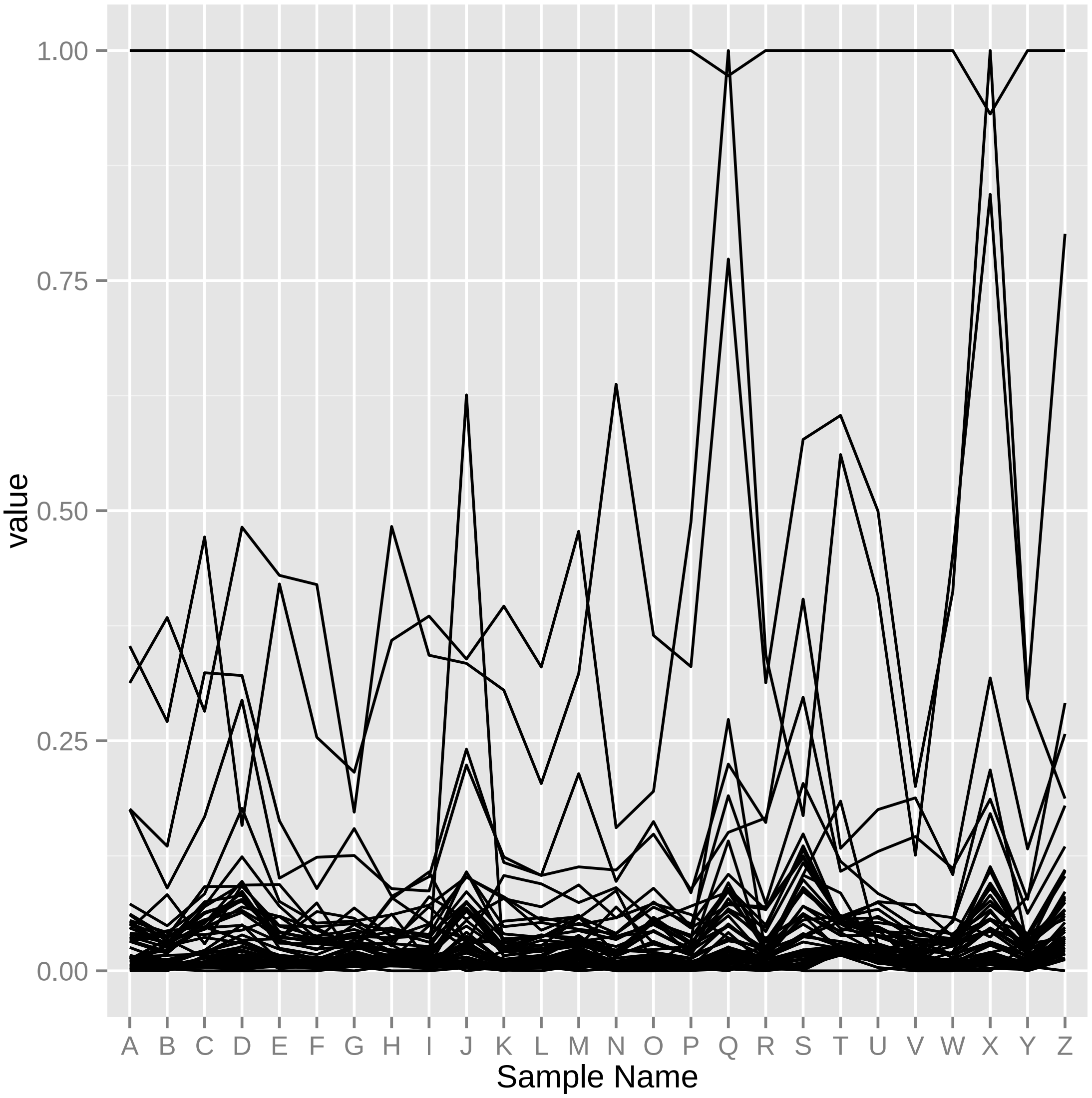


Figure 6.21: Parallel coordinate plot.

```r
autoplot(eset, type = "boxplot")
```

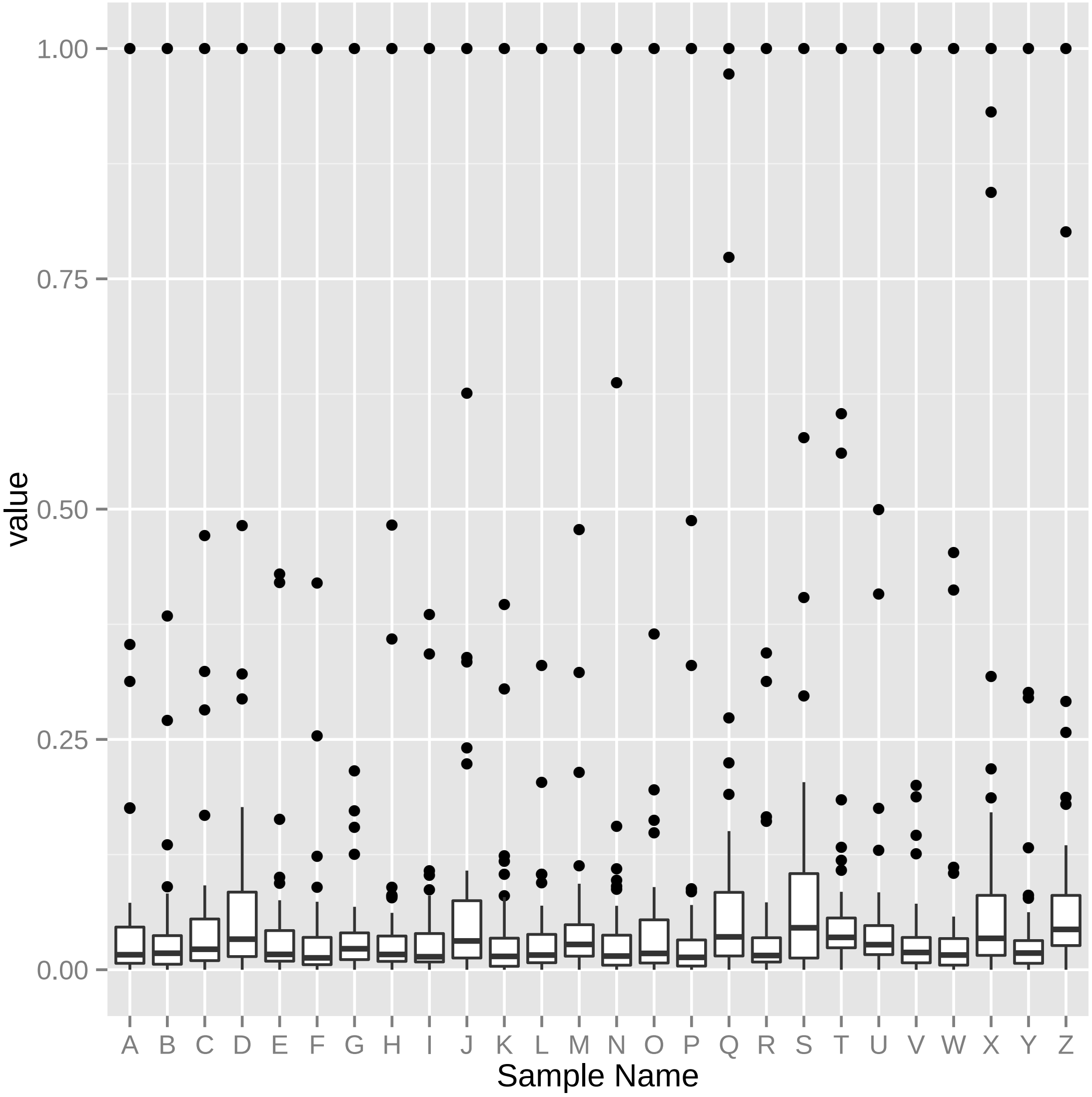


Figure 6.22: Boxplot.

```
autoplot(eset[, 1:7], type = "scatterplot.matrix")

    ## This function is deprecated.  For a replacement, see the ggpairs function in the
                    GGally package.  (Deprecated; last used in version 0.9.2)
```

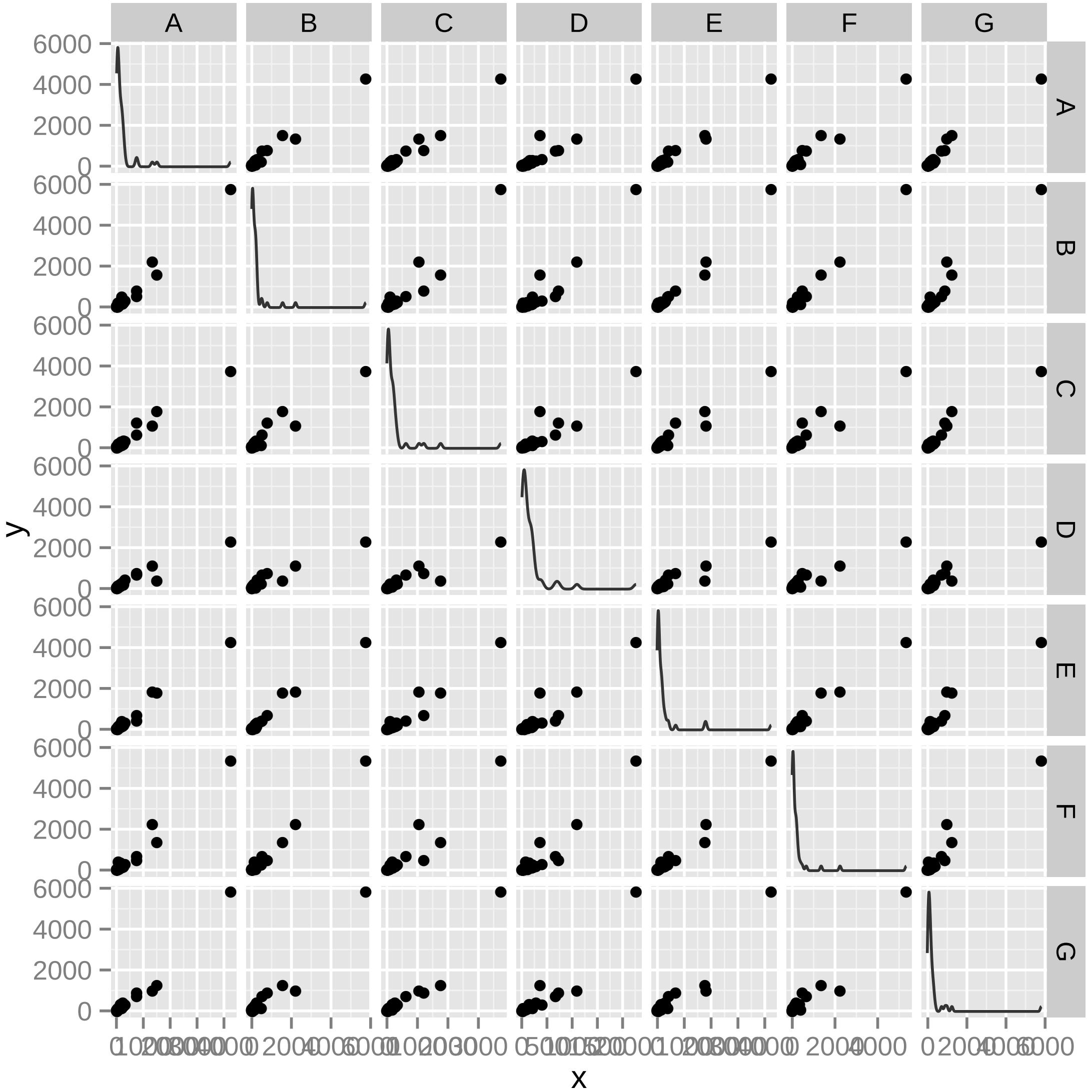


Figure 6.23: Scatterplot matrix.

```r
autoplot(eset, type = "mean-sd")

                                                   ## Loading required package:  vsn
```

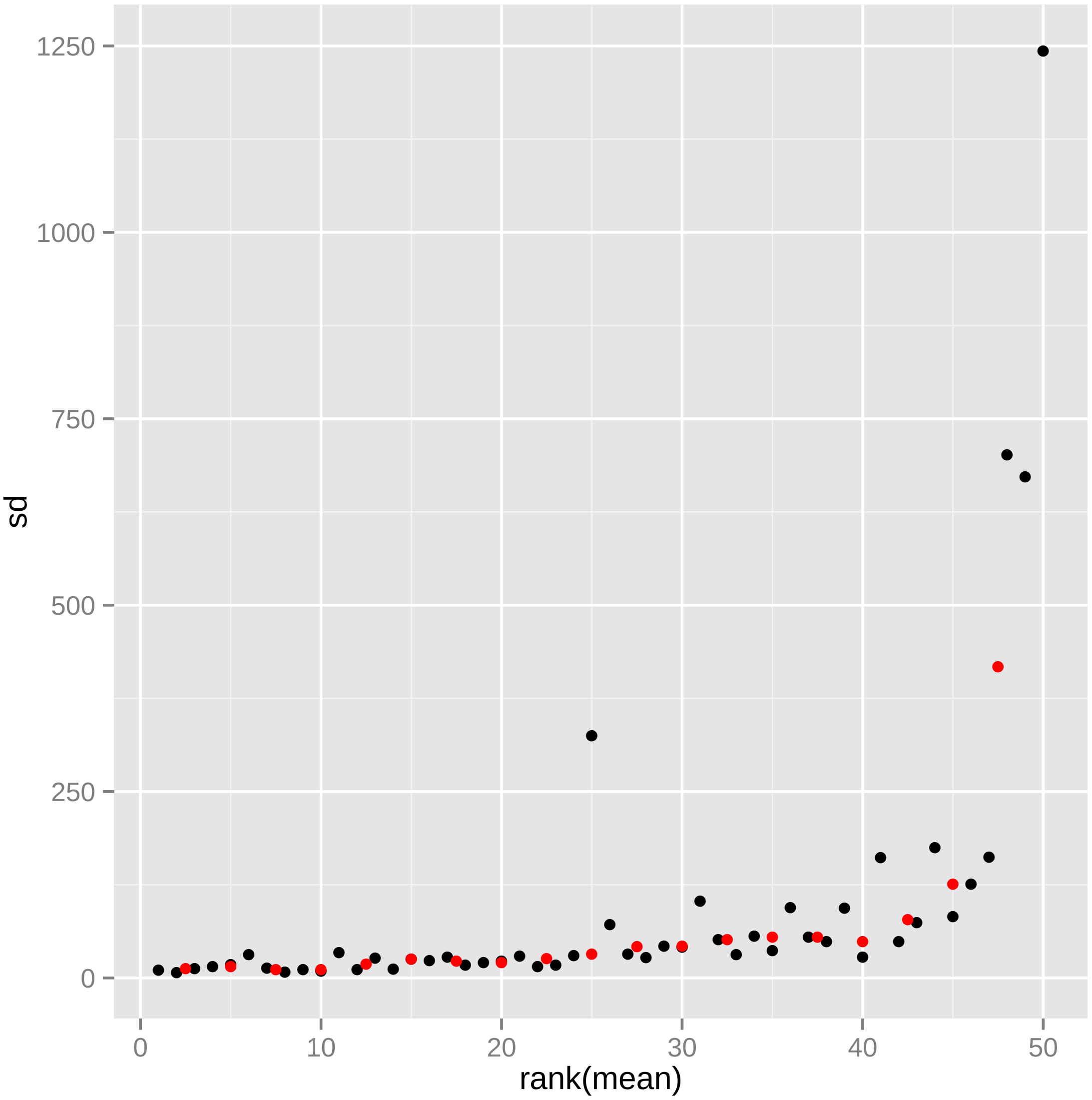


Figure 6.24: Scatterplot matrix.

```
autoplot(eset, type = "volcano", fac = pData(sample.ExpressionSet)$type)

## Loading required package:  genefilter
## genefilter::rowttests used
```

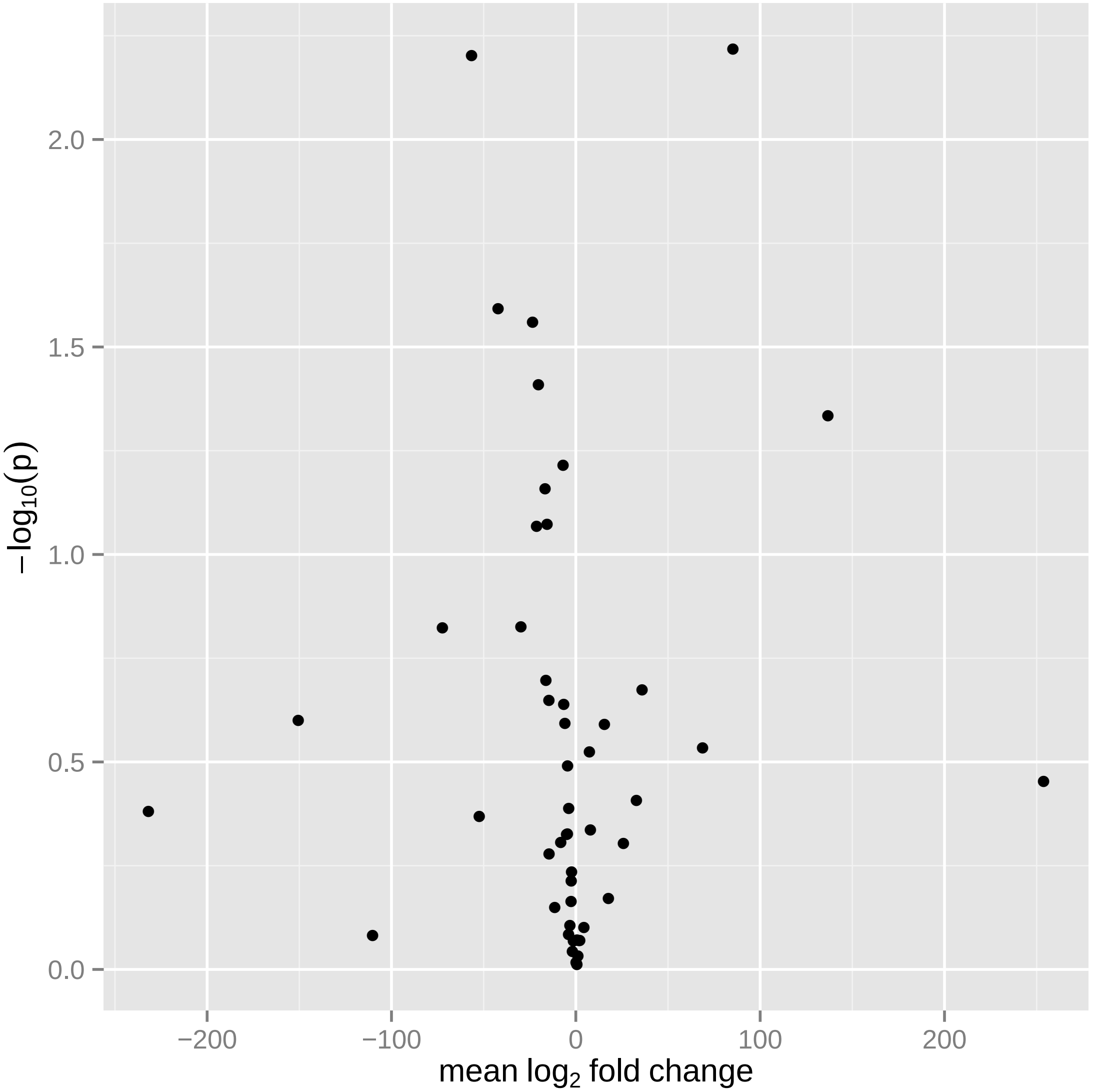


Figure 6.25: Scatterplot matrix.

```
autoplot(sset) + scale_fill_fold_change()

  ## Scale for 'x' is already present.  Adding another scale for 'x', which will replace
                                      the existing scale.
```

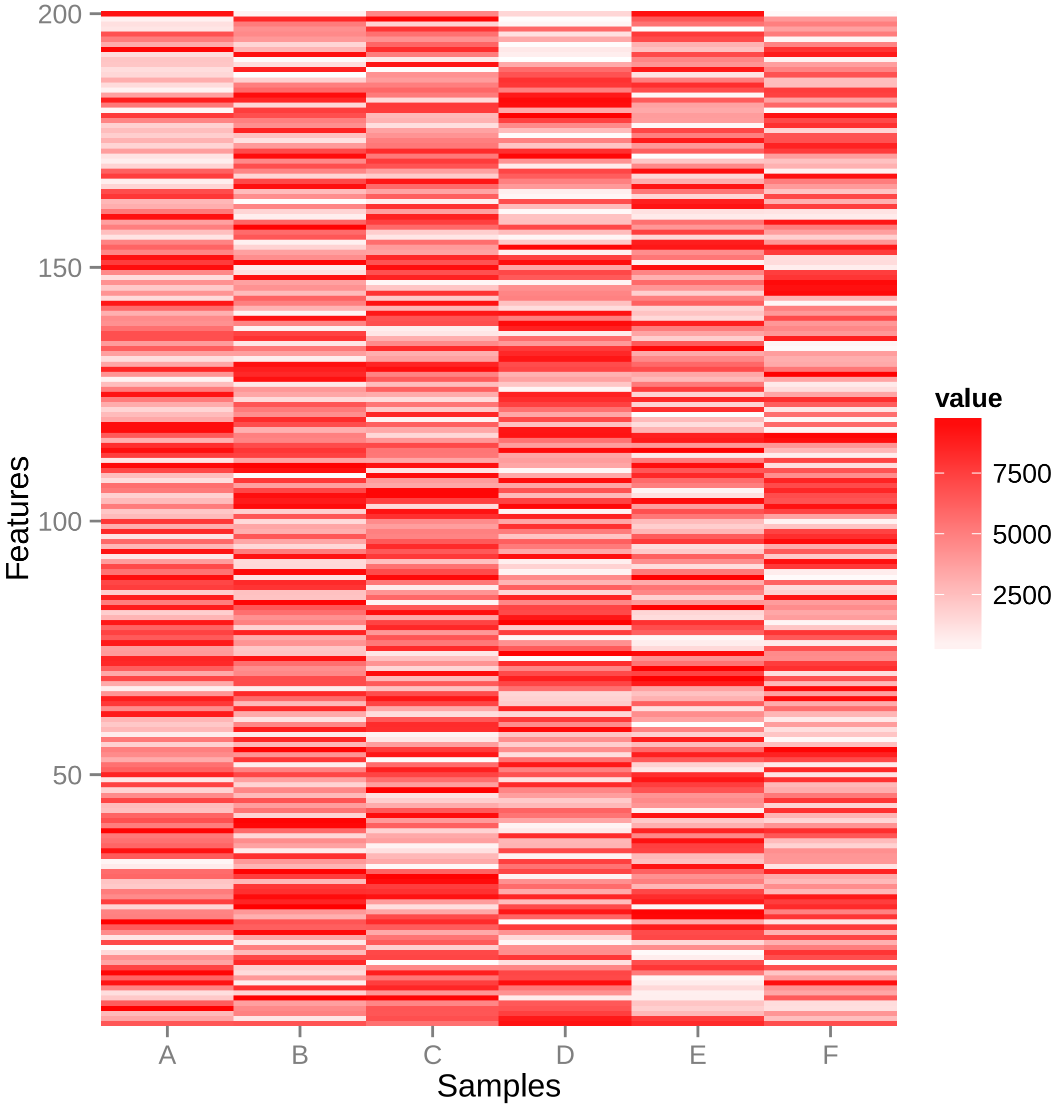


Figure 6.26: heatmap

```
autoplot(sset, type = "pcp")
```

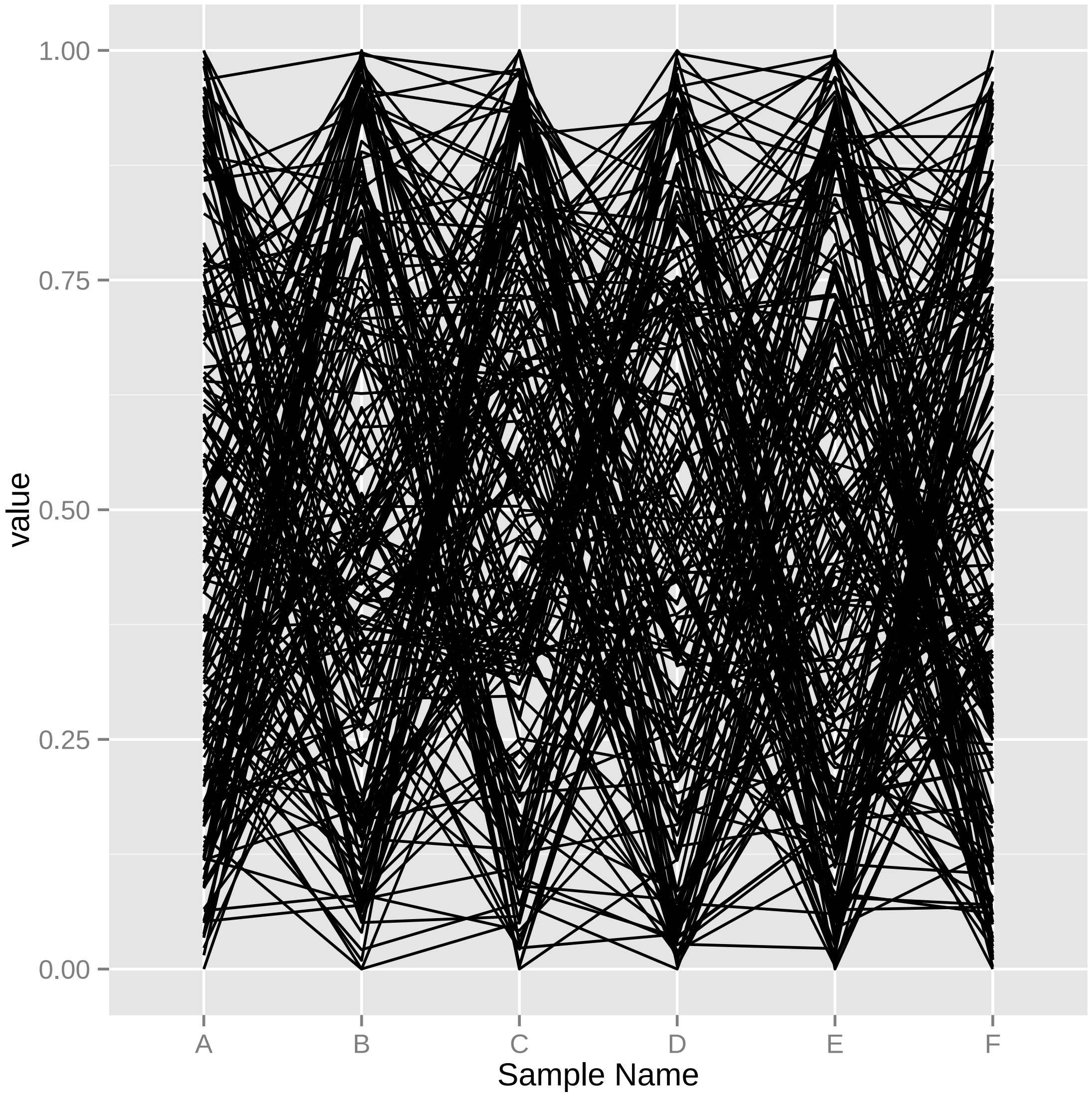


Figure 6.27: Parallel coordiante plot.

```
autoplot(sset, type = "boxplot")
```

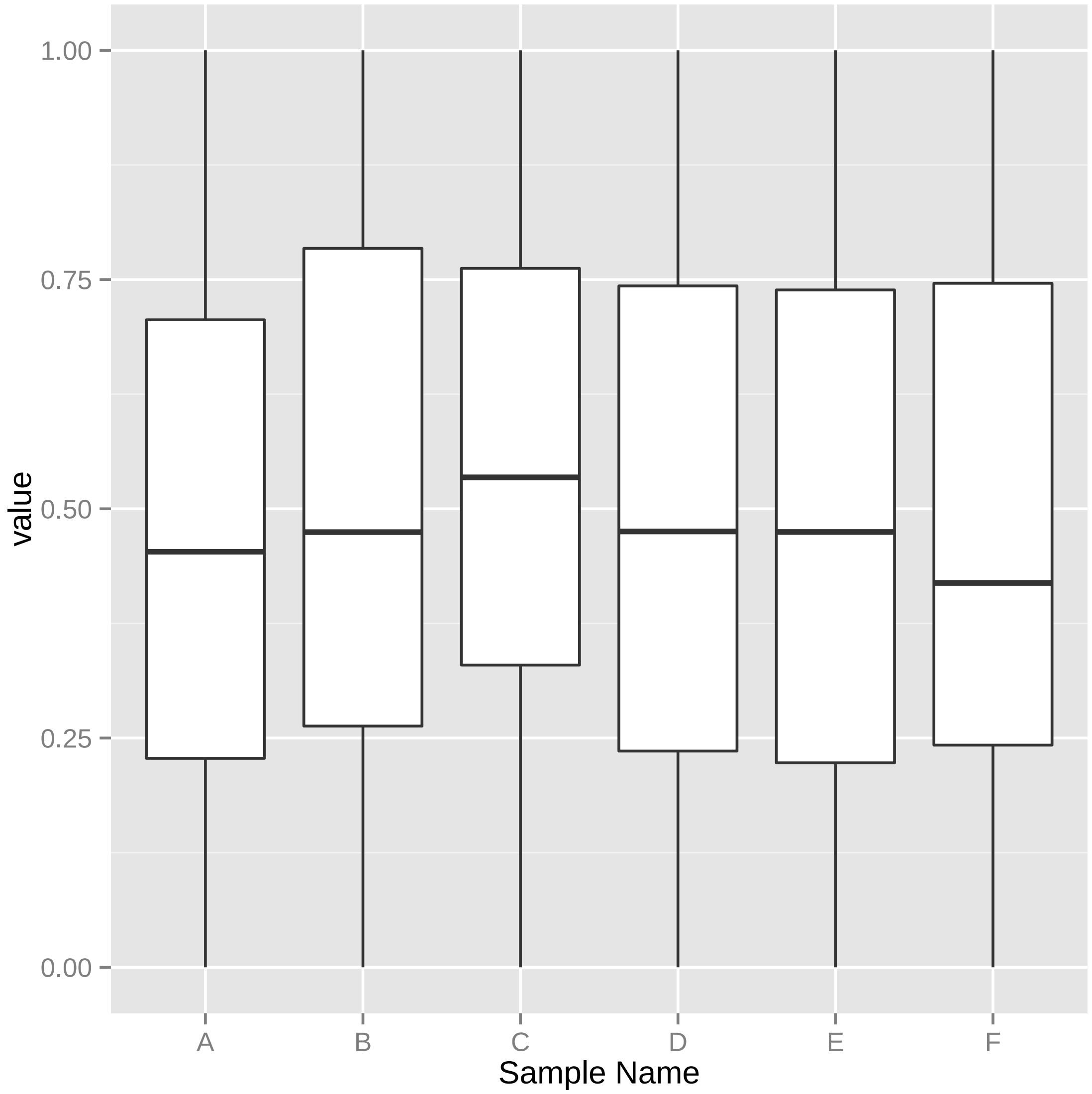


Figure 6.28: Boxplot.

```
autoplot(sset, type = "scatterplot.matrix")

   ## This function is deprecated.  For a replacement, see the ggpairs function in the
                   GGally package.  (Deprecated; last used in version 0.9.2)
```

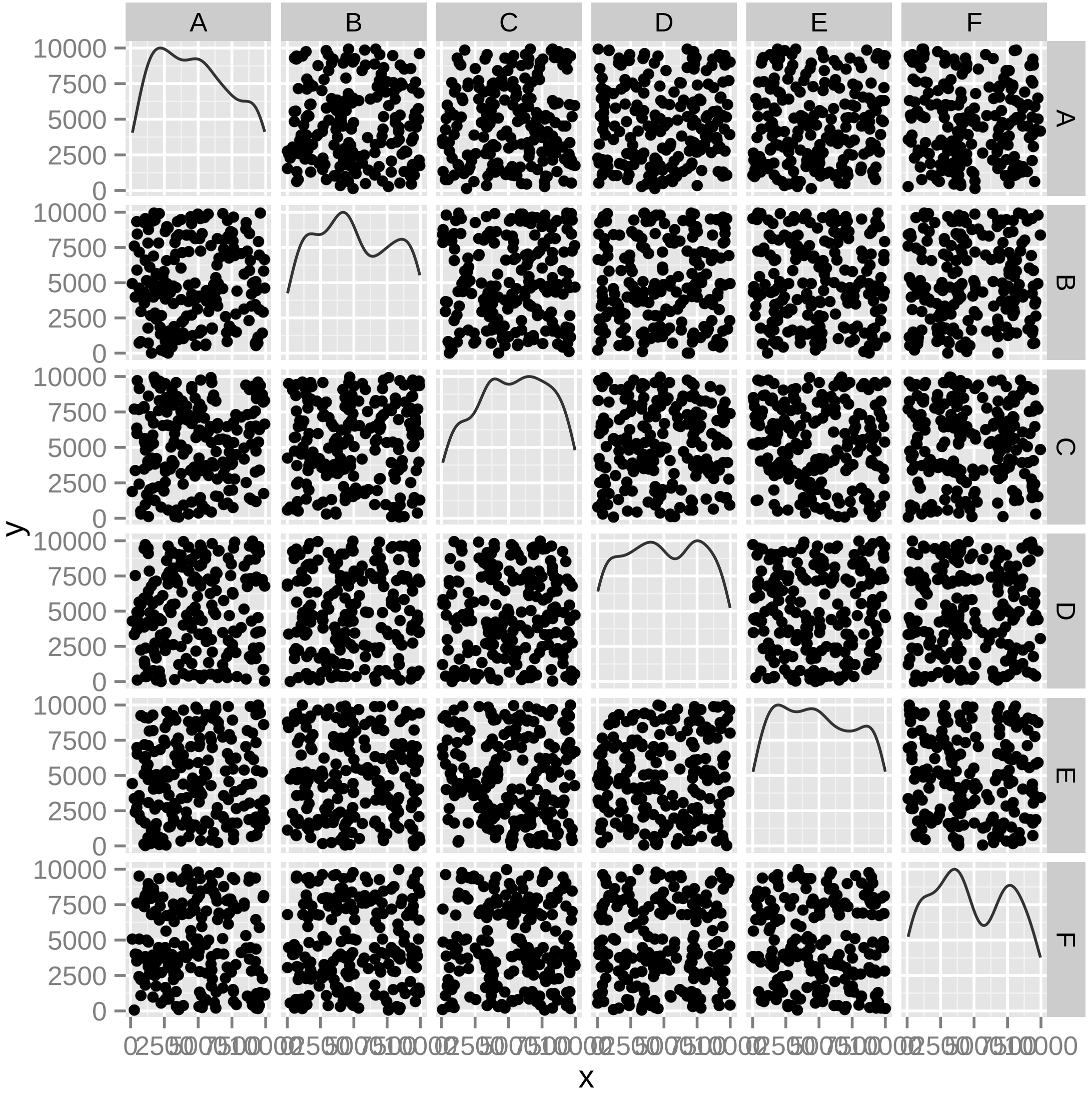


Figure 6.29: Scatterplot matrix

6.2.15 autoplot,VCF

VCF(Variant Call Format) class extends a class we have introduced SummarizedExperiment, and with additional slots, 'fixed' and 'info'. It's defined in package *VariantAnnotation*.

We have done some experimental visualization, and features are going to be extended or changed later.

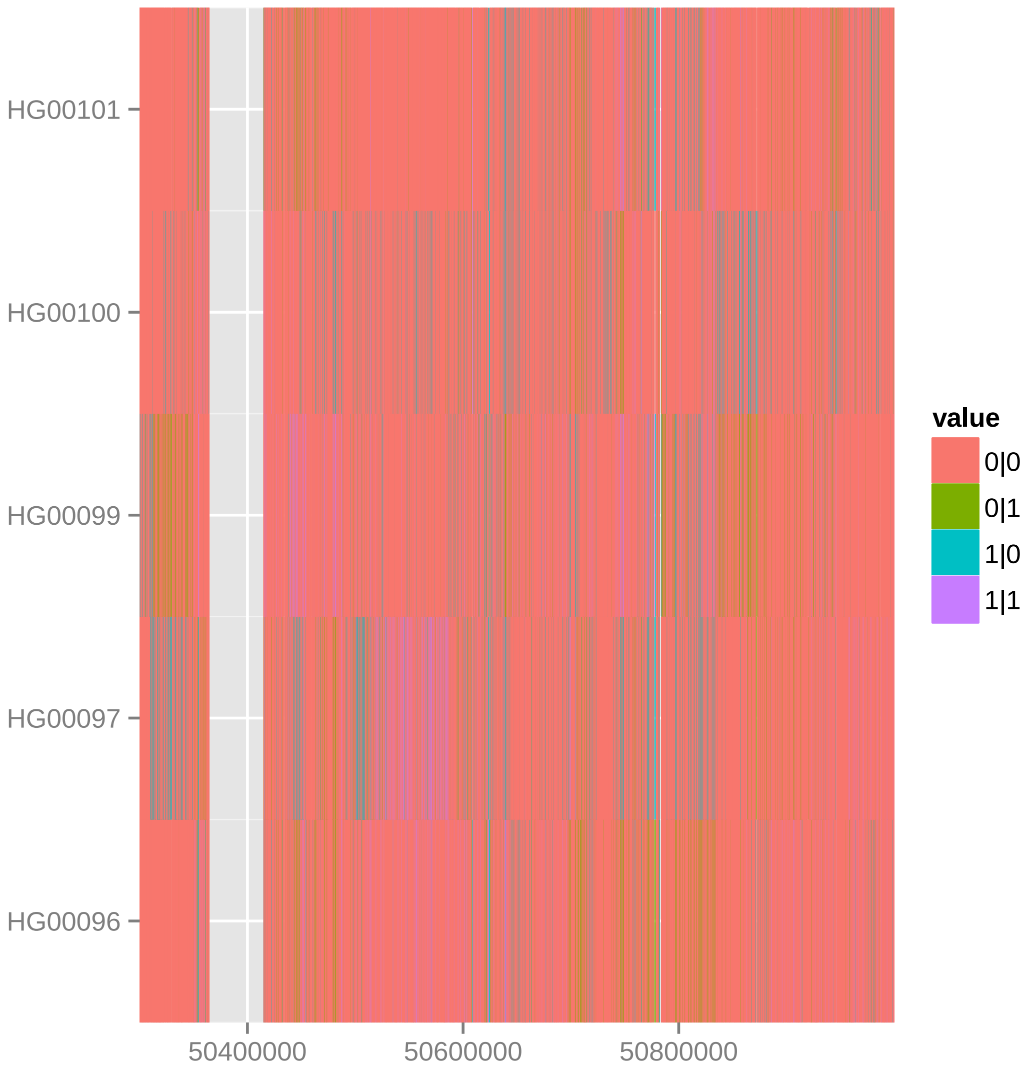
- For type 'geno': we get an assay, and test if 'GT' is in. Then we make a heatmap to show geno types.
- For type 'info': Please specify one variable as y to show as bars.
- For type 'fixed': You can plot ref/alt strings on the plot, default plot both reference and variants, and if one string is over 1, we will use a black 'I' to indicate that's an indel, then if at each position there are multiple data, we will show them in different y levels. Argument full.string control if you want to show full strings of indels or not, even they are shown as full string, they will still be in balck color, to indicate it's on the single position. ref.show controls if you want to show REF column in the data or not, sometimes people may want to just plot BSgenome object as reference track.

```
library(VariantAnnotation)


##
## Attaching package:  'VariantAnnotation'

## The following object(s) are masked from 'package:Biobase':
##
##    samples


vcffile <- system.file("extdata", "chr22.vcf.gz", package = "VariantAnnotation")
vcf <- readVcf(vcffile, "hg19")
hdr <- exptData(vcf)[["header"]]
```

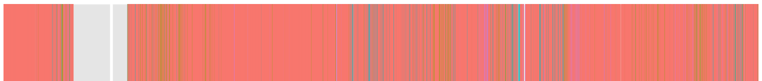
6.2.16 autoplot,BSgenome

The BSgenome class is a container for the complete genome sequence of a given organism, it's defined in package *BSgenome*. We use it to plot reference genome, along with other tracks.

```
autoplot(vcf)

## GT,DS,GL could be used for 'geno' type
## use GT for type geno as default
## Index:  744,3604,4738,7096,7287,10037,10198 snp with duplicated start position may be
masked by each other
## Scale for 'y' is already present.  Adding another scale for 'y', which will replace
the existing scale.
```



```
autoplot(vcf, genomic.pos = TRUE)

## GT,DS,GL could be used for 'geno' type
## use GT for type geno as default
## Index:  744,3604,4738,7096,7287,10037,10198 snp with duplicated start position may be
masked by each other
## Scale for 'y' is already present.  Adding another scale for 'y', which will replace
the existing scale.
```



```
autoplot(vcf, type = "info", aes(y = THETA))

    ## Other options for potential mapping(only keep numeric/integer/character/factor
                                    variable):
```

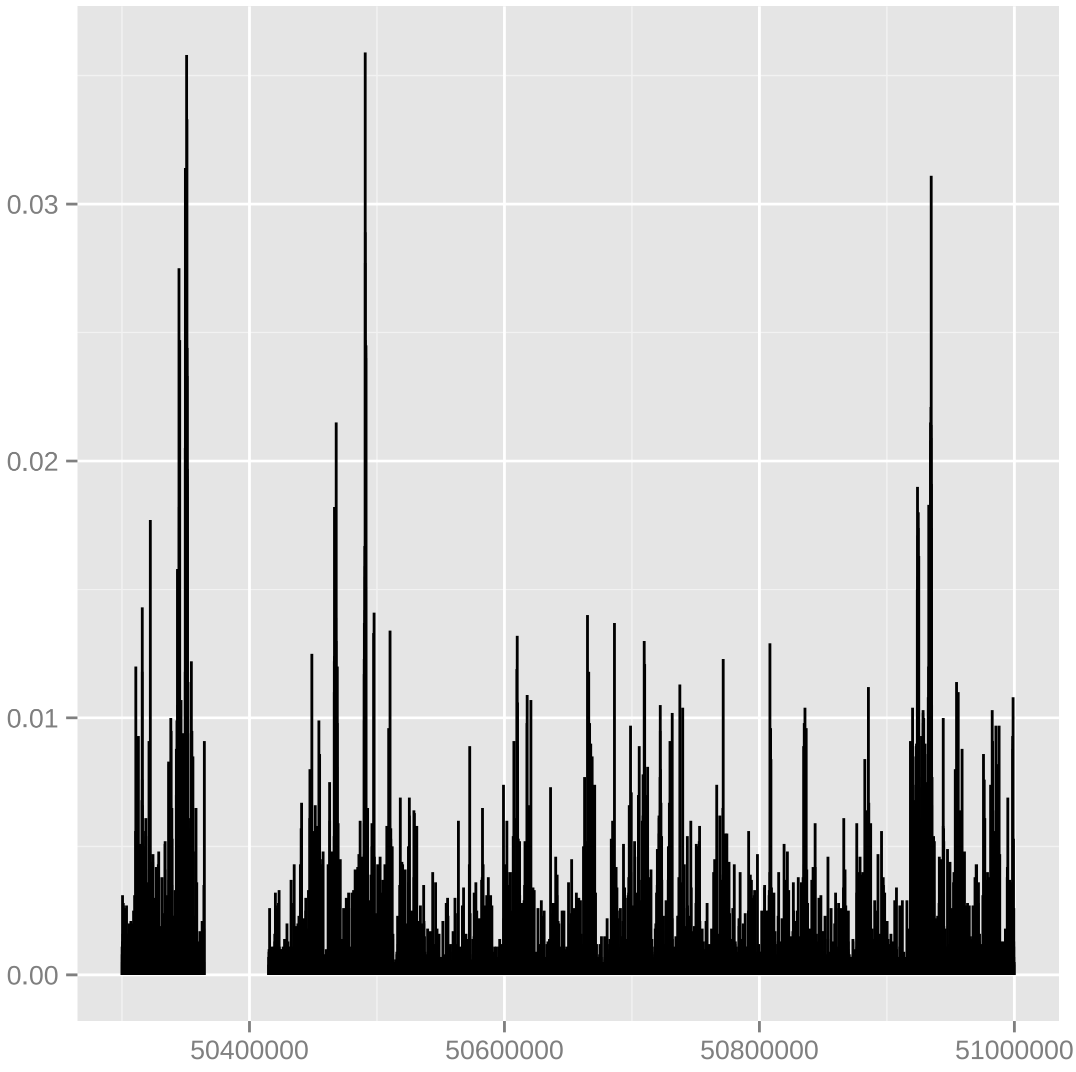


Figure 6.31: default bar chart for type info, use THETA as y.

```
autoplot(vcf, type = "fixed")
```

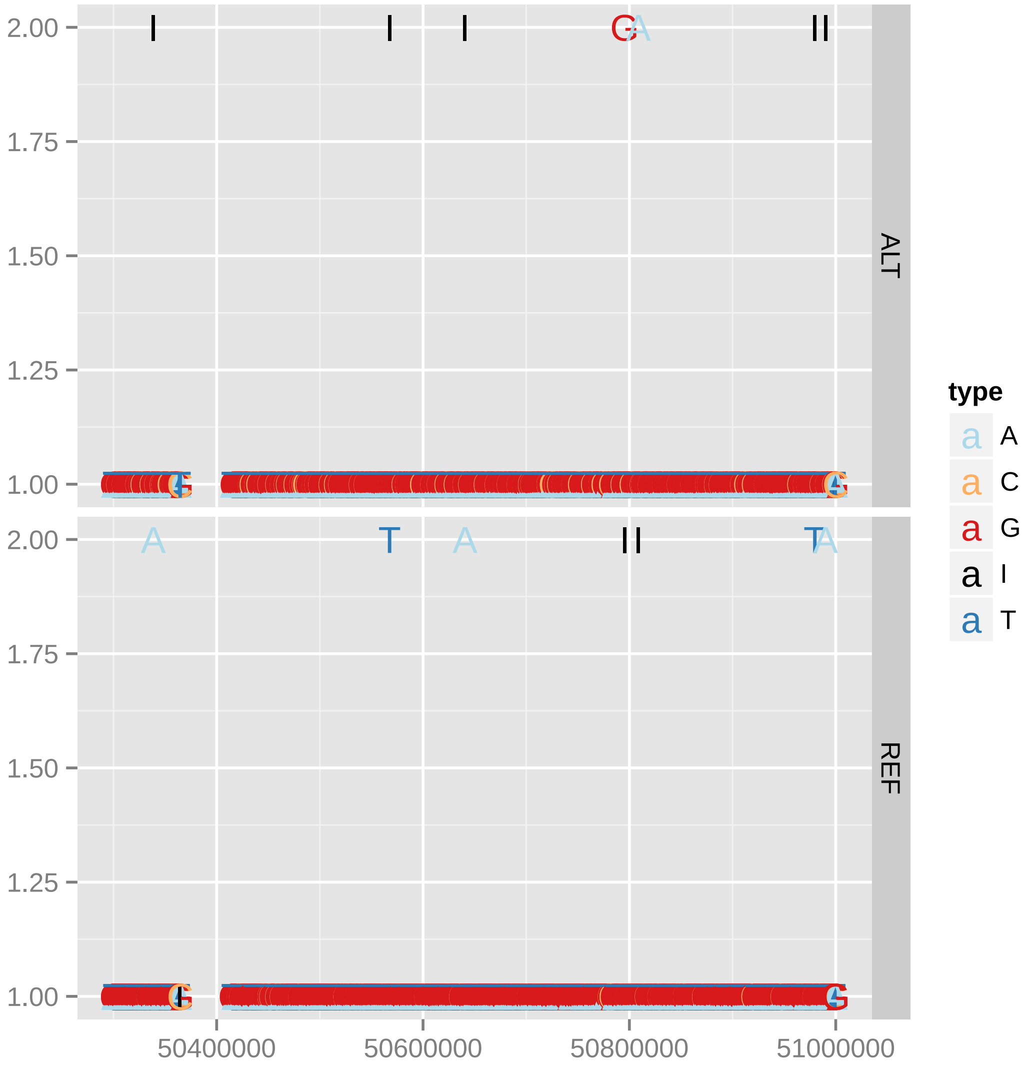


Figure 6.32: default heatmap to show GT as heatmap for type 'fixed'.

```r
p1 <- autoplot(vcf, type = "fixed") + xlim(50310860, 50310890)
p2 <- autoplot(vcf, type = "fixed", full.string = TRUE) + xlim(50310860,
    50310890)
tracks(`full.string = FALSE` = p1, `full.string = TRUE` = p2) + scale_y_continuous(breaks = NULL,
    limits = c(0, 3))

            ## Warning:  Removed 1 rows containing missing values (geom_text).
```

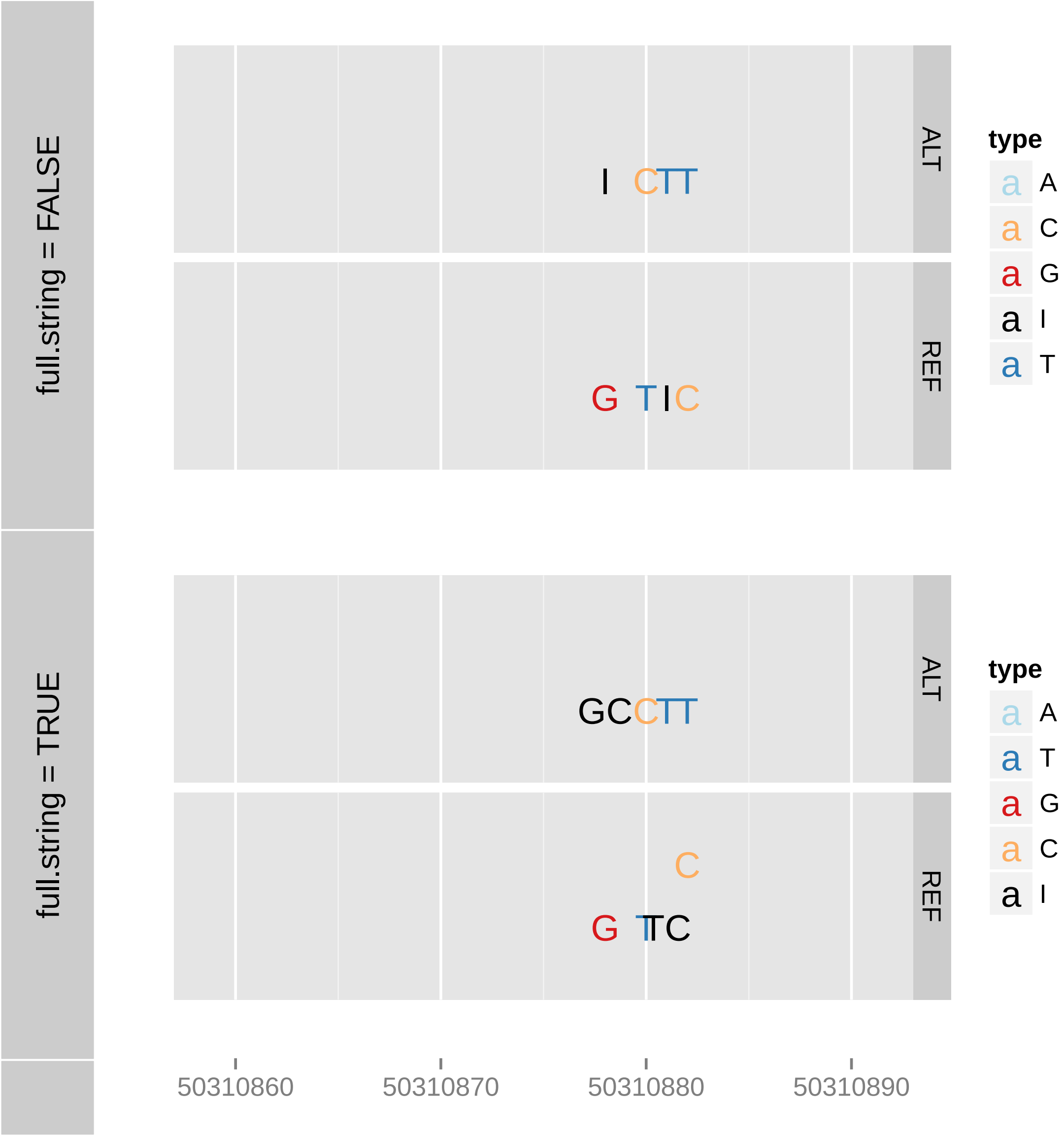


Figure 6.33: Demonstration of full.string.

```
p3 <- autoplot(vcf, type = "fixed", ref.show = FALSE) + xlim(50310860, 50310890) +
    scale_y_continuous(breaks = NULL, limits = c(0, 2))
p3
```

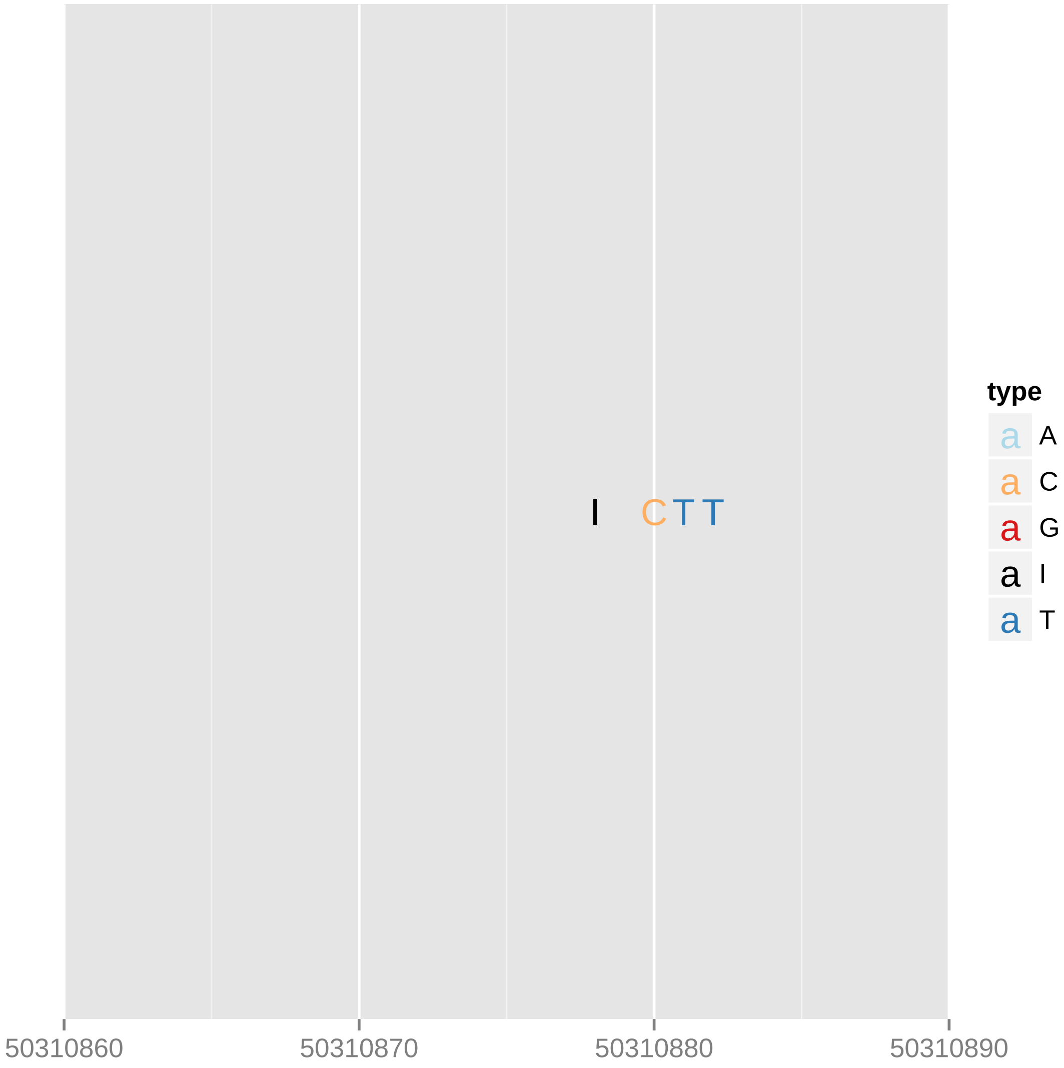


Figure 6.34: Demonstration of ref.show.

```
library(BSgenome.Hsapiens.UCSC.hg19)

## Loading required package:  BSgenome
##
## Attaching package:  'BSgenome'
## The following object(s) are masked from 'package:AnnotationDbi':
##
##     species

data(genesymbol, package = "biovizBase")
p1 <- autoplot(Hsapiens, which = resize(genesymbol["ALDOA"], width = 50))
p2 <- autoplot(Hsapiens, which = resize(genesymbol["ALDOA"], width = 50),
    geom = "rect")

## Scale for 'y' is already present.  Adding another scale for 'y', which will replace
the existing scale.

tracks(text = p1, rect = p2)
```

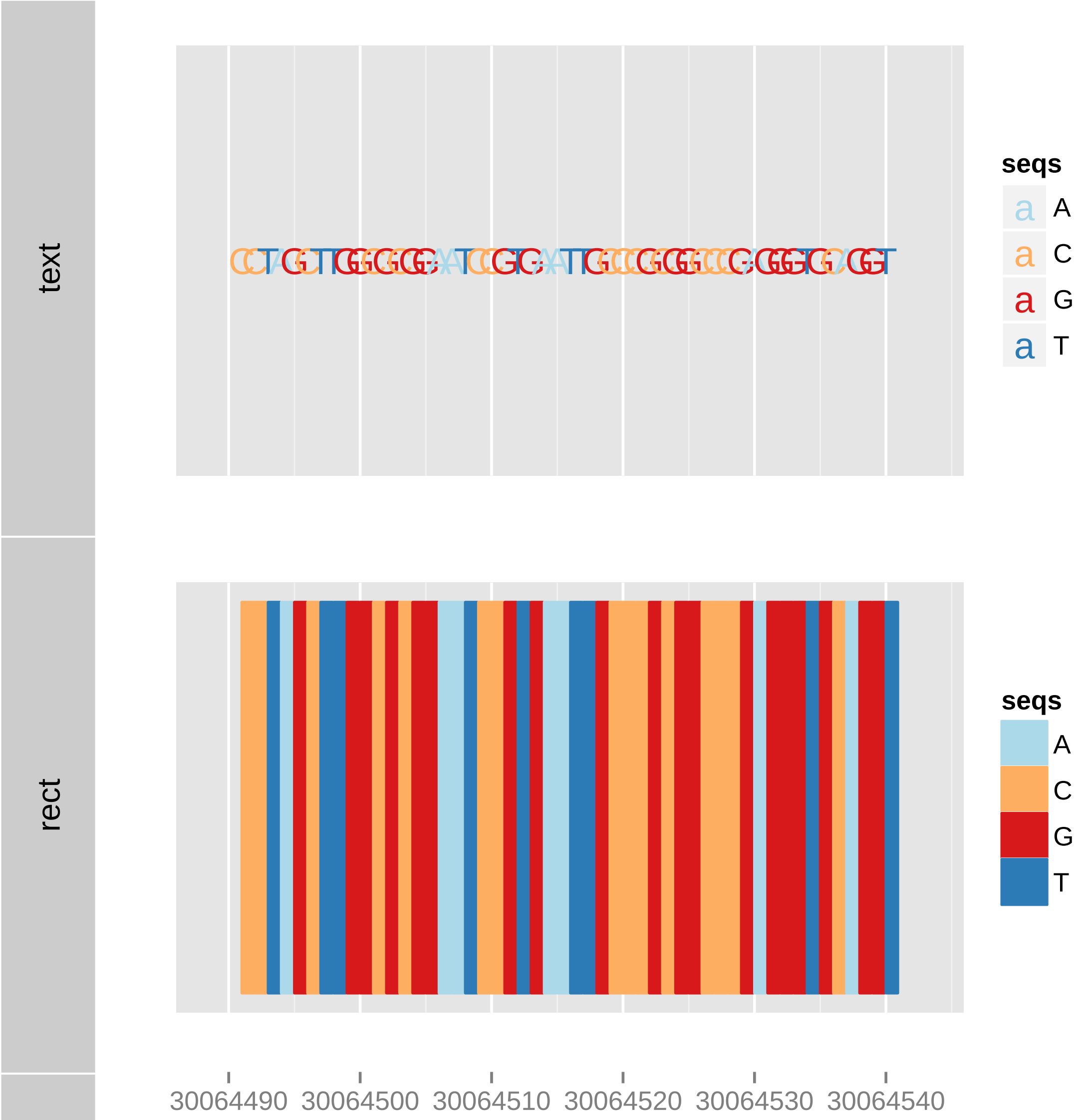


Figure 6.35: plot BSgenome

Chapter 7

Ideogram

7.1 Introduction

Ideograms are a schematic representation of chromosomes showing the relative size and banding patterns of the chromosomes. Single chromosome ideogram overview is widely used in most track-based genome browsers, usually on top of all tracks, and use a indicator such as a highlighted winbdow to indicate current region being viewed for tracks below, in this case, users won't lose too much context when zoomed into certain region.

7.2 Usage

7.2.1 Visualization of ideogram for single chromosome

For single chromosome ideogram, we require they have been arranged into a `GRanges` object in order to be visualized in *ggbio*. We will introduce how to get those ideogram on-line and manually later. Let's first take a look at what the data looks like.

We have two types of ideogram, which have different requirements for data, let's first introduce the most commonly used one: *Ideogram with cytoband*. It could be visualized with banding information, and require extra columns such as

- name: start with p or q. to tell the different arms of chromosomes. such as **p36.22** and **q12**.
- gieStain: dye color of cytoband. such as **gneg**.

Keep in mind, now, the data need to be transformed into a `GRanges` object. In the following example, we use a default data set in *ggbio* called *hg19IdeogramCyto* to show human ideogram. And a function called `isIdeogram` in package *biovizBase* could be used to check on your data, to see if it contain sufficient information about cytoband and arms or not.

Tips: after *ggbio* 1.5.14, if you doesn't provide any data, you can just provide genome to get it on the fly, or provide nothing, it will give you a list to choose from.

```
p <- plotIdeogram()
```

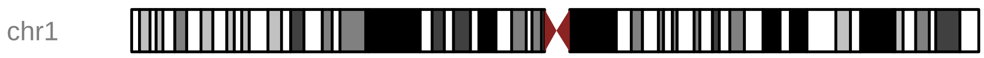
```
Please specify genome

  1: hg19       2: hg18       3: hg17       4: hg16       5: felCat4
  6: felCat3    7: galGal4    8: galGal3    9: galGal2   10: panTro3
 11: panTro2   12: panTro1   13: bosTau7   14: bosTau6   15: bosTau4
 16: bosTau3   17: bosTau2   18: canFam3   19: canFam2   20: canFam1
 21: loxAfr3   22: fr3       23: fr2       24: fr1       25: nomLeu1
 26: gorGor3   27: cavPor3   28: equCab2   29: equCab1   30: petMar1
 31: anoCar2   32: anoCar1   33: calJac3   34: calJac1   35: oryLat2
 36: myoLuc2   37: mm10      38: mm9       39: mm8       40: mm7
 41: hetGla1   42: monDom5   43: monDom4   44: monDom1   45: ponAbe2
 46: chrPic1   47: ailMel1   48: susScr2   49: ornAna1   50: oryCun2
 51: rn5       52: rn4       53: rn3       54: rheMac2   55: oviAri1
 56: gasAcu1   57: echTel1   58: tetNig2   59: tetNig1   60: melGal1
 61: macEug2   62: xenTro3   63: xenTro2   64: xenTro1   65: taeGut1
 66: danRer7   67: danRer6   68: danRer5   69: danRer4   70: danRer3
 71: ci2       72: ci1       73: braFlo1   74: strPur2   75: strPur1
 76: apiMel2   77: apiMel1   78: anoGam1   79: droAna2   80: droAna1
 81: droEre1   82: droGri1   83: dm3       84: dm2       85: dm1
 86: droMoj2   87: droMoj1   88: droPer1   89: dp3       90: dp2
 91: droSec1   92: droSim1   93: droVir2   94: droVir1   95: droYak2
 96: droYak1   97: caePb2    98: caePb1    99: cb3      100: cb1
101: ce10     102: ce6      103: ce4      104: ce2      105: caeJap1
106: caeRem3  107: caeRem2  108: priPac1  109: aplCal1  110: sacCer3
111: sacCer2  112: sacCer1

Selection:
```

Or you could specify genome argument, if you don't specify subchr argument, it will try to parse all information automatically.

> Tips: after first plotting, the data is automatically hooked with the graphic object, when you do edit and zooming, it will not download it anymore.

```
library(ggbio)
## requrie connection
p <- plotIdeogram(genome = "hg19", aspect.ratio = 1/20)


## Loading...

## Done

## use chr1 automatically


p
```

```
## Object of class "ideogram"
```



```
## NULL

## the data stored with p, won't download again for zooming
attr(p, "ideogram.data")

## GRanges with 862 ranges and 2 metadata columns:
##         seqnames               ranges strand   |     name gieStain
##            <Rle>            <IRanges>  <Rle>   | <factor> <factor>
##     [1]     chr1 [       0,  2300000]      *   |   p36.33     gneg
##     [2]     chr1 [ 2300000,  5400000]      *   |   p36.32   gpos25
##     [3]     chr1 [ 5400000,  7200000]      *   |   p36.31     gneg
##     [4]     chr1 [ 7200000,  9200000]      *   |   p36.23   gpos25
##     [5]     chr1 [ 9200000, 12700000]      *   |   p36.22     gneg
##     [6]     chr1 [12700000, 16200000]      *   |   p36.21   gpos50
##     [7]     chr1 [16200000, 20400000]      *   |   p36.13     gneg
##     [8]     chr1 [20400000, 23900000]      *   |   p36.12   gpos25
##     [9]     chr1 [23900000, 28000000]      *   |   p36.11     gneg
##     ...      ...                    ...    ... ...      ...      ...
##   [854]     chrY [ 3000000, 11600000]      *   |    p11.2     gneg
##   [855]     chrY [11600000, 12500000]      *   |    p11.1     acen
##   [856]     chrY [12500000, 13400000]      *   |    q11.1     acen
##   [857]     chrY [13400000, 15100000]      *   |   q11.21     gneg
##   [858]     chrY [15100000, 19800000]      *   |  q11.221   gpos50
##   [859]     chrY [19800000, 22100000]      *   |  q11.222     gneg
##   [860]     chrY [22100000, 26200000]      *   |  q11.223   gpos50
##   [861]     chrY [26200000, 28800000]      *   |   q11.23     gneg
##   [862]     chrY [28800000, 59373566]      *   |      q12     gvar
##   ---
##   seqlengths:
##     chr1 chr10 chr11 chr12 chr13 chr14 ...  chr7  chr8  chr9  chrX  chrY
##       NA    NA    NA    NA    NA    NA ...    NA    NA    NA    NA    NA
```

We will introduce a method to download the data manully and use it through the vignette by a function called `getIdoegram`. The data *hg19IdeogramCyto* is a default data with *ggbio* for convenient use. The argument `aspect.ratio` controls the height/width ratio if you want a fixed plot no matter how you resize the window.

> Tips: `aspect.ratio` by default is NULL, for the reason, when it's passed to `tracks` function, it will cause some issue if you pass a fixed aspect.ratio plot in it.

```
library(biovizBase)
data(hg19IdeogramCyto)
## data structure
hg19IdeogramCyto
```

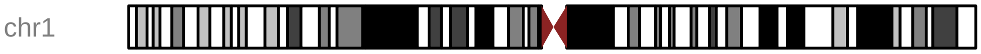
```
## GRanges with 862 ranges and 2 metadata columns:
##         seqnames               ranges strand   |     name gieStain
##            <Rle>            <IRanges>  <Rle>   | <factor> <factor>
##     [1]     chr1 [       0,  2300000]      *   |   p36.33     gneg
##     [2]     chr1 [ 2300000,  5400000]      *   |   p36.32   gpos25
##     [3]     chr1 [ 5400000,  7200000]      *   |   p36.31     gneg
##     [4]     chr1 [ 7200000,  9200000]      *   |   p36.23   gpos25
##     [5]     chr1 [ 9200000, 12700000]      *   |   p36.22     gneg
##     [6]     chr1 [12700000, 16200000]      *   |   p36.21   gpos50
##     [7]     chr1 [16200000, 20400000]      *   |   p36.13     gneg
##     [8]     chr1 [20400000, 23900000]      *   |   p36.12   gpos25
##     [9]     chr1 [23900000, 28000000]      *   |   p36.11     gneg
##     ...      ...                    ...    ... ...      ...      ...
##   [854]     chrY [ 3000000, 11600000]      *   |    p11.2     gneg
##   [855]     chrY [11600000, 12500000]      *   |    p11.1     acen
##   [856]     chrY [12500000, 13400000]      *   |    q11.1     acen
##   [857]     chrY [13400000, 15100000]      *   |   q11.21     gneg
##   [858]     chrY [15100000, 19800000]      *   |  q11.221   gpos50
##   [859]     chrY [19800000, 22100000]      *   |  q11.222     gneg
##   [860]     chrY [22100000, 26200000]      *   |  q11.223   gpos50
##   [861]     chrY [26200000, 28800000]      *   |   q11.23     gneg
##   [862]     chrY [28800000, 59373566]      *   |      q12     gvar
##   ---
##   seqlengths:
##     chr1 chr10 chr11 chr12 chr13 chr14 ...  chr7  chr8  chr9  chrX  chrY
##       NA    NA    NA    NA    NA    NA ...    NA    NA    NA    NA    NA


## return TRUE, if the object could be visualized by ggbio
biovizBase::isIdeogram(hg19IdeogramCyto)


## [1] TRUE
```

When the data is ready to be plotted, as you can tell, most time you only want to visualize a single chromosome, so you need to specify it. To visualize it, in *ggbio*, there two functions to do it, `plotIdeogram`, `plotSingleChrom`, they are just synonyms. If the graphic device is big, resize it to proper size or bind it in tracks use specified height. The plot is shown in Figure 7.1

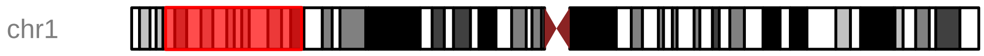
xlim accpet

- nuemric range
- IRanges
- GRanges object, when it's GRanges object, it will change the chromosome if it is not what it is before.

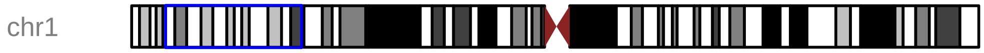
```
plotIdeogram(hg19IdeogramCyto, "chr1", aspect.ratio = 1/20)


## Object of class "ideogram"
```



```
## NULL

plotIdeogram(hg19IdeogramCyto, "chr1", aspect.ratio = 1/20, zoom.region = c(1e+07,
    5e+07))

## Object of class "ideogram"
```



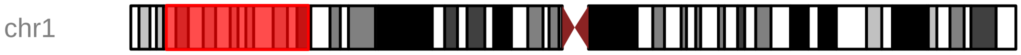
```
## NULL

plotIdeogram(hg19IdeogramCyto, "chr1", aspect.ratio = 1/20, zoom.region = c(1e+07,
    5e+07), fill = NA, color = "blue")

## Object of class "ideogram"
```



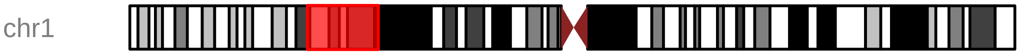
```
## NULL
```

```
p <- plotIdeogram(hg19IdeogramCyto, "chr1", aspect.ratio = 1/20)
p + xlim(1e+07, 5e+07)
```

```
## Object of class "ideogram"
```



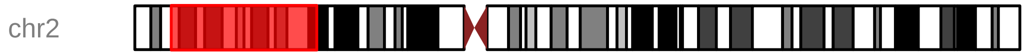
```
## NULL

library(GenomicRanges)
p + xlim(IRanges(5e+07, 7e+07))

## Object of class "ideogram"
```



```
## NULL
```

```
## change seqnames
p + xlim(GRanges("chr2", IRanges(1e+07, 5e+07)))
```

```
## Object of class "ideogram"
```



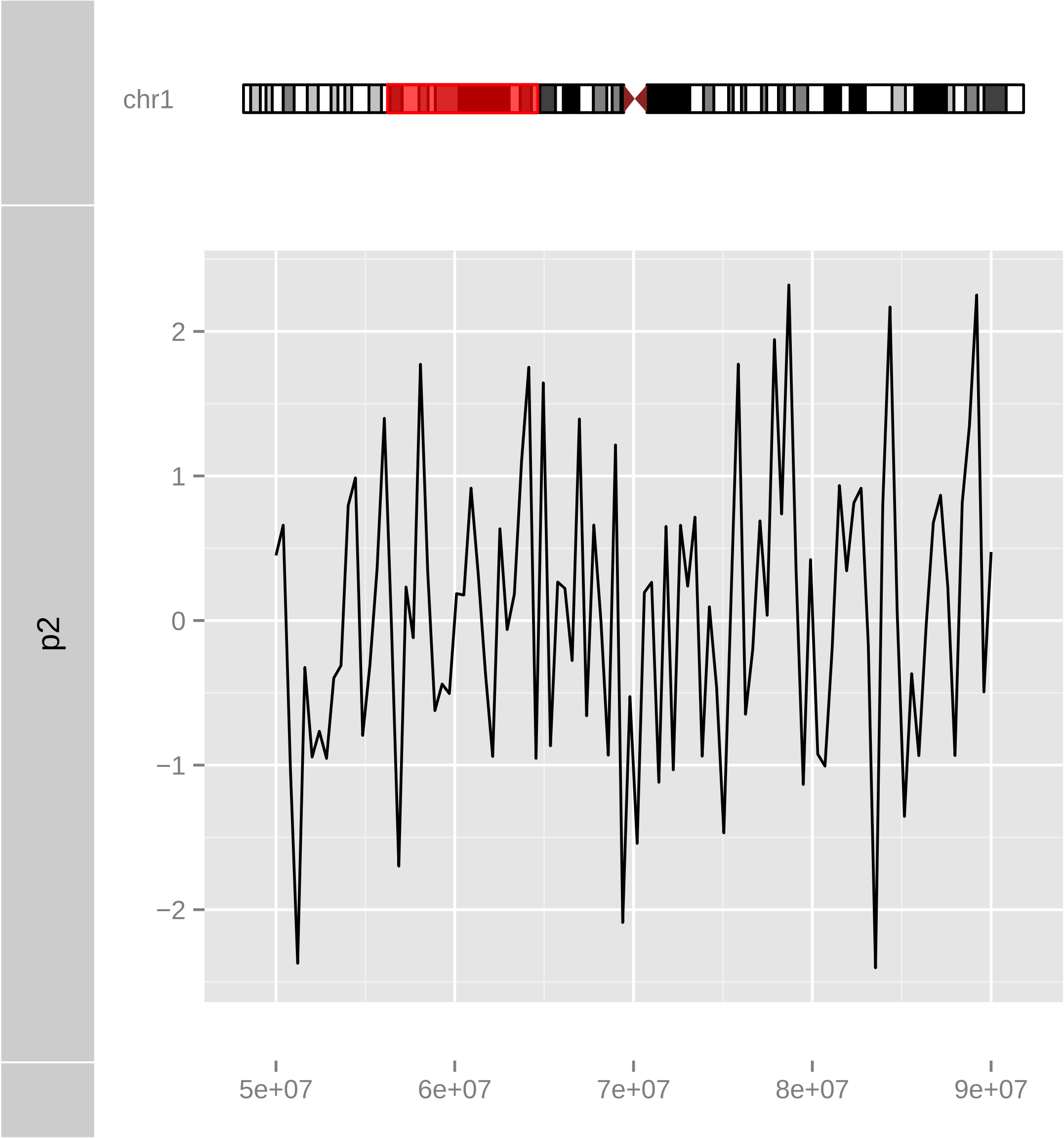
```
## NULL
```

Default ideogram has no X-scale label, to add axis text, you have to specify argument `xlabel` to `TRUE`.

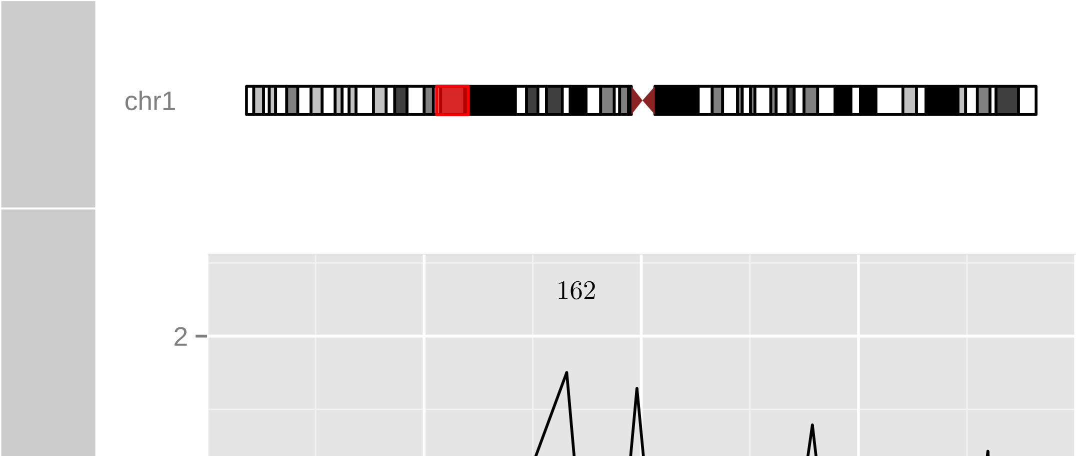
Some time, you don't want to visualize a chromosome with cytobands, or you cannot find any information about cytobands, in this case, you can simply visualize a blank chromosome just to indicate the position. *ggbio* has several ways to do it.

- Use argument `cytoband`. Set it to `FALSE`.
- Pass a GRanges with no extra column such as **name, gieStain**. it will automatically parse and estimate the chromosome lengths. It is **IMPORTANT** that to create an accurate lengths for chromosomes, you need to either make sure the ranges you passed covers all chromosomes or you need to specify the `seqlengths` for our `GRanges` object.

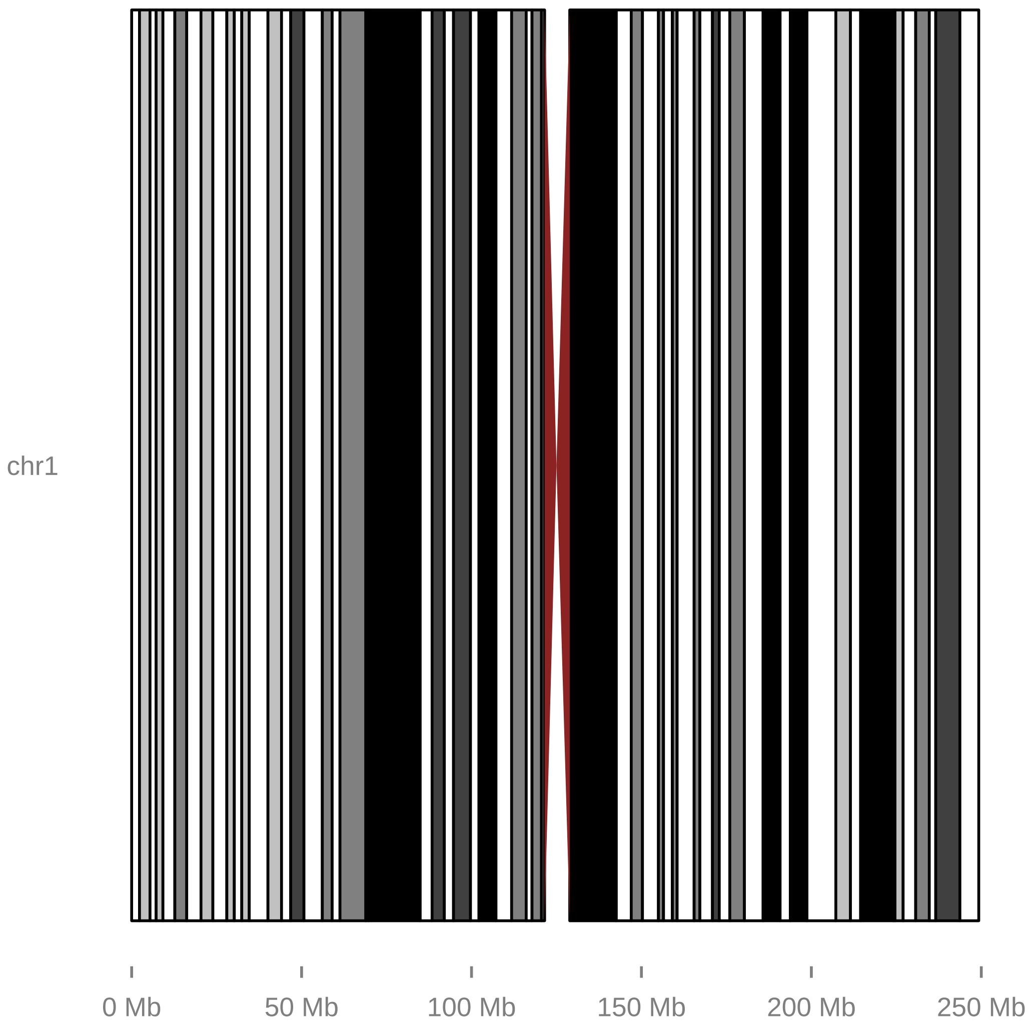
```
p <- plotIdeogram(hg19IdeogramCyto, "chr1")
df <- data.frame(x = seq(from = 5e+07, to = 9e+07, length = 100), y = rnorm(100))
p2 <- qplot(data = df, x = x, y = y, geom = "line") + ylab("")
tracks(p, p2 = p2, heights = c(1.2, 5))
```



```
tracks(p, p2 = p2, heights = c(1.2, 5)) + xlim(6e+07, 7e+07)
```



```
plotIdeogram(hg19IdeogramCyto, "chr1", xlabel = TRUE)

## Object of class "ideogram"
```



```
## NULL
```

Figure 7.2: Idoegram for human chromosome 1 with x scale labeled.

So please make sure the `GRanges` object you passed has an accurate seglengths information, or you are confident the ranges(for example, cytoband) will cover all the chromosome space, otherwise, you will end up with some very inaccurate chromosome lengths and you may NEVER notice from the plot. A example of this is shown in Figure 7.3.

There is another data set called **hg19Ideogram**, with no cytoband information, but with accurate seqlegnths information.

```
data(hg19Ideogram)
head(hg19Ideogram)


## GRanges with 6 ranges and 0 metadata columns:
##                   seqnames         ranges strand
##                      <Rle>      <IRanges>  <Rle>
##   [1]                 chr1 [1, 249250621]      *
##   [2] chr1_gl000191_random [1,    106433]      *
##   [3] chr1_gl000192_random [1,    547496]      *
##   [4]                 chr2 [1, 243199373]      *
##   [5]                 chr3 [1, 198022430]      *
##   [6]                 chr4 [1, 191154276]      *
##   ---
##   seqlengths:
##                     chr1  chr1_gl000191_random ...                  chrM
##                249250621                106433 ...                 16571
```

7.2.2 Get ideogram or customize the colors

We only provide default cytoband ideogram information for human, but what if you want to create your ideogram yourself? There is a high possibility that it's already in UCSC data base, and we can use package *rtracklayer* to download the data from the server.

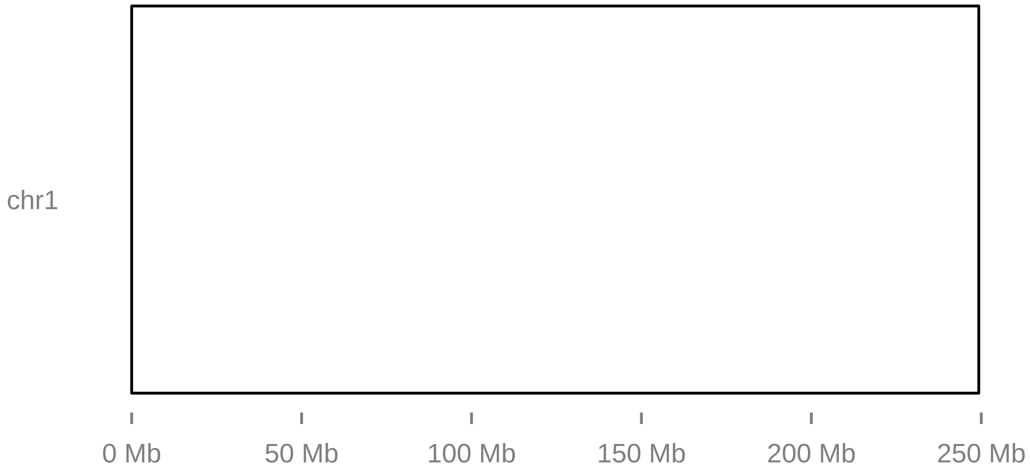
- `ucscGenomes` function in package *rtracklayer* will list all available UCSC genomes.
- You can also use *biovizBase*'s `getIdeogram` function without any arguments, that will give you some items names you can choose from. This function is a convenient wrapper over some functionality in *rtracklayer*.
- Keep in mind, not all available genomes have cytoband information and not all of them have the same default dye names as humans.

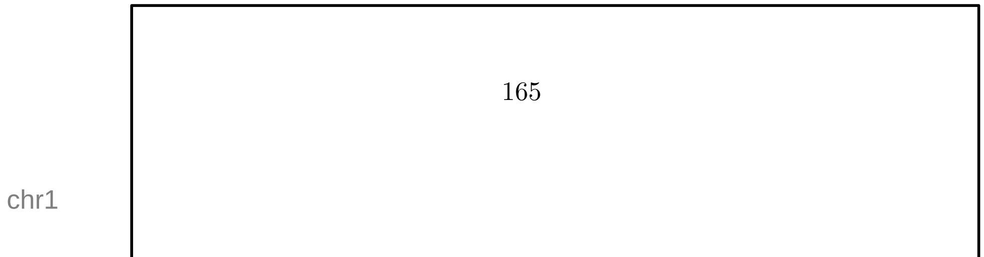
Let's first see how to get available genomes in following examples, we need the **db** column to use them in function `getIdeogram`.

```
library(rtracklayer)
## need UCSC connection
head(ucscGenomes())
```

```
> head(ucscGenomes())
```

```
library(GenomicRanges)
## there are no seqlengths
seqlengths(hg19IdeogramCyto)

##  chr1 chr10 chr11 chr12 chr13 chr14 chr15 chr16 chr17 chr18 chr19  chr2
##    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
## chr20 chr21 chr22  chr3  chr4  chr5  chr6  chr7  chr8  chr9  chrX  chrY
##    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA

## so directly plot will try to aggregate and estimate lengths of
## chromosomes, this is not accurate
data(hg19IdeogramCyto)
p1 <- plotIdeogram(hg19IdeogramCyto, "chr1", cytoband = FALSE, xlabel = TRUE)

  ## Warning:  geom(ideogram) need valid seqlengths information for accurate mapping,
       ##                  now use reduced information as ideogram...

## let's assign a short length to this object
hg19_fake_chr1 <- hg19IdeogramCyto
seqlengths(hg19_fake_chr1)[1] <- 1e+08

      ## Warning:  'ranges' contains values outside of sequence bounds
      ## Warning:  'ranges' contains values outside of sequence bounds

## this will use it's 'seqlengths' information to visualize the
## chromosome.
p2 <- plotIdeogram(hg19_fake_chr1, "chr1", cytoband = FALSE, xlabel = TRUE)

      ## Warning:  'ranges' contains values outside of sequence bounds

## see the difference
alignPlots(p1, p2)
```





```
       db species      date                               name
1    hg19   Human Feb. 2009 Genome Reference Consortium GRCh37
2    hg18   Human Mar. 2006                    NCBI Build 36.1
3    hg17   Human  May 2004                      NCBI Build 35
4    hg16   Human Jul. 2003                      NCBI Build 34
5 felCat4     Cat Dec. 2008                   NHGRI catChrV17e
6 felCat3     Cat Mar. 2006          Broad Institute Release 3
```

getIdeogram without arguments will give you choice to choose from too.

```r
library(biovizBase)
obj <- getIdeogram()
```

```
Please specify genome

  1: hg19       2: hg18       3: hg17       4: hg16       5: felCat4
  6: felCat3    7: galGal4    8: galGal3    9: galGal2   10: panTro3
 11: panTro2   12: panTro1   13: bosTau7   14: bosTau6   15: bosTau4
 16: bosTau3   17: bosTau2   18: canFam3   19: canFam2   20: canFam1
 21: loxAfr3   22: fr3       23: fr2       24: fr1       25: nomLeu1
 26: gorGor3   27: cavPor3   28: equCab2   29: equCab1   30: petMar1
 31: anoCar2   32: anoCar1   33: calJac3   34: calJac1   35: oryLat2
 36: myoLuc2   37: mm10      38: mm9       39: mm8       40: mm7
 41: hetGla1   42: monDom5   43: monDom4   44: monDom1   45: ponAbe2
 46: chrPic1   47: ailMel1   48: susScr2   49: ornAna1   50: oryCun2
 51: rn5       52: rn4       53: rn3       54: rheMac2   55: oviAri1
 56: gasAcu1   57: echTel1   58: tetNig2   59: tetNig1   60: melGal1
 61: macEug2   62: xenTro3   63: xenTro2   64: xenTro1   65: taeGut1
 66: danRer7   67: danRer6   68: danRer5   69: danRer4   70: danRer3
 71: ci2       72: ci1       73: braFlo1   74: strPur2   75: strPur1
 76: apiMel2   77: apiMel1   78: anoGam1   79: droAna2   80: droAna1
 81: droEre1   82: droGri1   83: dm3       84: dm2       85: dm1
 86: droMoj2   87: droMoj1   88: droPer1   89: dp3       90: dp2
 91: droSec1   92: droSim1   93: droVir2   94: droVir1   95: droYak2
 96: droYak1   97: caePb2    98: caePb1    99: cb3      100: cb1
101: ce10     102: ce6      103: ce4      104: ce2      105: caeJap1
106: caeRem3  107: caeRem2  108: priPac1  109: aplCal1  110: sacCer3
111: sacCer2  112: sacCer1

Selection:
```

Function getIdeogram have some control over it.

- subchr argument: to parse a subset of chromosomes information from genome.
- cytoband argument: default is TRUE, try to parse cytoband information, but sometimes you may came across errors when there is no data about cytoband available for certain genomes. We need to get that information manually somewhere else.

Let's try to get a mouse genome from the data base, we know the data base name is **mm9** from above listed choices.

```
library(biovizBase)
## just need information about chromosome lengths
mm9 <- getIdeogram("mm9", cytoband = FALSE)

## Loading...

## Done

## have
head(mm9)

## GRanges with 6 ranges and 0 metadata columns:
##          seqnames         ranges strand
##             <Rle>      <IRanges>  <Rle>
##   [1]        chr1 [1, 197195432]      *
##   [2] chr1_random [1,   1231697]      *
##   [3]        chr2 [1, 181748087]      *
##   [4]        chr3 [1, 159599783]      *
##   [5] chr3_random [1,     41899]      *
##   [6]        chr4 [1, 155630120]      *
##   ---
##   seqlengths:
##            chr1  chr1_random         chr2 ... chrUn_random         chrM
##       197195432      1231697    181748087 ...      5900358        16299

## need information about cytoband
mm9 <- getIdeogram("mm9")

## Loading...

## Done

head(mm9)

## GRanges with 6 ranges and 2 metadata columns:
##       seqnames               ranges strand |     name gieStain
##          <Rle>            <IRanges>  <Rle> | <factor> <factor>
##   [1]     chr1 [       0,  8918386]      * |      qA1  gpos100
##   [2]     chr1 [ 8918386, 12386647]      * |      qA2     gneg
##   [3]     chr1 [12386647, 20314102]      * |      qA3   gpos33
##   [4]     chr1 [20314102, 22295965]      * |      qA4     gneg
##   [5]     chr1 [22295965, 31214352]      * |      qA5  gpos100
##   [6]     chr1 [31214352, 43601000]      * |       qB     gneg
##   ---
##   seqlengths:
##     chr1 chr10 chr11 chr12 chr13 chr14 ...  chr7  chr8  chr9  chrX  chrY
##       NA    NA    NA    NA    NA    NA ...    NA    NA    NA    NA    NA
## with extra column 'name' and 'gieStain'.
```

Now we have to explain where define our default cytoband color, it's in an option list of package *biovizBase*. Later you will notice some of the staining color for 'mm9' is not defined in our color list.

```
cyto.def <- getOption("biovizBase")$cytobandColor
cyto.def

##      gneg     stalk      acen      gpos      gvar     gpos1     gpos2
## "grey100"  "brown3"  "brown4"   "grey0"   "grey0" "#FFFFFF" "#FCFCFC"
##     gpos3     gpos4     gpos5     gpos6     gpos7     gpos8     gpos9
## "#F9F9F9" "#F7F7F7" "#F4F4F4" "#F2F2F2" "#EFEFEF" "#ECECEC" "#EAEAEA"
##    gpos10    gpos11    gpos12    gpos13    gpos14    gpos15    gpos16
## "#E7E7E7" "#E5E5E5" "#E2E2E2" "#E0E0E0" "#DDDDDD" "#DADADA" "#D8D8D8"
##    gpos17    gpos18    gpos19    gpos20    gpos21    gpos22    gpos23
## "#D5D5D5" "#D3D3D3" "#D0D0D0" "#CECECE" "#CBCBCB" "#C8C8C8" "#C6C6C6"
##    gpos24    gpos25    gpos26    gpos27    gpos28    gpos29    gpos30
## "#C3C3C3" "#C1C1C1" "#BEBEBE" "#BCBCBC" "#B9B9B9" "#B6B6B6" "#B4B4B4"
##    gpos31    gpos32    gpos33    gpos34    gpos35    gpos36    gpos37
## "#B1B1B1" "#AFAFAF" "#ACACAC" "#AAAAAA" "#A7A7A7" "#A4A4A4" "#A2A2A2"
##    gpos38    gpos39    gpos40    gpos41    gpos42    gpos43    gpos44
## "#9F9F9F" "#9D9D9D" "#9A9A9A" "#979797" "#959595" "#929292" "#909090"
##    gpos45    gpos46    gpos47    gpos48    gpos49    gpos50    gpos51
## "#8D8D8D" "#8B8B8B" "#888888" "#858585" "#838383" "#808080" "#7E7E7E"
##    gpos52    gpos53    gpos54    gpos55    gpos56    gpos57    gpos58
## "#7B7B7B" "#797979" "#767676" "#737373" "#717171" "#6E6E6E" "#6C6C6C"
##    gpos59    gpos60    gpos61    gpos62    gpos63    gpos64    gpos65
## "#696969" "#676767" "#646464" "#616161" "#5F5F5F" "#5C5C5C" "#5A5A5A"
##    gpos66    gpos67    gpos68    gpos69    gpos70    gpos71    gpos72
## "#575757" "#545454" "#525252" "#4F4F4F" "#4D4D4D" "#4A4A4A" "#484848"
##    gpos73    gpos74    gpos75    gpos76    gpos77    gpos78    gpos79
## "#454545" "#424242" "#404040" "#3D3D3D" "#3B3B3B" "#383838" "#363636"
##    gpos80    gpos81    gpos82    gpos83    gpos84    gpos85    gpos86
## "#333333" "#303030" "#2E2E2E" "#2B2B2B" "#292929" "#262626" "#242424"
##    gpos87    gpos88    gpos89    gpos90    gpos91    gpos92    gpos93
## "#212121" "#1E1E1E" "#1C1C1C" "#191919" "#171717" "#141414" "#121212"
##    gpos94    gpos95    gpos96    gpos97    gpos98    gpos99   gpos100
## "#0F0F0F" "#0C0C0C" "#0A0A0A" "#070707" "#050505" "#020202" "#000000"

setdiff(unique(values(mm9)$gieStain), names(cyto.def))

## character(0)
```

We notice gieStain *gpos33, gpos66* is not defined in default, if we directly plot them, those region will be blank. Otherwise, we could

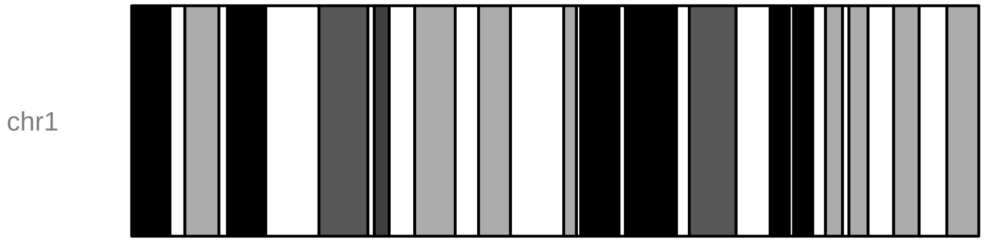
- either edited the default color option list
- or use *ggplot2* lower level utilities.

In the following code, we compare a incomplete color default with customized color.

```
p1 <- plotIdeogram(mm9, "chr1")
cyto.def
```

```
##      gneg     stalk      acen      gpos      gvar     gpos1     gpos2
## "grey100"  "brown3"  "brown4"   "grey0"   "grey0" "#FFFFFF" "#FCFCFC"
##     gpos3     gpos4     gpos5     gpos6     gpos7     gpos8     gpos9
## "#F9F9F9" "#F7F7F7" "#F4F4F4" "#F2F2F2" "#EFEFEF" "#ECECEC" "#EAEAEA"
##    gpos10    gpos11    gpos12    gpos13    gpos14    gpos15    gpos16
## "#E7E7E7" "#E5E5E5" "#E2E2E2" "#E0E0E0" "#DDDDDD" "#DADADA" "#D8D8D8"
##    gpos17    gpos18    gpos19    gpos20    gpos21    gpos22    gpos23
## "#D5D5D5" "#D3D3D3" "#D0D0D0" "#CECECE" "#CBCBCB" "#C8C8C8" "#C6C6C6"
##    gpos24    gpos25    gpos26    gpos27    gpos28    gpos29    gpos30
## "#C3C3C3" "#C1C1C1" "#BEBEBE" "#BCBCBC" "#B9B9B9" "#B6B6B6" "#B4B4B4"
##    gpos31    gpos32    gpos33    gpos34    gpos35    gpos36    gpos37
## "#B1B1B1" "#AFAFAF" "#ACACAC" "#AAAAAA" "#A7A7A7" "#A4A4A4" "#A2A2A2"
##    gpos38    gpos39    gpos40    gpos41    gpos42    gpos43    gpos44
## "#9F9F9F" "#9D9D9D" "#9A9A9A" "#979797" "#959595" "#929292" "#909090"
##    gpos45    gpos46    gpos47    gpos48    gpos49    gpos50    gpos51
## "#8D8D8D" "#8B8B8B" "#888888" "#858585" "#838383" "#808080" "#7E7E7E"
##    gpos52    gpos53    gpos54    gpos55    gpos56    gpos57    gpos58
## "#7B7B7B" "#797979" "#767676" "#737373" "#717171" "#6E6E6E" "#6C6C6C"
##    gpos59    gpos60    gpos61    gpos62    gpos63    gpos64    gpos65
## "#696969" "#676767" "#646464" "#616161" "#5F5F5F" "#5C5C5C" "#5A5A5A"
##    gpos66    gpos67    gpos68    gpos69    gpos70    gpos71    gpos72
## "#575757" "#545454" "#525252" "#4F4F4F" "#4D4D4D" "#4A4A4A" "#484848"
##    gpos73    gpos74    gpos75    gpos76    gpos77    gpos78    gpos79
## "#454545" "#424242" "#404040" "#3D3D3D" "#3B3B3B" "#383838" "#363636"
##    gpos80    gpos81    gpos82    gpos83    gpos84    gpos85    gpos86
## "#333333" "#303030" "#2E2E2E" "#2B2B2B" "#292929" "#262626" "#242424"
##    gpos87    gpos88    gpos89    gpos90    gpos91    gpos92    gpos93
## "#212121" "#1E1E1E" "#1C1C1C" "#191919" "#171717" "#141414" "#121212"
##    gpos94    gpos95    gpos96    gpos97    gpos98    gpos99   gpos100
## "#0F0F0F" "#0C0C0C" "#0A0A0A" "#070707" "#050505" "#020202" "#000000"
```

```
cyto.new <- c(cyto.def, c(gpos33 = "grey80", gpos66 = "grey60"))
## method 1:
optlist <- getOption("biovizBase")
optlist$cytobandColor <- cyto.new
options(biovizBase = optlist)
p2 <- plotIdeogram(mm9, "chr1")
p3 <- plotIdeogram(mm9, "chr1") + scale_fill_manual(values = cyto.new)
alignPlots(p1, p2, p3)
```



7.2.3 Plot ideogram directly from Seqinfo

More information could be found in autoplot tutorial.

```
data(hg19Ideogram)
seqs <- seqinfo(hg19Ideogram)
class(seqs)

## [1] "Seqinfo"
## attr(,"package")
## [1] "GenomicRanges"

p1 <- autoplot(seqs["chr1"])
p2 <- autoplot(seqs["chr1"], FALSE)
tracks(type1 = p1, type2 = p2)
```

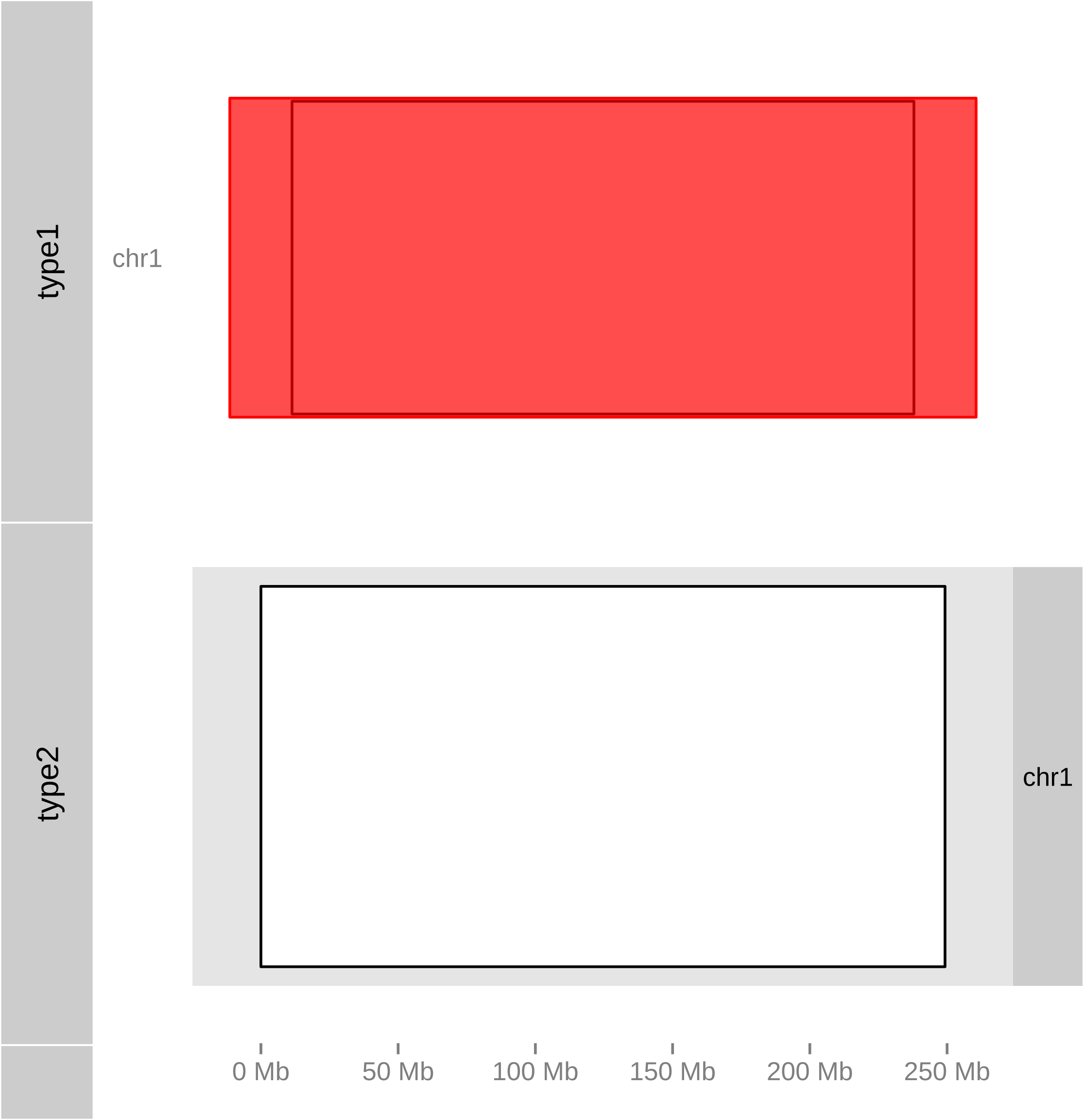


Figure 7.4: Plot Seqinfo object.

Chapter 8

Visualize genomic features

8.1 Introduction

Transcript-centric annotation is one of the most useful tracks that frequently aligned with other data in many genome browsers. In Bioconductor, you can either request data on the fly from UCSC or BioMart, which require internet connection, or you can save frequently used annotation data of particular organism, for example human genome, as a local data base. Package *GenomicFeatures* provides very convenient API for making and manipulating such database. Bioconductor also pre-built some frequently used genome annotation as packages for easy installation, for instance, for human genome(hg19), there is a meta data package called *TxDb.Hsapiens.UCSC.hg19.knownGene*, after you load this package, a `TranscriptDb` object called `TxDb.Hsapiens.UCSC.hg19.knownGene` will be visible from your workspace. This object contains information like coding regions, exons, introns, utrs, transcripts for this genome. If you cannot find the organism you want in Bioconductor meta packages, please refer to the vignette of package *GenomicFeatures* to check how to build your own data base manually.

ggbio providing visualization utilities based on this specific object, in the following tutorial we cover some usage:

- How to plot genomic features for certain region, including coding region, introns, utrs.
- How to change geom of introns, how to revise arrow size and density.
- How to change aesthetics such as colors.
- How to plot single genomic features by make statistical transformation of “reduce”.
- How to revise y label using expression and pattern.
- How to change x-scale unit to arbitrary *kb,bp*.
- How to use lower level API.

8.2 Usage

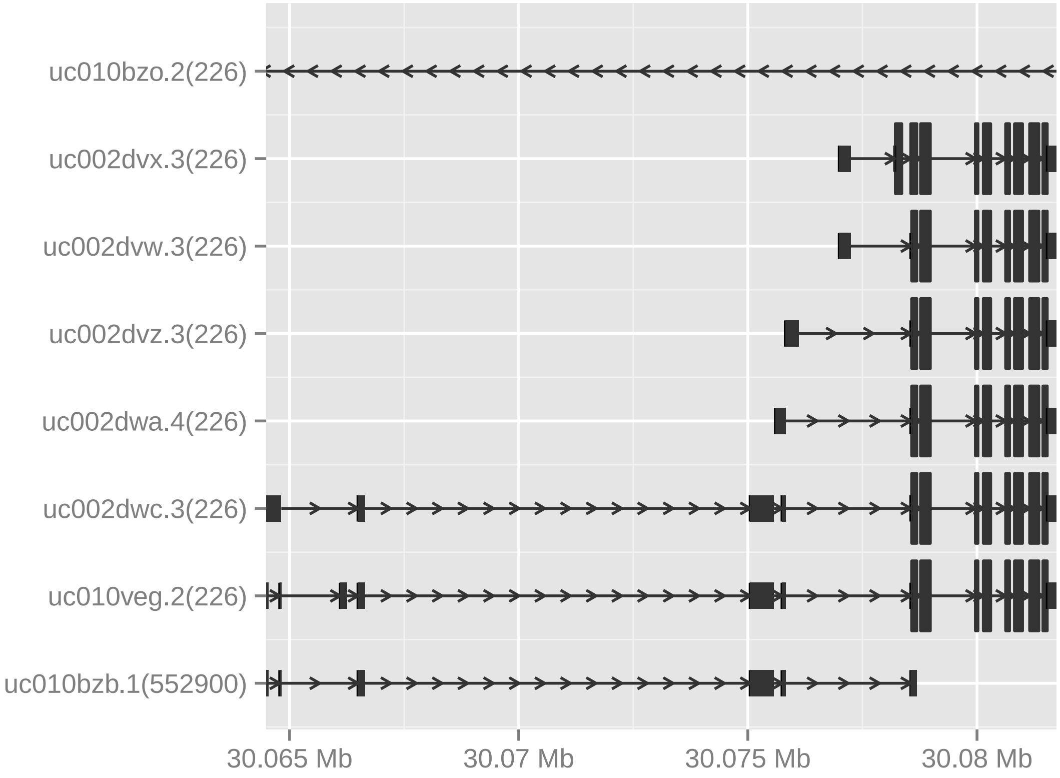
8.2.1 autoplot

autoplot API is higher level API in *ggbio* which tries to make smart decision for object-oriented graphics. Another vignette have more detailed introduction to this function.

In this tutorial, we solely focus on visualization of TranscriptDb object.

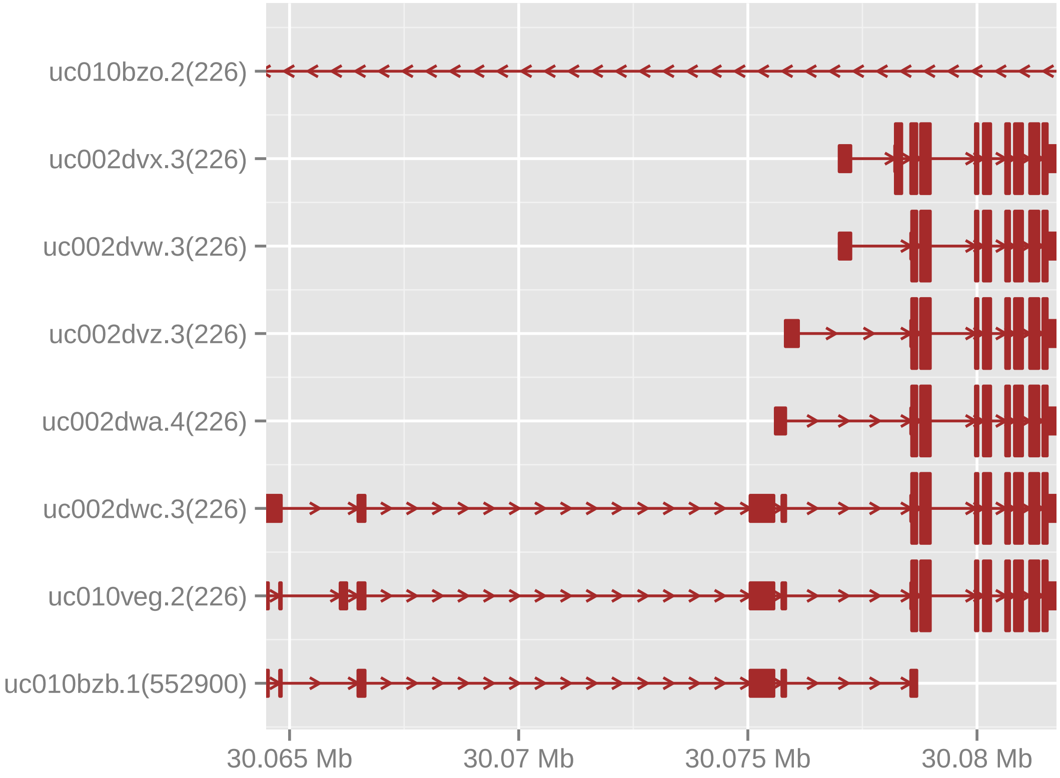
```
library(TxDb.Hsapiens.UCSC.hg19.knownGene)
txdb <- TxDb.Hsapiens.UCSC.hg19.knownGene
## suppose you already know the region you want to visualize or for
## human genome, you can try following commented code data(genesymbol,
## package = 'biovizBase') genesymbol['ALDOA']
aldoa.gr <- GRanges("chr16", IRanges(30064491, 30081734))
aldoa.gr


## GRanges with 1 range and 0 metadata columns:
##       seqnames               ranges strand
##          <Rle>            <IRanges>  <Rle>
##   [1]    chr16 [30064491, 30081734]      *
##   ---
##   seqlengths:
##    chr16
##       NA
```

```
library(ggbio)
p1 <- autoplot(txdb, which = aldoa.gr)


## Aggregating TranscriptDb...

## Parsing exons...

## Parsing cds...

## Parsing transcripts...

## Aggregating...

## Done

## Constructing graphics...


p1
```



You can changing some aesthetics like colors in `autoplot`, since rectangle is defined by 'color' which is border color and 'fill' for filled color.

```
library(ggbio)
p1 <- autoplot(txdb, which = aldoa.gr, fill = "brown", color = "brown")


## Aggregating TranscriptDb...

## Parsing exons...

## Parsing cds...

## Parsing transcripts...

## Aggregating...

## Done

## Constructing graphics...


p1
```

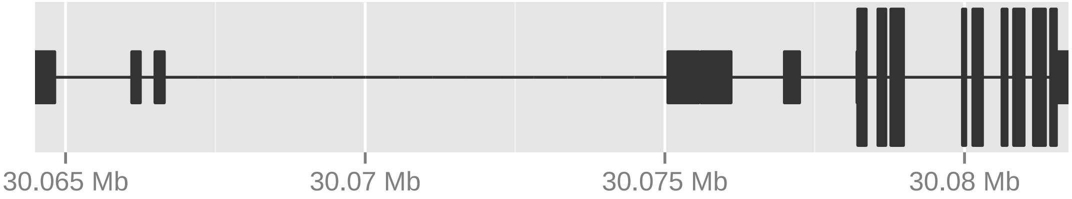


`autoplot` function for object `TranscriptDb` has two supported statistical transformation.

- **identity**: full model, show each transcript, parsing coding region, introns and utrs automatically from the database. interons are shown as small arrows to indicate the direction, exons are represented as wider rectangles and utrs are represented as narrow rectangles. This transformation is shown in Figure ??
- **reduce**: reduced model, show single reduced model, which take union of CDS, utrs and re-compute introns, as shown in Figure ??.

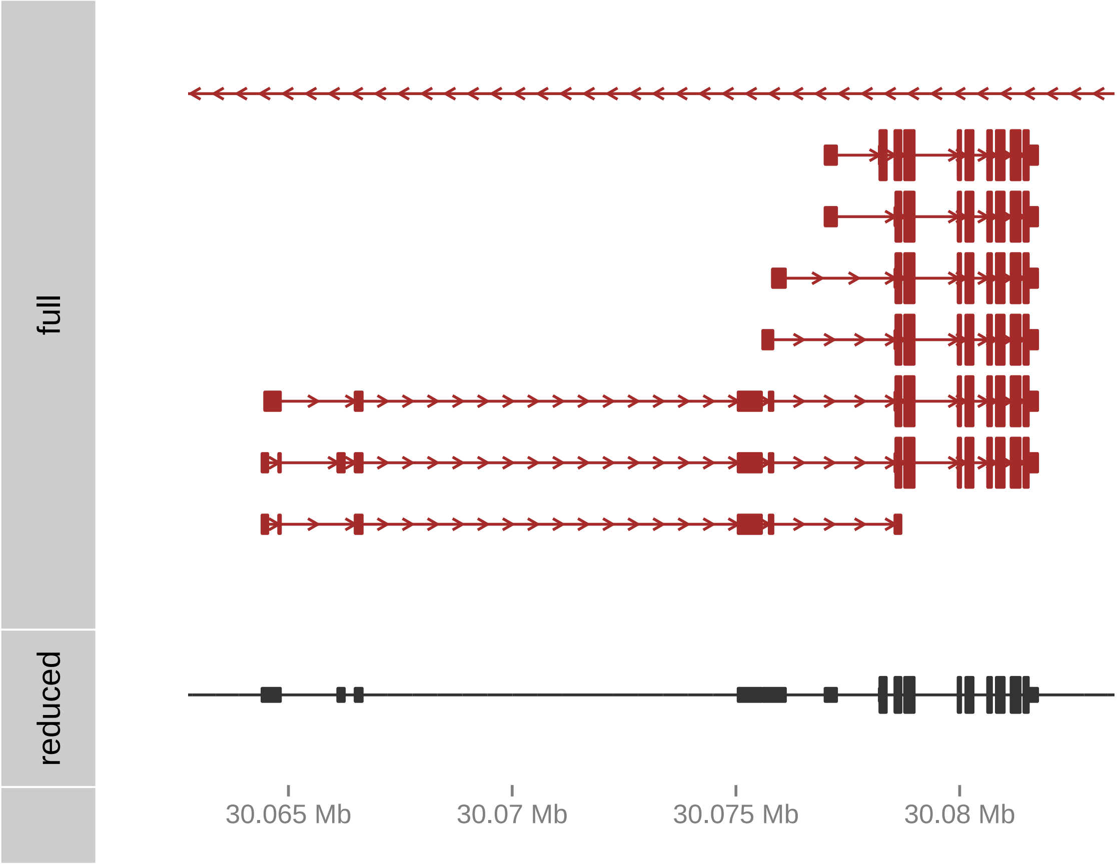
```
p2 <- autoplot(txdb, which = aldoa.gr, stat = "reduce")

## Aggregating TranscriptDb...

## Parsing exons...

## Parsing cds...

## Parsing transcripts...

## Aggregating...

## Done

## Constructing graphics...
```

```r
print(p2)
```



To better understand the behavior of "reduce" transformation, we layout these two graphics by tracks as shown in Figure ??. Function Tracks has been introduced in detail in another vignette.

```r
tracks(full = p1, reduced = p2, heights = c(4, 1)) + theme_alignment(grid = FALSE,
    border = FALSE)


## Scale for 'y' is already present.  Adding another scale for 'y', which will replace the
existing scale.

## Scale for 'y' is already present.  Adding another scale for 'y', which will replace the
existing scale.
```

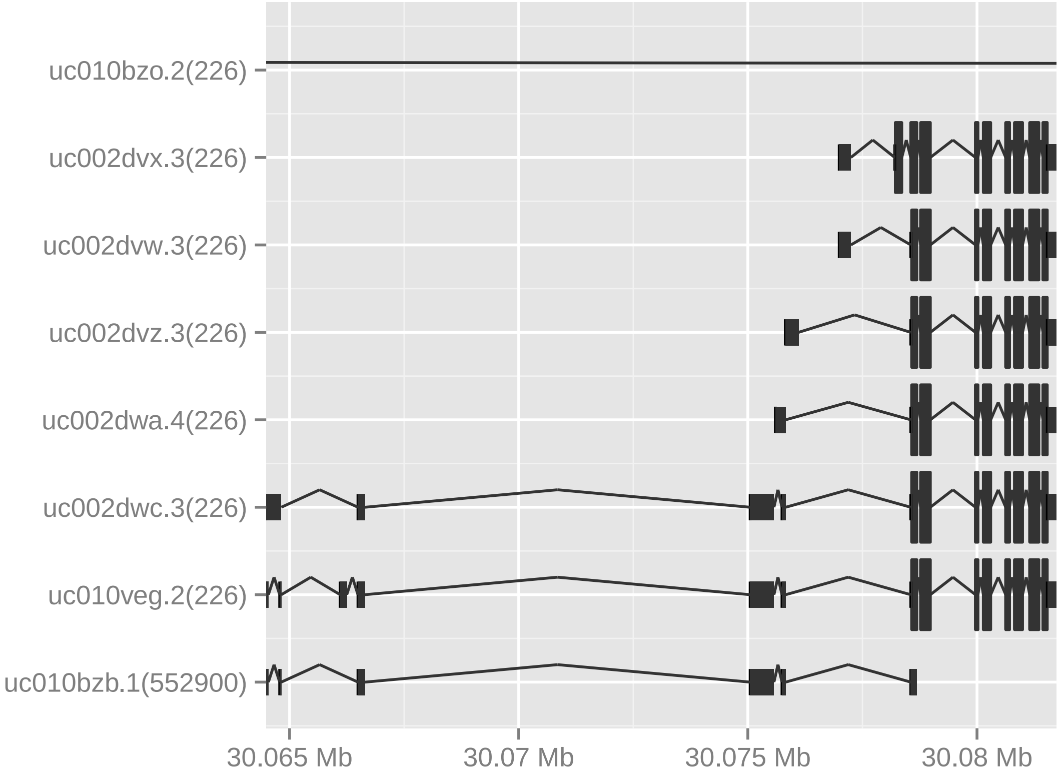


We allow users to change the way to visualization introns here, it's controlled by parameter "gap.geom", supported three geoms:

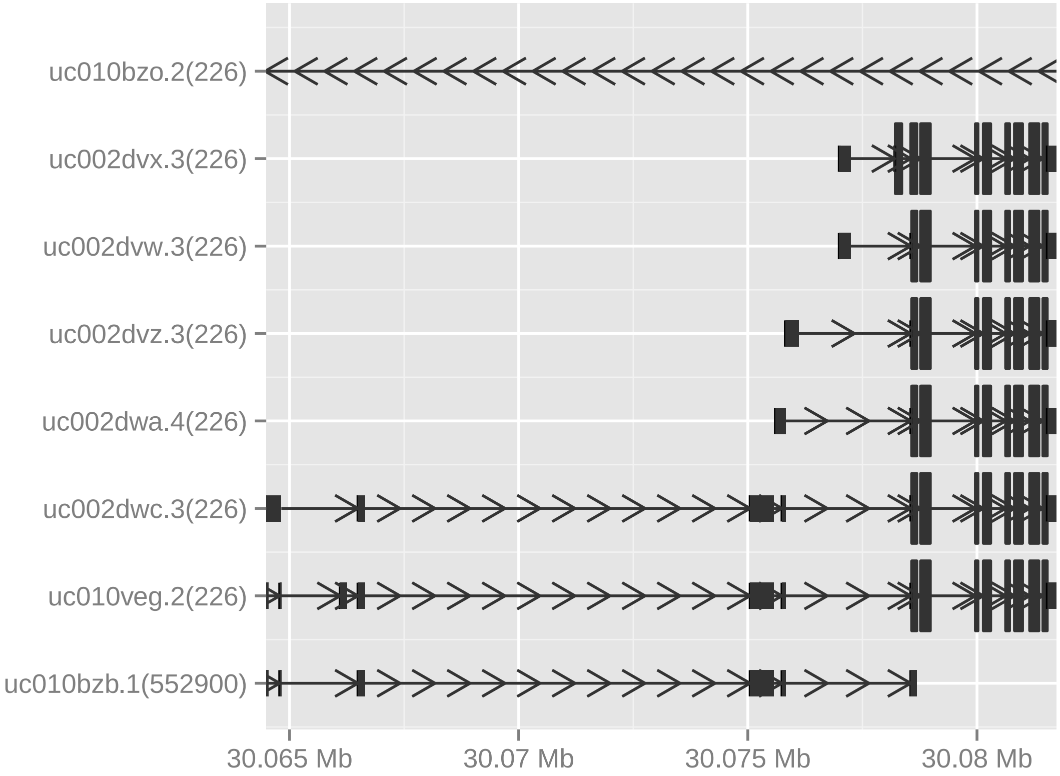
- **arrow**: with small arrow to indicate the strand direction, extra parameter existing to control the appearance of the arrow, as shown in Figure ??. **arrow.rate** control how dense the arrows shows in between.
- **chevron**:chevron to show as introns, no strand indication. please check geom_chevron.
- **segment**:segments to show as introns, no strand indication.

The geometric object for ranges, introns and uts are controled by parameters `range.geom, gap.geom, utr.geom`. For example if you want to change the geom for gap, just change the `gap.geom`.

```
autoplot(txdb, which = aldoa.gr, gap.geom = "chevron")

## Aggregating TranscriptDb...

## Parsing exons...

## Parsing cds...

## Parsing transcripts...

## Aggregating...
## Done

## Constructing graphics...
```

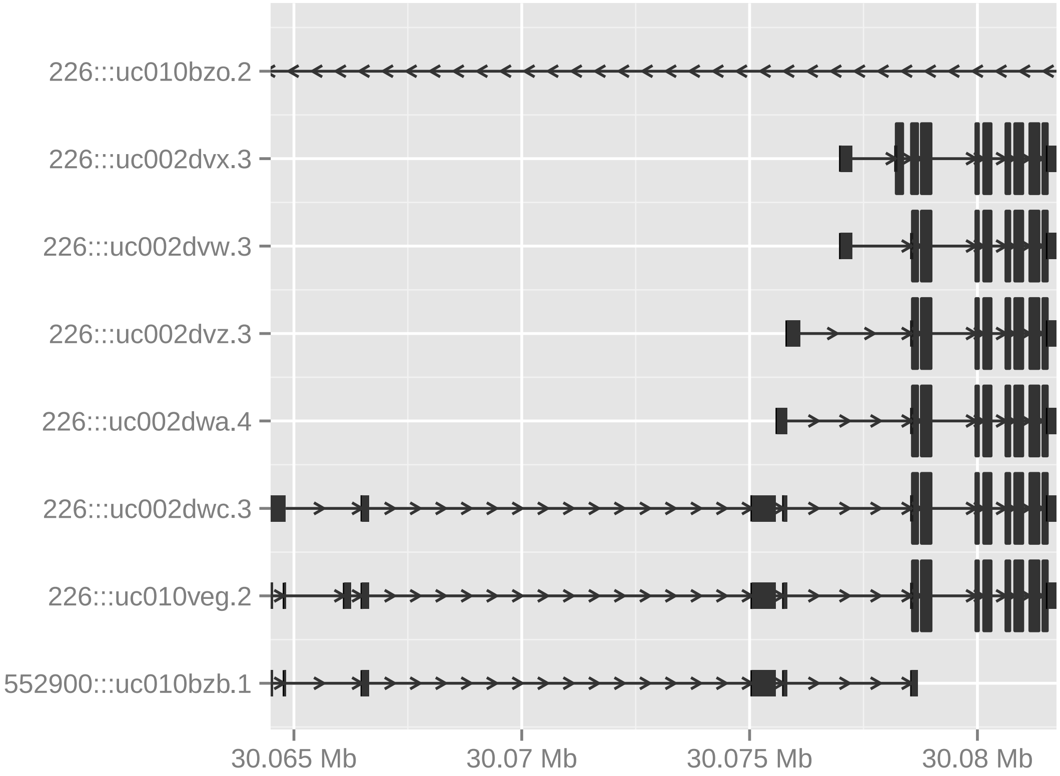


```
library(grid)
autoplot(txdb, which = aldoa.gr, arrow.rate = 0.001, length = unit(0.35,
    "cm"))


## Aggregating TranscriptDb...

## Parsing exons...

## Parsing cds...

## Parsing transcripts...

## Aggregating...

## Done

## Constructing graphics...
```



We also allow users to parse y labels from existing column in `TranscriptDb` object.

```
p <- autoplot(txdb, which = aldoa.gr, names.expr = "gene_id:::tx_name")

## Aggregating TranscriptDb...

## Parsing exons...

## Parsing cds...

## Parsing transcripts...

## Aggregating...

## Done

## Constructing graphics...

p
```


226:::uc010bzo.2
226:::uc002dvx.3
226:::uc002dvw.3
226:::uc002dvz.3
226:::uc002dwa.4
226:::uc002dwc.3
226:::uc010veg.2
552900:::uc010bzb.1
30.065 Mb
30.07 Mb
30.075 Mb
30.08 Mb

`scale_x_sequnit` is a add-on utility to revise the x-scale, it provides three unit

- **mb**: 1e6bp unit. default for autoplot,TranscriptDb.
- **kb**: 1e3bp unit.
- **bp**: 1bp unit

it's just post-graphic modification, won't re-load the parsing process. Figure

```
p + scale_x_sequnit("kb")

## Scale for ’x’ is already present.  Adding another scale for ’x’, which will replace
the existing scale.
```

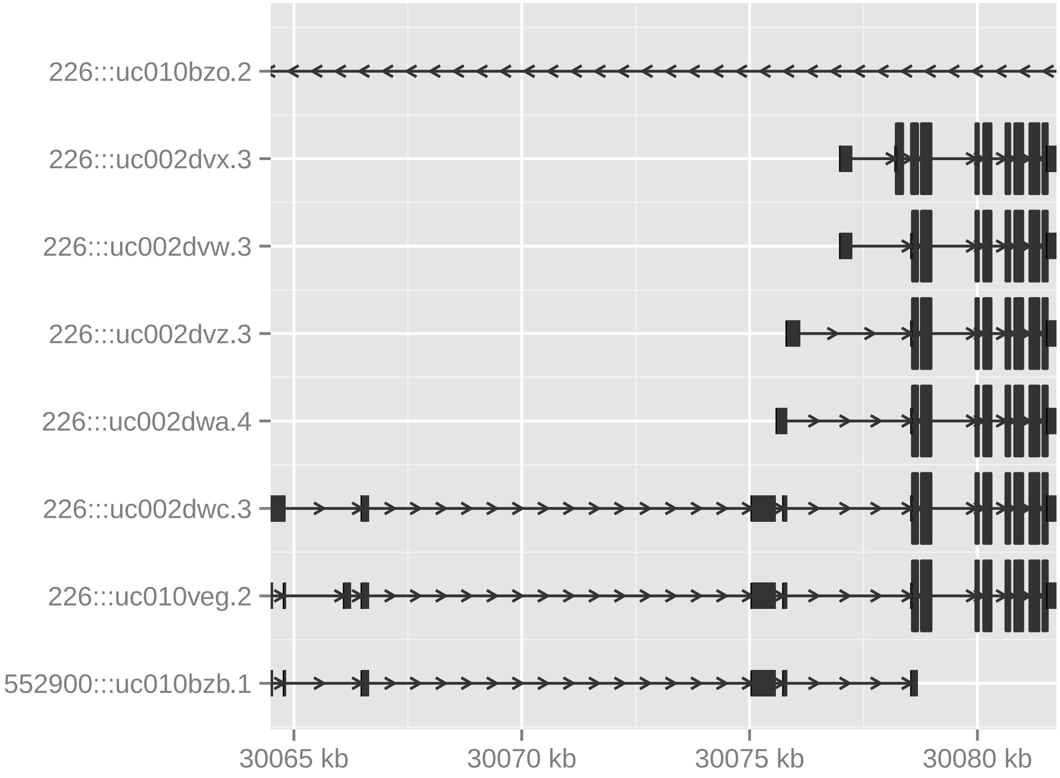


Figure 8.1: change the unit to kb.

8.2.2 geom_alignment

`stat_gene` is deprecated, and `geom_alignment` is the lower level API which facilitate construction layer by layer.

```r
p1 <- ggplot() + geom_alignment(txdb, which = aldoa.gr)
```

Chapter 9

Circular view

9.1 Introduction

Layout "circle" is inspired by *Circos*

graphics and make it a general layout. Layout is generally more complex than a coordinate transformation, it's a combination of different components like coordinate transformation(genome and polar), and tracks-based layout, etc. Especially, circular view is very useful to show links between different locations. Since we are following the grammar of graphics, aesthetics mapping are fairly easy in *ggbio*.

In this tutorial, we will start from the raw data, if you are already familiar with how to process your data into the right format, which here I mean `GRanges`,you can jump to 9.2.3 directly.

9.2 Tutorial

9.2.1 Step 1: understand the layout circle

We have discussed about the new coordinate "genome" in vignette about Manhattan plot before, now this time, it's one step further compared to genome coordinate transformation. We specify ring radius `radius` and track width `trackWidth` to help transform a linear genome coordinate system to a circular coordinate system. By using `layout_circle` function which we will introduce later.

Before we visualize our data, we need to have something in mind

- How many tracks we want?
- Can they be combined into the same data?
- Do I have chromosomes lengths information?
- Do I have interesting variables attached as one column?

9.2.2 Step 2: get your data ready to plot

Ok, let's start to process some raw data to the format we want. The data used in this study is from this a paper[1]. In this example, We are going to

1. Visualize somatic mutation as segment.
2. Visualize inter,intro-chromosome rearrangement as links.
3. Visualize mutation score as point tracks with grid-background.
4. Add scale and ticks and labels.
5. To arrange multiple plots and legend. create multiple sample comparison.

Notes: don't put too much tracks on it.

I simply put script here to get mutation data as 'GRanges' object.

```
crc1 <- system.file("extdata", "crc1-missense.csv", package = "biovizBase")
crc1 <- read.csv(crc1)
library(GenomicRanges)
mut.gr <- with(crc1, GRanges(Chromosome, IRanges(Start_position, End_position),
    strand = Strand))
values(mut.gr) <- subset(crc1, select = -c(Start_position, End_position,
    Chromosome))
data("hg19Ideogram", package = "biovizBase")
seqs <- seqlengths(hg19Ideogram)
## subset_chr
chr.sub <- paste("chr", 1:22, sep = "")
## levels tweak
seqlevels(mut.gr) <- c(chr.sub, "chrX")
mut.gr <- keepSeqlevels(mut.gr, chr.sub)
seqs.sub <- seqs[chr.sub]
## remove wrong position
bidx <- end(mut.gr) <= seqs.sub[match(as.character(seqnames(mut.gr)), names(seqs.sub))]
mut.gr <- mut.gr[which(bidx)]
## assign_seqlengths
seqlengths(mut.gr) <- seqs.sub
## reanme to shorter names
new.names <- as.character(1:22)
names(new.names) <- paste("chr", new.names, sep = "")
new.names
```

```
##  chr1  chr2  chr3  chr4  chr5  chr6  chr7  chr8  chr9 chr10 chr11 chr12
##   "1"   "2"   "3"   "4"   "5"   "6"   "7"   "8"   "9"  "10"  "11"  "12"
## chr13 chr14 chr15 chr16 chr17 chr18 chr19 chr20 chr21 chr22
##  "13"  "14"  "15"  "16"  "17"  "18"  "19"  "20"  "21"  "22"
```

[1] http://www.nature.com/ng/journal/v43/n10/full/ng.936.html

```
mut.gr.new <- renameSeqlevels(mut.gr, new.names)
head(mut.gr.new)

## GRanges with 6 ranges and 10 metadata columns:
##       seqnames                 ranges strand | Hugo_Symbol
##          <Rle>              <IRanges>  <Rle> |    <factor>
##   [1]        1 [ 11003085,  11003085]      + |      TARDBP
##   [2]        1 [ 62352395,  62352395]      + |       INADL
##   [3]        1 [194960885, 194960885]      + |         CFH
##   [4]        2 [ 10116508,  10116508]      - |        CYS1
##   [5]        2 [ 33617747,  33617747]      + |     RASGRP3
##   [6]        2 [ 73894280,  73894280]      + |     C2orf78
##       Entrez_Gene_Id   Center NCBI_Build   Strand
##            <integer> <factor>  <integer> <factor>
##   [1]          23435    Broad         36        +
##   [2]          10207    Broad         36        +
##   [3]           3075    Broad         36        +
##   [4]         192668    Broad         36        -
##   [5]          25780    Broad         36        +
##   [6]         388960    Broad         36        +
##       Variant_Classification Variant_Type Reference_Allele
##                     <factor>     <factor>         <factor>
##   [1]               Missense          SNP                G
##   [2]               Missense          SNP                T
##   [3]               Missense          SNP                G
##   [4]               Missense          SNP                C
##   [5]               Missense          SNP                C
##   [6]               Missense          SNP                T
##       Tumor_Seq_Allele1 Tumor_Seq_Allele2
##              <factor>          <factor>
##   [1]               G                 A
##   [2]               T                 G
##   [3]               G                 A
##   [4]               C                 T
##   [5]               C                 T
##   [6]               T                 C
##   ---
##   seqlengths:
##            1         2         3 ...        20        21        22
##    249250621 243199373 198022430 ...  63025520  48129895  51304566
```

To get ideogram track, we need to load human hg19 ideogram data, for details please check another vignette about getting ideogram.

```
hg19Ideo <- hg19Ideogram
hg19Ideo <- keepSeqlevels(hg19Ideogram, chr.sub)
hg19Ideo <- renameSeqlevels(hg19Ideo, new.names)
head(hg19Ideo)

## GRanges with 6 ranges and 0 metadata columns:
```

```
##       seqnames         ranges strand
##          <Rle>      <IRanges>  <Rle>
##   [1]        1 [1, 249250621]      *
##   [2]        2 [1, 243199373]      *
##   [3]        3 [1, 198022430]      *
##   [4]        4 [1, 191154276]      *
##   [5]        5 [1, 180915260]      *
##   [6]        6 [1, 171115067]      *
##   ---
##   seqlengths:
##            1         2         3 ...        20        21        22
##     249250621 243199373 198022430 ...  63025520  48129895  51304566
```

9.2.3 Step 3: low level API: `layout_circle`

`layout_circle` is a lower level API for creating circular plot, it accepts `Granges` object, and users need to specify radius, track width, and other aesthetics, it's very flexible. But keep in mind, you **have to** pay attention rules when you make circular plots.

- For now, `seqlengths`, `seqlevels` and chromosomes names should be exactly the same, so you have to make sure data on all tracks have this uniform information to make a comparison.
- Set arguments `space.skip` to the same value for all tracks, that matters for transformation, default is the same, so you don't have to change it, unless you want to add/remove space in between.
- `direction` argument should be exactly the same, either "clockwise" or "counterclockwise".
- Tweak with your radius and tracks width to get best results.

Since low level API leave you as much flexibility as possible, this may looks hard to adjust, but it can produce various types of graphics which higher levels API like `autoplot` hardly can, for instance, if you want to overlap multiple tracks or fine-tune your layout.

Ok, let's start to add tracks one by one.

First to add a "ideo" track

Then a "scale" track with ticks

Then a "text" track to label chromosomes. *NOTICE*, after genome coordinate transformation, original data will be stored in column ".ori", and for mapping, just use ".ori" prefix to it. Here we use '.ori.seqnames', if you use 'seqnames', that is going to be just "genome" character.

Then a "rectangle" track to show somatic mutation, this will looks like vertical segments.

Next, we need to add some "links" to show the rearrangement, of course, links can be used to map any kind of association between two or more different locations to indicate relationships like copies or fusions.

```
rearr <- read.csv(system.file("extdata", "crc-rearrangment.csv", package = "biovizBase"))
## start position
```

```
library(ggbio)
p <- ggplot() + layout_circle(hg19Ideo, geom = "ideo", fill = "gray70", radius = 30,
    trackWidth = 4)
p
```

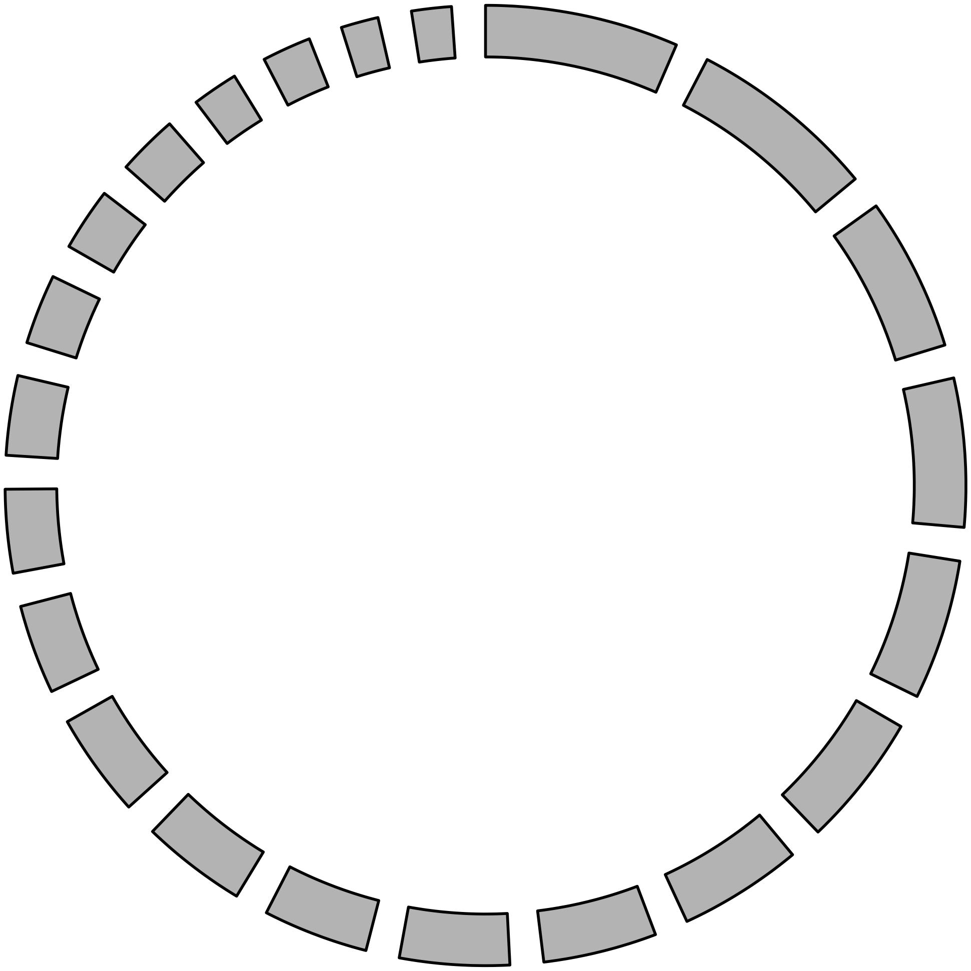


Figure 9.1: Adding 'ideogram' track.

```
p <- p + layout_circle(hg19Ideo, geom = "scale", size = 2, radius = 35, trackWidth = 2)
p
```

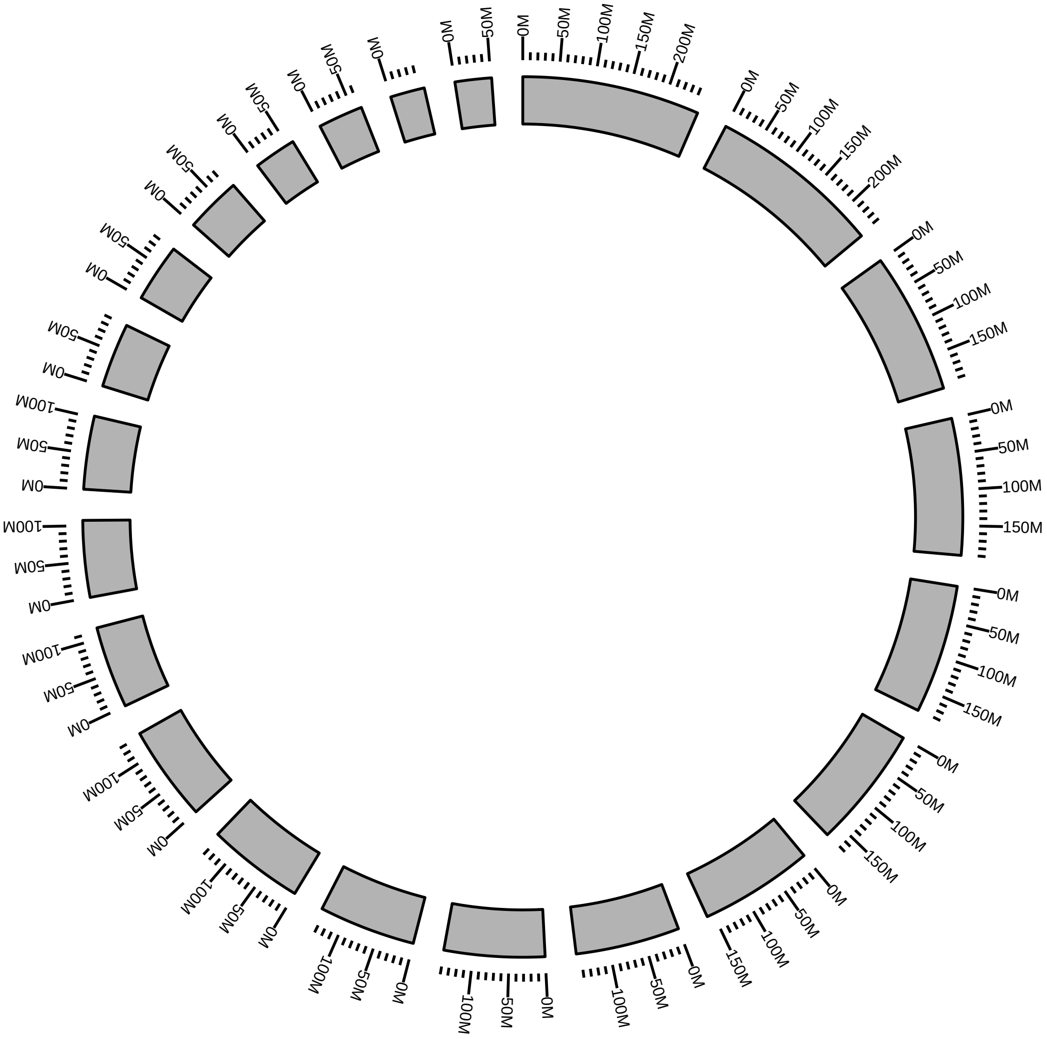


Figure 9.2: Adding a 'scale' track.

```
p <- p + layout_circle(hg19Ideo, geom = "text", aes(label = seqnames), vjust = 0,
    radius = 38, trackWidth = 7)
p
```

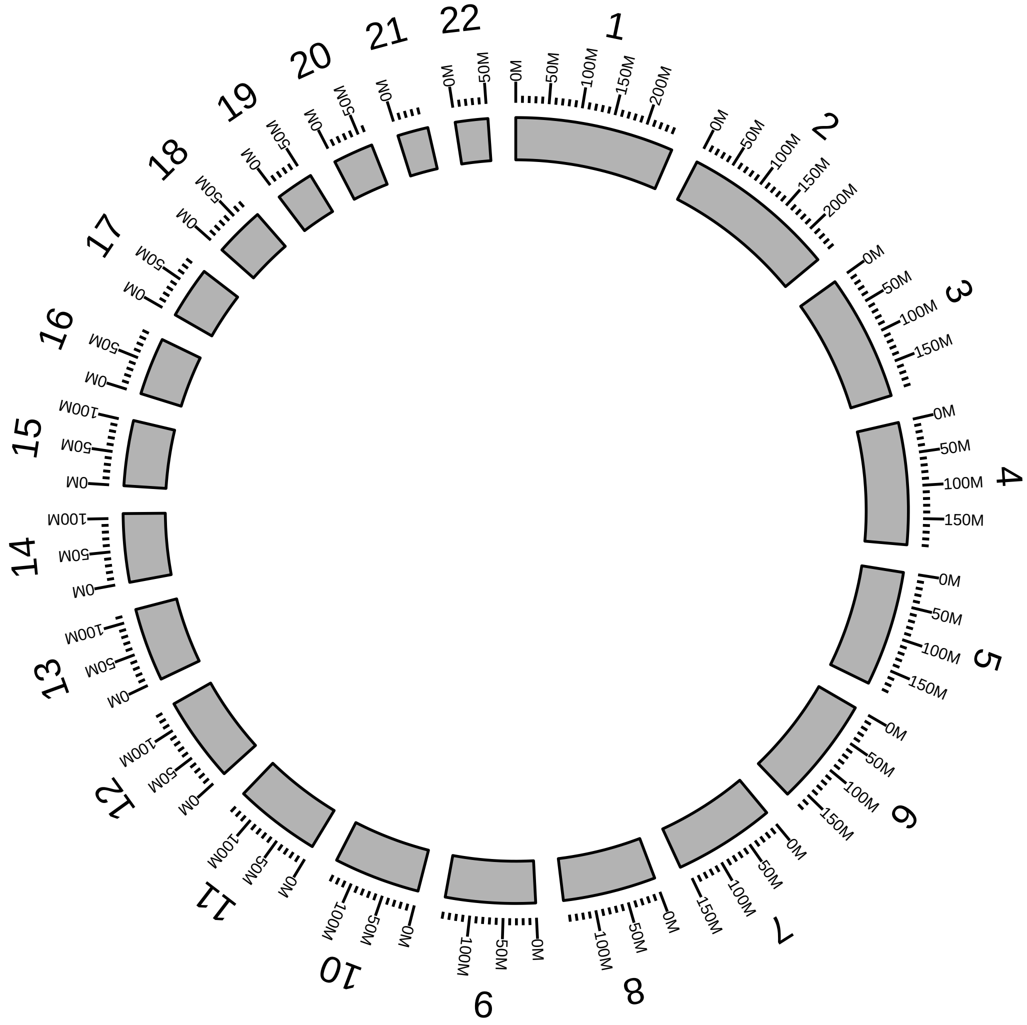


Figure 9.3: Adding a 'text' track.

```
p <- p + layout_circle(mut.gr, geom = "rect", color = "steelblue", radius = 23,
    trackWidth = 6)
p
```

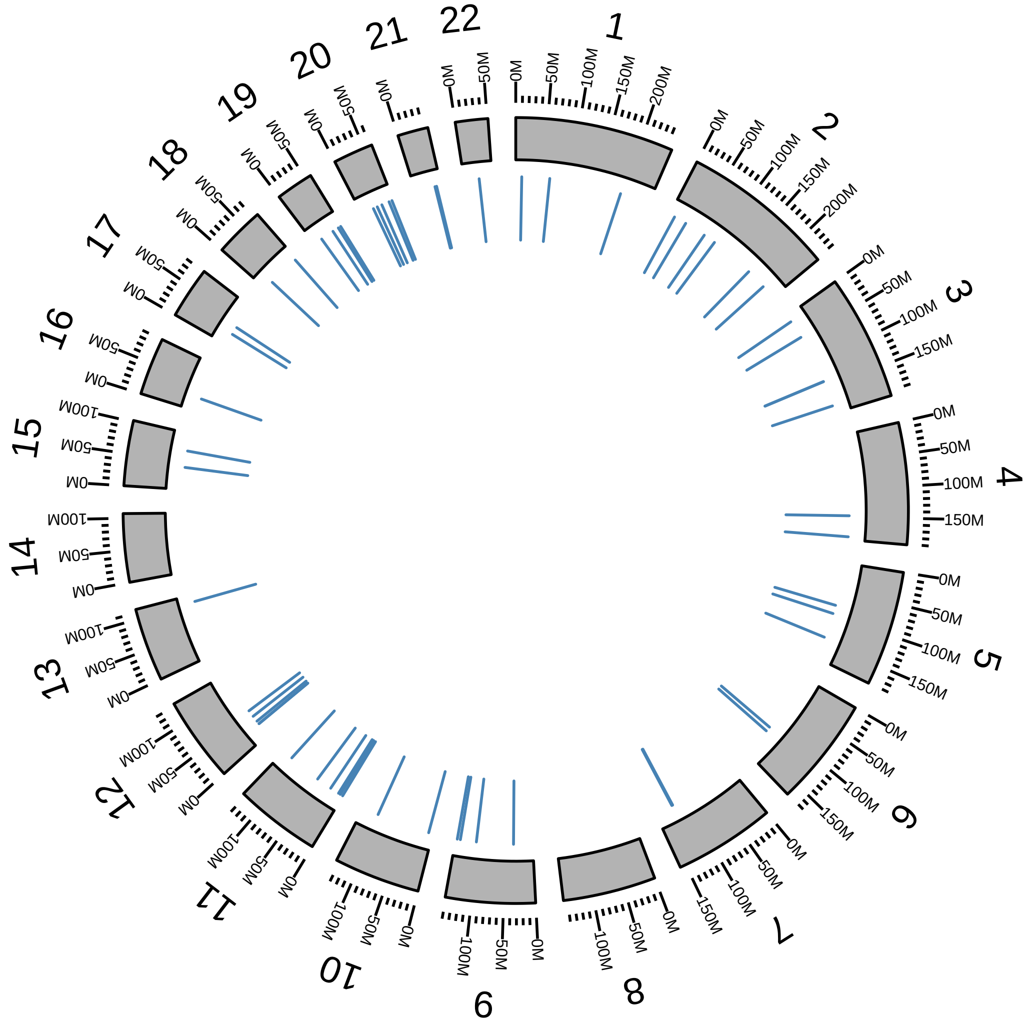


Figure 9.4: Adding a segment track to show mutation.

```
gr1 <- with(rearr, GRanges(chr1, IRanges(pos1, width = 1)))
## end position
gr2 <- with(rearr, GRanges(chr2, IRanges(pos2, width = 1)))
## add extra column
nms <- colnames(rearr)
.extra.nms <- setdiff(nms, c("chr1", "chr2", "pos1", "pos2"))
values(gr1) <- rearr[, .extra.nms]
## remove out-of-limits data
seqs <- as.character(seqnames(gr1))
.mx <- seqlengths(hg19Ideo)[seqs]
idx1 <- start(gr1) > .mx
seqs <- as.character(seqnames(gr2))
.mx <- seqlengths(hg19Ideo)[seqs]
idx2 <- start(gr2) > .mx
idx <- !idx1 & !idx2
gr1 <- gr1[idx]
seqlengths(gr1) <- seqlengths(hg19Ideo)
gr2 <- gr2[idx]
seqlengths(gr2) <- seqlengths(hg19Ideo)
```

To create a suitable structure to plot, please use another ‘GRanges‘ to represent the end of the links, and stored as elementMetadata for the "start point" ‘GRanges‘. Here we named it as "to.gr" and will be used later.

```
values(gr1)$to.gr <- gr2
## rename to gr
gr <- gr1
```

Here we show the flexibility of *ggbio*, for example, if you want to use color to indicate your links, make sure you add extra information in the data, used for mapping later. Here in this example, we use "intrachromosomal" to label rearrangement within the same chromosomes and use "interchromosomal" to label rearrangement in different chromosomes.

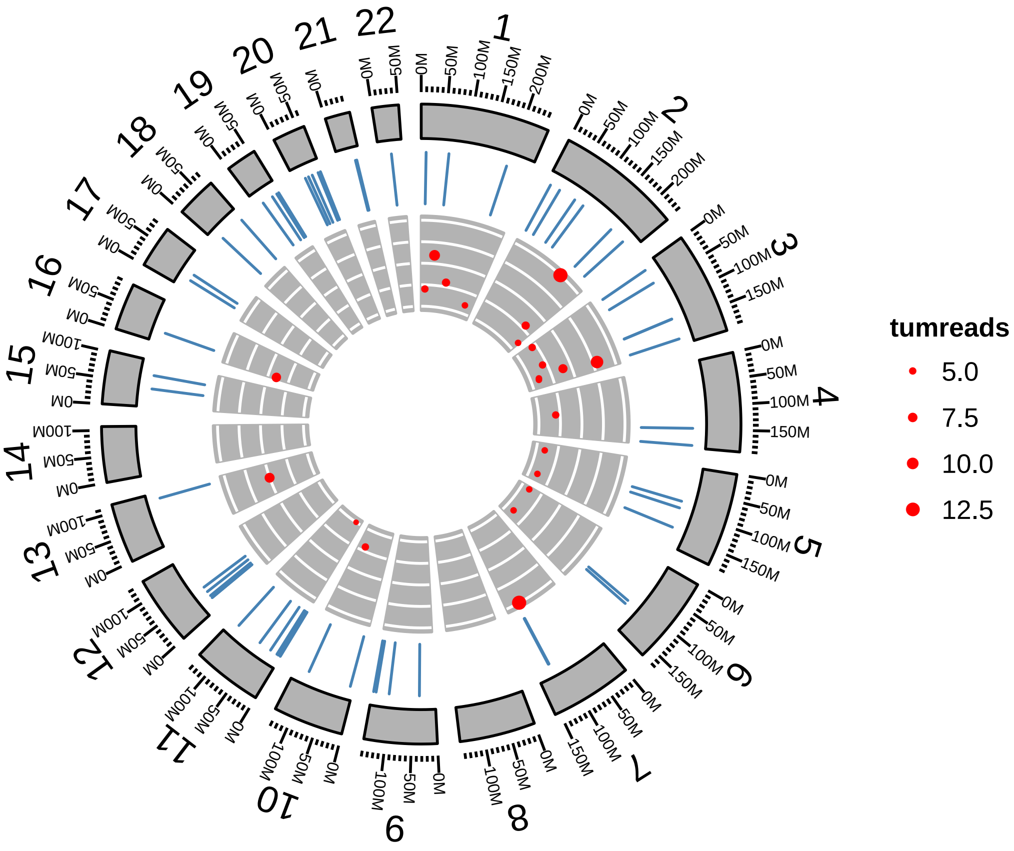
```
values(gr)$rearrangements <- ifelse(as.character(seqnames(gr)) == as.character(seqnames((values(gr)$to.gr
    "intrachromosomal", "interchromosomal")
```

Get subset of links data for only one sample "CRC1"

```
gr.crc1 <- gr[values(gr)$individual == "CRC-1"]
```

Ok, add a "point" track with grid background for rearrangement data and map ‘y‘ to variable "score", map ‘size‘ to variable "tumreads", rescale the size to a proper size range.

```
p <- p + layout_circle(gr.crc1, geom = "point", aes(y = score, size = tumreads),
    color = "red", radius = 12, trackWidth = 10, grid = TRUE) + scale_size(range = c(1,
    2.5))
p
```



Finally, let's add links and map color to rearrangement types. Remember you need to specify 'linked.to' to the column that contain end point of the data.

9.2.4 Step 4: Complex arragnment of plots

In this step, we are going to make multiple sample comparison, this may require some knowledge about package *grid* and *gridExtra*. We will introduce a more easy way to combine your graphics later after this.

We just want 9 single circular plots put together in one page, since we cannot keep too many tracks, we only keep ideogram and links. Here is one sample.

```
grl <- split(gr, values(gr)$individual)
## need 'unit', load grid
```

```
p <- p + layout_circle(gr.crc1, geom = "link", linked.to = "to.gr", aes(color = rearrangements),
    radius = 10, trackWidth = 1)
p
```

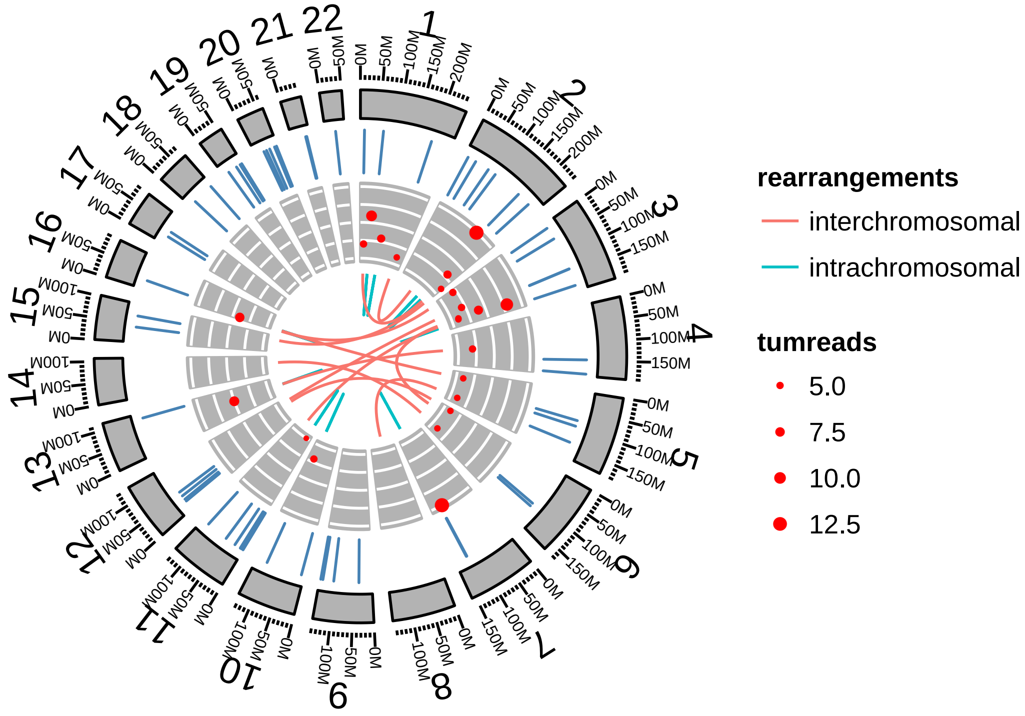


Figure 9.5: A link track is added to the circular plot.

```
cols <- RColorBrewer::brewer.pal(3, "Set2")[2:1]
names(cols) <- c("interchromosomal", "intrachromosomal")

p0 <- ggplot() + layout_circle(gr.crc1, geom = "link", linked.to = "to.gr",
    aes(color = rearrangements), radius = 7.1) + layout_circle(hg19Ideo,
    geom = "ideo", trackWidth = 1.5, color = "gray70", fill = "gray70") +
    scale_color_manual(values = cols)
p0
```

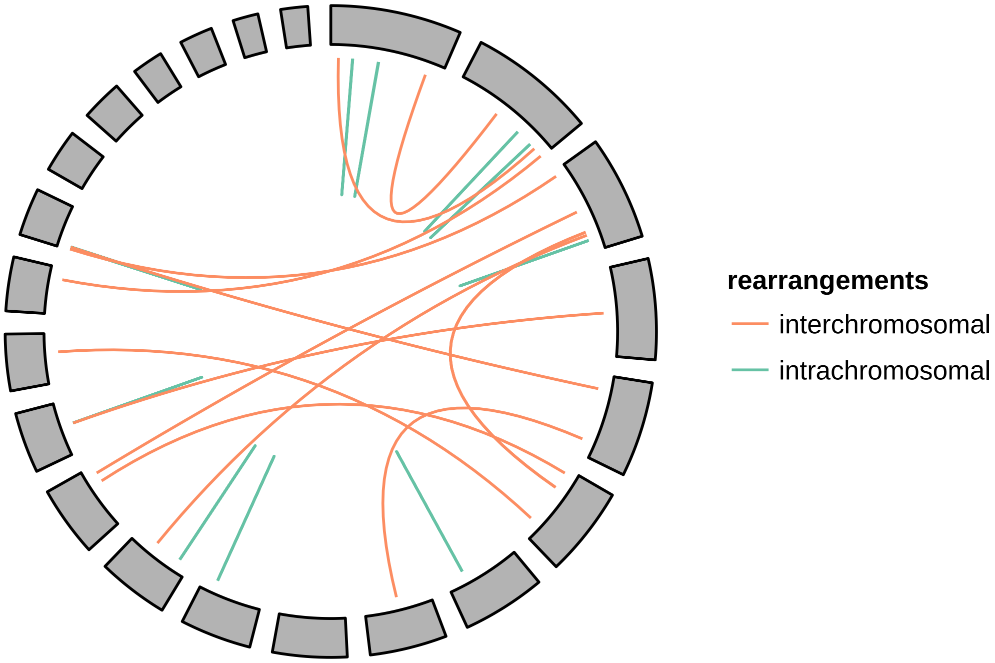


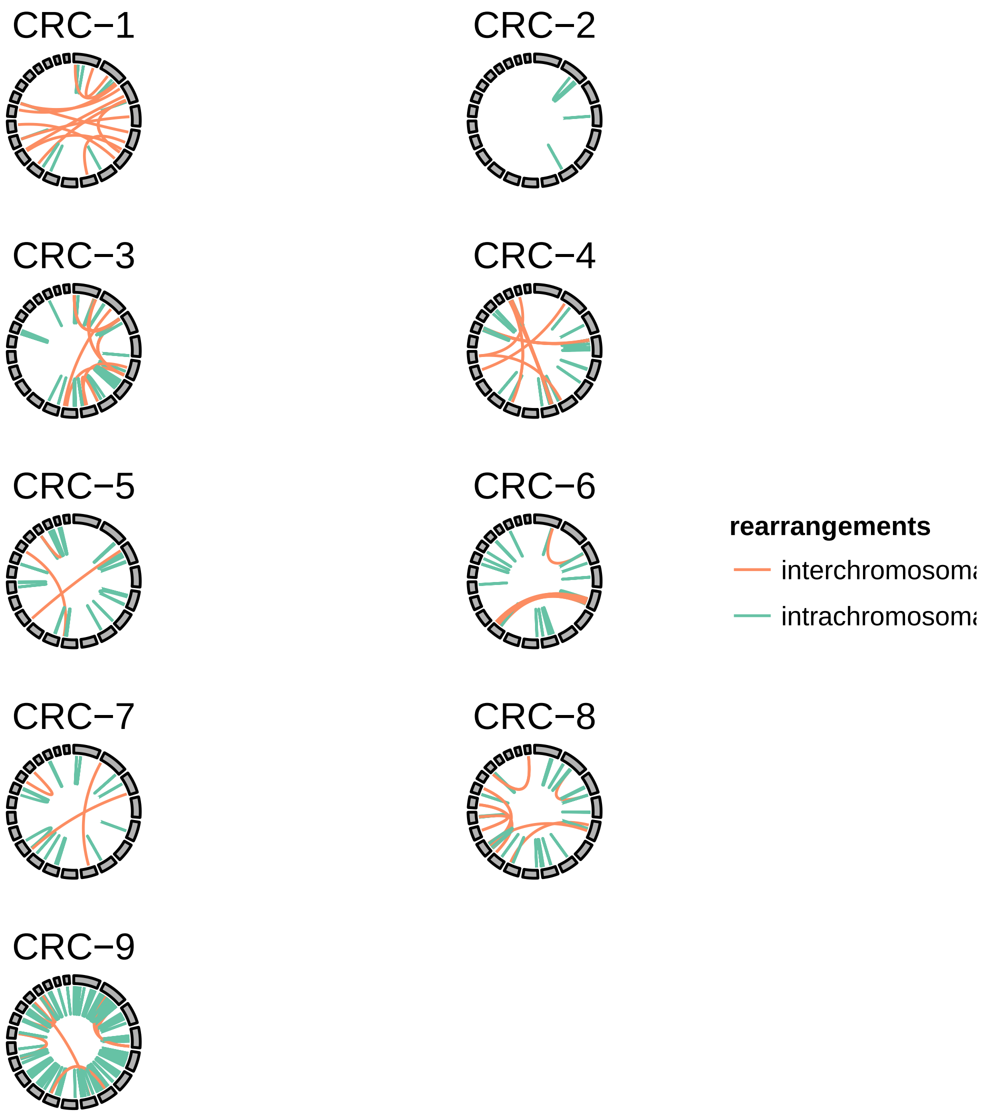
Figure 9.6: Just to show single individuals crc1.

```
library(grid)
lst <- lapply(grl, function(gr.cur) {
    print(unique(as.character(values(gr.cur)$individual)))
    cols <- RColorBrewer::brewer.pal(3, "Set2")[2:1]
    names(cols) <- c("interchromosomal", "intrachromosomal")
    p <- ggplot() + layout_circle(gr.cur, geom = "link", linked.to = "to.gr",
        aes(color = rearrangements), radius = 7.1) + layout_circle(hg19Ideo,
        geom = "ideo", trackWidth = 1.5, color = "gray70", fill = "gray70") +
        scale_color_manual(values = cols) + labs(title = (unique(values(gr.cur)$individual))) +
        theme(plot.margin = unit(rep(0, 4), "lines"))
})
```

```
## [1] "CRC-1"
## [1] "CRC-2"
## [1] "CRC-3"
## [1] "CRC-4"
## [1] "CRC-5"
## [1] "CRC-6"
## [1] "CRC-7"
## [1] "CRC-8"
## [1] "CRC-9"
```

We wrap the function in grid level to a more user-friendly high level function, called arrangeGrobByParsingLegend. You can pass your ggplot2 graphics to this function , specify the legend you want to keep on the right, you can also specify the column/row numbers. Here we assume all plots we have passed follows the same color scale and have the same legend, so we only have to keep one legend on the right.

```
arrangeGrobByParsingLegend(lst, widths = c(4, 1), legend.idx = 1, ncol = 2)
```


CRC−1
CRC−2
CRC−3
CRC−4
CRC−5
CRC−6
CRC−7
CRC−8
CRC−9
rearrangements
interchromosoma
intrachromosoma

```
## NULL
```

Chapter 10

Manhattan plot

10.1 Introduction

In this tutorial, we introduce a new coordinate system called "genome" for genomic data. This transformation is to put all chromosomes on the same genome coordinates following specified orders and adding buffers in between. One may think about facet ability based on *seqnames*, it can produce something similar to *Manhattan plot*[1], but the view will not be compact. What's more, genome transformation is previous step to form a circular view. In this tutorial, we will simulate some SNP data and use this special coordinate and a specialized function plotGrandLinear to make a Manhattan plot.

Manhattan plot is just a special use design with this coordinate system.

10.2 Understand the new coordinate

Let's load some packages and data first

```
library(ggbio)
data(hg19IdeogramCyto, package = "biovizBase")
data(hg19Ideogram, package = "biovizBase")
library(GenomicRanges)
```

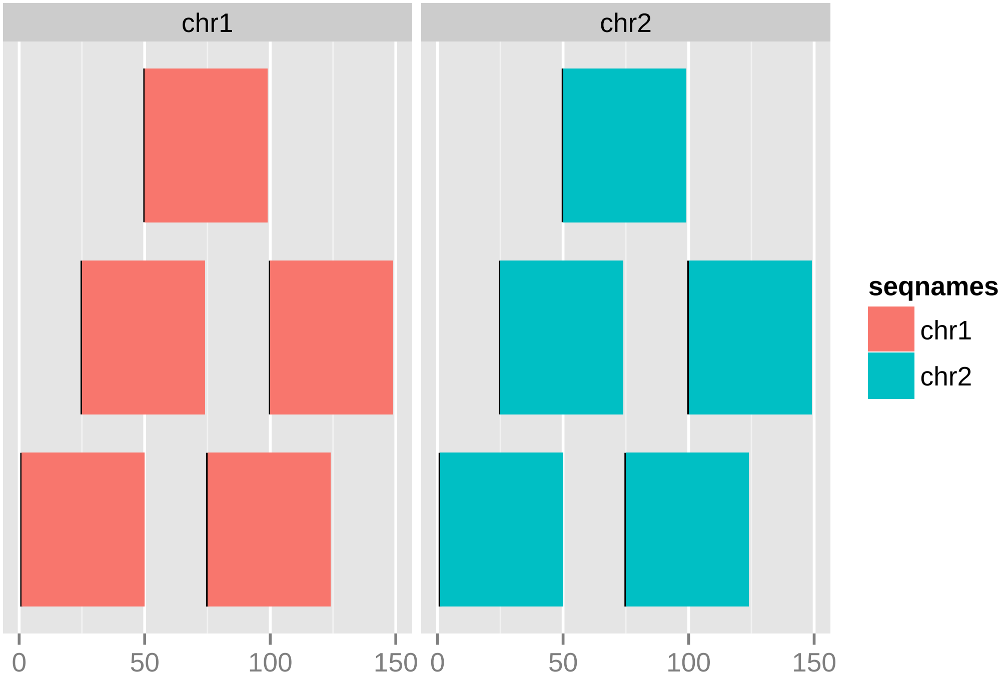
Make a minimal example 'GRanges', and see what the default coordiante looks like, pay attention that, by default, the graphics are faceted by 'seqnames' as shown in Figure 10.1

What if we specify the coordinate system to be "genome" in autoplot function, there is no faceting anymore, the two plots are merged into one single genome space, and properly labeled as shown in Figure 10.2

The internal transformation are implemented into the function transformToGenome. And there is some simple way to test if a GRanges object is transformed to coordinate "genome" or not

[1] http://en.wikipedia.org/wiki/Manhattan

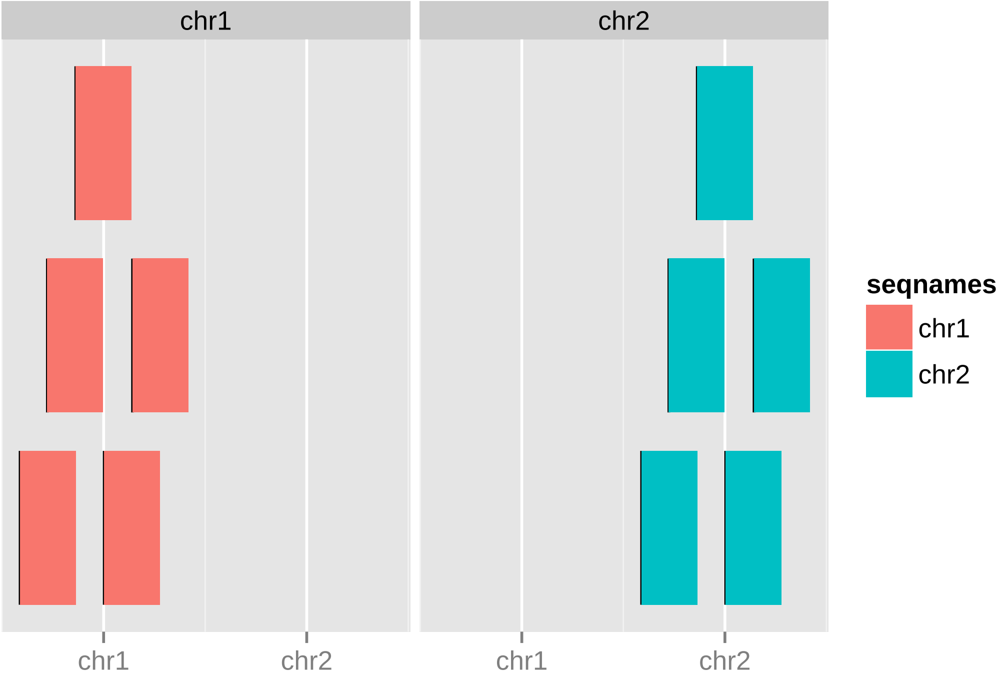
```
library(biovizBase)
gr <- GRanges(rep(c("chr1", "chr2"), each = 5), IRanges(start = rep(seq(1,
    100, length = 5), times = 2), width = 50))
autoplot(gr, aes(fill = seqnames))

## Object of class "ggbio"
```



```
## NULL
```

Figure 10.1: Default grahpics is faceted by seqnames

```
autoplot(gr, coord = "genome", aes(fill = seqnames))

                                     ## using coord:genome to parse x scale

## Object of class "ggbio"
```



```
## NULL
```

Figure 10.2: Coordinate genome

```
gr.t <- transformToGenome(gr)
head(gr.t)

## GRanges with 6 ranges and 2 metadata columns:
##       seqnames     ranges strand |    .start      .end
##          <Rle>  <IRanges>  <Rle> | <numeric> <numeric>
##   [1]     chr1 [  1,  50]      * |         1        50
##   [2]     chr1 [ 25,  74]      * |        25        74
##   [3]     chr1 [ 50,  99]      * |        50        99
##   [4]     chr1 [ 75, 124]      * |        75       124
##   [5]     chr1 [100, 149]      * |       100       149
##   [6]     chr2 [  1,  50]      * |       180       229
##   ---
##   seqlengths:
##    chr1 chr2
##      NA   NA

is_coord_genome(gr.t)

## [1] TRUE

metadata(gr.t)$coord

## [1] "genome"
```

10.3 Step 2: Simulate a SNP data set

Let's use the real human genome space to simulate a SNP data set.

```
chrs <- as.character(levels(seqnames(hg19IdeogramCyto)))
seqlths <- seqlengths(hg19Ideogram)[chrs]
set.seed(1)
nchr <- length(chrs)
nsnps <- 100
gr.snp <- GRanges(rep(chrs, each = nsnps), IRanges(start = do.call(c, lapply(chrs,
    function(chr) {
        N <- seqlths[chr]
        runif(nsnps, 1, N)
    })), width = 1), SNP = sapply(1:(nchr * nsnps), function(x) paste("rs",
    x, sep = "")), pvalue = -log10(runif(nchr * nsnps)), group = sample(c("Normal",
    "Tumor"), size = nchr * nsnps, replace = TRUE))
genome(gr.snp) <- "hg19"
gr.snp

## GRanges with 2400 ranges and 3 metadata columns:
```

```
##          seqnames                 ranges strand   |         SNP
##             <Rle>              <IRanges>  <Rle>   | <character>
##      [1]     chr1 [ 66178199,  66178199]      *   |         rs1
##      [2]     chr1 [ 92752113,  92752113]      *   |         rs2
##      [3]     chr1 [142784056, 142784056]      *   |         rs3
##      [4]     chr1 [226371355, 226371355]      *   |         rs4
##      [5]     chr1 [ 50269347,  50269347]      *   |         rs5
##      [6]     chr1 [223924186, 223924186]      *   |         rs6
##      [7]     chr1 [235460897, 235460897]      *   |         rs7
##      [8]     chr1 [164704260, 164704260]      *   |         rs8
##      [9]     chr1 [156807066, 156807066]      *   |         rs9
##      ...      ...                    ...    ... ...         ...
##   [2392]     chrY   [36501485, 36501485]      *   |      rs2392
##   [2393]     chrY   [30054272, 30054272]      *   |      rs2393
##   [2394]     chrY   [20065602, 20065602]      *   |      rs2394
##   [2395]     chrY   [19541601, 19541601]      *   |      rs2395
##   [2396]     chrY   [34038689, 34038689]      *   |      rs2396
##   [2397]     chrY   [ 3010837,  3010837]      *   |      rs2397
##   [2398]     chrY   [23806602, 23806602]      *   |      rs2398
##   [2399]     chrY   [15474595, 15474595]      *   |      rs2399
##   [2400]     chrY   [10016302, 10016302]      *   |      rs2400
##             pvalue       group
##          <numeric> <character>
##      [1]   1.22380      Normal
##      [2]   1.27916      Normal
##      [3]   0.01199       Tumor
##      [4]   0.09985      Normal
##      [5]   1.49938       Tumor
##      [6]   0.26497       Tumor
##      [7]   1.75456       Tumor
##      [8]   0.10976       Tumor
##      [9]   0.12073       Tumor
##      ...       ...         ...
##   [2392]   0.93515      Normal
##   [2393]   0.08353       Tumor
##   [2394]   0.05148      Normal
##   [2395]   0.01483      Normal
##   [2396]   0.17601      Normal
##   [2397]   0.78685       Tumor
##   [2398]   0.48952      Normal
##   [2399]   0.60000      Normal
##   [2400]   0.03967      Normal
##   ---
##   seqlengths:
##     chr1 chr10 chr11 chr12 chr13 chr14 ...  chr7  chr8  chr9  chrX  chrY
##       NA    NA    NA    NA    NA    NA ...    NA    NA    NA    NA    NA
```

We use the some trick to make a shorter names.

```
seqlengths(gr.snp)

##  chr1 chr10 chr11 chr12 chr13 chr14 chr15 chr16 chr17 chr18 chr19  chr2
##    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
## chr20 chr21 chr22  chr3  chr4  chr5  chr6  chr7  chr8  chr9  chrX  chrY
##    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA

nms <- seqnames(seqinfo(gr.snp))
nms.new <- gsub("chr", "", nms)
names(nms.new) <- nms
gr.snp <- renameSeqlevels(gr.snp, nms.new)
seqlengths(gr.snp)

##  1 10 11 12 13 14 15 16 17 18 19  2 20 21 22  3  4  5  6  7  8  9  X  Y
## NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA
```

10.4 Step 3: Start to make Manhattan plot by using `autoplot`

wrapped basic functions into `autoplot`, you can specify the coordinate. Figure 10.3 shows what the unordered object looks like.

That's probably not what you want, if you want to change to specific order, just sort them by hand and use ‘keepSeqlevels‘. Figure 10.4 shows a sorted plot.

NOTICE: the data now doesn't have information about lengths of each chromosomes, this is allowed to be plotted, but it's misleading sometimes, without chromosomes lengths information, *ggbio* use data space to make estimated lengths for you, this is not accurate! So let's just assign `seqlengths` to the object. Then you will find the data space now is distributed proportional to real space as shown in Figure 10.5.

In `autoplot`, argument `coord` is just used to transform the data, after that, you can use it as common `GRanges`, all other geom/stat works for it. Here just show a simple example for another geom "line" as shown in Figure 10.6

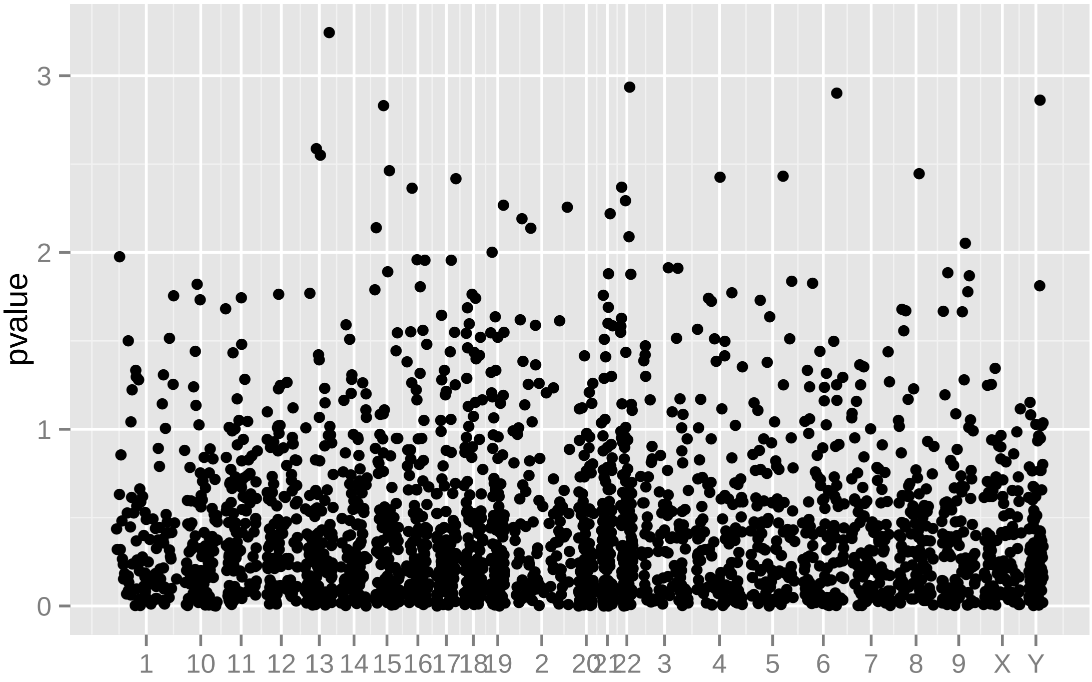
10.5 Convenient `plotGrandLinear` function

In *ggbio*, sometimes we develop specialized function for certain types of plots, it's basically a wrapper over lower level API and `autoplot`, but more convenient to use. Here for *Manhattan plot*, we have a function called `plotGrandLinear` used for it. aes(y =) is required to indicate the y value, e.g. p-value. Figure 10.7 shows a defalut graphic.

Color mapping is automatically figured out by *ggbio* following the rules

- if `color` present in `aes()`, like `aes(color = seqnames)`, it will assume it's mapping to data column called 'seqnames'.
- if `color` is not wrapped in `aes()`, then this function will **recylcle** them to all chromosomes.

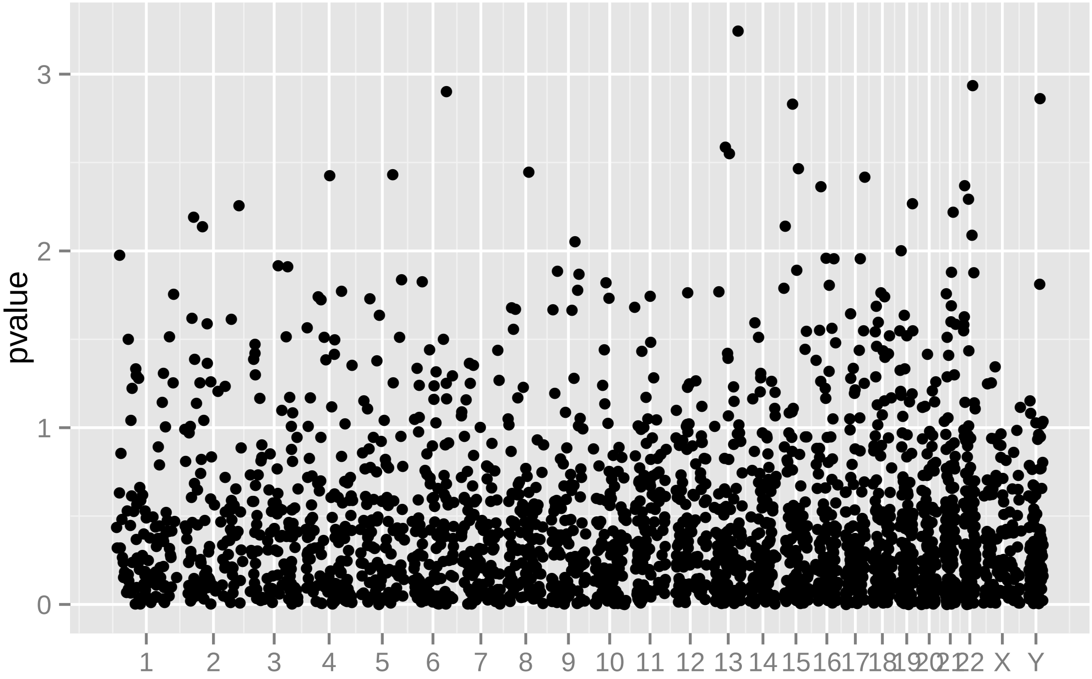
```
autoplot(gr.snp, coord = "genome", geom = "point", aes(y = pvalue), space.skip = 0.01)

                                      ## using coord:genome to parse x scale

## Object of class "ggbio"
```



```
## NULL
```

Figure 10.3: Unordred Manhattan plot

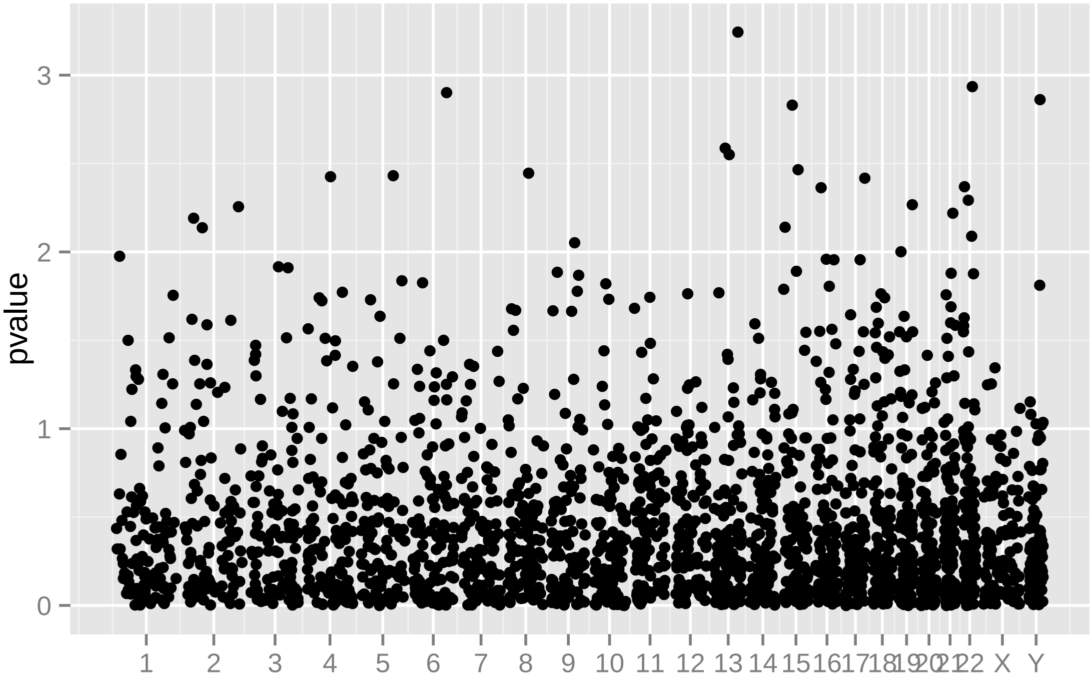
```
gr.snp <- keepSeqlevels(gr.snp, c(1:22, "X", "Y"))
autoplot(gr.snp, coord = "genome", geom = "point", aes(y = pvalue), space.skip = 0.01)

                                ## using coord:genome to parse x scale

## Object of class "ggbio"
```



```
## NULL
```

Figure 10.4: Sorted data for Manhattan plot

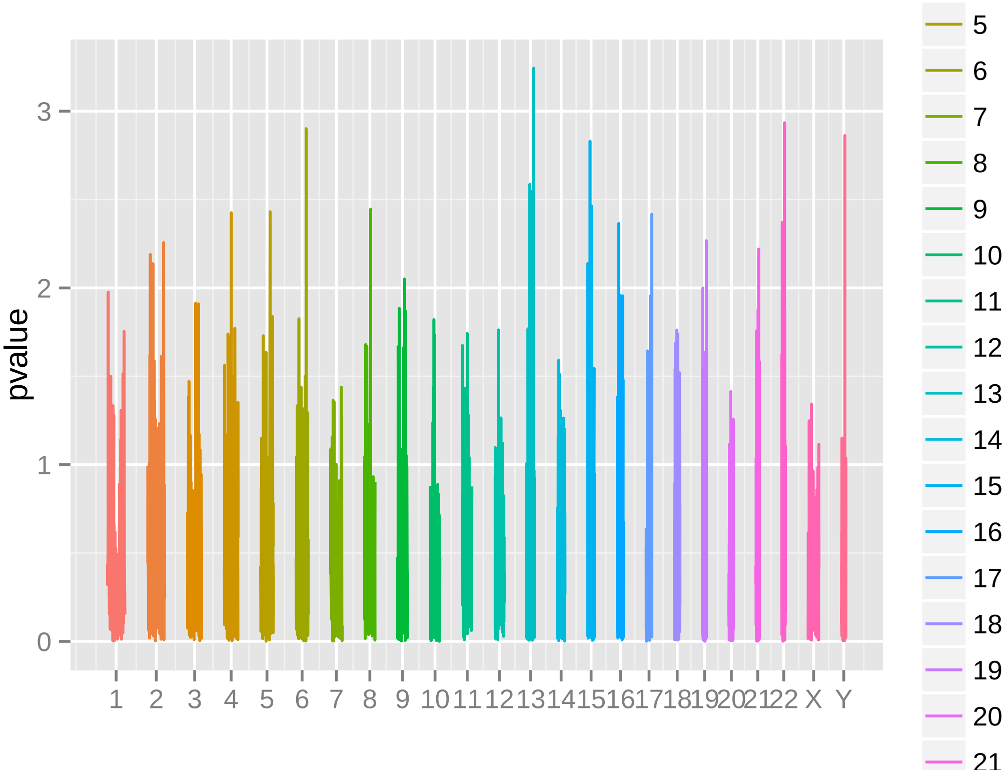
```
names(seqlths) <- gsub("chr", "", names(seqlths))
seqlengths(gr.snp) <- seqlths[names(seqlengths(gr.snp))]
autoplot(gr.snp, coord = "genome", geom = "point", aes(y = pvalue), space.skip = 0.01)

                          ## using coord:genome to parse x scale

## Object of class "ggbio"
```



```
## NULL
```

Figure 10.5: Manhattan plot after setting seqlengths to the data, the data space now is distributed proportional to real chromosome space.

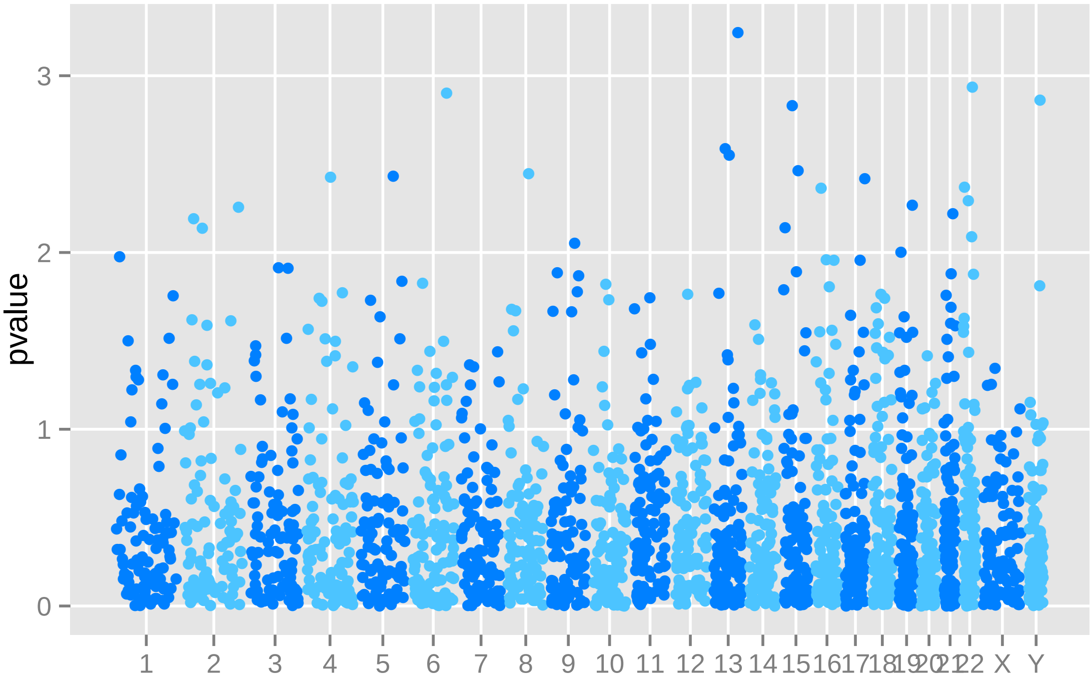
```
autoplot(gr.snp, coord = "genome", geom = "line", aes(y = pvalue, group = seqnames,
    color = seqnames))

                                ## using coord:genome to parse x scale

## Object of class "ggbio"
```



```
## NULL
```

Figure 10.6: Use line to represent the data in typical Manhattan plot.

```
plotGrandLinear(gr.snp, aes(y = pvalue))

                                    ## using coord:genome to parse x scale

## Object of class "ggbio"
```



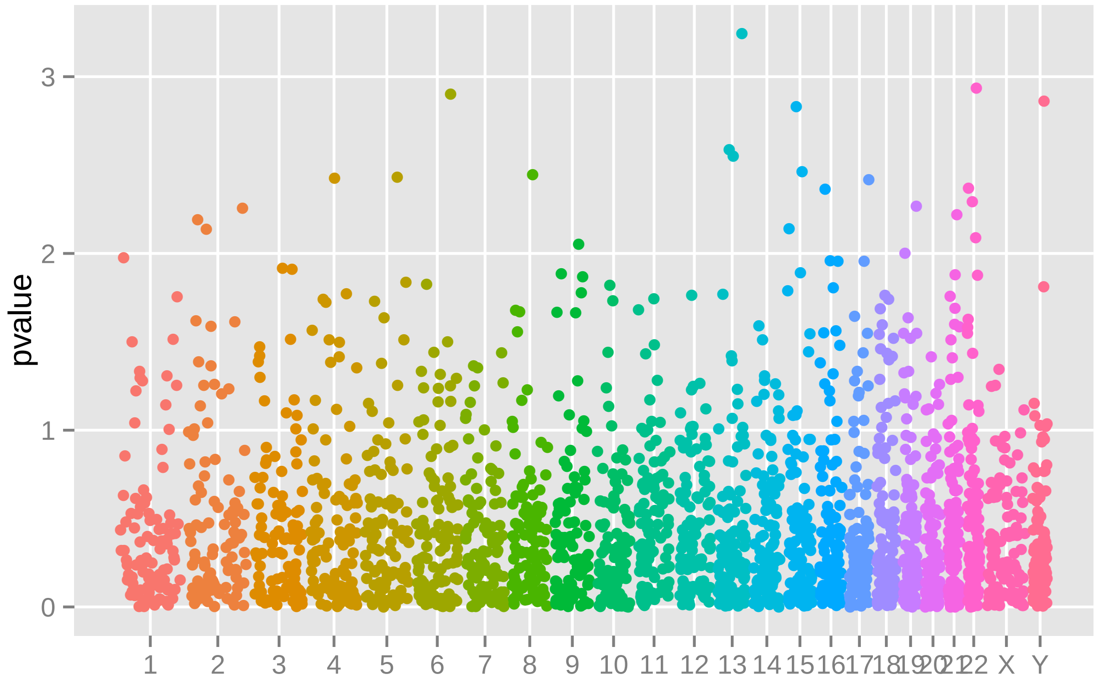
```
## NULL
```

Figure 10.7: Default Manhattan plot by calling plotGrandLinear function

- if `color` is single character representing color, then just use one arbitrary color.

Let's test some examples for controling colors.

```
plotGrandLinear(gr.snp, aes(y = pvalue, color = seqnames))

                                    ## using coord:genome to parse x scale

## Object of class "ggbio"
```



```
## NULL
```

Figure 10.8: Color mapped to chromosome names.

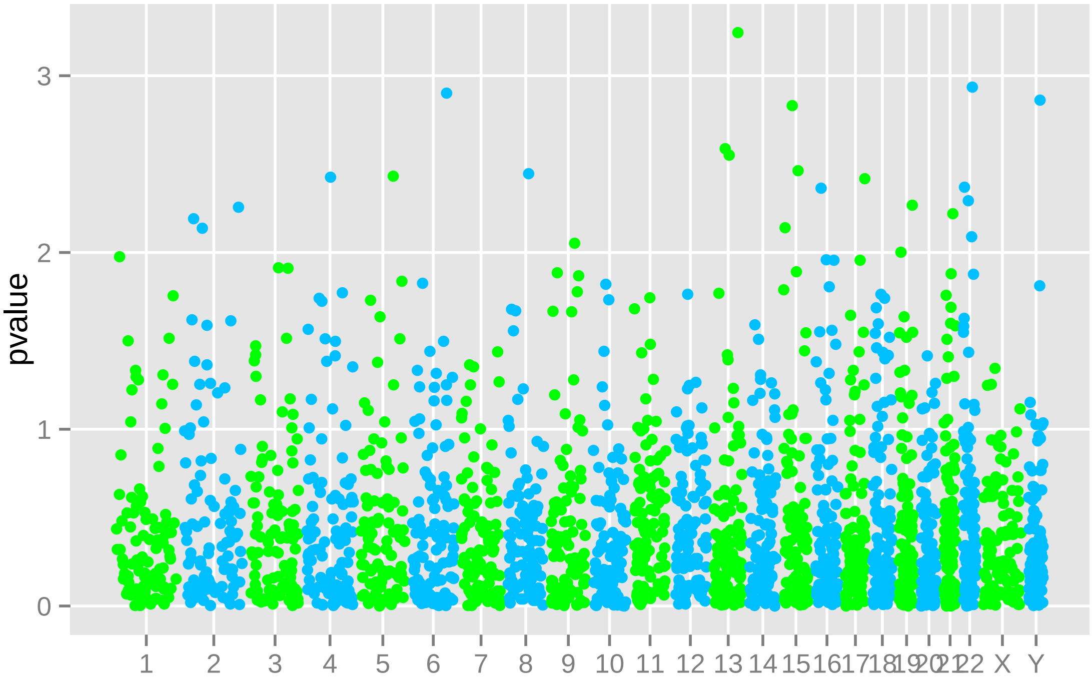
You can also add cutoff line as shown in Figure 10.12.

This is equivalent to *ggplot2* 's API.

```
plotGrandLinear(gr.snp, aes(y = pvalue)) + geom_hline(yintercept = 3, color = "blue",
    size = 4)
```

Sometimes the names of chromosomes maybe very long, you may want to rotate them, let's make a longer name first

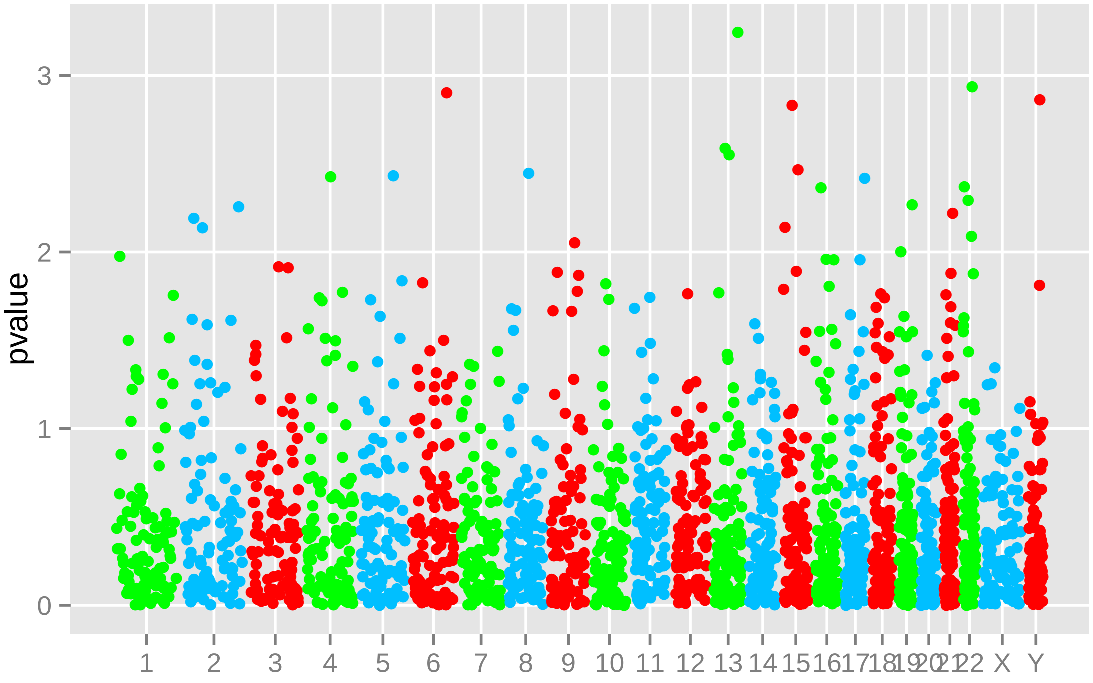
```
plotGrandLinear(gr.snp, aes(y = pvalue), color = c("green", "deepskyblue"))

                                        ## using coord:genome to parse x scale

## Object of class "ggbio"
```



```
## NULL
```

Figure 10.9: Color follow 'green' and 'deepskyblue' order for all chromosome space.

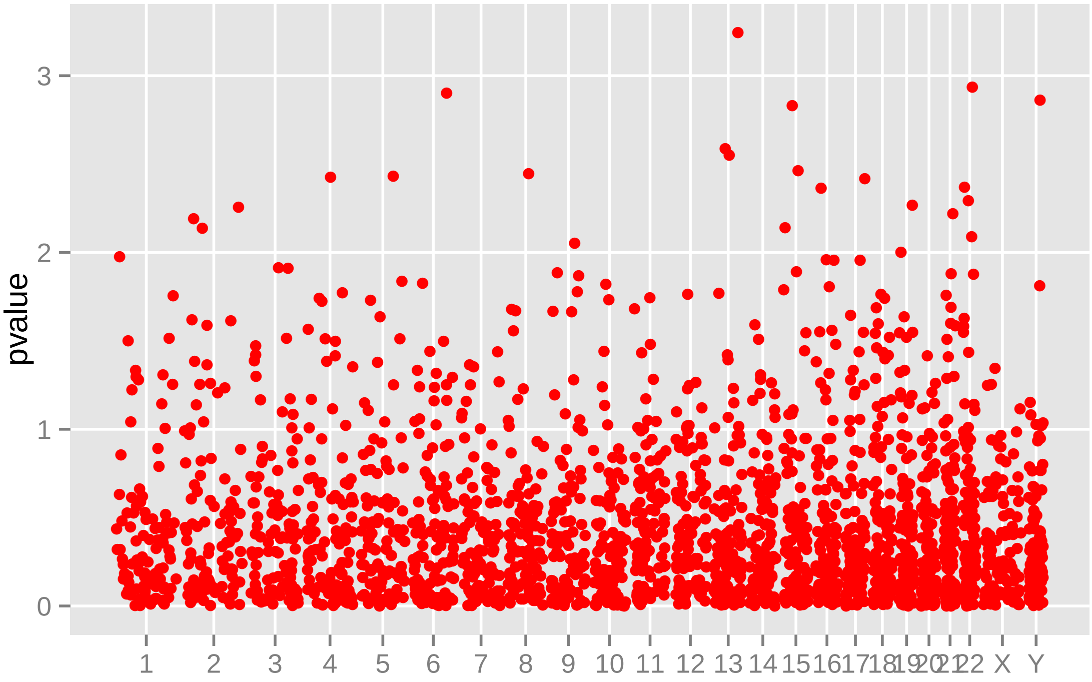
```
plotGrandLinear(gr.snp, aes(y = pvalue), color = c("green", "deepskyblue",
    "red"))

                        ## using coord:genome to parse x scale

## Object of class "ggbio"
```



```
## NULL
```

Figure 10.10: Color follow three colors pattern: 'green','deepskyblue', 'red'

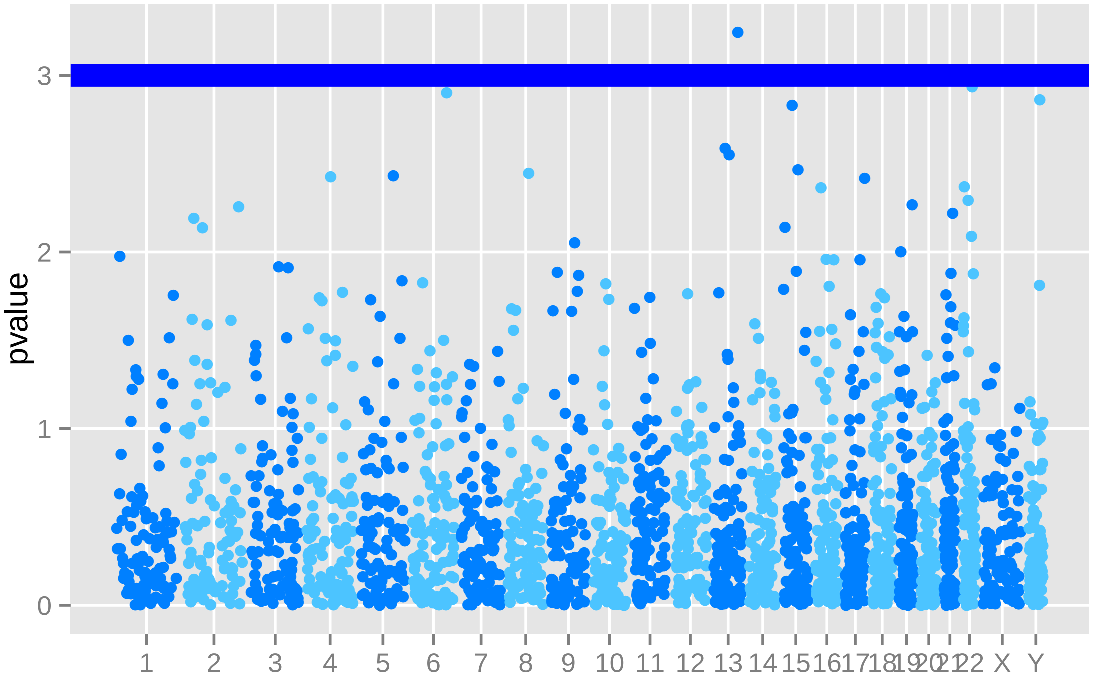
```
plotGrandLinear(gr.snp, aes(y = pvalue), color = "red")

                                        ## using coord:genome to parse x scale

## Object of class "ggbio"
```



```
## NULL
```

Figure 10.11: Unique color for all.

```
plotGrandLinear(gr.snp, aes(y = pvalue), cutoff = 3, cutoff.color = "blue",
    cutoff.size = 4)

                                    ## using coord:genome to parse x scale

## Object of class "ggbio"
```



```
## NULL
```

Figure 10.12: Set cutoff on the Manhattan plot. The 'blue' line shows cutoff at value 3.

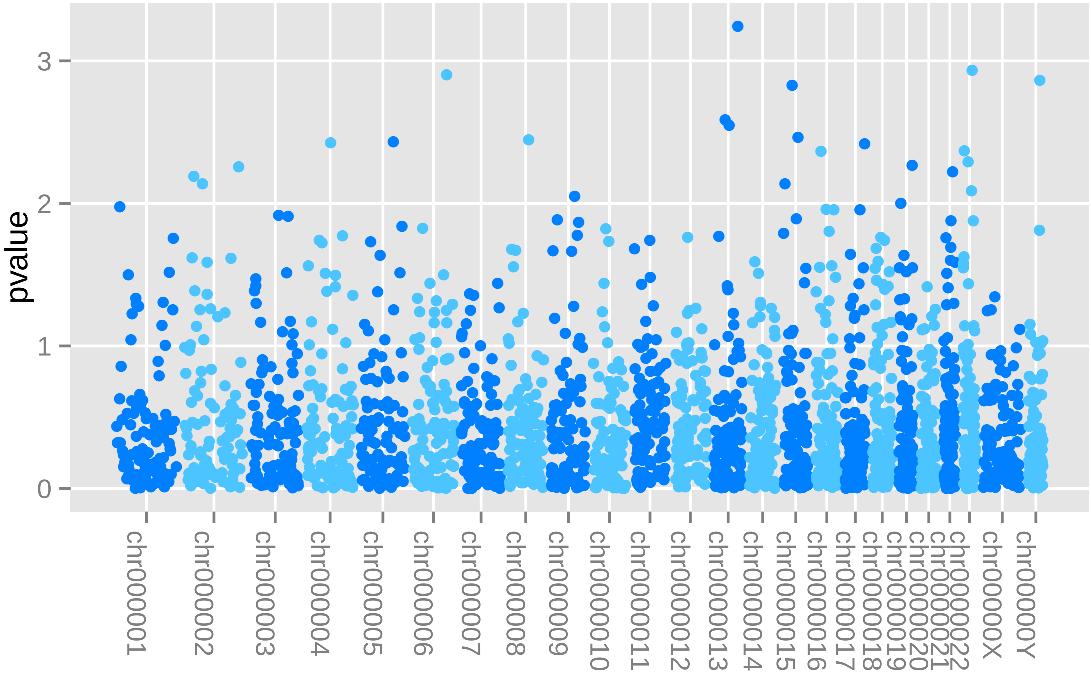
```
## let's make a long name
nms <- seqnames(seqinfo(gr.snp))
nms.new <- paste("chr00000", nms, sep = "")
names(nms.new) <- nms
gr.snp <- renameSeqlevels(gr.snp, nms.new)
seqlengths(gr.snp)
```

```
##  chr000001  chr000002  chr000003  chr000004  chr000005  chr000006
##  249250621  243199373  198022430  191154276  180915260  171115067
##  chr000007  chr000008  chr000009 chr0000010 chr0000011 chr0000012
##  159138663  146364022  141213431  135534747  135006516  133851895
## chr0000013 chr0000014 chr0000015 chr0000016 chr0000017 chr0000018
##  115169878  107349540  102531392   90354753   81195210   78077248
## chr0000019 chr0000020 chr0000021 chr0000022  chr00000X  chr00000Y
##   59128983   63025520   48129895   51304566  155270560   59373566
```

Then rotate it!

As you can tell from above examples, all utilities works for *ggplot2* will work for *ggbio* too.

```
plotGrandLinear(gr.snp, aes(y = pvalue)) + theme(axis.text.x = theme_text(angle = -90,
    hjust = 0))

                              ## using coord:genome to parse x scale
   ## theme_text is deprecated.  Use 'element_text' instead.  (Deprecated; last used in
                                         version 0.9.1)

## Object of class "ggbio"
```



```
## NULL
```

Figure 10.13: Rotate the x lable to save space.

Chapter 11

Karyogram overview

11.1 Introduction

A karyotype is the number and appearance of chromosomes in the nucleus of a eukaryotic cell[1]. It's one overview option when we want to show distribution of certain events on the genome, for example, binding sites for one protein. Particular pattern might be easier to observe from graphics, such as

- Clusterred events.
- Large missing chunk of data on particular chromosome.

GRanges object is also an ideal container for storing data needed for karyogram plot. Here is the strategy we used for generating ideogram templates.

- seqlengths is not required, but highly recommended for plotting karyogram. If a GRanges object contains seqlengths, we know exactly how long each chromosome is, and will use this information to plot genome space, particularly we plot all levels included in it, not just DATA space.
- If a GRanges has no seqlengths, we will issue a warning and try to estimate the chromosome lengths from data included. This is NOT accurate most time, so please pay attention to what you are going to visualize and make sure set seqlengths before hand.

11.2 Usage

11.2.1 autoplot

Let's first introduce how to use autoplot to generate karyogram graphic. To understand why we call it kayogram, let's first visualize some cytoband. We use layout argument to specify this special layout "karyogram". And under this layout, cytoband argument is acceptable, default is FALSE, if set to TRUE, we assume your have additional information associated with the data, stored in column gieStain, it will try to fill colors based on

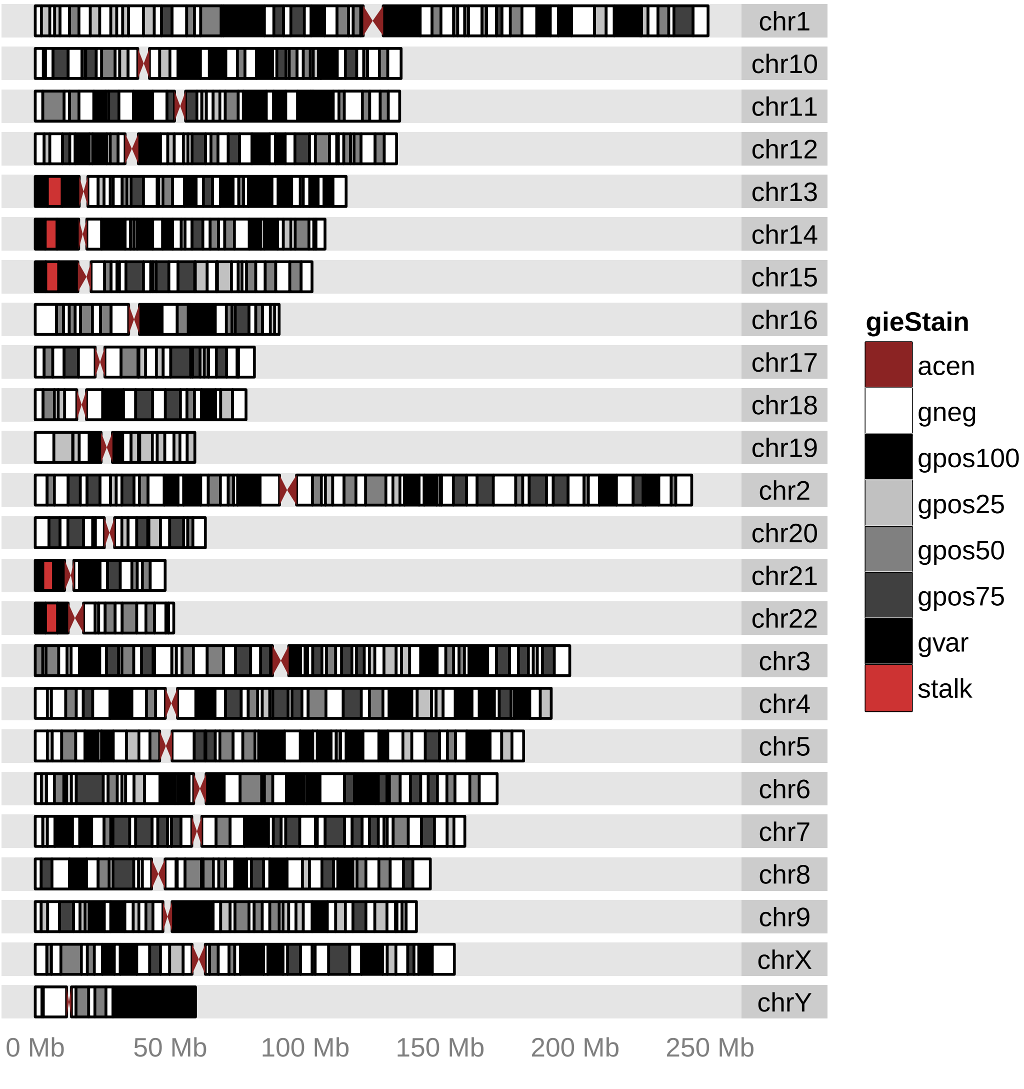
[1] http://en.wikipedia.org/wiki/Karyotype

this variable according to a pre-set staining colors. You may notice, this data set doesn't contain seqlengths information, but the data space actually cover the real space, so it's not going to be a problem.

```
library(ggbio)
data(hg19IdeogramCyto, package = "biovizBase")
head(hg19IdeogramCyto)

## GRanges with 6 ranges and 2 metadata columns:
##       seqnames              ranges strand |     name gieStain
##          <Rle>           <IRanges>  <Rle> | <factor> <factor>
##   [1]     chr1 [       0,  2300000]     * |   p36.33     gneg
##   [2]     chr1 [ 2300000,  5400000]     * |   p36.32   gpos25
##   [3]     chr1 [ 5400000,  7200000]     * |   p36.31     gneg
##   [4]     chr1 [ 7200000,  9200000]     * |   p36.23   gpos25
##   [5]     chr1 [ 9200000, 12700000]     * |   p36.22     gneg
##   [6]     chr1 [12700000, 16200000]     * |   p36.21   gpos50
##   ---
##   seqlengths:
##     chr1 chr10 chr11 chr12 chr13 chr14 ...  chr7  chr8  chr9  chrX  chrY
##       NA    NA    NA    NA    NA    NA ...    NA    NA    NA    NA    NA

## default pre-set color stored in
getOption("biovizBase")$cytobandColor

##      gneg     stalk      acen      gpos      gvar     gpos1     gpos2
## "grey100"  "brown3"  "brown4"   "grey0"   "grey0" "#FFFFFF" "#FCFCFC"
##     gpos3     gpos4     gpos5     gpos6     gpos7     gpos8     gpos9
## "#F9F9F9" "#F7F7F7" "#F4F4F4" "#F2F2F2" "#EFEFEF" "#ECECEC" "#EAEAEA"
##    gpos10    gpos11    gpos12    gpos13    gpos14    gpos15    gpos16
## "#E7E7E7" "#E5E5E5" "#E2E2E2" "#E0E0E0" "#DDDDDD" "#DADADA" "#D8D8D8"
##    gpos17    gpos18    gpos19    gpos20    gpos21    gpos22    gpos23
## "#D5D5D5" "#D3D3D3" "#D0D0D0" "#CECECE" "#CBCBCB" "#C8C8C8" "#C6C6C6"
##    gpos24    gpos25    gpos26    gpos27    gpos28    gpos29    gpos30
## "#C3C3C3" "#C1C1C1" "#BEBEBE" "#BCBCBC" "#B9B9B9" "#B6B6B6" "#B4B4B4"
##    gpos31    gpos32    gpos33    gpos34    gpos35    gpos36    gpos37
## "#B1B1B1" "#AFAFAF" "#ACACAC" "#AAAAAA" "#A7A7A7" "#A4A4A4" "#A2A2A2"
##    gpos38    gpos39    gpos40    gpos41    gpos42    gpos43    gpos44
## "#9F9F9F" "#9D9D9D" "#9A9A9A" "#979797" "#959595" "#929292" "#909090"
##    gpos45    gpos46    gpos47    gpos48    gpos49    gpos50    gpos51
## "#8D8D8D" "#8B8B8B" "#888888" "#858585" "#838383" "#808080" "#7E7E7E"
##    gpos52    gpos53    gpos54    gpos55    gpos56    gpos57    gpos58
## "#7B7B7B" "#797979" "#767676" "#737373" "#717171" "#6E6E6E" "#6C6C6C"
##    gpos59    gpos60    gpos61    gpos62    gpos63    gpos64    gpos65
## "#696969" "#676767" "#646464" "#616161" "#5F5F5F" "#5C5C5C" "#5A5A5A"
##    gpos66    gpos67    gpos68    gpos69    gpos70    gpos71    gpos72
## "#575757" "#545454" "#525252" "#4F4F4F" "#4D4D4D" "#4A4A4A" "#484848"
##    gpos73    gpos74    gpos75    gpos76    gpos77    gpos78    gpos79
## "#454545" "#424242" "#404040" "#3D3D3D" "#3B3B3B" "#383838" "#363636"
##    gpos80    gpos81    gpos82    gpos83    gpos84    gpos85    gpos86
## "#333333" "#303030" "#2E2E2E" "#2B2B2B" "#292929" "#262626" "#242424"
##    gpos87    gpos88    gpos89    gpos90    gpos91    gpos92    gpos93
## "#212121" "#1E1E1E" "#1C1C1C" "#191919" "#171717" "#141414" "#121212"
##    gpos94    gpos95    gpos96    gpos97    gpos98    gpos99   gpos100
## "#0F0F0F" "#0C0C0C" "#0A0A0A" "#070707" "#050505" "#020202" "#000000"
##    gpos33    gpos66
##  "grey80"  "grey60"
```

```
autoplot(hg19IdeogramCyto, layout = "karyogram", cytoband = TRUE)
```

```
## Object of class "ggbio"
```



```
## NULL
```

You may want to change the order of chromosomes, keepSeqlevels are convenient for this purpose, it's defined in package *GenomicRanges*.

This GRanges object is special, it's a 'ideogram' we expected, in this case, cytoband argument could set to TRUE, and we draw special ideogram not just rectangles but show centromere as possible.

If we set it to FALSE, we treat it as a normal GRanges, nothing special as ideogram. So to show the cytoband, we need to specify which color column variable to fill as cytoband, function aes use an unevaluated expression like fill = gieStain, *gieStain* is column name which store cytoband color, notice that we don't use quotes around it, this means it's not something defined globally, but some column name defined in the data. The system will usually automatically assign categorical colors to represent this variable. But instead, cytoband already have some pre-defined colors which mimic the color you observed under microscope. Function scale_fill_giemsa did this trick to correct the color. If it's first time you observe usage by +, it's a very popular API in package *ggplot2*[2], which could add graphics layer by layer or revise a existing graphic.

Let's try a different data set which is not an 'ideogram', but a normal GRanges object that most people will have, extra data such as statistical values or categorical levels are stored in element data columns used for aesthetics mapping.

We use a default data in package *biovizBase*, which is a subset of RNA editing set in human. The data involved in this GRanges is sparse, so we cannot simply use it to make karyogram, otherwise, the estimated chromosome lengths will be very rough and inaccurate. So what we need to do is:

1. Adding seqlegnths to this GRanges object. If you adding seqlengths to object, we have two ways to show chromosome space as karyogram.
   autoplot(object, layout = 'karyogram') or
   autoplot(seqinfo(object)).
2. Changing the order of chromosomes.
3. Visualize it and map variable to different aesthetics.

Then we take one step further, the power of *ggplot2* or *ggbio* is the flexible multivariate data mapping ability in graphics, make data exploration much more convenient. In the following example, we are trying to map a categorical variable 'exReg' to color, this variable is included in the data, and have three levels, '3' indicate 3' utr, '5' means 5' utr and 'C' means coding region. We have some missing values indicated as NA, in default, it's going to be shown in gray color, and keep in mind, since the basic geom(geometric object) is rectangle, and genome space is very large, so change both color/fill color of the rectangle to specify both border and filled color is necessary to get the data shown as different color, otherwise if the region is too small, border color is going to override the fill color.

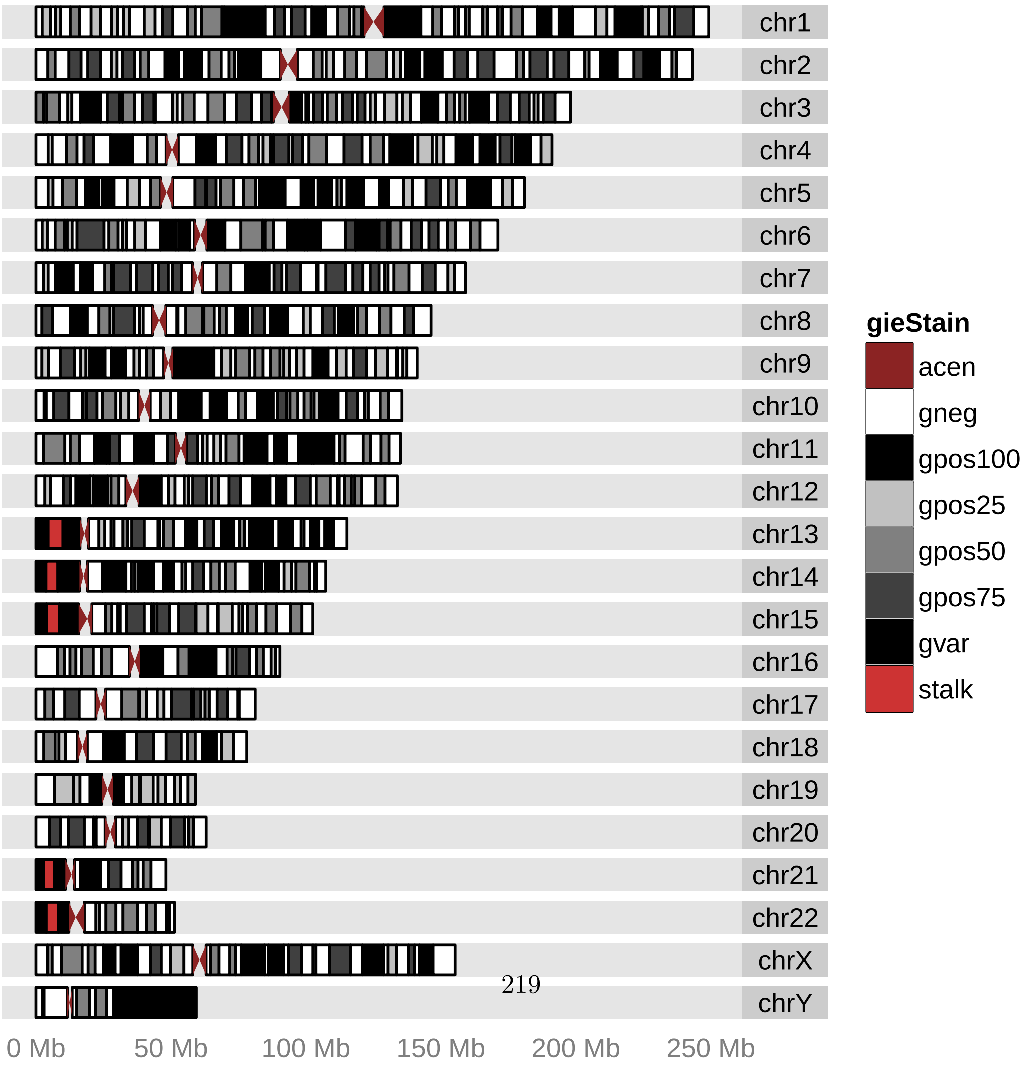
Or you can set the missing value to particular color you want.

A test could be performed to demonstrate why 'seqlengths' of object GRanges is important. Let's assume we set wrong chromosome lengths by accident, lengths are all equal to chromosome 1. We arbitrarily set it to the same number so that every chromosome are of equal length. From Figure 11.6, it's clear that this will affect what we see. So please make sure

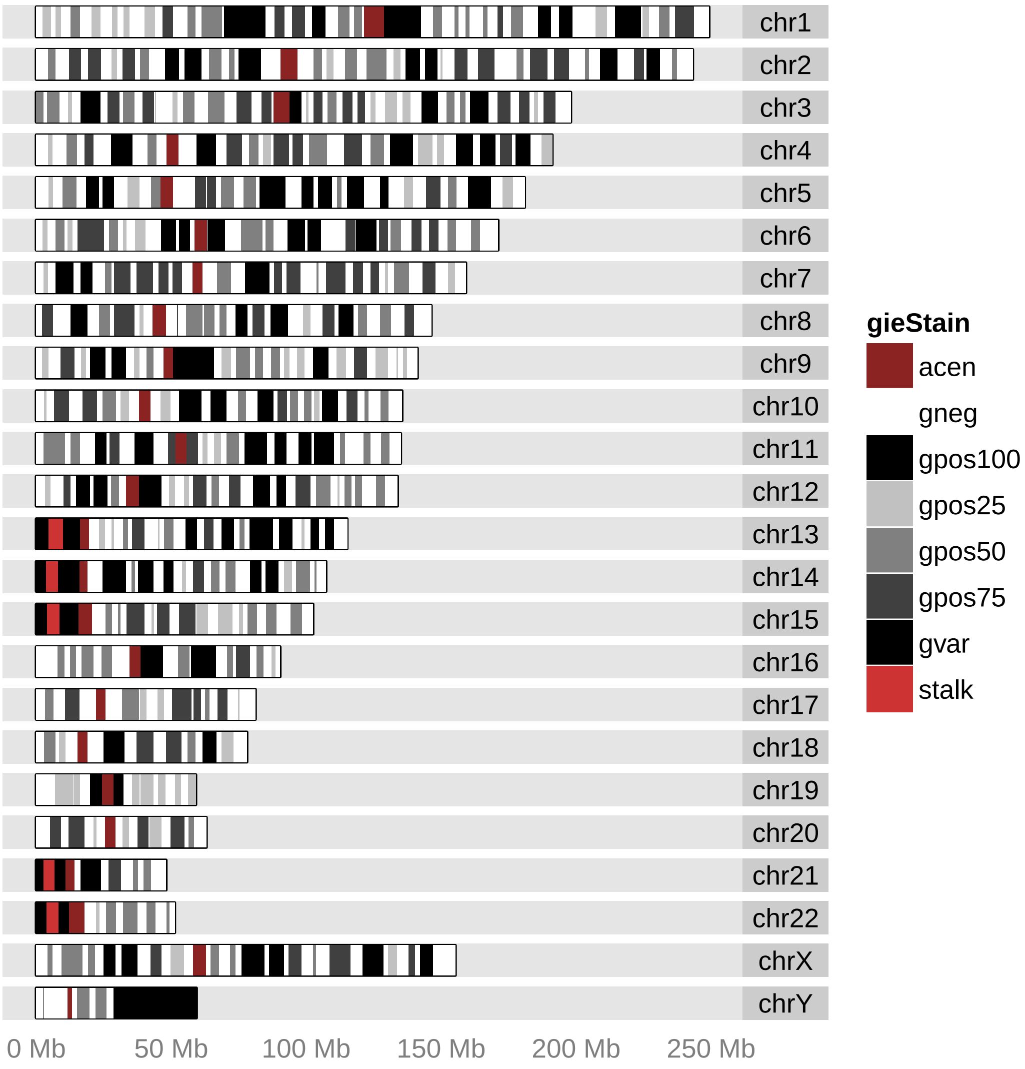
- You get data space cover exactly the same chromosome space for each chromosome. or

[2] http://had.co.nz/ggplot2/

```
library(GenomicRanges)
hg19 <- keepSeqlevels(hg19IdeogramCyto, paste0("chr", c(1:22, "X", "Y")))
head(hg19)

## GRanges with 6 ranges and 2 metadata columns:
##       seqnames               ranges strand |     name gieStain
##          <Rle>            <IRanges>  <Rle> | <factor> <factor>
##   [1]     chr1 [       0,  2300000]      * |   p36.33     gneg
##   [2]     chr1 [ 2300000,  5400000]      * |   p36.32   gpos25
##   [3]     chr1 [ 5400000,  7200000]      * |   p36.31     gneg
##   [4]     chr1 [ 7200000,  9200000]      * |   p36.23   gpos25
##   [5]     chr1 [ 9200000, 12700000]      * |   p36.22     gneg
##   [6]     chr1 [12700000, 16200000]      * |   p36.21   gpos50
##   ---
##   seqlengths:
##     chr1  chr2  chr3  chr4  chr5  chr6 ... chr20 chr21 chr22  chrX  chrY
##       NA    NA    NA    NA    NA    NA ...    NA    NA    NA    NA    NA

autoplot(hg19, layout = "karyogram", cytoband = TRUE)

## Object of class "ggbio"
```

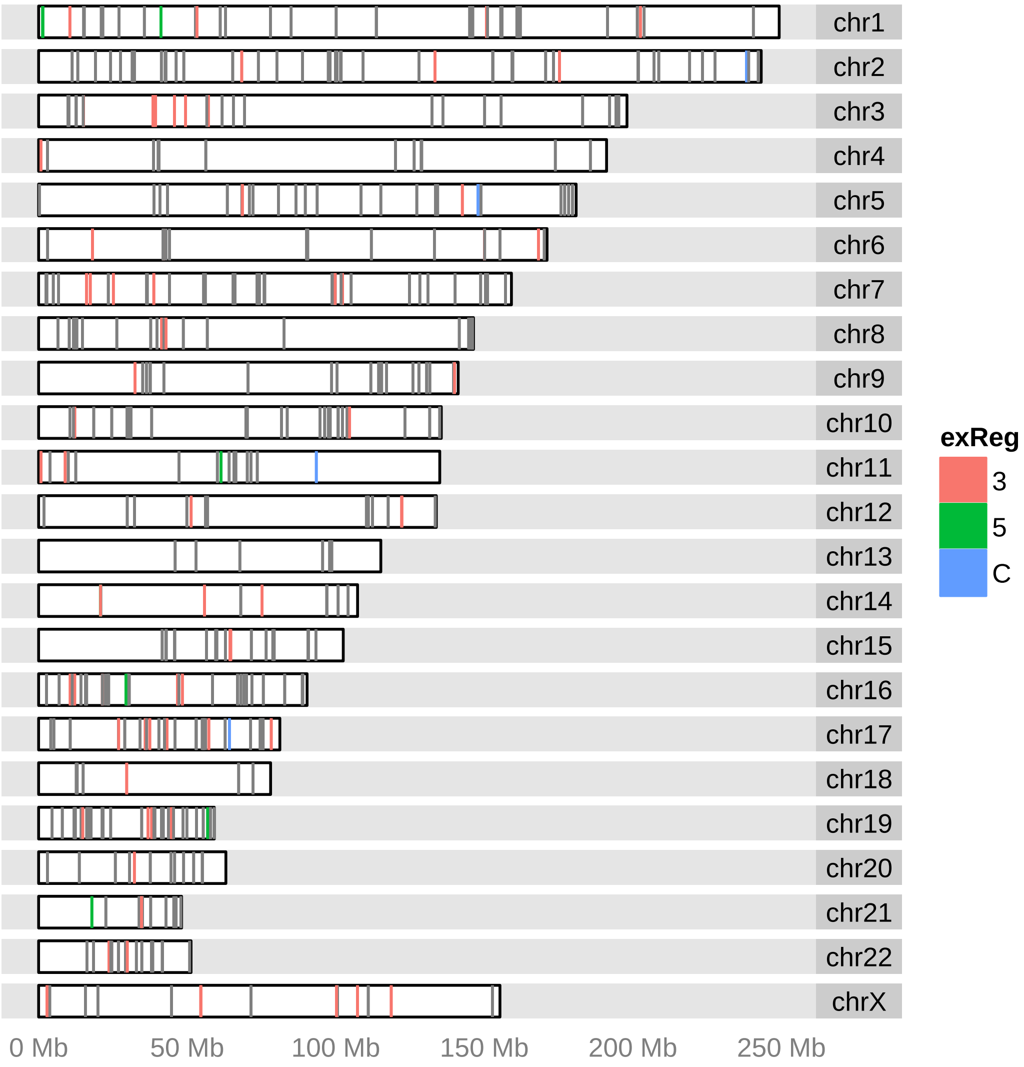


```
library(GenomicRanges)
## it's a 'ideogram'
biovizBase::isIdeogram(hg19)

## [1] TRUE

## set to FALSE
autoplot(hg19, layout = "karyogram", cytoband = FALSE, aes(fill = gieStain)) +
    scale_fill_giemsa()

   ## Warning:  geom(ideogram) need valid seqlengths information for accurate mapping,
             ##                     now use reduced information as ideogram...
   ## Warning:  geom(ideogram) need valid seqlengths information for accurate mapping,
             ##                     now use reduced information as ideogram...
  ## Scale for 'x' is already present.  Adding another scale for 'x', which will replace
                                  the existing scale.

## Object of class "ggbio"
```



```
## NULL
```

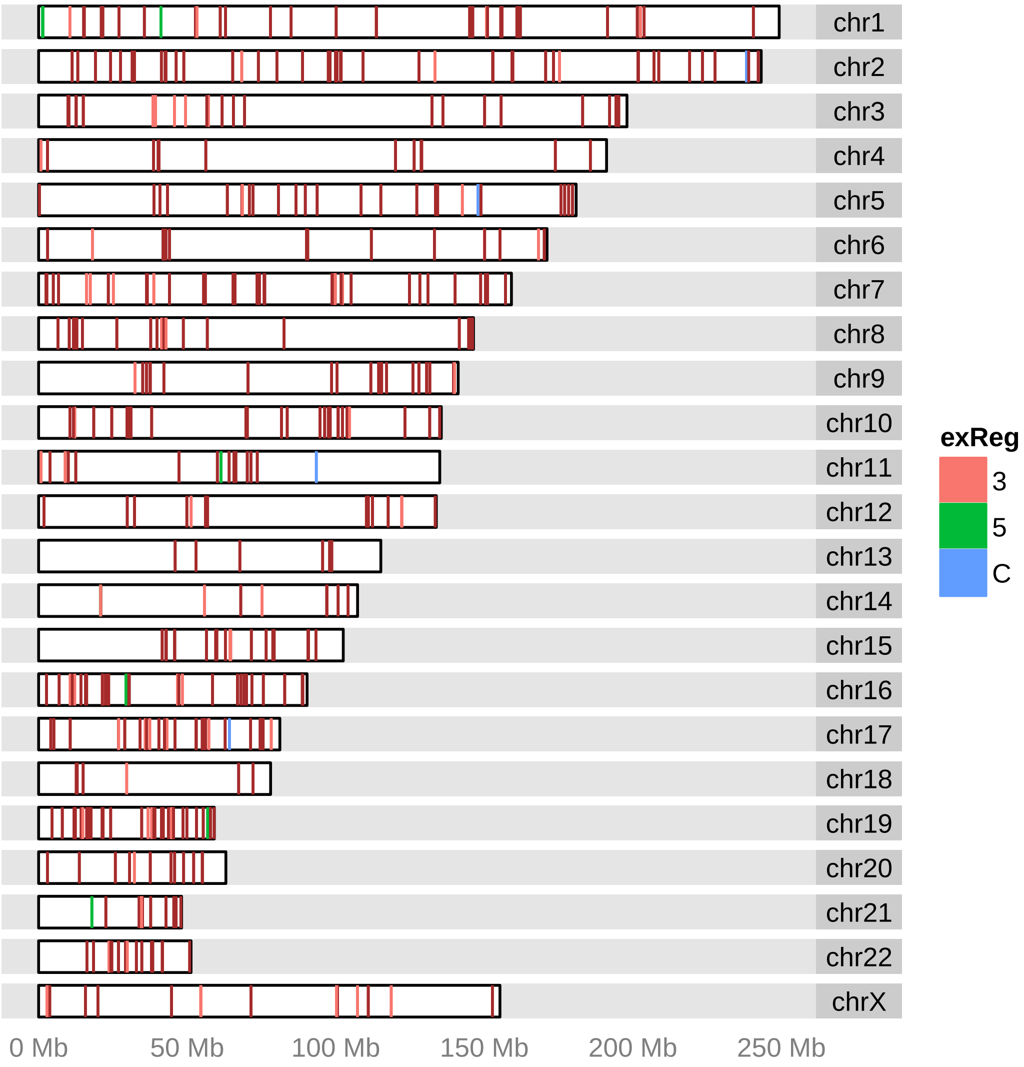
```
data(darned_hg19_subset500, package = "biovizBase")
dn <- darned_hg19_subset500
head(dn)

## GRanges with 6 ranges and 10 metadata columns:
##       seqnames                 ranges strand |       inchr       inrna
##          <Rle>              <IRanges>  <Rle> | <character> <character>
##   [1]     chr5 [ 86618225,  86618225]      - |           A           I
##   [2]     chr7 [ 99792382,  99792382]      - |           A           I
##   [3]    chr12 [110929076, 110929076]      - |           A           I
##   [4]    chr20 [ 25818128,  25818128]      - |           A           I
##   [5]     chr3 [132397992, 132397992]      + |           A           I
##   [6]    chr19 [ 59078471,  59078471]      - |           A           I
##               snp        gene      seqReg       exReg
##       <character> <character> <character> <character>
##   [1]        <NA>        <NA>           O        <NA>
##   [2]        <NA>        <NA>           O        <NA>
##   [3]        <NA>        <NA>           O        <NA>
##   [4]        <NA>        <NA>           O        <NA>
##   [5]        <NA>        <NA>           O        <NA>
##   [6]        <NA>        MZF1           I        <NA>
##                    source      ests      esta      author
##               <character> <integer> <integer> <character>
##   [1]           amygdala         0         0    15342557
##   [2]               <NA>         0         0    15342557
##   [3]     salivary gland         0         0    15342557
##   [4] brain, hippocampus         0         0    15342557
##   [5]    small intestine         0         0    15342557
##   [6]               <NA>         0         0    15258596
##   ---
##   seqlengths:
##     chr1 chr10 chr11 chr12 chr13 chr14 ...  chr6  chr7  chr8  chr9  chrX
##       NA    NA    NA    NA    NA    NA ...    NA    NA    NA    NA    NA

## add seqlengths we have seqlegnths information in another data set
data(hg19Ideogram, package = "biovizBase")
seqlengths(dn) <- seqlengths(hg19Ideogram)[names(seqlengths(dn))]
## now we have seqlengths
head(dn)

## GRanges with 6 ranges and 10 metadata columns:
##       seqnames                 ranges strand |       inchr       inrna
##          <Rle>              <IRanges>  <Rle> | <character> <character>
##   [1]     chr5 [ 86618225,  86618225]      - |           A           I
##   [2]     chr7 [ 99792382,  99792382]      - |           A           I
##   [3]    chr12 [110929076, 110929076]      - |           A           I
##   [4]    chr20 [ 25818128,  25818128]      - |           A           I
##   [5]     chr3 [132397992, 132397992]      + |           A           I
##   [6]    chr19 [ 59078471,  59078471]      - |           A           I
##               snp        gene      seqReg       exReg
##       <character> <character> <character> <character>
##   [1]        <NA>        <NA>           O        <NA>
##   [2]        <NA>        <NA>           O        <NA>
##   [3]        <NA>        <NA>           O        <NA>
##   [4]        <NA>        <NA>           O        <NA>
##   [5]        <NA>        <NA>           O        <NA>
##   [6]        <NA>        MZF1           I        <NA>
##                    source      ests      esta      author
##               <character> <integer> <integer> <character>
## since default is geom rectangle, even though it's looks like segment
## we still use both fill/color to map colors
autoplot(dn, layout = "karyogram", aes(color = exReg, fill = exReg))

  ## Scale for 'x' is already present.  Adding another scale for 'x', which will replace
                                    the existing scale.

## Object of class "ggbio"
```



```
## NULL
```

Figure 11.4: Karyogram for RNA-editing sites, and map color to exReg column, which means exon region. '3' indicate 3' utr, '5' means 5' utr and 'C' means coding region, NA indicate missing value(or not in other three levels) shown as gray color.

```
## since default is geom rectangle, even though it's looks like segment
## we still use both fill/color to map colors
autoplot(dn, layout = "karyogram", aes(color = exReg, fill = exReg)) + scale_color_discrete(na.value = "b

  ## Scale for 'x' is already present.  Adding another scale for 'x', which will replace
                                the existing scale.

## Object of class "ggbio"
```



```
## NULL
```

Figure 11.5: Karyogram for RNA-editing sites, and map color to exReg column, which means exon region. '3' indicate 3' utr, '5' means 5' utr and 'C' means coding region, we force the missing value(NA) shown as brown color.

- You set the seqlengths to the right number.

Otherwise you will see weird pattern from your results, so actually it's a good way to test your raw data too, if you raw data have something beyond chromosome space, you need to dig into it to see what happened.

11.2.2 `plotKaryogram`

`plotKaryogram` (or `plotStackedOverview`) are specialized function to draw karyogram graphics. It's actually what function `autoplot` calls inside. API is a littler simpler because layout 'karyogram' is default in these two functions. So equivalent usage is like

```
plotKaryogram(dn)
plotKaryogram(dn, aes(color = exReg, fill = exReg))
```

11.2.3 `layout_karyogram`

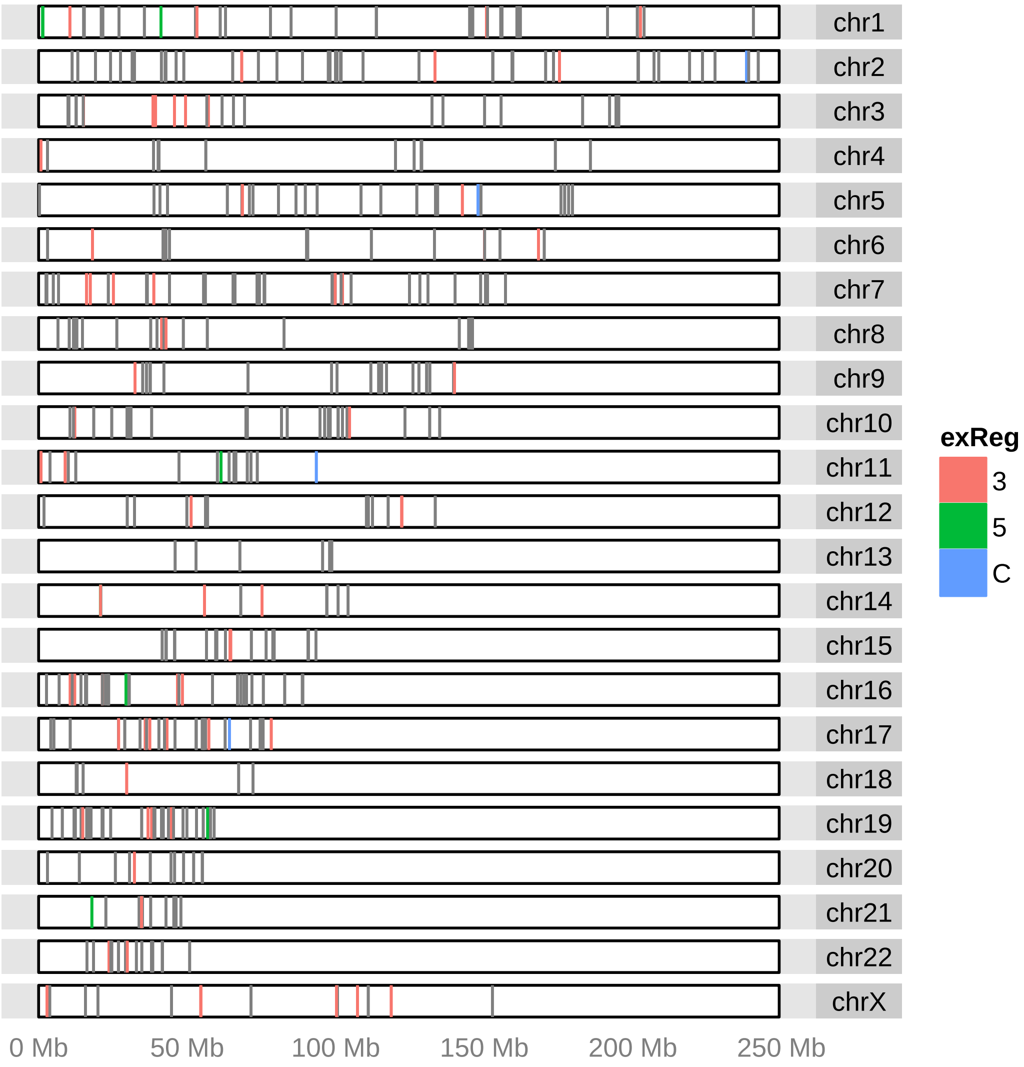
In this section, a lower level function `layout_karyogram` is going to be introduced. This is convenient API for constructing karyogram plot and adding more data layer by layer. Function `ggplot` is just to create blank object to add layer on.

You need to pay attention to

- when you add plots layer by layer, seqnames of different data must be the same to make sure the data are mapped to the same chromosome. For example, if you name chromosome following schema like *chr1* and use just number *1* to name other data, they will be treated as different chromosomes.
- cannot use the same aesthetics mapping multiple time for different data. For example, if you have used aes(color =), for one data, you cannot use aes(color =) anymore for mapping variables from other add-on data, this is currently not allowed in *ggplot2*, even though you expect multiple color legend shows up, this is going to confuse people which is which. HOWEVER, `color` or `fill` without `aes()` wrap around, is allowed for any track, it's set single arbitrary color. This is shown in Figure 11.8
- Default rectangle y range is [0, 10], so when you add on more data layer by layer on existing graphics, you can use `ylim` to control how to normalize your data and plot it relative to chromosome space. For example, with default, chromosome space is plotted between y [0, 10], if you use `ylim = c(10 , 20)`, you will stack data right above each chromosomes and with equal width. For geom like 'point', which you need to specify 'y' value in `aes()`, we will add 5% margin on top and at bottom of that track.

Then we construct another multiple layer graphics for multiple data using different geom, suppose we want to show RNA-editing sites on chromosome space as rectangle(looks like segment in graphic) and stack a line for another track above.

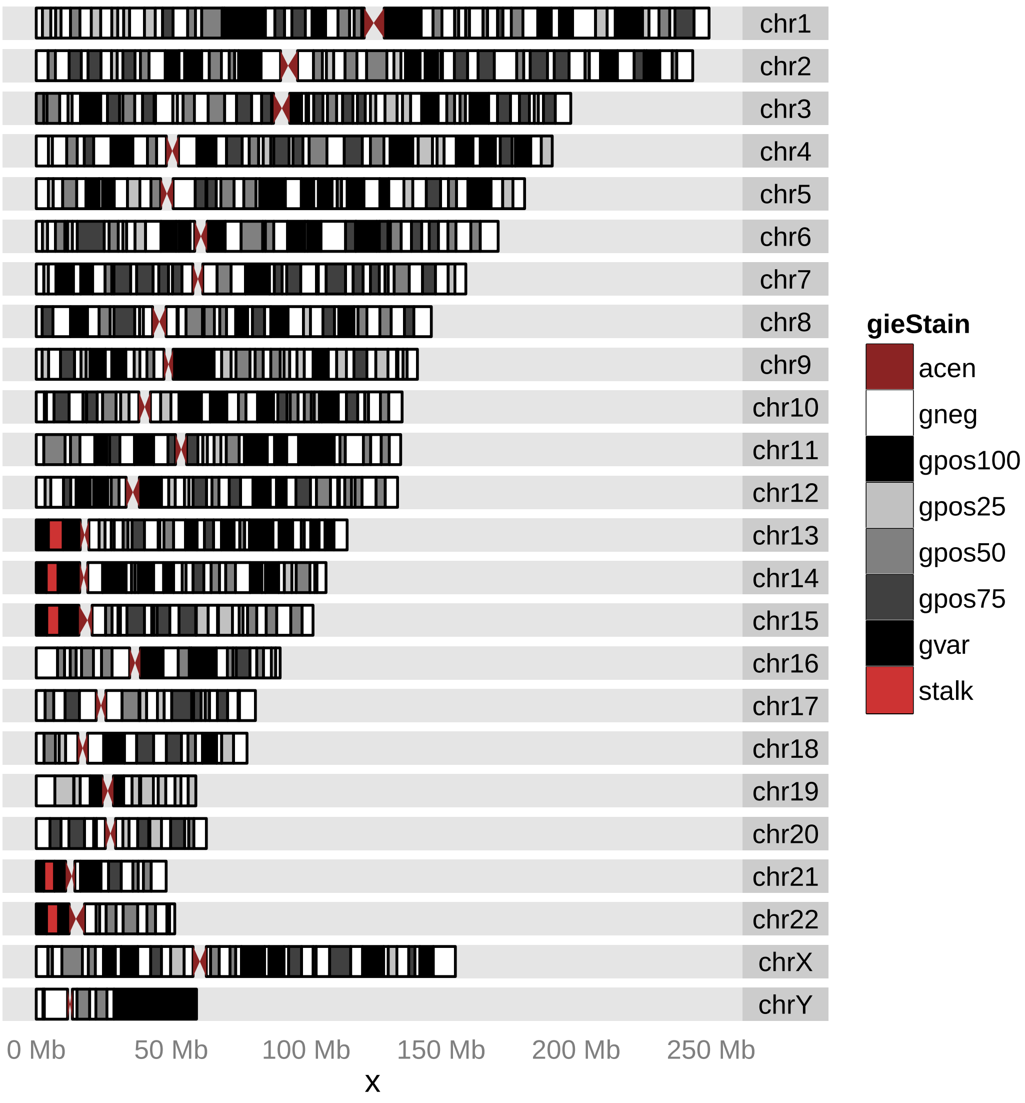
```
dn2 <- dn
seqlengths(dn2) <- rep(max(seqlengths(dn2)), length(seqlengths(dn2)))
autoplot(dn2, layout = "karyogram", aes(color = exReg, fill = exReg))

  ## Scale for 'x' is already present.  Adding another scale for 'x', which will replace
                                   the existing scale.

## Object of class "ggbio"
```



```
## NULL
```

Figure 11.6: Karyogram for RNA-editing sites, and map color to exReg column, which means exon region. '3' indicate 3' utr, '5' means 5' utr and 'C' means coding region, we force the missing value(NA) shown as brown color.

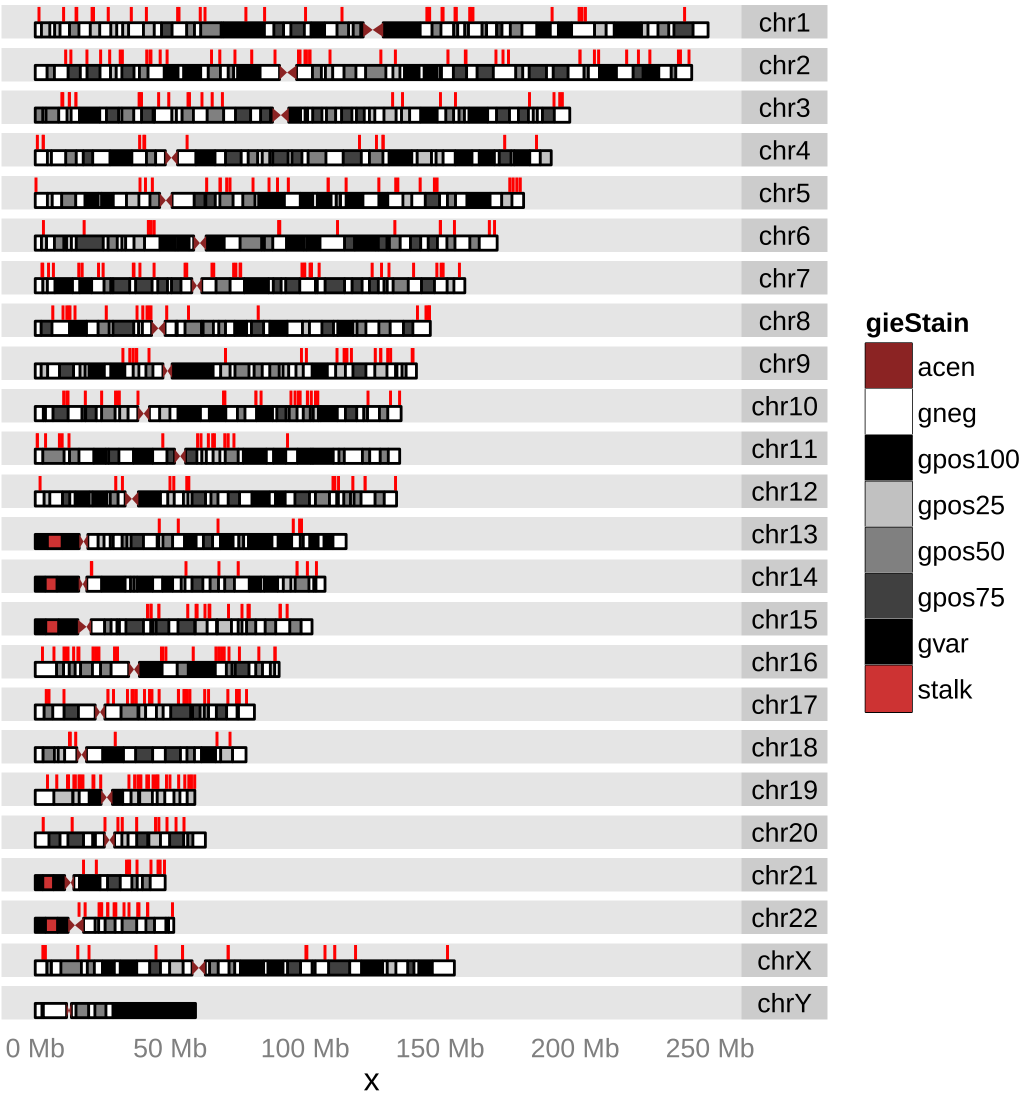
```
## plot ideogram
p <- ggplot(hg19) + layout_karyogram(cytoband = TRUE)
p

## Object of class "ggbio"
```



```
## NULL

## eqevelant autoplot(hg19, layout = 'karyogram', cytoband = TRUE)
```

Figure 11.7: Ideogram overview by using the function layout_karyogram

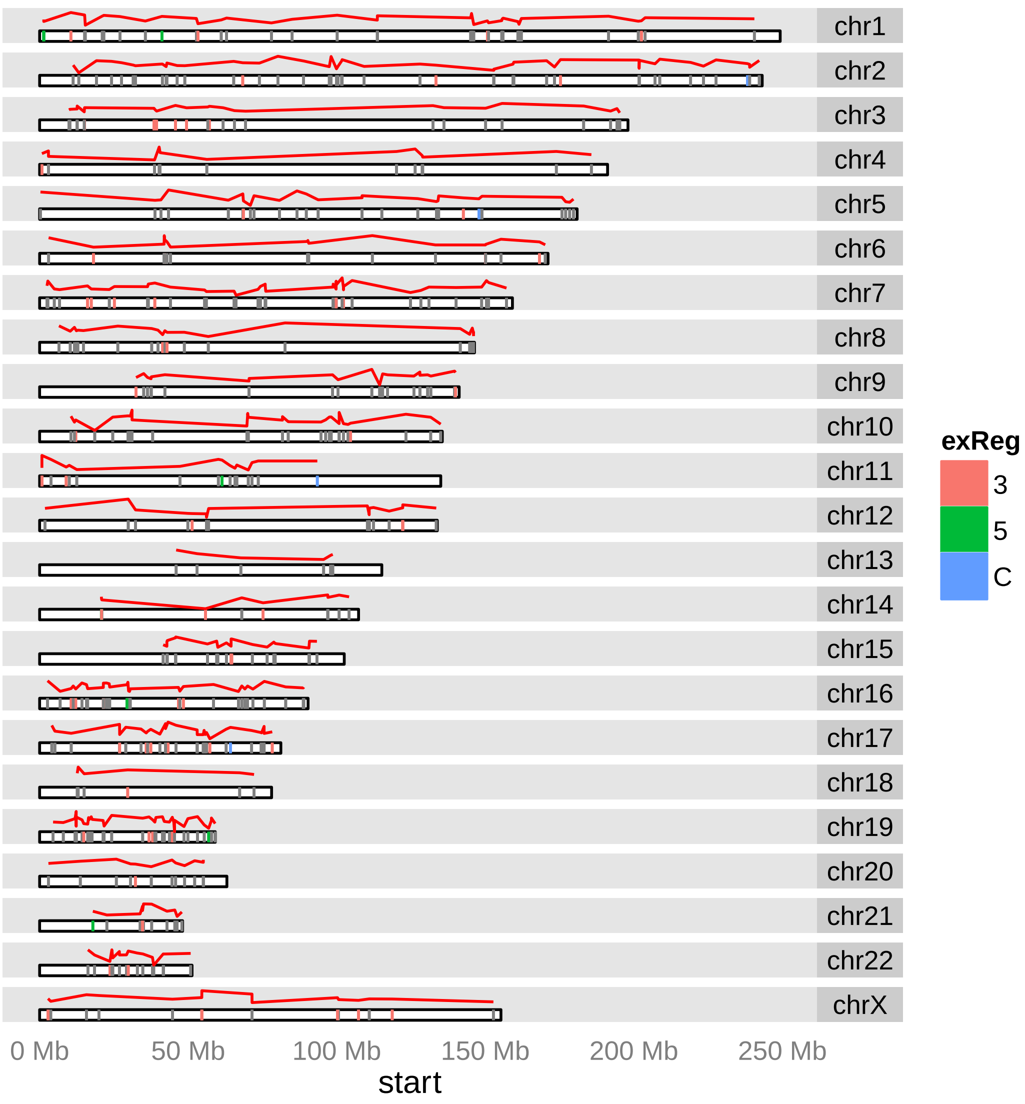
```
p <- p + layout_karyogram(dn, geom = "rect", ylim = c(11, 21), color = "red")

  ## Scale for 'x' is already present.  Adding another scale for 'x', which will replace
                                  the existing scale.

## commented line below won't work the cytoband fill color has been
## used already.  p <- p + layout_karyogram(dn, aes(fill = exReg, color
## = exReg), geom = 'rect')
p

## Object of class "ggbio"
```



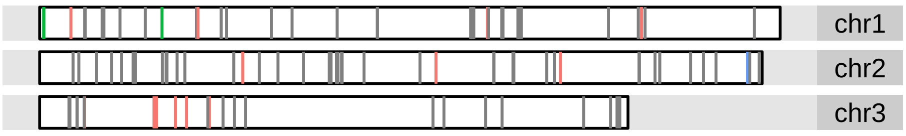
```
## NULL
```

Figure 11.8: We layout another track(data) which is RNA-editing sites on top of ideogram. Notice since legend fill and color is used, we cannot specify that in RNA-editing track, we could only set it to arbitrary color.

```
## plot chromosome space
p <- autoplot(seqinfo(dn))
## make sure you pass rect as geom otherwise you just get background
p <- p + layout_karyogram(dn, aes(fill = exReg, color = exReg), geom = "rect")

  ## Scale for 'x' is already present.  Adding another scale for 'x', which will replace
                                      the existing scale.

values(dn)$pvalue <- rnorm(length(dn))
p + layout_karyogram(dn, aes(x = start, y = pvalue), ylim = c(10, 30), geom = "line",
    color = "red")

  ## Scale for 'x' is already present.  Adding another scale for 'x', which will replace
                                      the existing scale.
```



```
p
```



Chapter 12

Ranges-link-to-data plot

12.1 Introduction

Ranges linked data is similar to parallel coordinate plots, could be used to transform information from sparse or uneven space to uniformed space, then observe multivariate data change patterns by linking the value within group. This view is inspired by a view in package *DEXseq*. Here we generalize it first to GRanges then later other more convenient object.

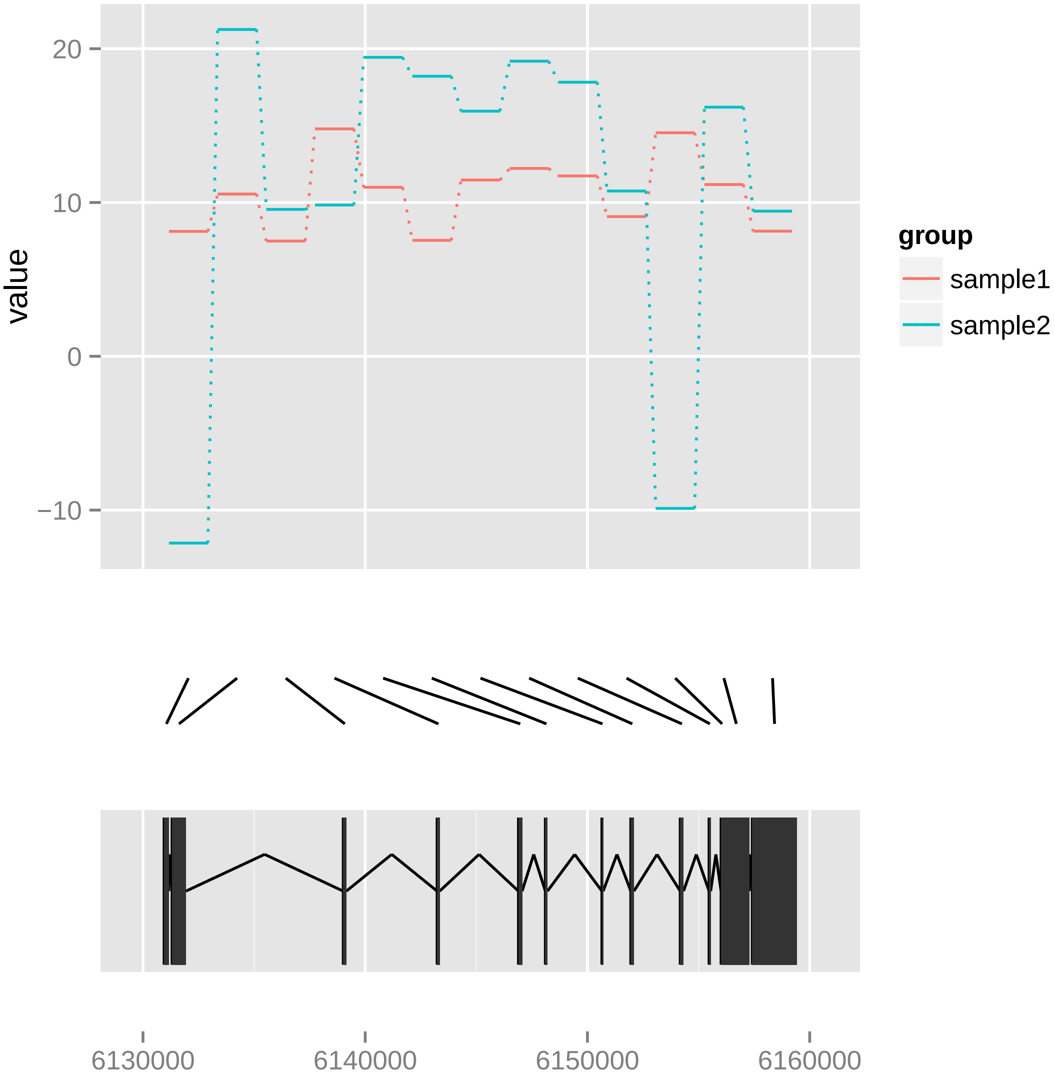
Suppose we have a matrix storing expression levels for each exons, each row indicate the interval, each column indicate the sample. we can store these values into elementMetadata of GRanges object.

First we simulated a data like this, suppose we have two samples, named 'sample1' and 'sample2', then we create a column to indicate they are significant or not, named 'significant' filled with value TRUE/FALSE or 1/0.

```
library(TxDb.Hsapiens.UCSC.hg19.knownGene)
library(ggbio)
data(genesymbol, package = "biovizBase")
txdb <- TxDb.Hsapiens.UCSC.hg19.knownGene
model <- exonsBy(txdb, by = "tx")
model17 <- subsetByOverlaps(model, genesymbol["RBM17"])
exons <- exons(txdb)
exon17 <- subsetByOverlaps(exons, genesymbol["RBM17"])
## reduce to make sure there is no overlap just for example
exon.new <- reduce(exon17)
## suppose
set.seed(1)
values(exon.new)$sample1 <- rnorm(length(exon.new), 10, 3)
values(exon.new)$sample2 <- rnorm(length(exon.new), 10, 10)
values(exon.new)$significant <- c(TRUE, rep(FALSE, length(exon.new) - 1))
head(exon.new)


## GRanges with 6 ranges and 3 metadata columns:
##       seqnames             ranges strand |          sample1
##          <Rle>          <IRanges>  <Rle> |        <numeric>
##   [1]    chr10 [6130949, 6131156]      + |   8.120638567773
##   [2]    chr10 [6131309, 6131934]      + | 10.5509299726662
##   [3]    chr10 [6139011, 6139151]      + | 7.49311416276986
##   [4]    chr10 [6143234, 6143350]      + | 14.7858424064134
##   [5]    chr10 [6146894, 6147060]      + | 10.9885233154461
##   [6]    chr10 [6148104, 6148201]      + | 7.53859484764595
##                 sample2 significant
##               <numeric>   <logical>
##   [1] -12.146998871775           1
##   [2] 21.2493091814311           0
##   [3] 9.55066390984769           0
##   [4] 9.83809736901054           0
##   [5]  19.438362106853           0
##   [6] 18.2122119509809           0
##   ---
##   seqlengths:
##                    chr1                 chr2 ...       chrUn_gl000249
##               249250621            243199373 ...                38502
```

`stat.col` accept column names or column index in *element meta data*, so 1 doesn't mean 'seqnames'.

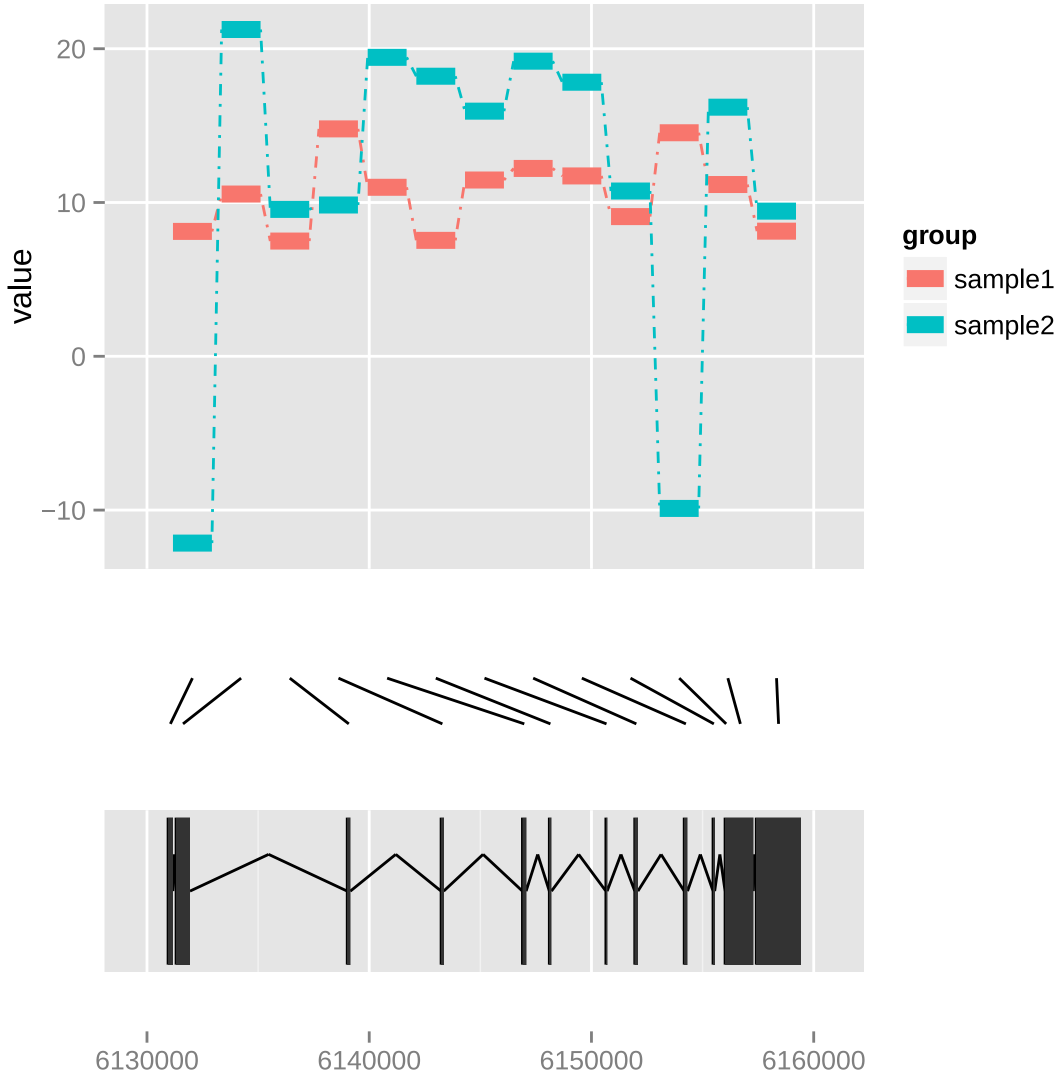
```
plotRangesLinkedToData(exon.new, stat.col = 1:2)


## Scale for 'y' is already present.  Adding another scale for 'y', which will replace the
existing scale.
```



```
## equivilent to plotRangesLinkedToData(exon.new, stat.col =
## c('sample1', 'sample2'))
```

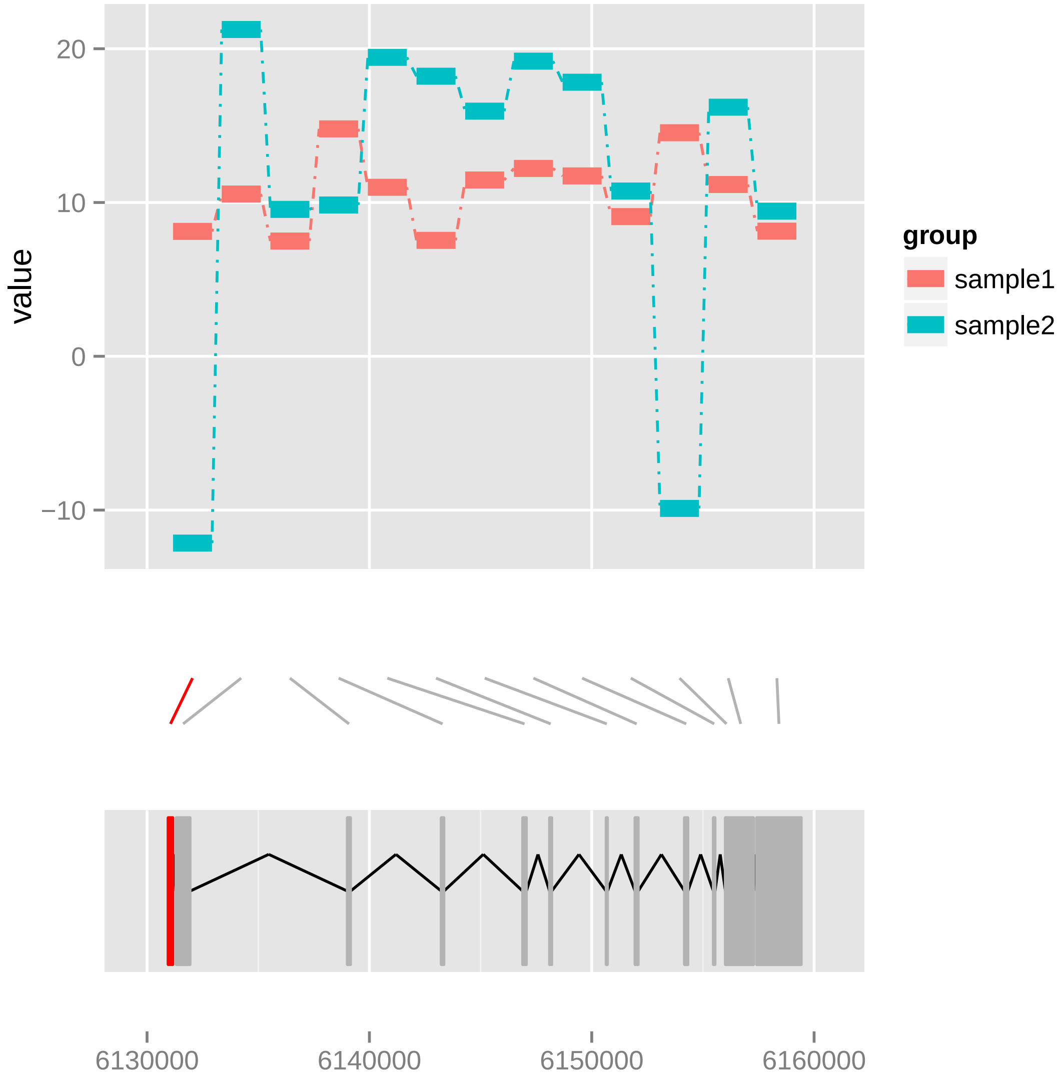
we can specify line size and type.

```
plotRangesLinkedToData(exon.new, stat.col = 1:2, size = 3, linetype = 4)


## Scale for ’y’ is already present.  Adding another scale for ’y’, which will replace the
existing scale.
```



And use argument `sig` to indicate the column name which stored the significance bool value, and `sig.col` indicate the insignificant color and significant color.

```
plotRangesLinkedToData(exon.new, stat.col = 1:2, size = 3, linetype = 4,
    sig = "significant", sig.col = c("gray70", "red"))


## Scale for 'y' is already present.  Adding another scale for 'y', which will replace the
existing scale.
```


20
10
0
−10
value
group
sample1
sample2
6130000
6140000
6150000
6160000

Chapter 13

Case studies

13.1 Chip-seq

13.1.1 Introduction

In this tutorial, we are going to use *chipseq* package to analyze some example ChIP-seq data and explore them by visualization of *ggbio*

Example data we used in this tutorial, is called *cstest*, which is a data set in package *chipseq*. This is a small subset of data downloaded fro SRA data base includes two lanes representing CTCF and GFP pull-down in mouse. More information about this data could be found in the manual of package *chipseq*.

13.1.2 Usage

Processing and fragment estimation

Firstly, we mainly follow workflow described in vignette of package *chipseq*, except we remove unused seqnames(chromosome names) in the data, from the data we could see that only chromosome 10, 11, 12 involved, the reason we removed too many unused seq levels from the data is that, in *ggbio*, most time, it will plot every chromosomes in the data even there is no data at all, this will take too much space for visualization.

```
library(chipseq)


## Loading required package:  ShortRead

## Loading required package:  lattice

## Loading required package:  latticeExtra

## Loading required package:  RColorBrewer

##
## Attaching package:  'latticeExtra'
## The following object(s) are masked from 'package:ggplot2':
##
##     layer

##
## Attaching package:  'ShortRead'

## The following object(s) are masked from 'package:VariantAnnotation':
##
##     compose, name, stats
```

```
library(GenomicFeatures)
data(cstest)
unique(seqnames(cstest))
```

```
## CompressedCharacterList of length 2
## [["ctcf"]] chr10 chr11 chr12
## [["gfp"]] chr10 chr11 chr12
```

```
## subset
chrs <- c("chr10", "chr11", "chr12")
cstest <- keepSeqlevels(cstest, chrs)
## estimate fragment length
fraglen <- estimate.mean.fraglen(cstest$ctcf)
fraglen[!is.na(fraglen)]
```

```
## chr10 chr11 chr12
## 179.7 172.5 181.7
```

```
## that's around 200 cstest.gr <- stack(cstest) head(cstest.gr)
## cstest.ext <- resize(cstest.gr, width = 200) extending them
ctcf.ext <- resize(cstest$ctcf, width = 200)
cov.ctcf <- coverage(ctcf.ext)
gfp.ext <- resize(cstest$gfp, width = 200)
cov.gfp <- coverage(gfp.ext)
## estimate peak cutoff
peakCutoff(cov.ctcf, fdr = 1e-04)
```

```
## [1] 6.96
```

```
## we can use 8
```

To understand why we are extending reads to estimated fragment lengths, we first find two peaks that from negative/positive strands separately which close to each other. Then we simply visualize that region and compare it to what it is after extending.

```r
c.p <- cstest$ctcf[seqnames(cstest$ctcf) == "chr10" & strand(cstest$ctcf) ==
    "+", ]

c.n <- cstest$ctcf[seqnames(cstest$ctcf) == "chr10" & strand(cstest$ctcf) ==
    "-", ]


cov.p <- coverage(c.p)
cov.n <- coverage(c.n)
v1 <- viewWhichMaxs(slice(cov.p, lower = 8))$chr10
v2 <- viewWhichMaxs(slice(cov.n, lower = 8))$chr10
res <- expand.grid(v1, v2)
wh <- as.numeric(res[order(abs(res[, 1] - res[, 2]))[1], ])
gr.wh <- GRanges("chr10", IRanges(wh[1], wh[2]))
gr.wh <- resize(gr.wh, width(gr.wh) + 200, fix = "center")
```

Then we use *ggbio* to visualize those short reads first, notice they are shorter(width:24) than esitmated fragment lengths(200). That may make one single peak looks like two peaks. Here we use `autoplot` for object `GRanges`. To tell different reads from different strand, we facet and filled the rectangles by strands. Figure 13.1 shows the effect of resizing.

Keep in mind, you don't want to viualize all the short reads at once, that's going to be crazily slow for NGS data, and it's not useful for exploration. In this example, we subset the reads by small region, that will give you quick response. For genome-wide visualization, you should try from `autoplot` for `Rle` or `RleList` method, which is lots faster and meaningful, we will introduce this method soon in this tutorial.

```
library(ggbio)
ctcf.sub <- subsetByOverlaps(cstest$ctcf, gr.wh)
p1 <- autoplot(ctcf.sub, aes(fill = strand), facets = strand ~ .)
ctcf.ext.sub <- subsetByOverlaps(ctcf.ext, gr.wh)
p2 <- autoplot(ctcf.ext.sub, aes(fill = strand), facets = strand ~ .)
tracks(original = p1, extending = p2, heights = c(3, 5))
```

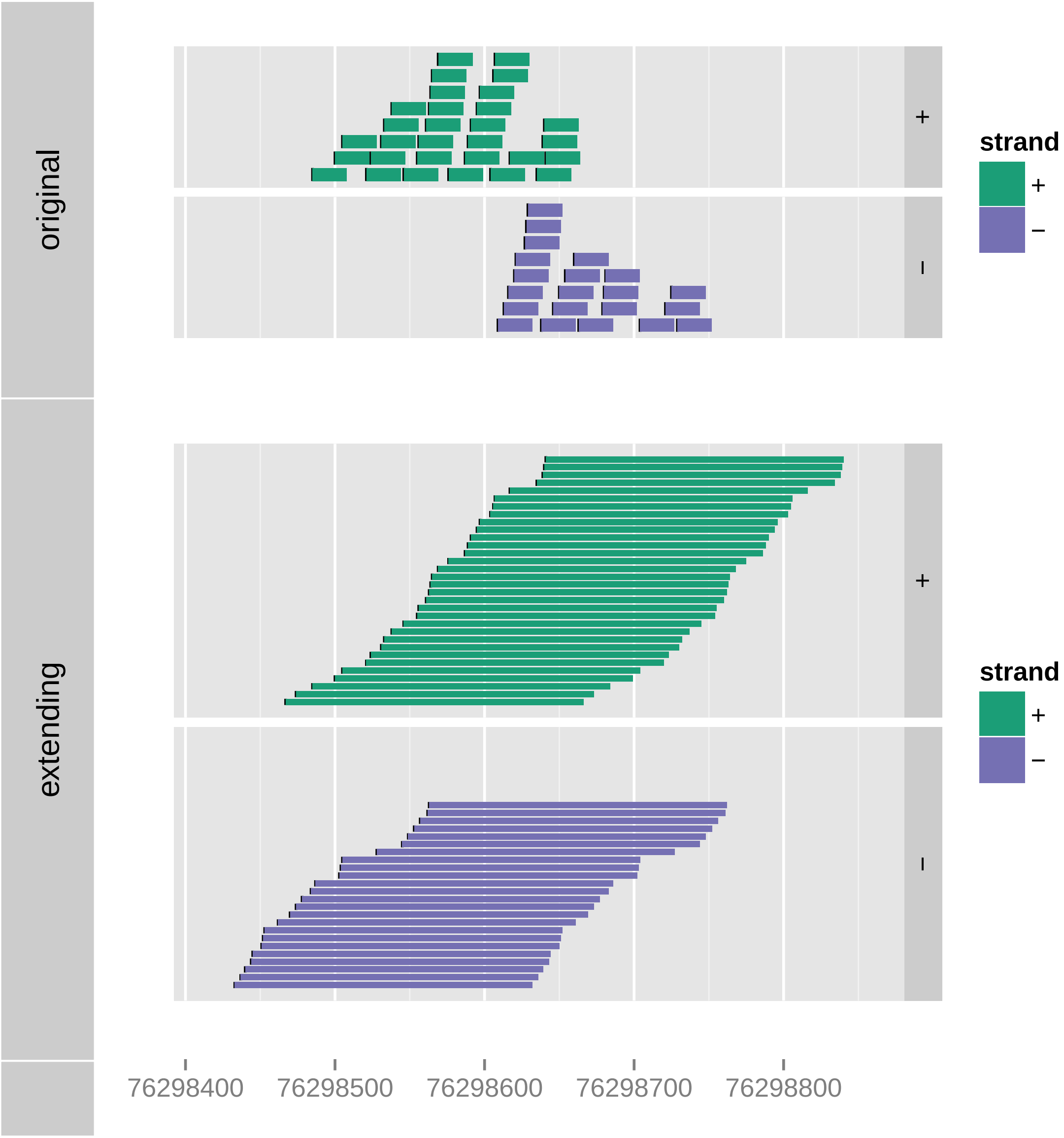


Figure 13.1: A small region on chromosome 10, each track are faceted by strand. Top track shows short reads of around width 24, and bottom track shows the same data with extending width to 200. The order of short reads are randomly assigned at different levels, so hard to match each reads at exactly the same position.

Maybe reads are not quite easy to perceive the effect of resizing, we use statistical transformation "coverage" to make better illustration. In Figure 13.2, you can clearly see why we need to extending reads to get a better estimation of peaks. In this plot, two peaks are about to merge together to one single peak. That means most possible, there are only one binding site.

```
ctcf.sub <- subsetByOverlaps(cstest$ctcf, gr.wh)
p1 <- autoplot(ctcf.sub, aes(fill = strand), facets = strand ~ ., stat = "coverage",
    geom = "area")
ctcf.ext.sub <- subsetByOverlaps(ctcf.ext, gr.wh)
p2 <- autoplot(ctcf.ext.sub, aes(fill = strand), facets = strand ~ ., stat = "coverage",
    geom = "area")
tracks(original = p1, extending = p2)
```

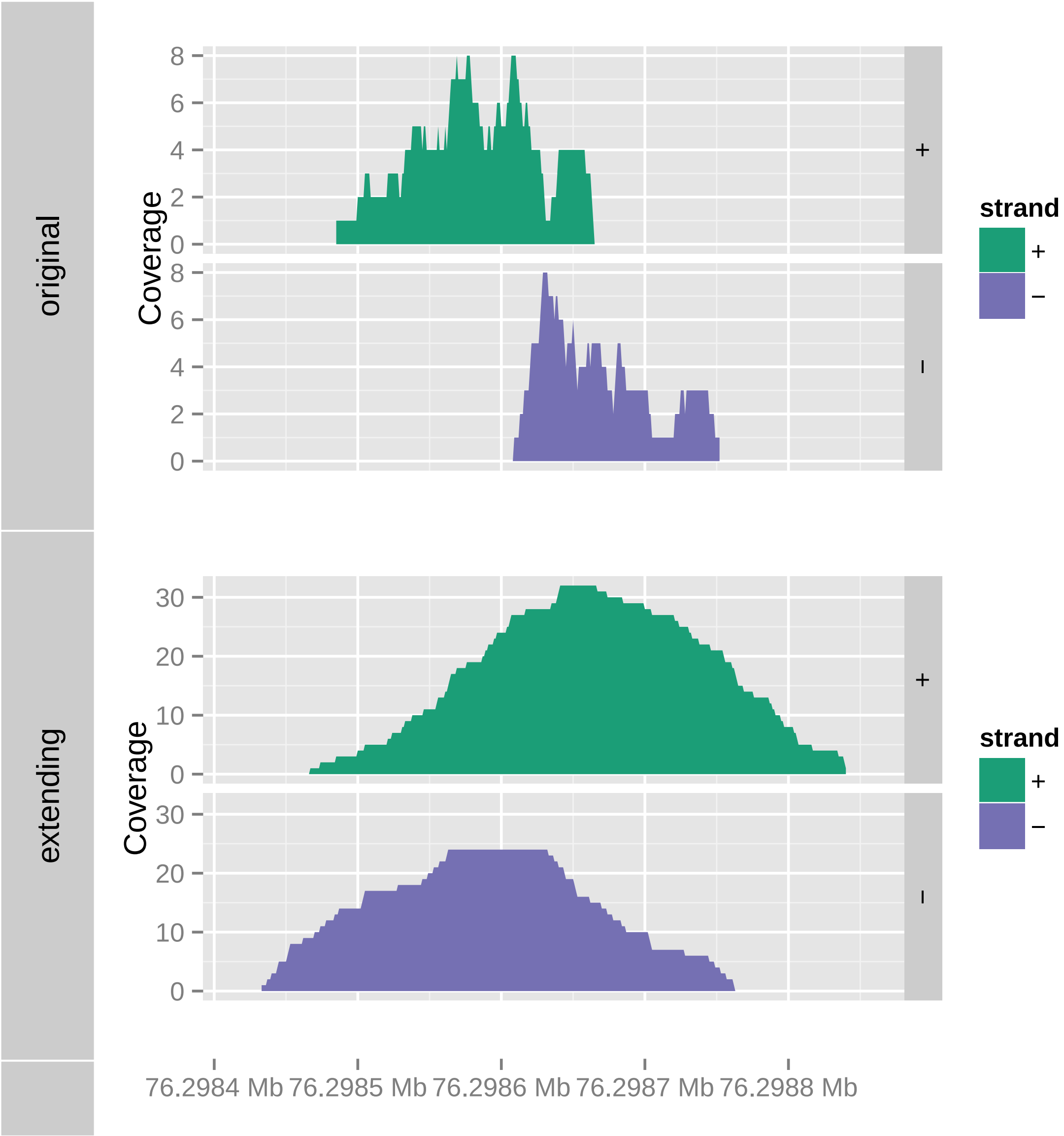


Figure 13.2: A small region on chromosome 10, each track are faceted by strand. Top track shows coverage of short reads of around width 24, and bottom track shows the same data with extending width to 200. Clearly two peaks are tend to merge to one single peak after resizing.

Finding islands and genome-wide visualization

As mentioned before, to visualize genonme-wide information, short-reads visualization is absolutely not recommended, a summary is way much better. We can compuate coveage as `Rle` (Run Length Encode), so we can perform efficient summary transformation like finding islands over certain cutoff, or bin them and show summary value as heatmap or bar chart.

In the following examples, we tried different visualization method.

There are three statistical transformation for object `Rle` and `Rle`:

- **bin**:(default). Bin the object and compute summary based on summary types mentioned below. `nbin` controls how many bins you want. geom *heatmap* and *bar*(default) supported.
- **slice**: slice the object based on certain cutoffs to generate islands, use specified summary method to generate values. geom *rect, bar, heatmap* to many other geoms such as *point, line, area* are supported.
- **identity**: raw sequence. Be careful if this object is too long to be visualized!

There are four types of summary method for statistical transformation **bin** and **slice**

- **ViewSums:** Sums for each sliced island or bins.
- **ViewMaxs:** Maxs for each sliced island or bins.
- **ViewMeans:** Means for each sliced island or bins.
- **ViewMins:** Mins for each sliced island or bins.

Figure 13.3 shows a default track.

```
library(ggbio)
p1 <- autoplot(cov.ctcf)

                              ## Default use binwidth:  range/30

p2 <- autoplot(cov.gfp)

                              ## Default use binwidth:  range/30

tracks(ctcf = p1, gfp = p2)
```

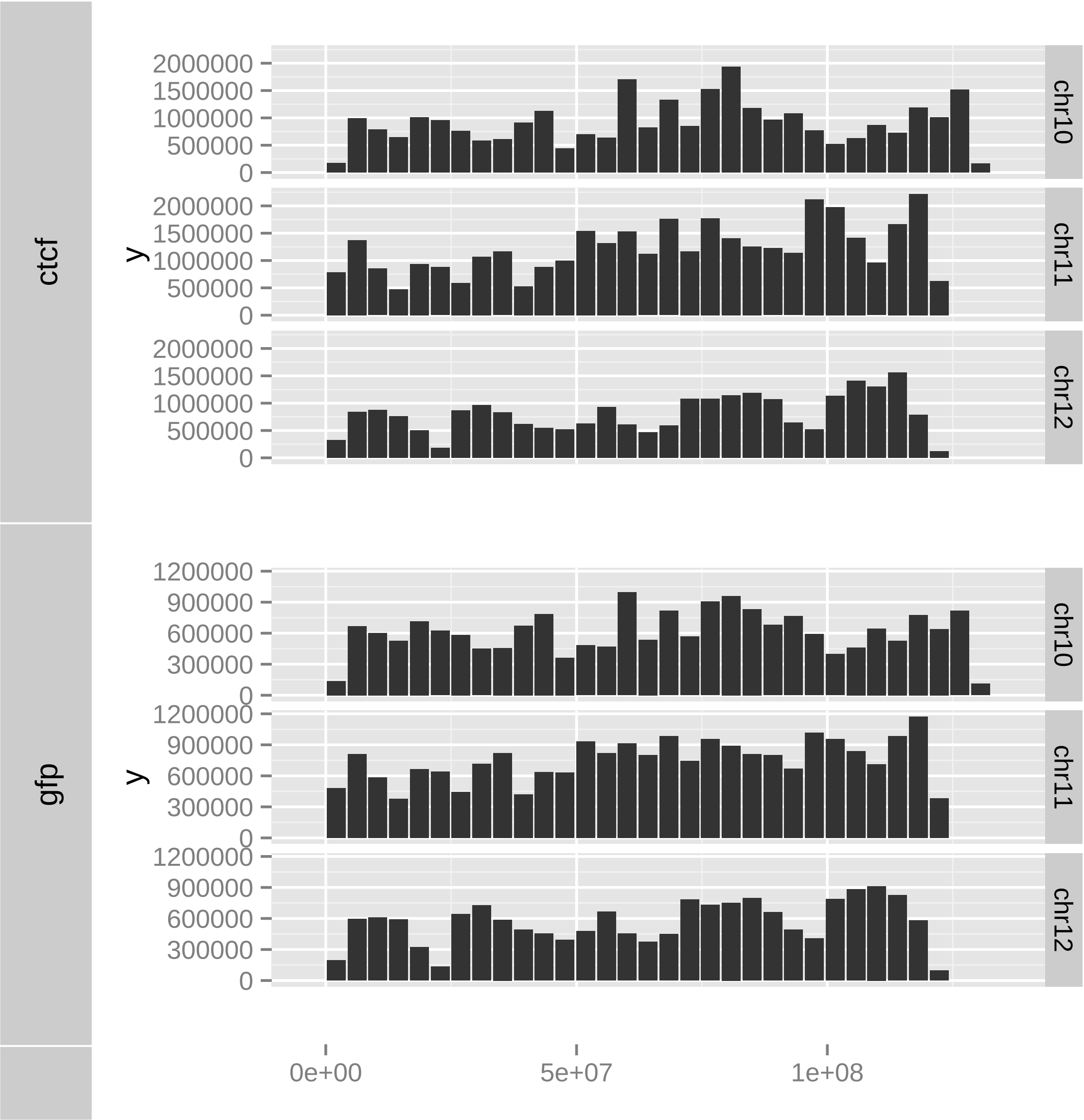
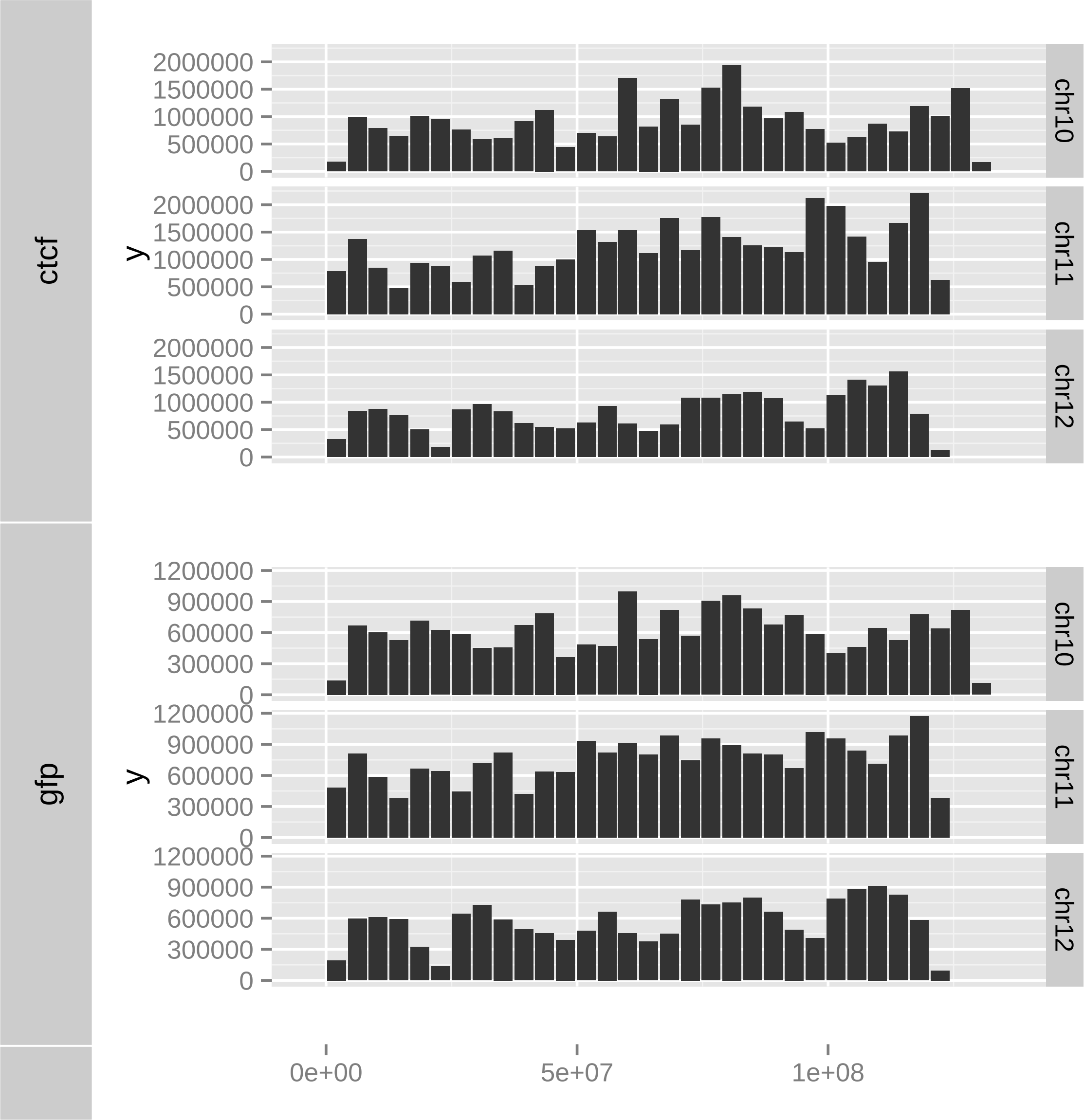


Figure 13.3: Use default statistical transformation "bin" and geom "bar" to represent default smumary ViewSums.

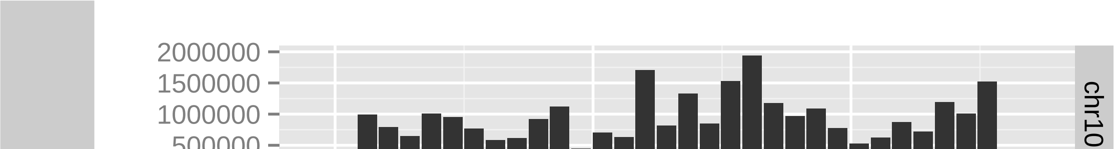
We may notice it's hard to compare the summary if limits on y are different, we have two ways to make them on the same scale. Because tracks by default keep original plots' y scale while change and align their x-scale.

- Aggregate all data into one single data and facet by samples.
- use "+" method to change overall y limits as shown in Figure 13.4.

```
library(ggbio)
p1 <- autoplot(cov.ctcf)

                               ## Default use binwidth:  range/30

p2 <- autoplot(cov.gfp)

                               ## Default use binwidth:  range/30

## doesn't work?
tracks(ctcf = p1, gfp = p2) + coord_cartesian(ylim = c(0, 2e+06))
```



```
## work with ylim
tracks(ctcf = p1, gfp = p2) + ylim(0, 2e+06)

          ## Warning:  Removed 2 rows containing missing values (position_stack).
          ## Warning:  Removed 2 rows containing missing values (position_stack).
```



Let's view maxs instead of sums as shown in Figure 13.5.

```
p1 <- autoplot(cov.ctcf, type = "viewMaxs")

                             ## Default use binwidth:  range/30

p2 <- autoplot(cov.gfp, type = "viewMaxs")

                             ## Default use binwidth:  range/30

tracks(ctcf = p1, gfp = p2) + ylim(c(0, 2e+06))

         ## Warning:  Removed 2 rows containing missing values (position_stack).
         ## Warning:  Removed 2 rows containing missing values (position_stack).
```

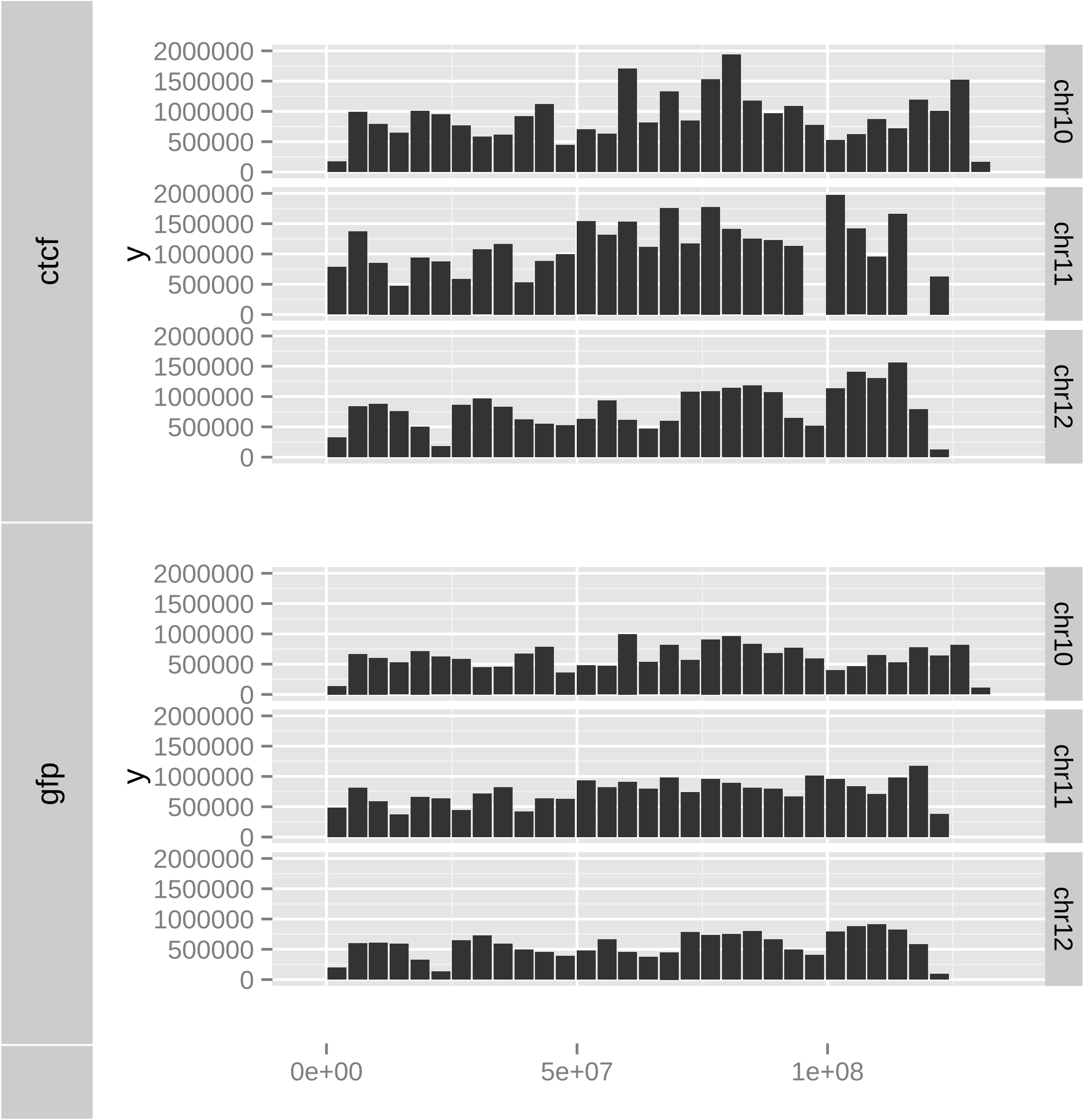


Figure 13.5: Use default statistical transformation ”bin” and geom ”bar” to represent summary ViewMaxs, and keep y limits on the same scale.

We can change bin numbers as shown in Figure 13.6

```
p1 <- autoplot(cov.ctcf, type = "viewMaxs", nbin = 100)

                                ## Default use binwidth:  range/100

p2 <- autoplot(cov.gfp, type = "viewMaxs", nbin = 100)

                                ## Default use binwidth:  range/100

tracks(ctcf = p1, gfp = p2) + ylim(0, 1e+06)
```

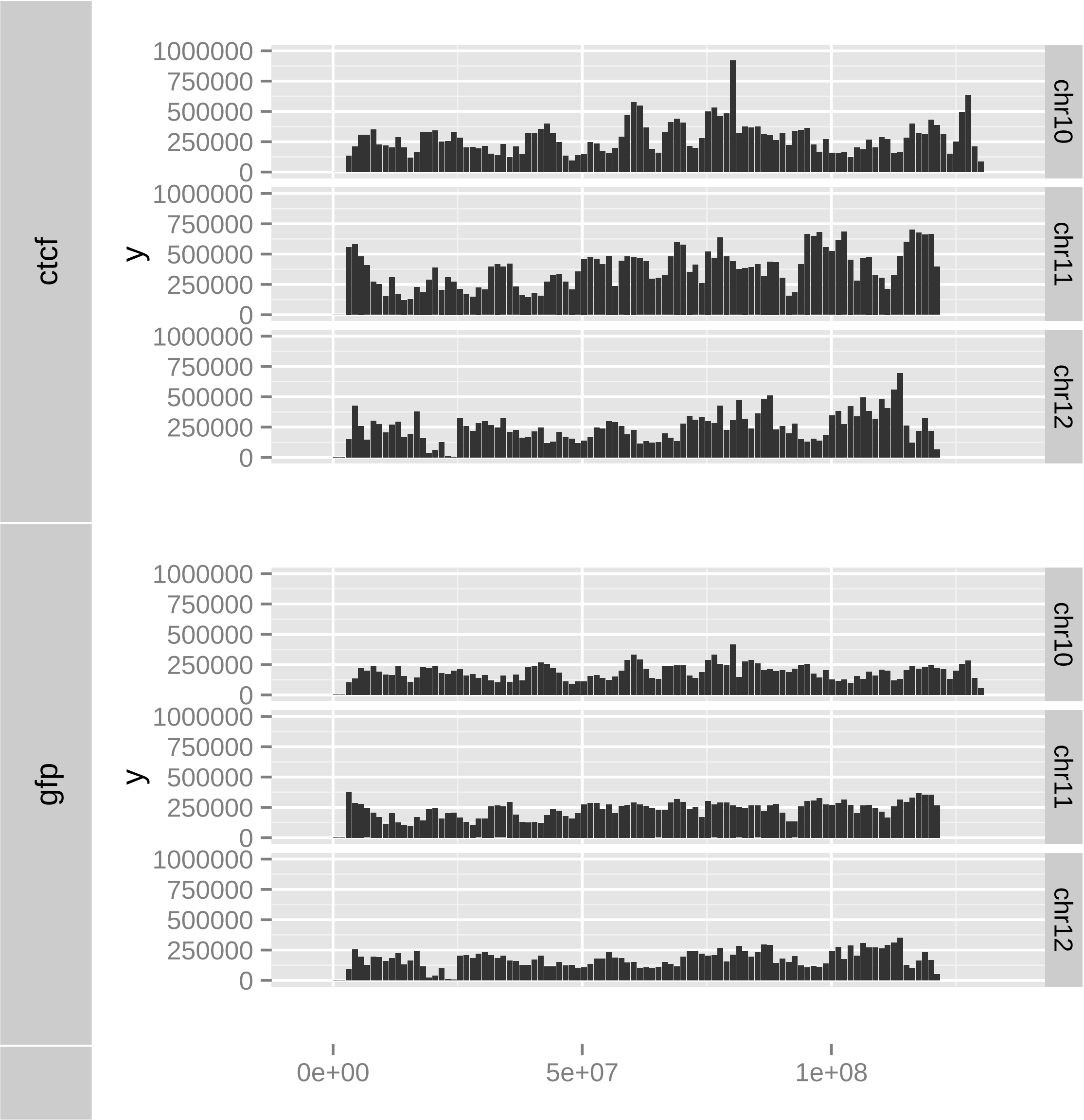


Figure 13.6: Use default statistical transformation "bin" and geom "bar" to represent summary ViewMaxs, with bin number changed to 100, and keep y limits on the same scale.

Try heatmap as shown in Figure 13.7, When you use tracks function to bind plots, please pay attention that, the color scale might be different which is critical for your judge. So in the following code, I add some add-on control to make sure it's on the same scale.

```
p1 <- autoplot(cov.ctcf, type = "viewMeans", nbin = 100, geom = "heatmap")

                              ## Default use binwidth:  range/100

p2 <- autoplot(cov.gfp, type = "viewMeans", nbin = 100, geom = "heatmap")

                              ## Default use binwidth:  range/100

tracks(ctcf = p1, gfp = p2) + scale_fill_continuous(limits = c(0, 8e+05)) +
    scale_color_continuous(limits = c(0, 8e+05))
```

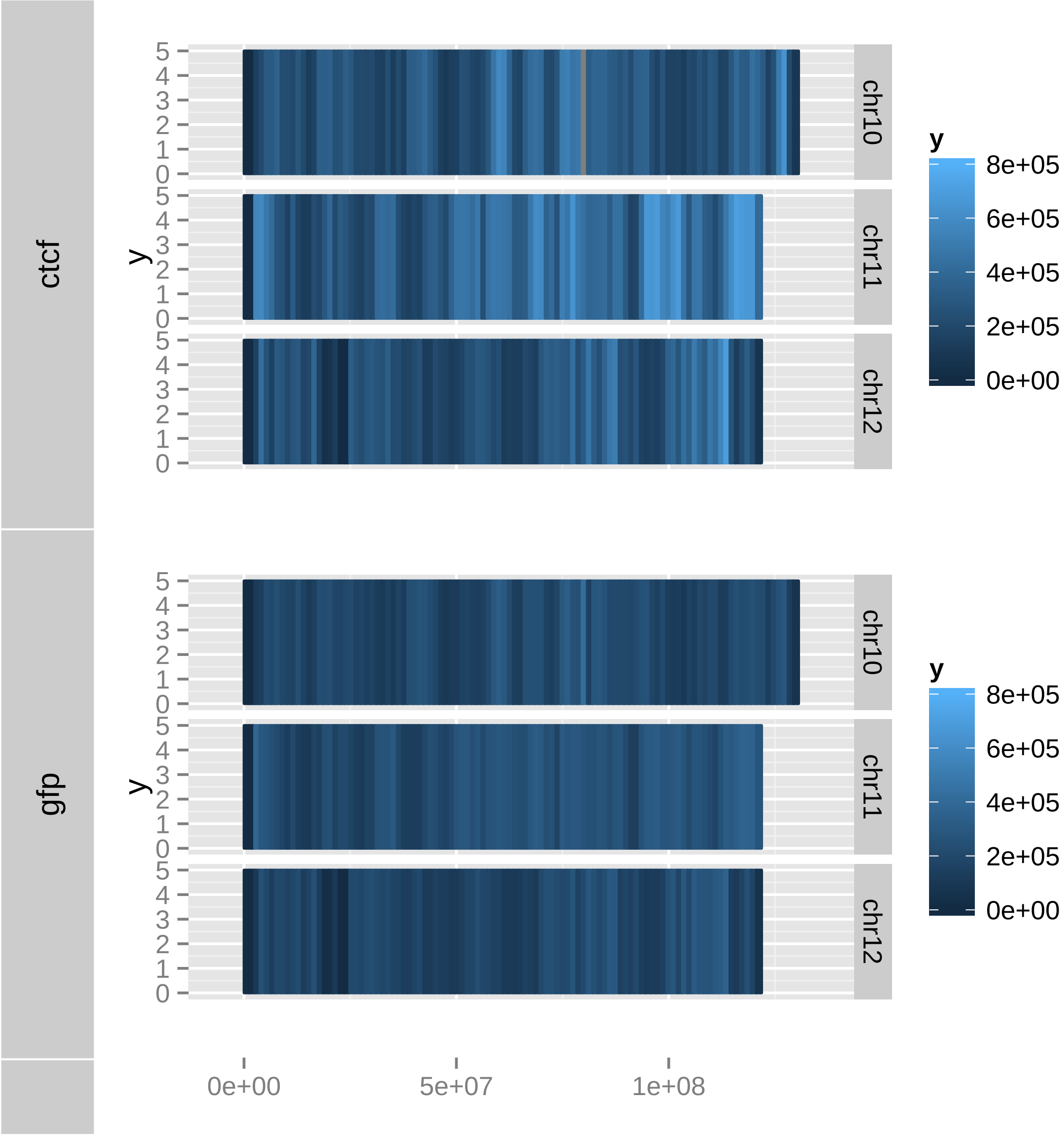


Figure 13.7: Use default statistical transformation ”bin” and geom ”heatmap” to represent summary ViewMaxs, with bin number changed to 100

Try statistical transformation "slice" as shown in Figure 13.8, we use an estimated cutoff 8 to define islands.

```
p1 <- autoplot(cov.ctcf, type = "viewMaxs", stat = "slice", lower = 8)
p2 <- autoplot(cov.gfp, type = "viewMaxs", stat = "slice", lower = 8)
tracks(ctcf = p1, gfp = p2) + ylim(0, 15000)

          ## Warning:  Removed 2 rows containing missing values (geom_segment).
          ## Warning:  Removed 2 rows containing missing values (geom_segment).
```

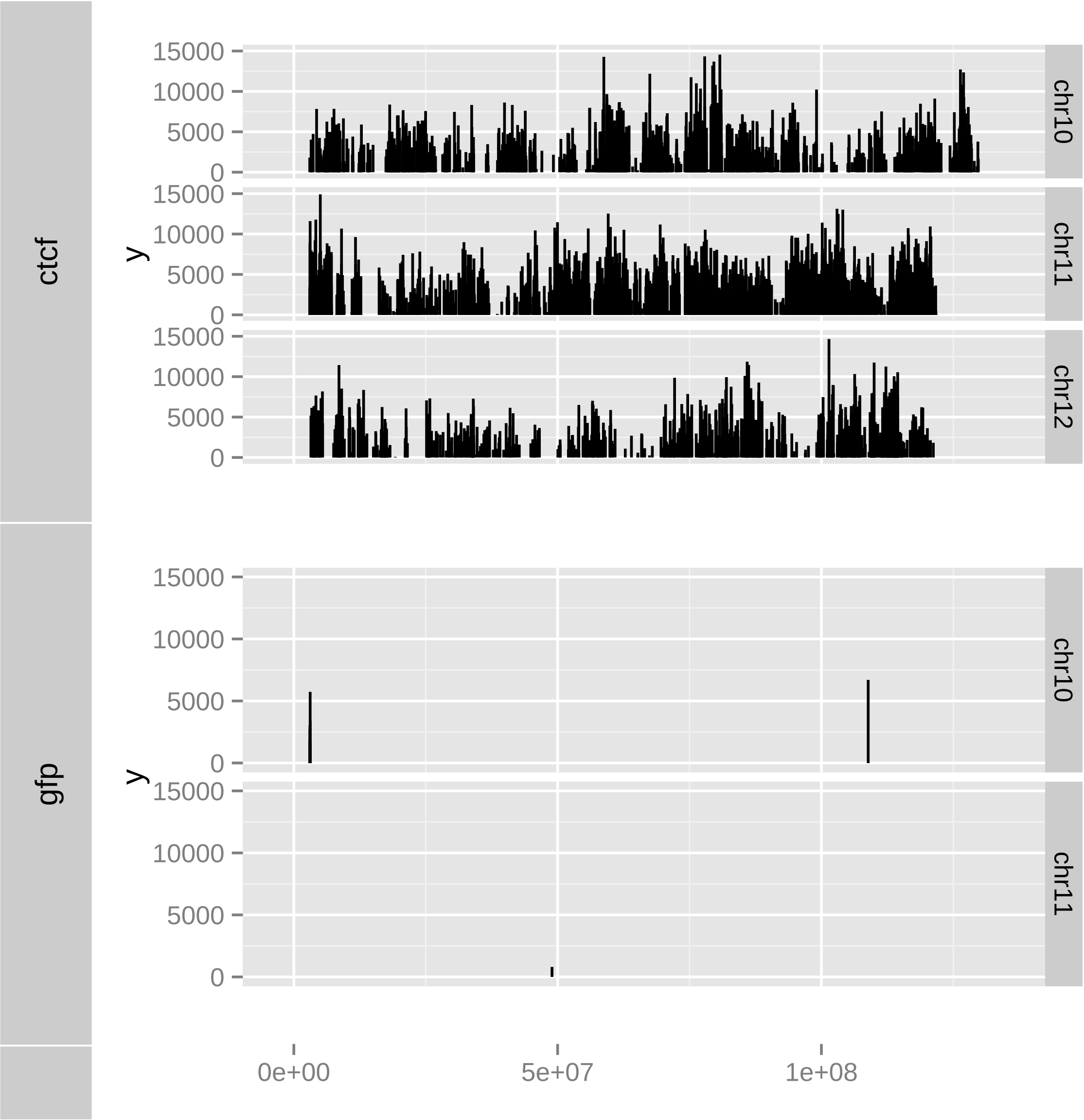


Figure 13.8: Use default statistical transformation "slice" and geom vertical "segment" to represent summary ViewMaxs, with lower cutoff 8

Notice in Figure 13.8, chromosome with no data are dropped automatically, if you want to keep the chromosomes based on chromosome levels you passed, you can use argument drop to control this as shown in Figure 13.9

```
p1 <- autoplot(cov.ctcf, type = "viewMaxs", stat = "slice", lower = 8)
p2 <- autoplot(cov.gfp, type = "viewMaxs", stat = "slice", lower = 8, drop = FALSE)
tracks(ctcf = p1, gfp = p2) + ylim(0, 15000)

          ## Warning:  Removed 2 rows containing missing values (geom_segment).
          ## Warning:  Removed 2 rows containing missing values (geom_segment).
```

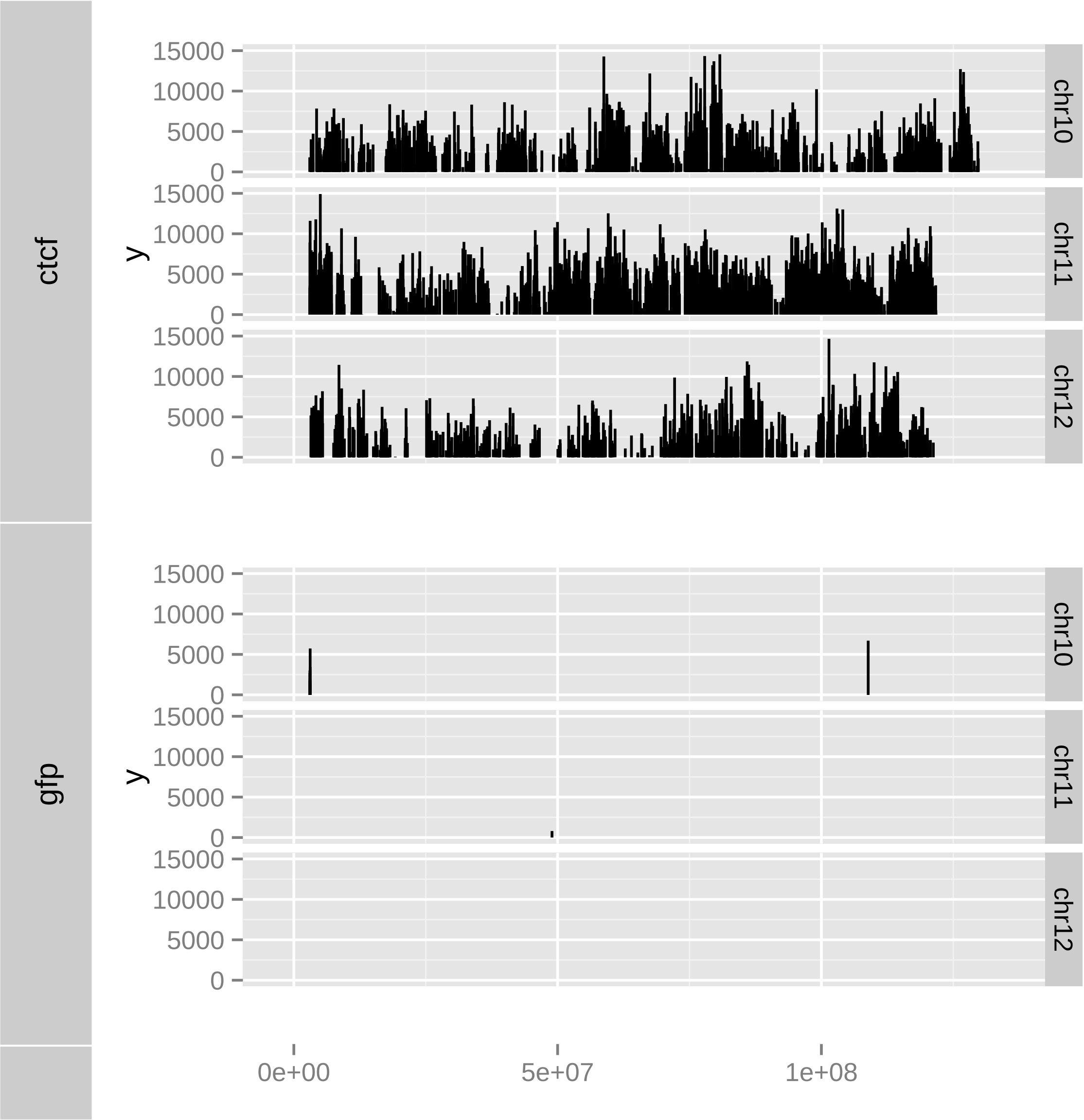


Figure 13.9: Use default statistical transformation ”slice” and geom vertical ”segment” to represent summary ViewMaxs, with lower cutoff 8

We can use geom “rect” to just see the region of island as shown in Figure 13.10

```
p1 <- autoplot(cov.ctcf, stat = "slice", lower = 5, geom = "rect")
p2 <- autoplot(cov.gfp, stat = "slice", lower = 5, geom = "rect")
tracks(ctcf = p1, gfp = p2)
```

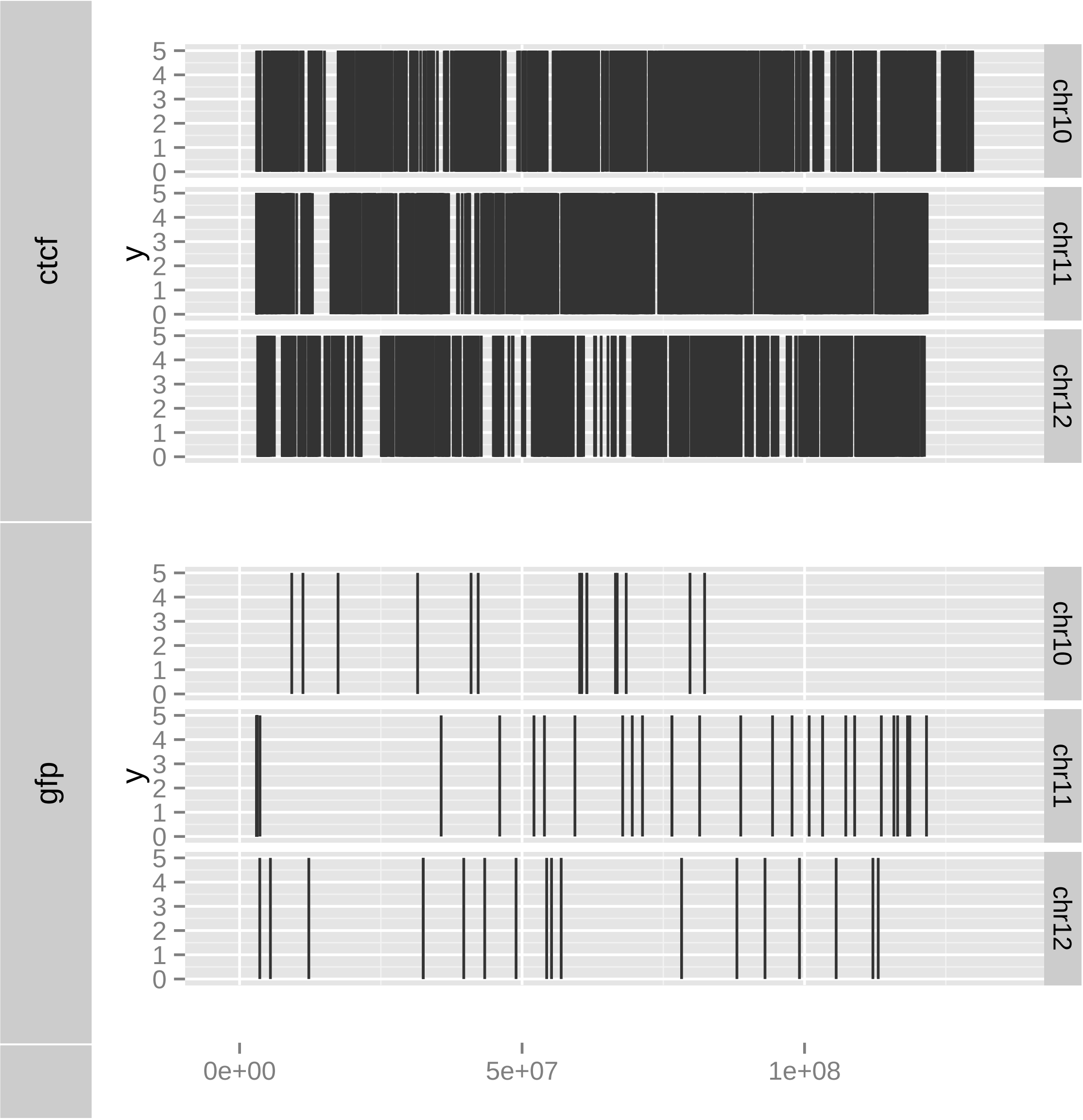


Figure 13.10: Use default statistical transformation ”slice” and geom vertical ”rect” to represent island region. Wider rectangle means wider island.

Finally, let's try geom "heatmap".

13.10

```
p1 <- autoplot(cov.ctcf, type = "viewMaxs", stat = "slice", lower = 8, geom = "heatmap")
p2 <- autoplot(cov.gfp, type = "viewMaxs", stat = "slice", lower = 8, geom = "heatmap",
    drop = FALSE)
tracks(ctcf = p1, gfp = p2) + scale_fill_continuous(limits = c(1000, 6000)) +
    scale_color_continuous(limits = c(1000, 6000))
```

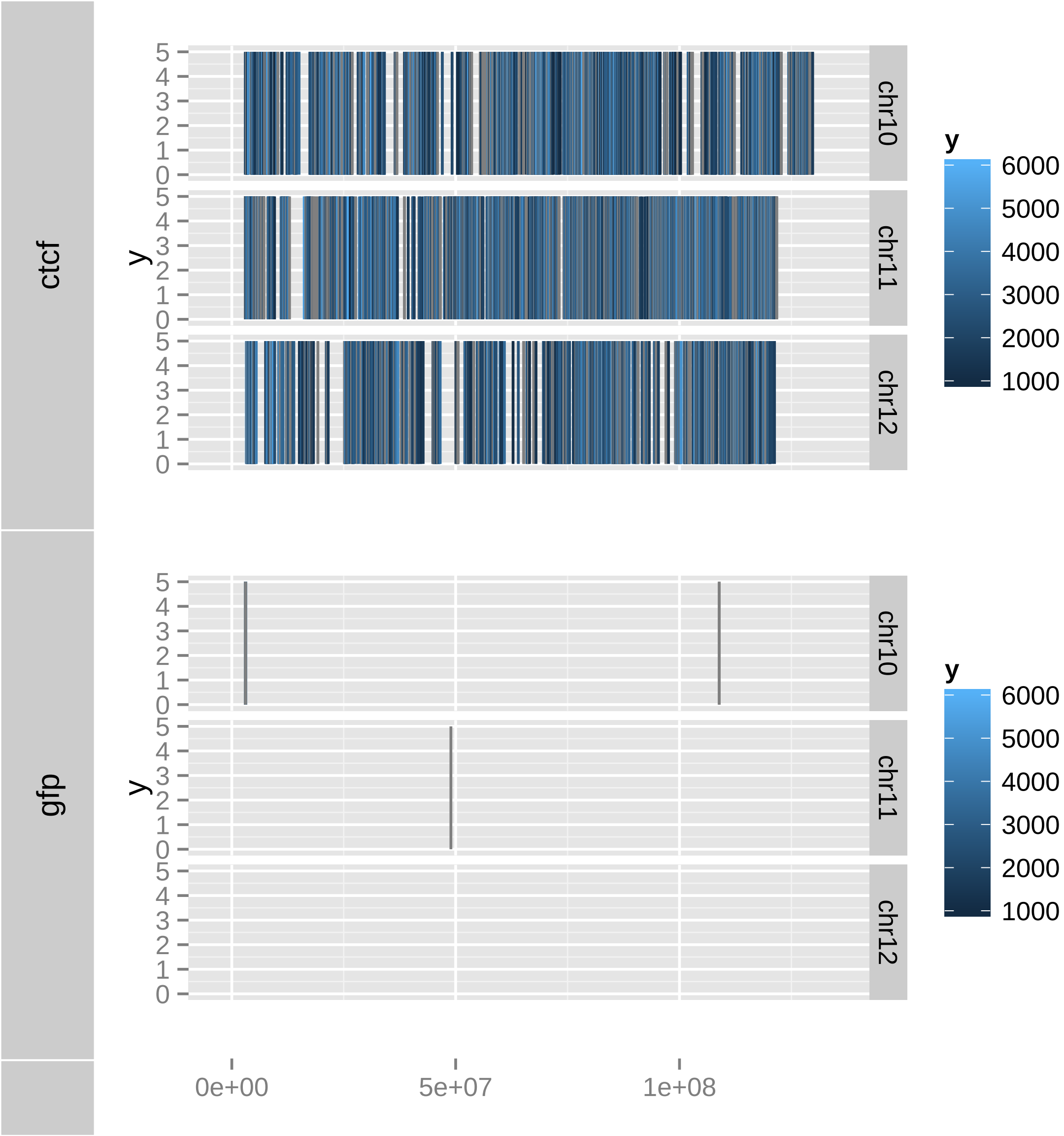


Figure 13.11: Use default statistical transformation "slice" and geom vertical "rect" to represent island region. Wider rectangle means wider island.

Constructing tracks with ideogram and genomic features

Most time, we only want to visualize a small region on the genome with annotation data to help us understand the biological significance or form hypothesis.

In this section, we try to find a region that ...,

```
peaks.ctcf <- slice(cov.ctcf, lower = 8)
peaks.gfp <- slice(cov.gfp, lower = 8)
## this function is from vignette of chipseq
peakSummary <- diffPeakSummary(peaks.gfp, peaks.ctcf)
peakSummary <- within(peakSummary, {
    diffs <- asinh(sums2) - asinh(sums1)
    resids <- (diffs - median(diffs))/mad(diffs)
    up <- resids > 2
    down <- resids < -2
    change <- ifelse(up, "up", ifelse(down, "down", "flat"))
})
ps.down <- peakSummary[peakSummary$change == "down" & peakSummary$space ==
    "chr11", ]
pk.down <- ps.down[order(ps.down$diffs), ]
pk.down
```

```
## RangedData with 22 rows and 10 value columns across 3 spaces
##        space             ranges   |  comb.max     sums1     sums2
##     <factor>          <IRanges>   | <integer> <integer> <integer>
## 1      chr11 [3082329, 3082373]   |         9       360        11
## 2      chr11 [3096046, 3096092]   |         9       376        47
## 3      chr11 [3078056, 3078070]   |         9       120        15
## 4      chr11 [3074248, 3074257]   |         9        80        10
## 5      chr11 [3060223, 3060234]   |        10        96        17
## 6      chr11 [3064802, 3064843]   |        10       336        71
## 7      chr11 [3078072, 3078123]   |        11       436        98
## 8      chr11 [3053546, 3053565]   |        10       160        40
## 9      chr11 [3054994, 3054997]   |        10        32         8
## ...      ...                ... ...       ...       ...       ...
## 14     chr11 [3043396, 3043497]   |        14       867       299
## 15     chr11 [3053477, 3053510]   |        12       282       102
## 16     chr11 [3077140, 3077157]   |        11       144        54
## 17     chr11 [3042270, 3042278]   |        11        72        27
## 18     chr11 [3077516, 3077671]   |        14      1422       540
## 19     chr11 [3082415, 3082505]   |        14       804       326
## 20     chr11 [3092597, 3092611]   |        12       120        51
## 21     chr11 [3079874, 3079905]   |        12       256       115
## 22     chr11 [3064758, 3064781]   |        12       192        87
##         maxs1     maxs2      change     diffs      down    resids
##     <integer> <integer> <character> <numeric> <logical> <numeric>
## 1           8         1        down    -3.486      TRUE    -2.779
## 2           8         1        down    -2.079      TRUE    -2.386
## 3           8         1        down    -2.078      TRUE    -2.386
## 4           8         1        down    -2.077      TRUE    -2.385
## 5           8          2        down    -1.730      TRUE     -2.288
## 6           8          2        down    -1.554      TRUE     -2.239
## 7           9          3        down    -1.493      TRUE     -2.222
## 8           8          2        down    -1.386      TRUE     -2.192
## 9           8          2        down    -1.383      TRUE     -2.191
## ...       ...        ...         ...       ...       ...        ...
## 14         10          4        down   -1.0646      TRUE     -2.102
## 15          9          3        down   -1.0169      TRUE     -2.089
## 16          8          3        down   -0.9808      TRUE     -2.079
## 17          8          3        down   -0.9805      TRUE     -2.079
## 18         11          5        down   -0.9682      TRUE     -2.076
## 19          9          5        down   -0.9027      TRUE     -2.057
## 20          8          4        down   -0.8556      TRUE     -2.044
## 21          8          4        down   -0.8002      TRUE     -2.029
## 22          8          4        down   -0.7916      TRUE     -2.026
##            up
##     <logical>
## 1       FALSE
## 2       FALSE
## 3       FALSE
## 4       FALSE
## 5       FALSE
## 6       FALSE
## 7       FALSE
## 8       FALSE
## 9       FALSE
## ...       ...
## 14      FALSE
## 15      FALSE
## 16      FALSE
## 17      FALSE
## 18      FALSE
## 19      FALSE
## 20      FALSE
## 21      FALSE
## 22      FALSE


##
library(TxDb.Mmusculus.UCSC.mm9.knownGene)
txdb <- TxDb.Mmusculus.UCSC.mm9.knownGene
tx <- transcripts(txdb)
gn <- transcriptsBy(txdb, by = "gene")
fu <- fiveUTRsByTranscript(txdb)

idx <- which(countOverlaps(as(pk.down, "GRanges"), flank(fu, width = 100)) ==
    1)
wh.p <- as(pk.down[idx[2], ], "GRanges")
wh.pw <- resize(wh.p, width = 30000, fix = "center")
```

Since mouse ideogram is not default data in *ggbio*, you need to get that information from UCSC, there is another vignette talking about how to create ideogram.

We create this ideogram with zoomed region.

```
library(biovizBase)
mm9 <- getIdeogram("mm9")

                                        ## Loading...
                                           ## Done

cyto.def <- getOption("biovizBase")$cytobandColor
cyto.new <- c(cyto.def, c(gpos33 = "grey80", gpos66 = "grey60"))
p.ideo <- plotIdeogram(mm9, "chr10", zoom = c(start(wh.pw), end(wh.pw))) +
    scale_fill_manual(values = cyto.new)
print(p.ideo)
```

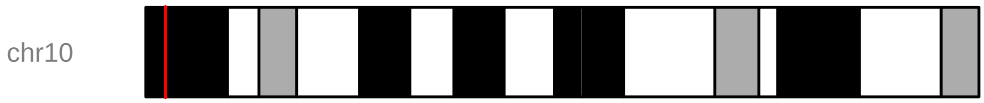


Figure 13.12: Ideogram for mouse chromosome 10

```
p.gene <- autoplot(txdb, which = wh.pw)

## Aggregating TranscriptDb...

## Parsing exons...

## Parsing cds...

## Parsing transcripts...

## Aggregating...

## Done

## Constructing graphics...
```

13.2 Mismatch summary

13.2.1 Introduction

stat_mismatch is lower level API to read in a bam file and show mismatch summary for certain region, counts at each position are summarized, those reads which are identical as reference will be either shown as gray background or removed, it's controled by argument ‘show.coverage‘, mismatched part will be shown as color-coded bar or segment.

Objects supported:

- Bamfile
- GRanges. this will pass interval checking which make sure the GRanges has required columns.

13.2.2 Usage

Low level API: stat_mismatch

Load packages

```
library(ggbio)
library(BSgenome.Hsapiens.UCSC.hg19)
data("genesymbol", package = "biovizBase")
```

Load example bam file

```
bamfile <- system.file("extdata", "SRR027894subRBM17.bam", package = "biovizBase")
library(Rsamtools)
bf <- BamFile(bamfile)
```

```
## coverage
cstest.s <- stack(cstest)
cstest.s <- resize(cstest.s, width = 200)
cstest.sub <- subsetByOverlaps(cstest.s, wh.pw)
p.cov <- autoplot(cstest.sub, stat = "coverage", facets = sample ~ ., geom = "area")
## ideogram
tracks(p.ideo, coverage = p.cov, gene = p.gene, xlim = as(wh.pw, "GRanges"),
    heights = c(1, 5, 5))
```

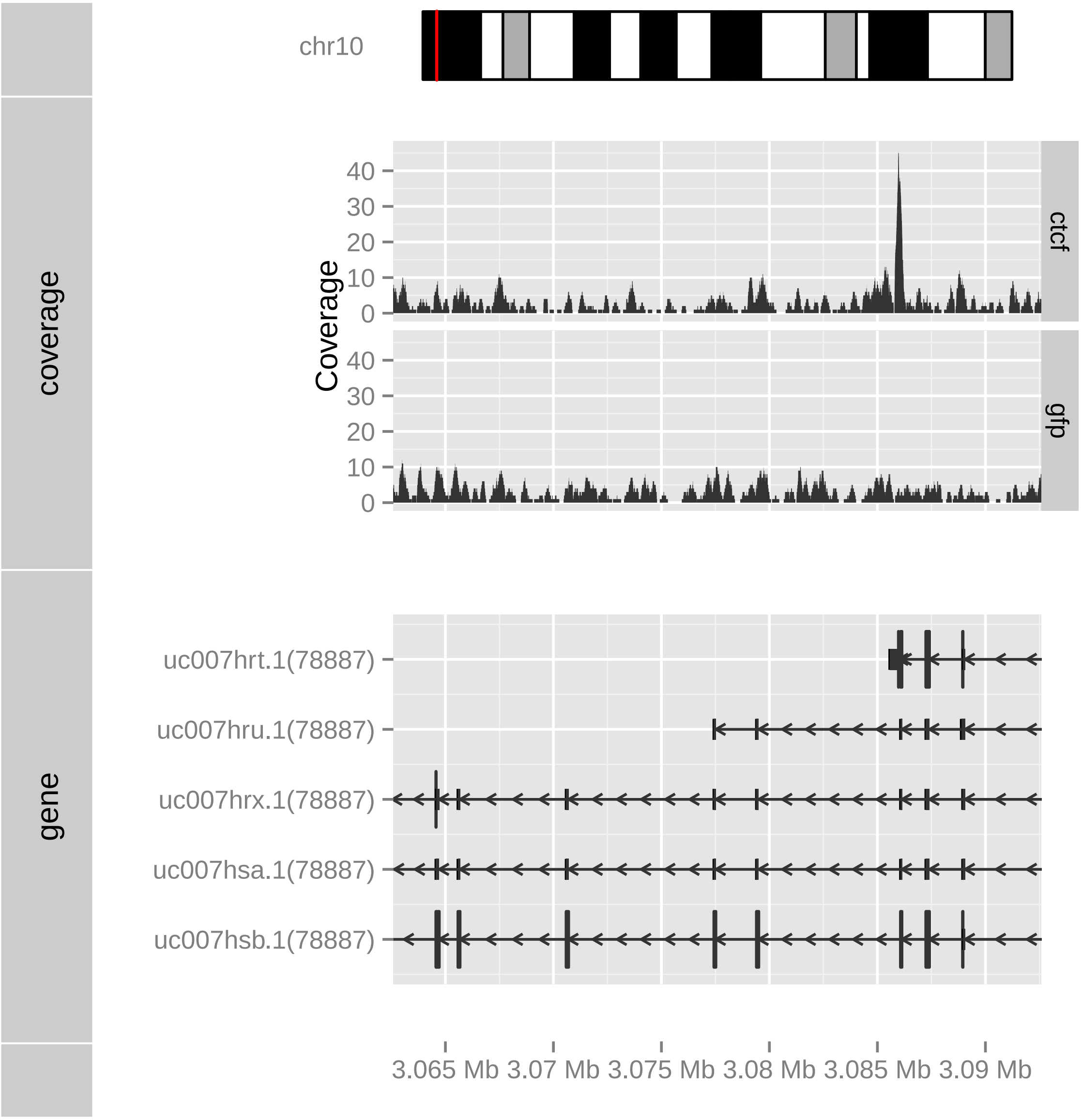


Figure 13.13: Tracks showing one strong peak in cfcf.

If the object is BamFile, a BSgenome object is required to compute the mismatch summary. in the following code, coord_cartesian function is a *ggplot2* function which zoom in/out, function theme_bw is a customized theme in *ggplot2* which will give you a grid and white background.

Sometimes bam file and BSgenome object might have a different naming schema for chromosomes, currently, stat_mismatch is not smart enough to deal with complicated cases, in this way, you may want to get mismatch summary as GRanges yourself and correct the names, with keepSeqlevels or renamesSeqleves functions in package *GenomicRanges*. Following examples doesn't show you how to manipulate seqnames, but just show you how to compute mismatch summary.

```
library(biovizBase)
pgr <- pileupAsGRanges(bamfile, region = genesymbol["RBM17"])
pgr.match <- pileupGRangesAsVariantTable(pgr, genome = Hsapiens)
```

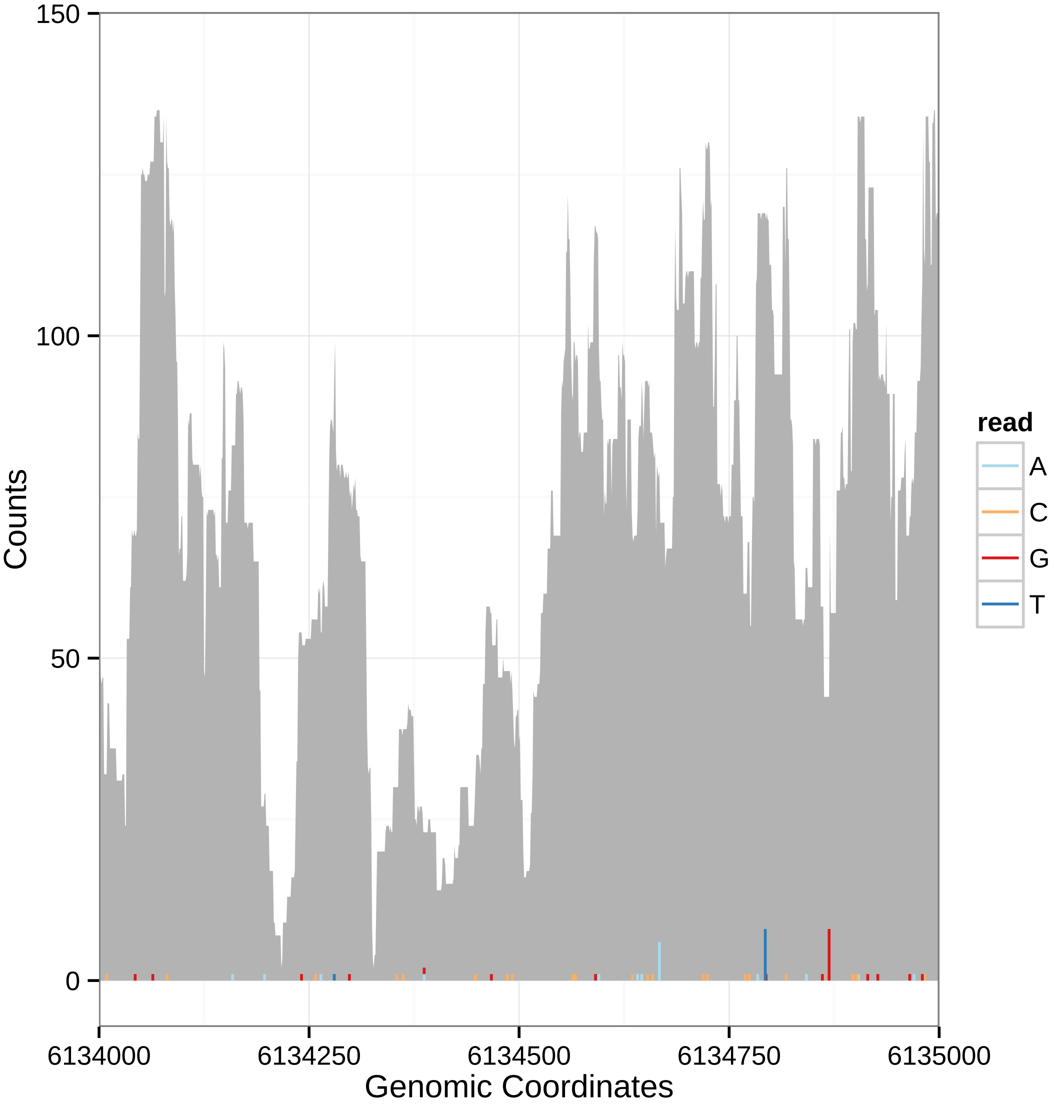
And directly plot the mismatch GRanges object, at the same time hide coverage background.

Then we compare geom 'bar' and 'segment', 'bar' is useful when zoomed in to a small region.

autoplot

autoplot for object Bamfile have a statistical transformation called *mismatch*, this is a wrapper over lower level function stat_mismatch.

```
ggplot(bf) + stat_mismatch(which = genesymbol["RBM17"], bsgenome = Hsapiens,
    show.coverage = TRUE) + coord_cartesian(xlim = c(6134000, 6135000)) +
    theme_bw()

## Object of class "ggbio"
```



```
## NULL
```

Figure 13.14: Mismatch summary for gene RBM17. Background is coverage shown as gray color, and only mismatched reads are shown with different color.

```
ggplot() + stat_mismatch(pgr.match)
```

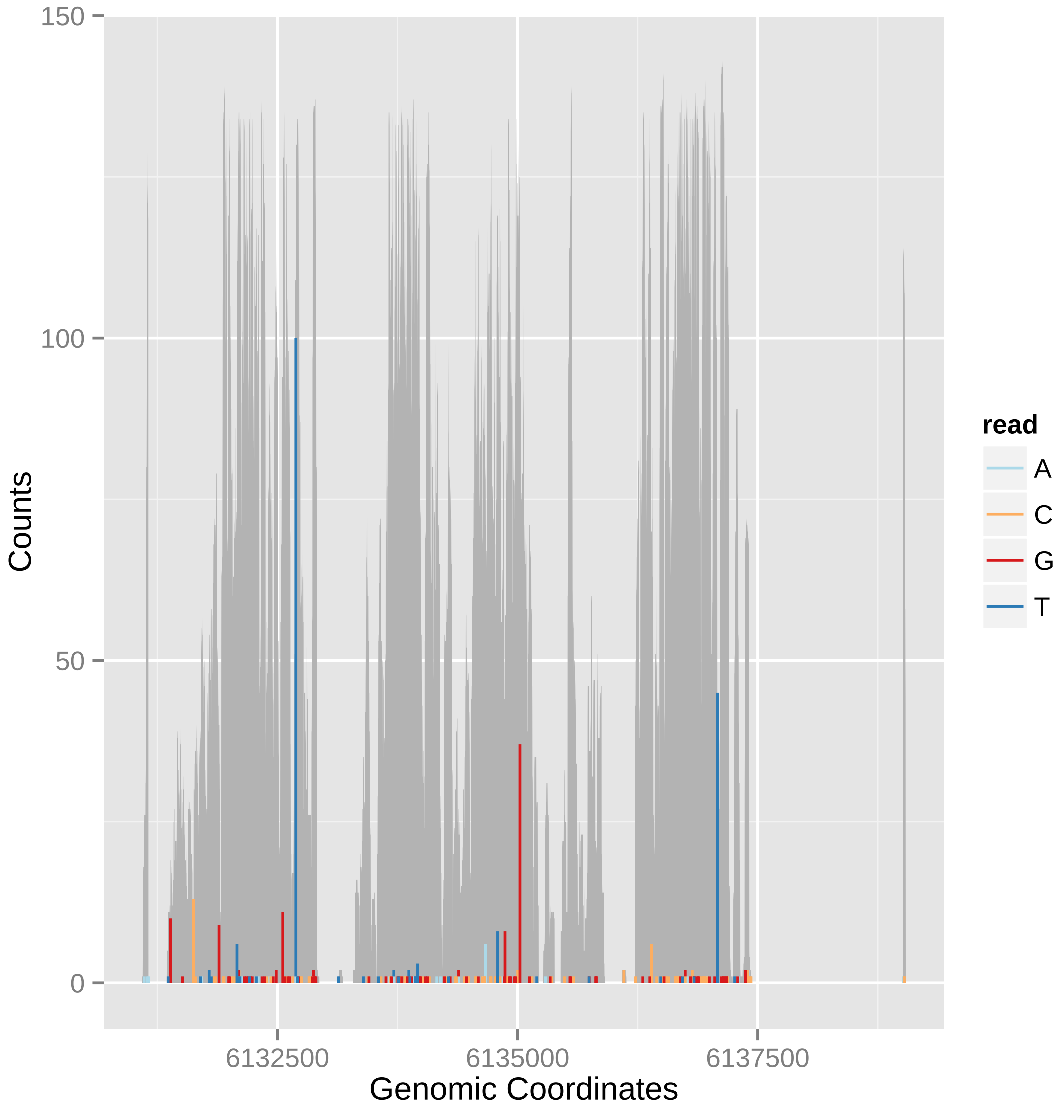


Figure 13.15: Mismatch summary without coverage

```
p1 <- ggplot() + stat_mismatch(pgr.match, show.coverage = FALSE, geom = "bar") +
    xlim(6132060, 6132120) + ylim(0, 10)
p2 <- ggplot() + stat_mismatch(pgr.match, geom = "segment") + xlim(6132060,
    6132120) + ylim(0, 10)
tracks(segment = p2, bar = p1) + scale_x_sequnit("Mb")

           ## Warning:  Removed 1 rows containing missing values (geom_segment).
           ## Warning:  Removed 2 rows containing missing values (geom_segment).
           ## Warning:  Removed 2 rows containing missing values (geom_segment).
           ## Warning:  Removed 1 rows containing missing values (geom_segment).
           ## Warning:  Removed 2 rows containing missing values (geom_segment).
           ## Warning:  Removed 2 rows containing missing values (geom_segment).
```

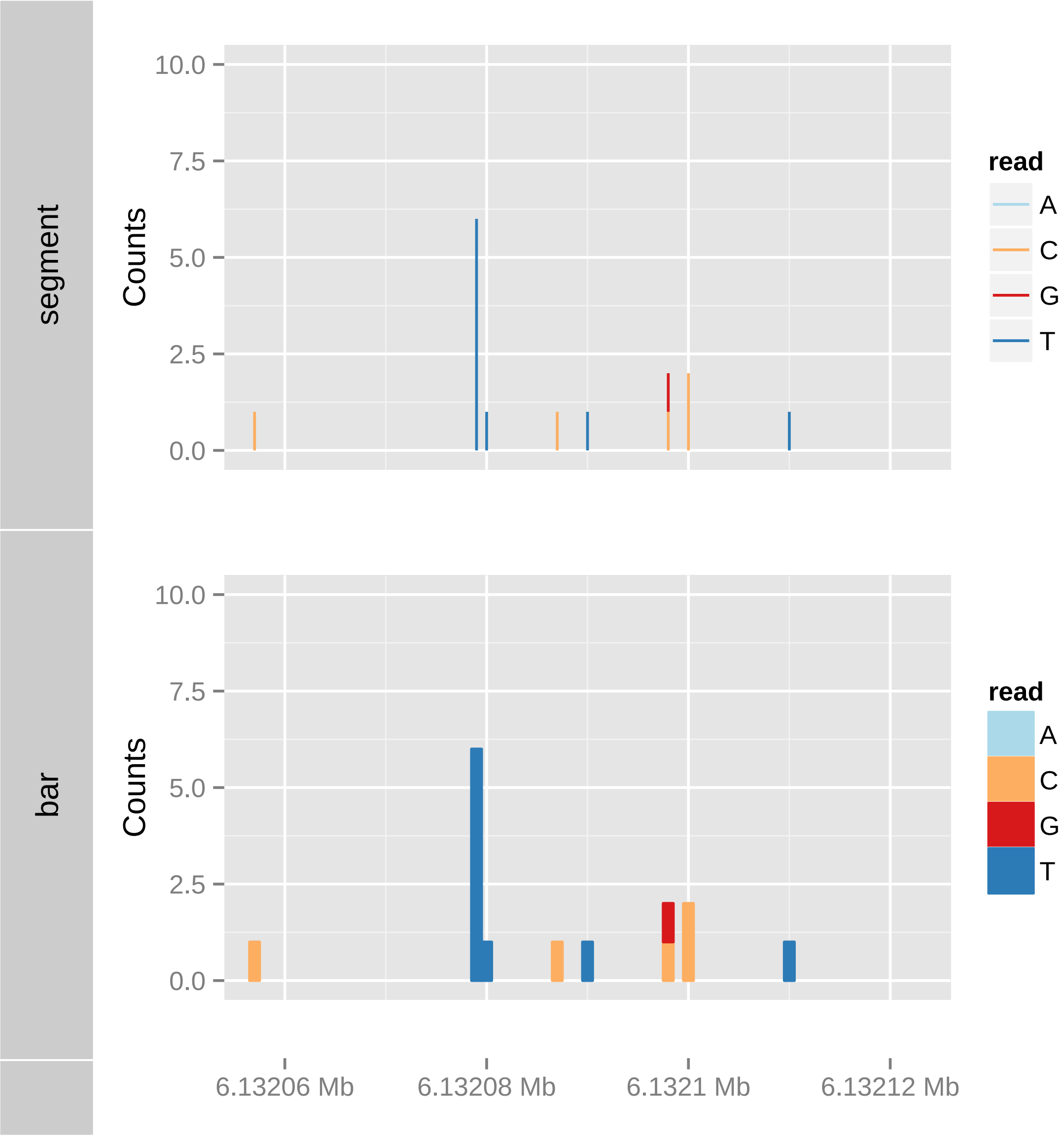


Figure 13.16: Mismatch summary without coverage

```
autoplot(bf, which = genesymbol["RBM17"], bsgenome = Hsapiens, show.coverage = TRUE,
    stat = "mismatch", geom = "bar") + xlim(6132060, 6132120) + ylim(0, 10)
```

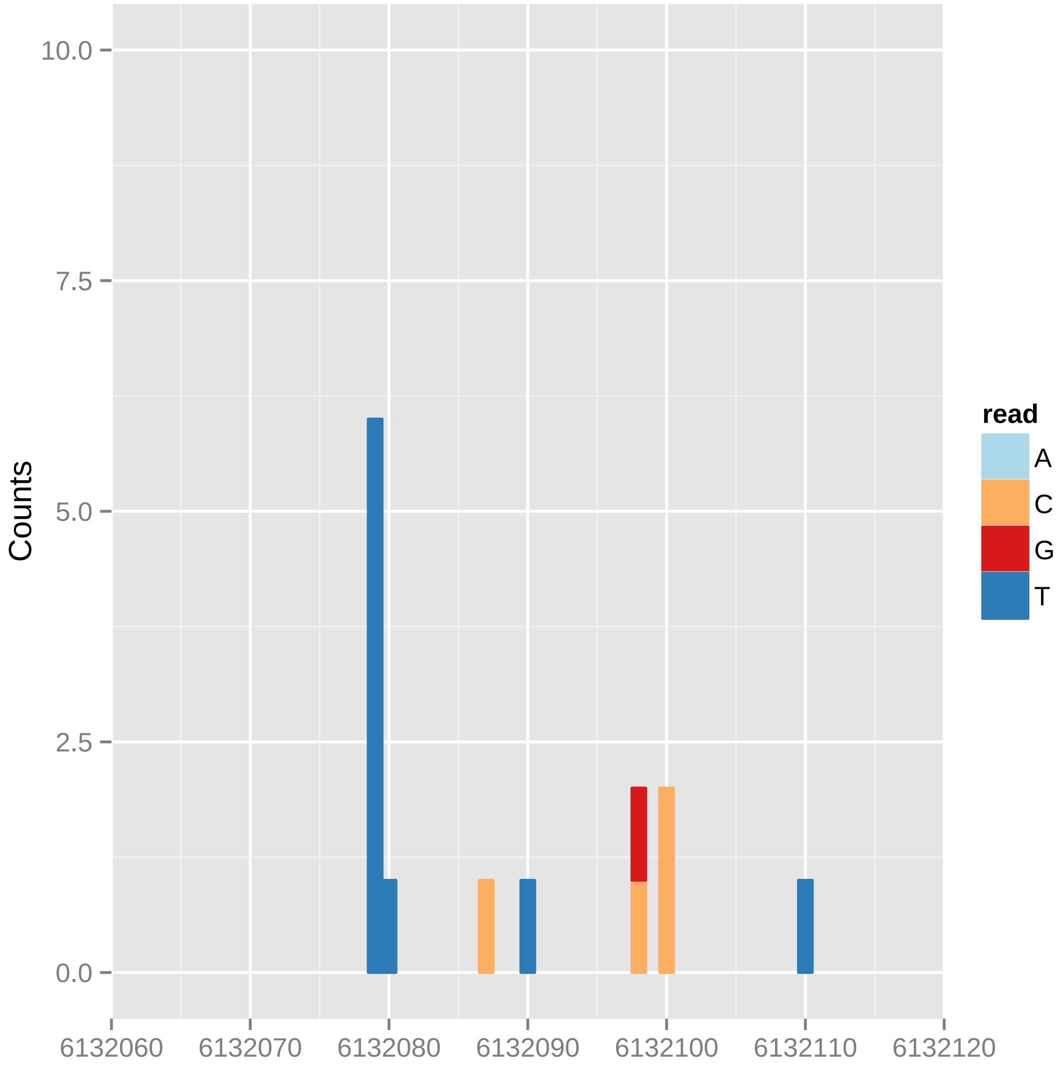


Figure 13.17: autoplot API to show the same plot

Chapter 14

Reference

Chapter 15

Appendix

15.1 Session Information

```
sessionInfo()


## R version 2.15.2 (2012-10-26)
## Platform: x86_64-unknown-linux-gnu (64-bit)
##
## locale:
##  [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
##  [3] LC_TIME=en_US.UTF-8        LC_COLLATE=C
##  [5] LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
##  [7] LC_PAPER=C                 LC_NAME=C
##  [9] LC_ADDRESS=C               LC_TELEPHONE=C
## [11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C
##
## attached base packages:
## [1] grid      stats     graphics  grDevices utils     datasets
## [7] methods   base
##
## other attached packages:
##  [1] TxDb.Mmusculus.UCSC.mm9.knownGene_2.8.0
##  [2] chipseq_1.8.0
##  [3] ShortRead_1.16.3
##  [4] latticeExtra_0.6-24
##  [5] RColorBrewer_1.0-5
##  [6] lattice_0.20-13
##  [7] biovizBase_1.6.2
##  [8] BSgenome.Hsapiens.UCSC.hg19_1.3.19
##  [9] BSgenome_1.26.1
## [10] VariantAnnotation_1.4.6
## [11] genefilter_1.40.0
## [12] vsn_3.26.0
## [13] Rsamtools_1.10.2
## [14] Biostrings_2.26.2
## [15] TxDb.Hsapiens.UCSC.hg19.knownGene_2.8.0
## [16] GenomicFeatures_1.10.1
## [17] AnnotationDbi_1.20.3
## [18] Biobase_2.18.0
## [19] rtracklayer_1.18.2
## [20] GenomicRanges_1.10.6
## [21] IRanges_1.16.4
## [22] BiocGenerics_0.4.0
## [23] ggbio_1.6.6
## [24] ggplot2_0.9.3
## [25] codetools_0.2-8
## [26] knitr_1.0
##
## loaded via a namespace (and not attached):
##  [1] BiocInstaller_1.8.3   DBI_0.2-5             Hmisc_3.10-1
##  [4] MASS_7.3-23           RCurl_1.95-3          RSQLite_0.11.2
##  [7] XML_3.95-0.1          affy_1.36.0           affyio_1.26.0
## [10] annotate_1.36.0       biomaRt_2.14.0        bitops_1.0-5
## [13] cluster_1.14.3        colorspace_1.2-0      dichromat_1.2-4
## [16] digest_0.6.0          evaluate_0.4.3        formatR_0.7
## [19] gridExtra_0.9.1       gtable_0.1.2          hwriter_1.3
## [22] labeling_0.1          limma_3.14.4          munsell_0.4
## [25] parallel_2.15.2       plyr_1.8              preprocessCore_1.20.0
## [28] proto_0.3-10          reshape2_1.2.2        scales_0.2.3
## [31] splines_2.15.2        stats4_2.15.2         stringr_0.6.2
## [34] survival_2.37-2       tools_2.15.2          xtable_1.7-0
## [37] zlibbioc_1.4.0
```