

ggbio: visualization toolkits for genomic data

Tengfei Yin¹

August 26, 2014

¹tengfei.yin@sbgenomics.com

Contents

1	Getting started	3
1.1	Citation	3
1.2	Introduction	3
2	Case study: building your first tracks	4
2.1	Add an ideogram track	4
2.2	Add a gene model track	5
2.2.1	Introduction	5
2.2.2	Make gene model from <i>OrganismDb</i> object	5
2.2.3	Make gene model from <i>TranscriptDb</i> object	9
2.2.4	Make gene model from <i>GRangesList</i> object	10
2.3	Add a reference track	13
2.3.1	Semantic zoom	13
2.4	Add an alignment track	15
2.5	Add a variants track	19
2.6	Building your tracks	22
3	Simple navigation	24
4	Overview plots	26
4.1	how to make circular plots	26
4.1.1	Introduction	26
4.1.2	Building circular plot layer by layer	26
4.1.3	Complex arrangement of plots	34
4.2	How to make grandlinear plots	35
4.2.1	Introduction	36
4.2.2	Coordinate genome	37
4.2.3	Convenient <code>plotGrandLinear</code> function	38
4.2.4	How to highlight some points?	40
4.3	How to make stacked karyogram overview plots	41
4.3.1	Introduction	41
4.3.2	Create karyogram template	41
4.3.3	Add data on karyogram layout	43
4.3.4	Add more data using <code>layout_karyogram</code> function	46
4.3.5	More flexible layout of karyogram	48
5	Link ranges to your data	50
6	Miscellaneous	52
6.1	Themes	52
6.1.1	Plot theme	52
6.1.2	Track theme	58

Chapter 1

Getting started

1.1 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},
##   }
```

1.2 Introduction

ggbio is a *Bioconductor* package building on top of *ggplot2*(), leveraging the rich objects defined by *Bioconductor* and its statistical and computational power, it provides a flexible genomic visualization framework, extends the grammar of graphics into genomic data, try to delivers high quality, highly customizable graphics to the users.

What it features

- `autoplot` function provides ready-to-use template for *Bioconductor* objects and different types of data.
- flexible low level components to use grammar of graphics to build you graphics layer by layer.
- layout transformation, so you could generate circular plot, grandlinear plot, stacked overview more easily.
- flexible tracks function to bind any *ggplot2*(), *ggbio* based plots.

Chapter 2

Case study: building your first tracks

In this chapter, you will learn

- how to add ideogram track.
- How to add gene model track.
- how to add track for bam files to visualize coverage and mismatch summary.
- how to add track for vcf file to visualize the variants.

2.1 Add an ideogram track

Ideogram provides functionality to construct ideogram, check the manual for more flexible methods. We build genome *hg19*, *hg18*, *mm10*, *mm9* inside, so you don't have download it on the fly. When embed with tracks, ideogram show zoomed region highlights automatically. `xlim` has special function here, is too changed highlighted zoomed region on the ideogram.

```
library(ggbio)
p.ideo <- Ideogram(genome = "hg19")
p.ideo
```



```
library(GenomicRanges)
## special highlights instead of zoomin!
p.ideo + xlim(GRanges("chr2", IRanges(1e8, 1e8+1000000)))
```



2.2 Add a gene model track

2.2.1 Introduction

Gene model track is one of the most frequently used track in genome browser, it is composed of genetic features CDS, UTR, introns, exons and non-genetic region. In *ggbio* we support three methods to make gene model track:

- *OrganismDb* object: recommended, support gene symbols and other combination of columns as label.
- *TranscriptDb* object: don't support gene symbol labeling.
- *GRangesList* object: flexible, if you don't have annotation package available for the first two methods, you could prepare a data set parsed from gtf file, you can simply use it and plot it as gene model track.

2.2.2 Make gene model from OrganismDb object

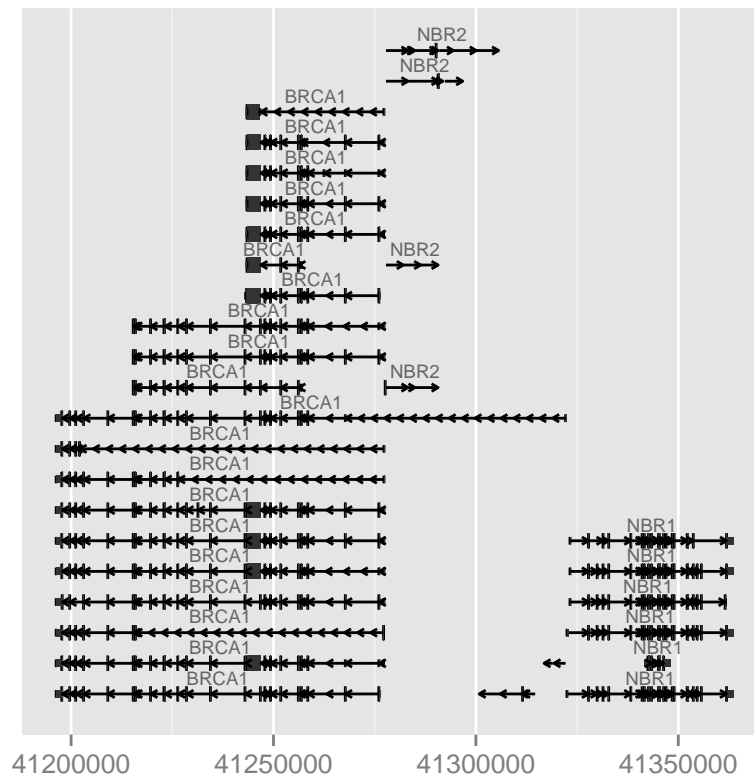
OrganismDb object has a simpler API to retrieve data from different annotation resources, so we could label our transcripts in different ways

```
library(ggbio)
library(Homo.sapiens)
class(Homo.sapiens)

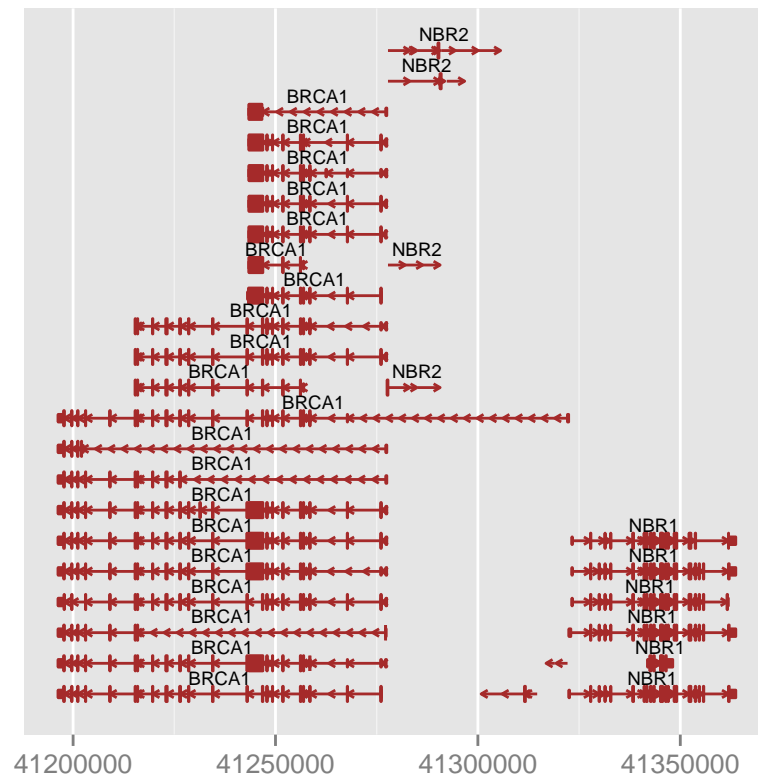
## [1] "OrganismDb"
## attr(,"package")
## [1] "OrganismDbi"

##
data(genesymbol, package = "biovizBase")
wh <- genesymbol[c("BRCA1", "NBR1")]
wh <- range(wh, ignore.strand = TRUE)

p.txdb <- autoplot(Homo.sapiens, which = wh)
p.txdb
```

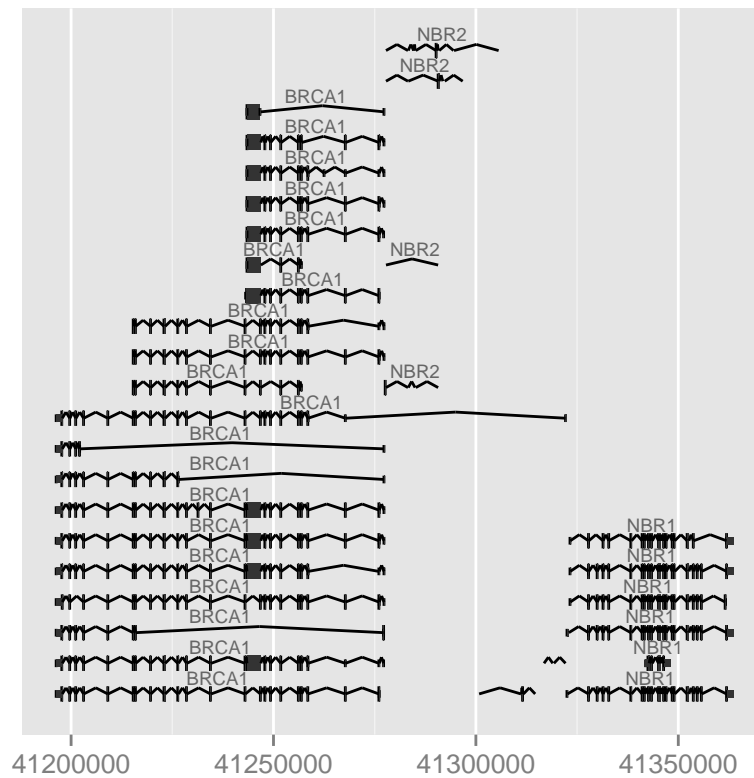


```
autoplot(Homo.sapiens, which = wh, label.color = "black", color = "brown",
         fill = "brown")
```



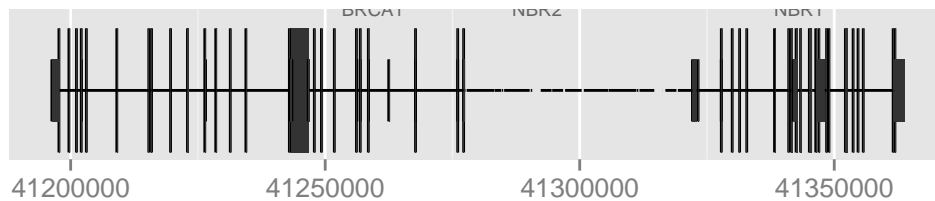
To change the intron geometry, use `gap.geom` to control it, check out `geom_alignment` for more control parameters.

```
autoplot(Homo.sapiens, which = wh, gap.geom = "chevron")
```

To collapse all features, use stat 'reduce'

```
autoplot(Homo.sapiens, which = wh, stat = "reduce")
```



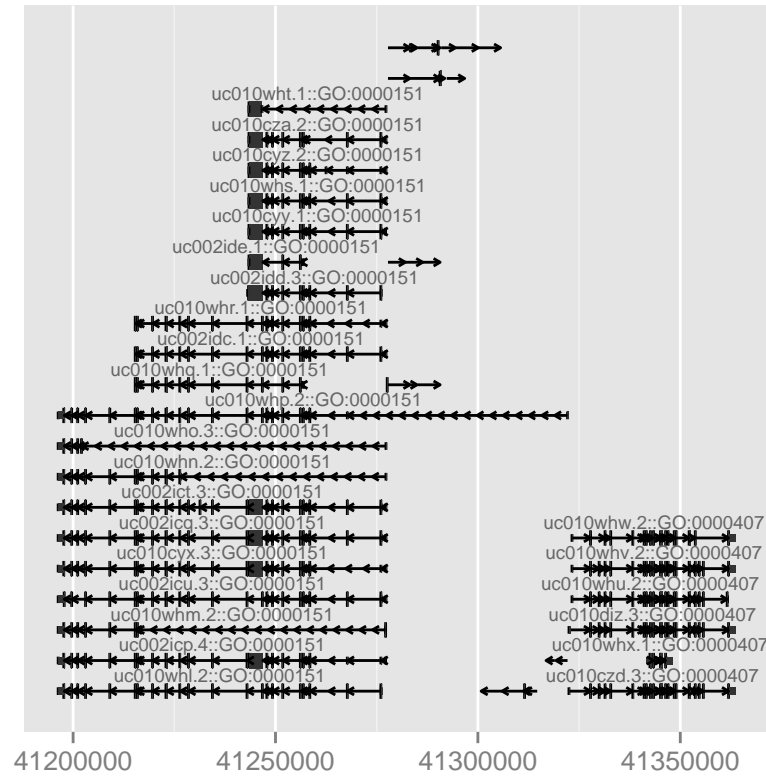
Label could be turned off by setting it to FALSE, you could also use expression to make a flexible label combination from column names.

```
columns(Homo.sapiens)
```

## [1]	"GOID"	"TERM"	"ONTOLOGY"	"DEFINITION"	"ENTREZID"
## [6]	"PFAM"	"IPI"	"PROSITE"	"ACCNUM"	"ALIAS"
## [11]	"CHR"	"CHRLOC"	"CHRLOCEND"	"ENZYME"	"MAP"
## [16]	"PATH"	"PMID"	"REFSEQ"	"SYMBOL"	"UNIGENE"
## [21]	"ENSEMBL"	"ENSEMBLPROT"	"ENSEMBLTRANS"	"GENENAME"	"UNIPROT"
## [26]	"GO"	"EVIDENCE"	"GOALL"	"EVIDENCEALL"	"ONTOLOGYALL"
## [31]	"OMIM"	"UCSKG"	"CDSID"	"CDSNAME"	"CDSCHROM"
## [36]	"CDSSTRAND"	"CDSSTART"	"CSEND"	"EXONID"	"EXONNAME"
## [41]	"EXONCHROM"	"EXONSTRAND"	"EXONSTART"	"EXONEND"	"GENEID"

```
## [46] "TXID"          "EXONRANK"      "TXNAME"        "TXCHROM"       "TXSTRAND"
## [51] "TXSTART"       "TXEND"

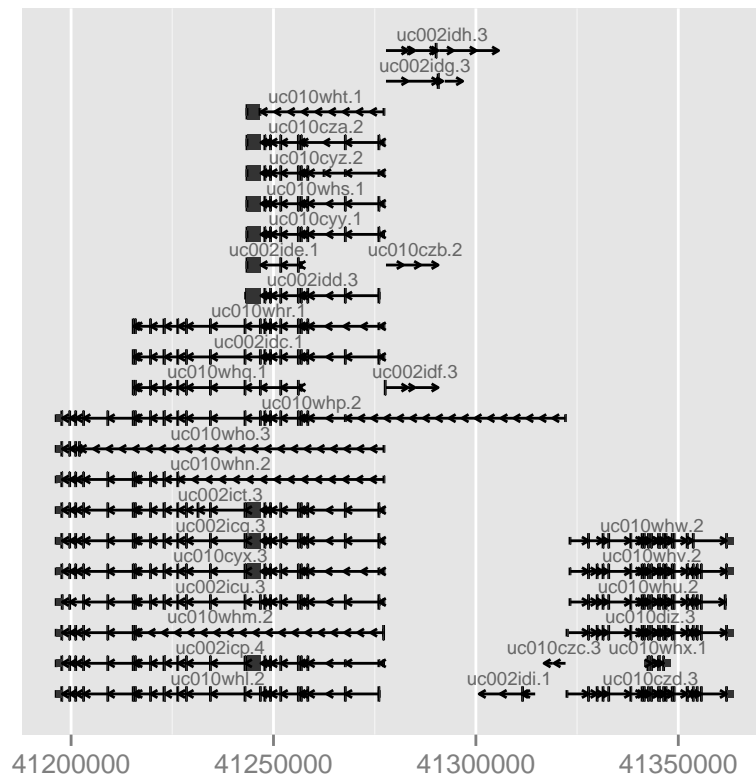
autoplot(Homo.sapiens, which = wh, columns = c("TXNAME", "GO"), names.expr = "TXNAME::GO")
```



2.2.3 Make gene model from TranscriptDb object

TranscriptDb doesn't contain any gene symbol information, so we use tx.id as default for label.

```
library(TxDb.Hsapiens.UCSC.hg19.knownGene)
txdb <- TxDb.Hsapiens.UCSC.hg19.knownGene
autoplot(txdb, which = wh)
```



2.2.4 Make gene model from GRangesList object

Sometimes your gene model is not available as none of *OrganismDb* or *TranscriptDb* object, it's may be stored in a table, you could simple parse it into a *GRangesList* object.

- each group indicate one transcripts
- names of group are shown as labels
- this object must has a column contains following key word: cds, exon, intron, and it's not case sensitive. use type to map this column. By default, we will try to parse 'type' column.

Let's make a sample *GRangesList* object which contains all information, and fake some labels.

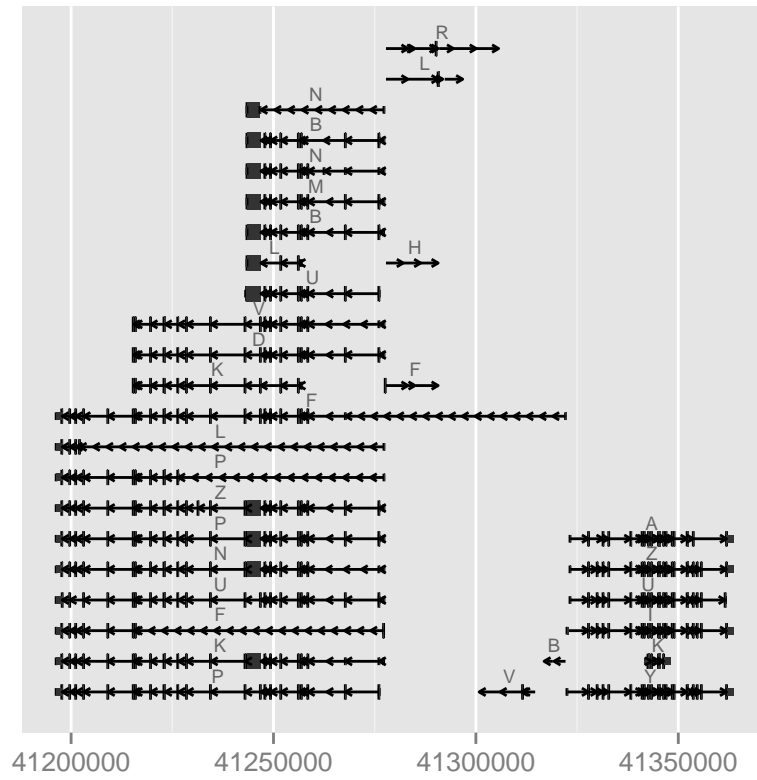
```
library(biovizBase)
gr.txdB <- crunch(txdB, which = wh)
## change column to 'model'
colnames(values(gr.txdB))[4] <- "model"
grl <- split(gr.txdB, gr.txdB$tx_id)
## fake some random names
names(grl) <- sample(LETTERS, size = length(grl), replace = TRUE)
grl

## GRangesList of length 32:
## $F
## GRanges with 7 ranges and 4 metadata columns:
##      seqnames      ranges strand | tx_id  tx_name  gene_id
##      <Rle>        <IRanges> <Rle> | <factor> <factor> <factor>
## [1] chr17 [41277600, 41277787] + | 61241 uc002idf.3 10230
```

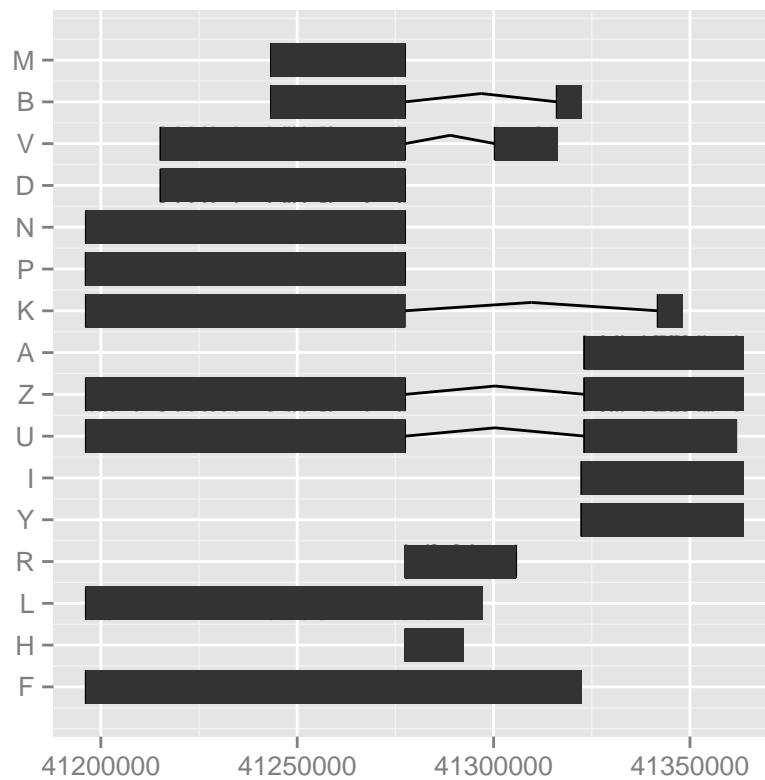
```
## [2] chr17 [41283225, 41283287] + | 61241 uc002idf.3 10230
## [3] chr17 [41284973, 41285154] + | 61241 uc002idf.3 10230
## [4] chr17 [41290674, 41292342] + | 61241 uc002idf.3 10230
## [5] chr17 [41277788, 41283224] * | 61241 uc002idf.3 10230
## [6] chr17 [41283288, 41284972] * | 61241 uc002idf.3 10230
## [7] chr17 [41285155, 41290673] * | 61241 uc002idf.3 10230
##      model
##      <factor>
## [1] exon
## [2] exon
## [3] exon
## [4] exon
## [5] gap
## [6] gap
## [7] gap
##
## $H
## GRanges with 3 ranges and 4 metadata columns:
##      seqnames      ranges strand | tx_id  tx_name gene_id model
## [1] chr17 [41277600, 41277787] + | 61242 uc010czb.2 10230 exon
## [2] chr17 [41290674, 41292342] + | 61242 uc010czb.2 10230 exon
## [3] chr17 [41277788, 41290673] * | 61242 uc010czb.2 10230 gap
##
## $L
## GRanges with 9 ranges and 4 metadata columns:
##      seqnames      ranges strand | tx_id  tx_name gene_id model
## [1] chr17 [41277600, 41277787] + | 61243 uc002idg.3 10230 exon
## [2] chr17 [41283225, 41283287] + | 61243 uc002idg.3 10230 exon
## [3] chr17 [41290674, 41290939] + | 61243 uc002idg.3 10230 exon
## [4] chr17 [41291833, 41292300] + | 61243 uc002idg.3 10230 exon
## [5] chr17 [41296745, 41297125] + | 61243 uc002idg.3 10230 exon
## [6] chr17 [41277788, 41283224] * | 61243 uc002idg.3 10230 gap
## [7] chr17 [41283288, 41290673] * | 61243 uc002idg.3 10230 gap
## [8] chr17 [41290940, 41291832] * | 61243 uc002idg.3 10230 gap
## [9] chr17 [41292301, 41296744] * | 61243 uc002idg.3 10230 gap
##
## ...
## <29 more elements>
## ---
## seqlengths:
##      chr17
## 81195210
```

We get our example data ready, it meets all requirements, to make it a gene model track it's pretty simple to use autoplot, but don't forget mapping because we changed our column names, assume you store you model key words in column 'model'.

```
autoplot(gr1, aes(type = model))
```



```
ggplot() + geom_alignment(gr1, type = "model")
```



2.3 Add a reference track

To add a reference track, we need to load a *BSgenome* object from the annotation package. You can choose to plot the sequence as *text*, *rect*, *segment*.

2.3.1 Semantic zoom

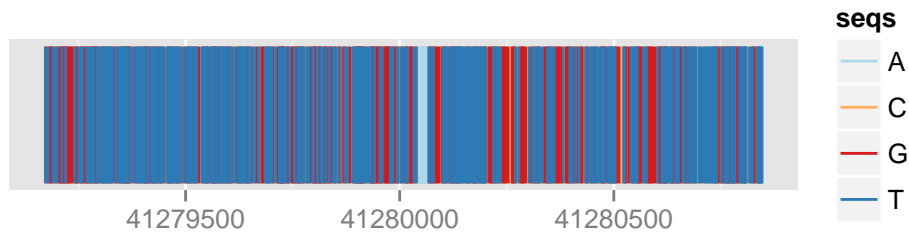
Here we introduce semantic zoom in *ggbio*, for some plots like reference sequence, we use pre-defined zoom level threshold to automatically assign geom to the track, unless the geom is explicitly specified. In the example below, when your region is too wide we show text 'zoom in to see text', when you zoom into different level, it shows you different details. *zoom* is a function we will introduce more in chapter 3 when we introduce more about navigation.

You can pass a zoom in factor into *zoom* function, if it's over 1 it's zooming out, if it's smaller than 1 it's zooming in.

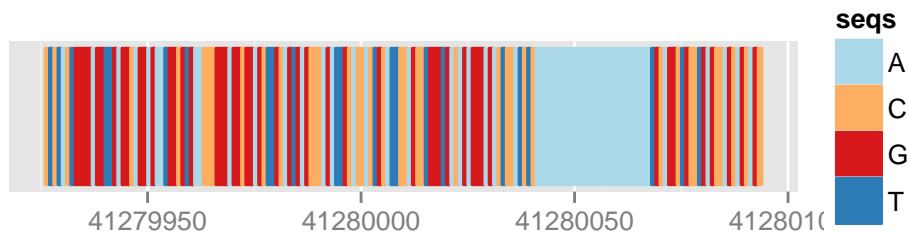
```
library(BSgenome.Hsapiens.UCSC.hg19)
bg <- BSgenome.Hsapiens.UCSC.hg19
p.bg <- autoplot(bg, which = wh)
## no geom
p.bg
```



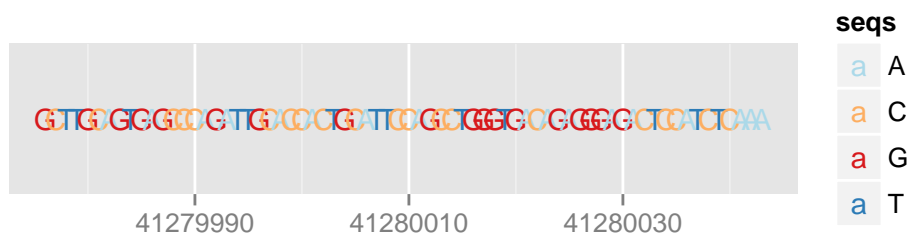
```
## segment
p.bg + zoom(1/100)
```



```
## rectangle
p.bg + zoom(1/1000)
```



```
## text
p.bg + zoom(1/2500)
```



To override a semantic zoom threshold, you simply provide a geom explicitly.

```
library(BSgenome.Hsapiens.UCSC.hg19)
bg <- BSgenome.Hsapiens.UCSC.hg19
## force to use geom 'segment' at this level
autoplot(bg, which = resize(w, width = width(w)/2000), geom = "segment")
```

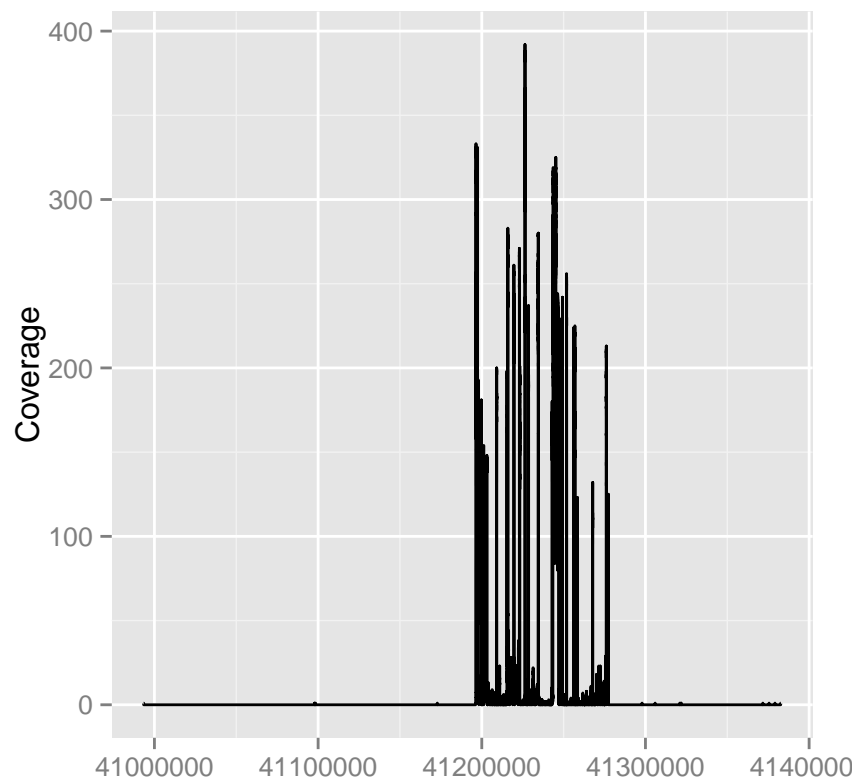
2.4 Add an alignment track

`ggbio` supports visualization of alignments file stored in bam, `autoplot` method accepts

- bam file path (indexed)
- *BamFile* object
- *GappedAlignemnt* object

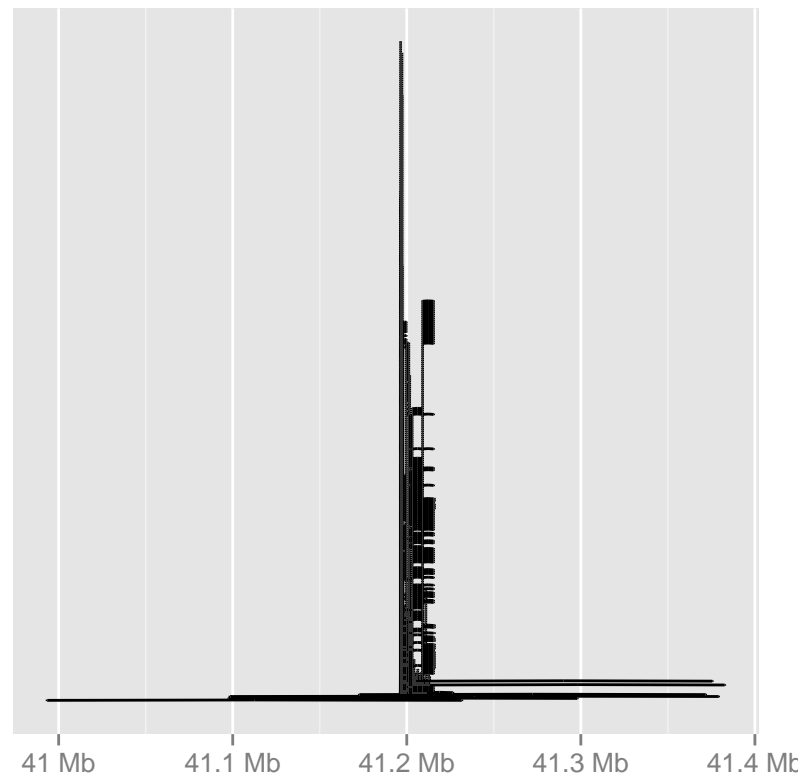
It's simple to just pass a file path to `autoplot` function, you can stream a chunk of region by providing 'which' parameter. Otherwise please use method 'estiamte' to show overall estiamted coverage.

```
f1.bam <- system.file("extdata", "wg-brca1.sorted.bam", package = "biovizBase")
wh <- keepSeqlevels(wh, "chr17")
autoplot(f1.bam, which = wh)
```



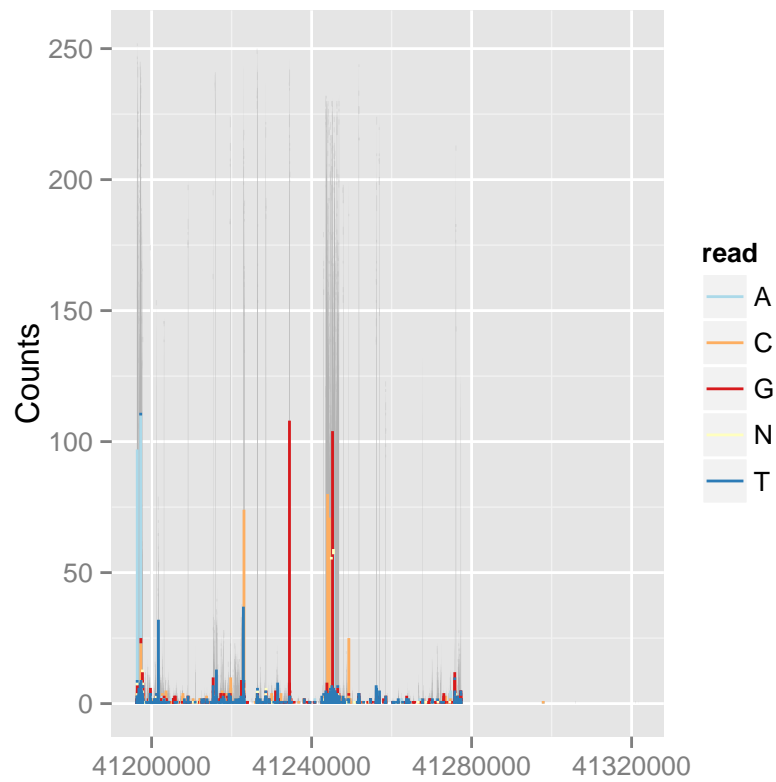
geom 'gapped pair' will show you alignments.

```
f1.bam <- system.file("extdata", "wg-brca1.sorted.bam", package = "biovizBase")
wh <- keepSeqlevels(wh, "chr17")
autoplot(f1.bam, which = resize(wh, width = width(wh)/10), geom = "gapped.pair")
```

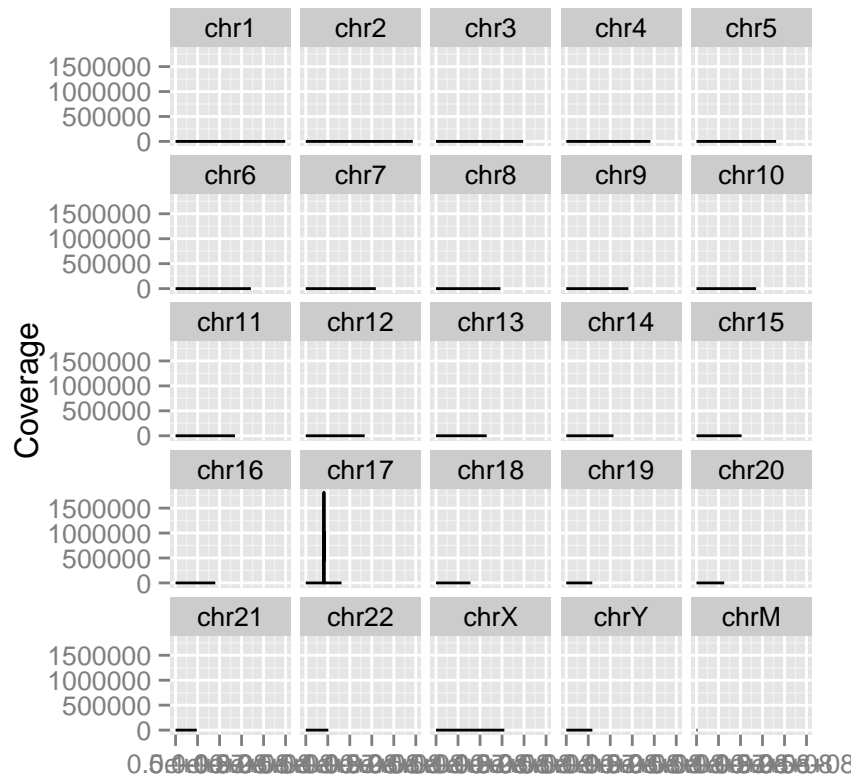
To show mismatch proportion, you have to provide reference sequence, the mismatched proportion is color coded in the bar chart.

```
library(BSgenome.Hsapiens.UCSC.hg19)
bg <- BSgenome.Hsapiens.UCSC.hg19
p.mis <- autoplot(fl.bam, bsgenome = bg, which = wh, stat = "mismatch")
p.mis
```

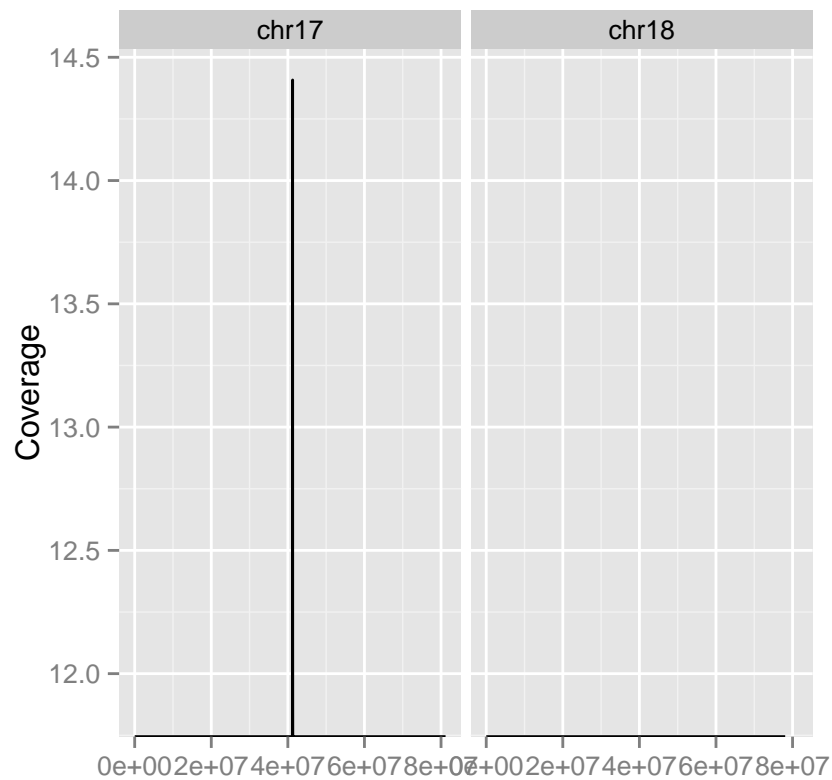


To view overall estimated coverage distribution, please use method 'estimate'. 'which' parameter also accept characters. And there is a hidden value called '..coverage..' to let you do simple transformation in aes().

```
autoplot(fl.bam, method = "estimate")
```



```
autoplot(fl.bam, method = "estimate", which = paste0("chr", 17:18), aes(y = log(..coverage..)))
```



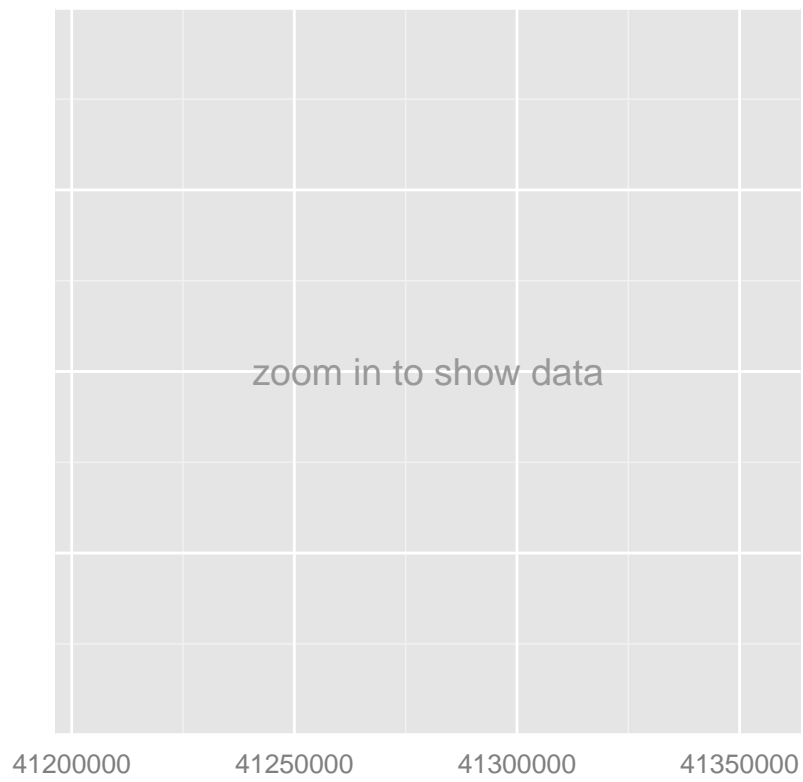
2.5 Add a variants track

This track is supported by semantic zoom.

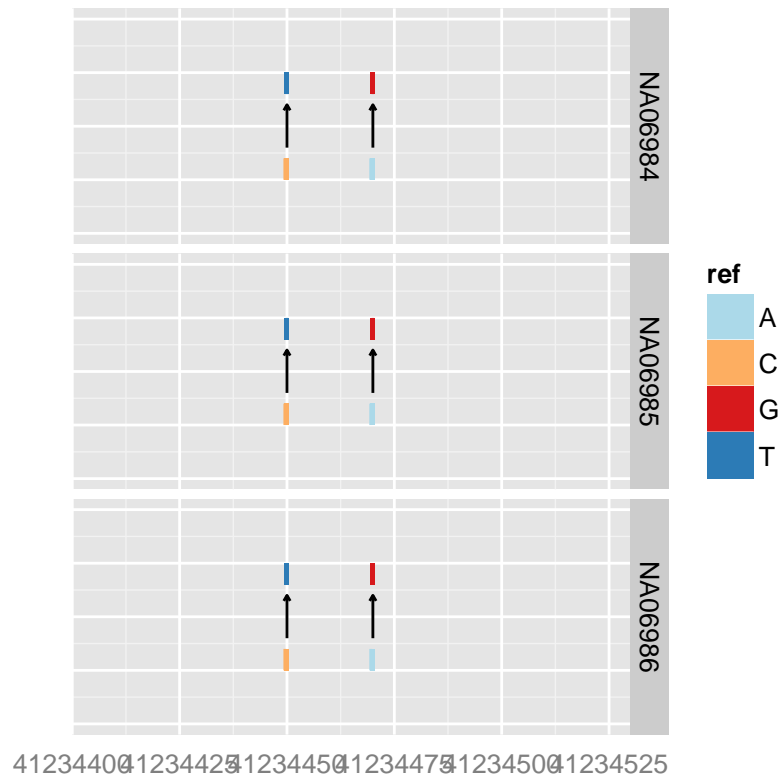
To view your variants file, you could

- Import it using package [VariantAnnotation](#) as *VCF* object, then use `autoplot`
- Convert it into *VRanges* object and use `autoplot`.
- Simply provide `vcf` file path in `autoplot()`.

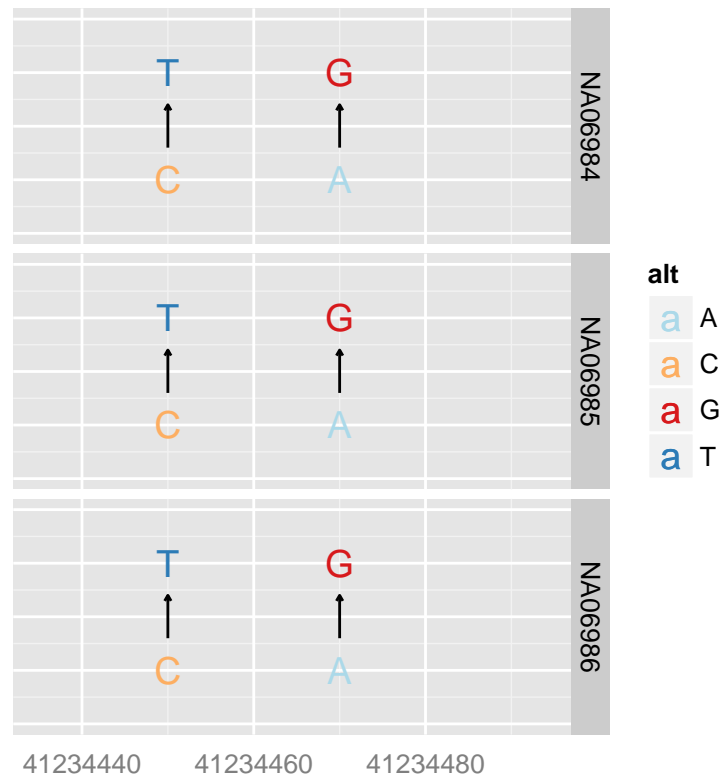
```
library(VariantAnnotation)
fl.vcf <- system.file("extdata", "17-1409-CEU-brca1.vcf.bgz", package="biovizBase")
vcf <- readVcf(fl.vcf, "hg19")
vr <- as(vcf[, 1:3], "VRanges")
vr <- renameSeqlevels(vr, value = c("17" = "chr17"))
## small region contains data
gr17 <- GRanges("chr17", IRanges(41234400, 41234530))
p.vr <- autoplot(vr, which = wh)
## none geom
p.vr
```



```
## rect geom  
p.vr + xlim(gr17)
```



```
## text geom
p.vr + xlim(gr17) + zoom()
```

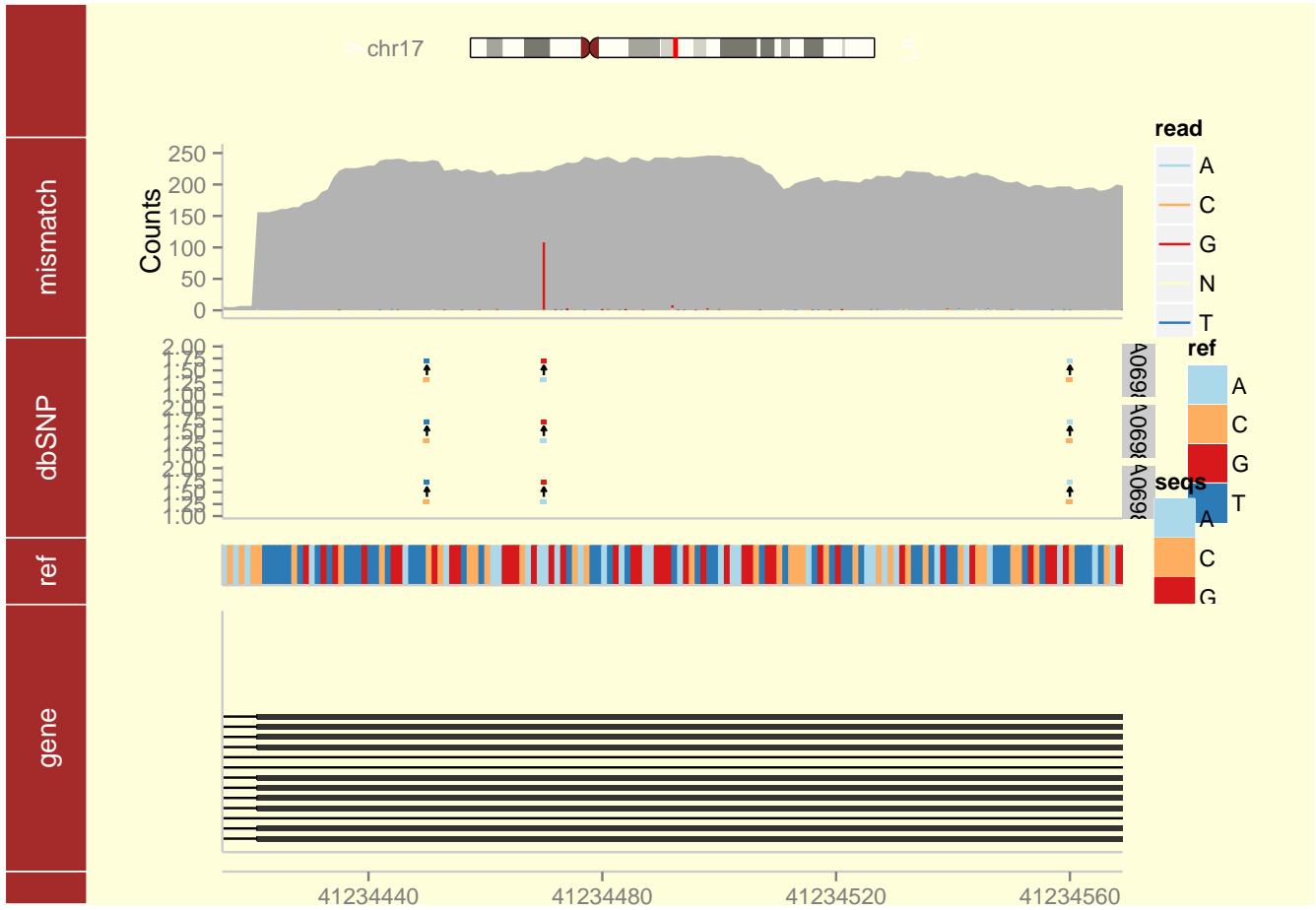


You can simply override geom

```
autoplot(vr, which = wh, geom = "rect", arrow = FALSE)
```

2.6 Building your tracks

```
## tks <- tracks(p.ideo, mismatch = p.mis, dbSNP = p.vr, ref = p.bs, gene = p.txdb)
## tks <- tracks(fl.bam, fl.vcf, bs, Homo.sapiens) ## default ideo = FALSE, turned on
## tks <- tracks(fl.bam, fl.vcf, bs, Homo.sapiens, ideo = TRUE)
## tks + xlim(gr17)
gr17 <- GRanges("chr17", IRanges(41234415, 41234569))
tks <- tracks(p.ideo, mismatch = p.mis, dbSNP = p.vr, ref = p.bg, gene = p.txdb,
             heights = c(2, 3, 3, 1, 4)) + xlim(gr17) + theme_tracks_sunset()
tks
```



Chapter 3

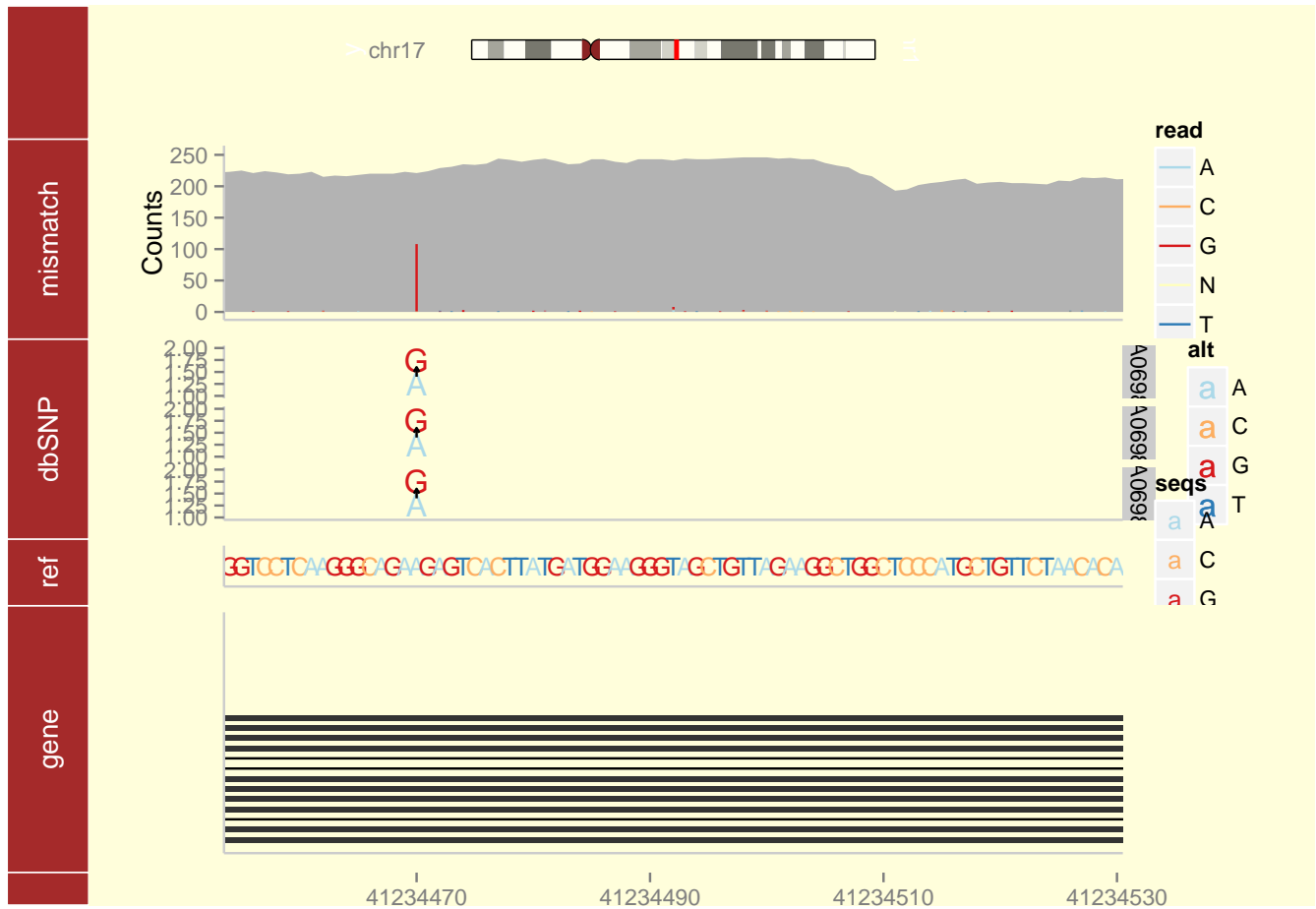
Simple navigation

We try to provide a simple navigation API for your plot, so you could zoom in and zoom out, or go through view chunks one by one.

- `zoom`: put a factor inside and you can zoom in or zoom out
- `nextView`: switch to next view
- `prevView`: switch to previous view

Navigation function also works for tracks plot too.

```
## zoom in  
tks + zoom()
```



Try following command yourself.

```
## zoom in with scale
p.txdb + zoom(1/8)
## zoom out
p.txdb + zoom(2)
## next view page
p.txdb + nextView()
## previous view page
p.txdb + prevView()
```

Don't forget `xlim` accept *GRanges* object (single row), so you could simply prepare a *GRanges* to store the region of interests and go through them one by one.

Chapter 4

Overview plots

Overview is a good way to show all events at the same time, give overall summary statistics for the whole genome. In this chapter, we will introduce three different layouts that are used a lot in genomic data visualization.

4.1 how to make circular plots

4.1.1 Introduction

Circular view is a special layout in *ggbio*, this idea has been implemented in many different software, for example, the *Circos* project. However, we keep the grammar of graphics for users, so mapping variables to aesthetics is very easy, *ggbio* leverage the data structure defined in *Bioconductor* to make this process as simple as possible.

4.1.2 Building circular plot layer by layer

Ok, let's start to process some raw data to the format we want. The data used in this study is from this paper¹. In this tutorial, we are going to

1. Visualize somatic mutation as segment.
2. Visualize inter, intra-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.

All the raw data processed and stored in *GRanges* ready for use, you can simply load the sample data from *biovizBase*

```
data("CRC", package = "biovizBase")
```

`layout_circle` is deprecated, because you have to set up radius and trackWidth manually with this function for creating circular plot.

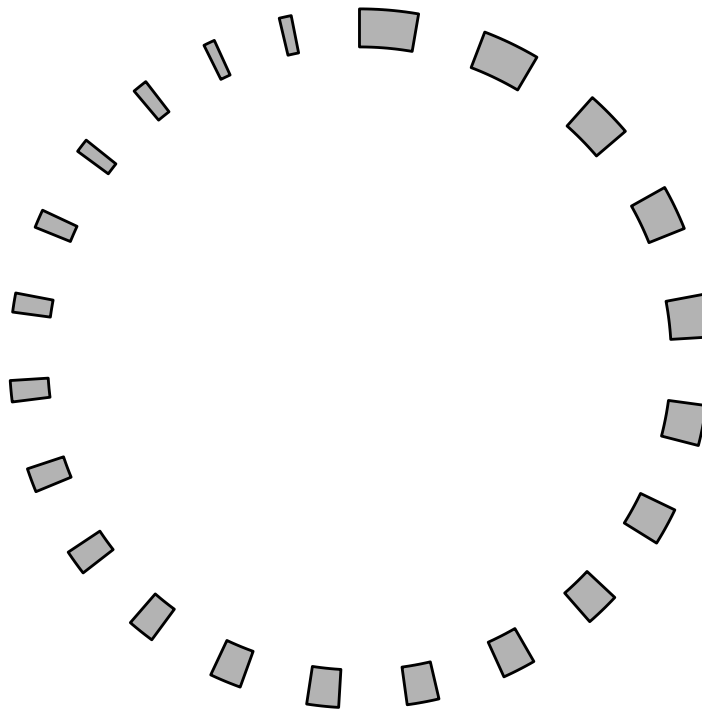
We now present the new `circle` function, it accepts *GRanges* object, and users don't have to specify radius, track width, you just add them one by one, it will be automatically created from inner circle to outside, unless you specify trackWidth and radius manually. To change default radius and trackWidth for all tracks, you simply put them in *ggbio* function.

- rule of thumb seqLengths, seqLevels and chromosomes names should be exactly the same.
- to use `circle`, you have to use *ggbio* constructor at the beginning instead of *ggplot*.

¹<http://www.nature.com/ng/journal/v43/n10/full/ng.936.html>

You can use autoplot to create single track easily like

```
head(hg19sub)
## 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      4 ...      20      21      22
## 249250621 243199373 198022430 191154276 ... 63025520 48129895 51304566
autoplot(hg19sub, layout = "circle", fill = "gray70")
```

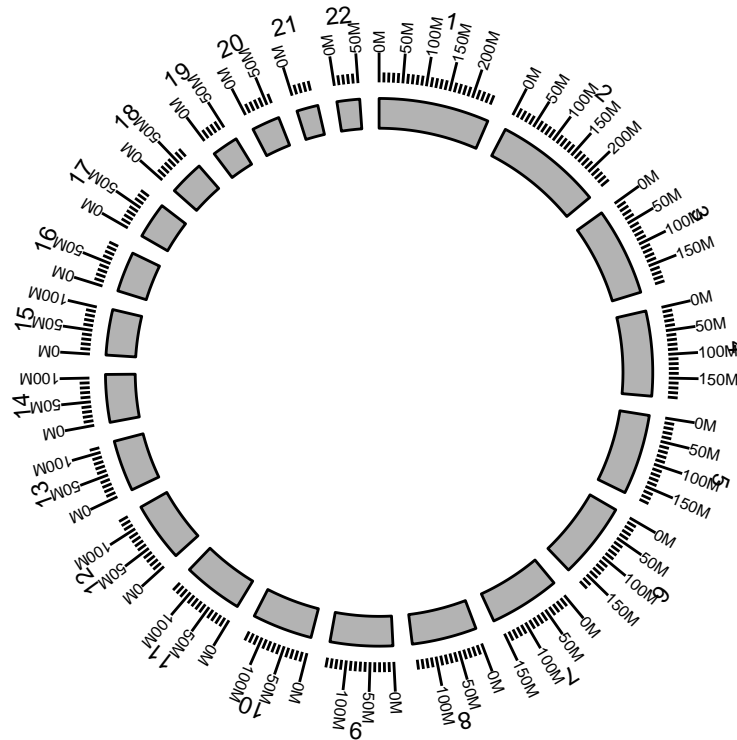


Hoever, the low level circle function leave you more flexibility to build circular plot one by one. Let's start to add tracks one by one.

Let's use the same data to create ideogram, label and scale track, it layouts the circle by the order you created from inside to outside.

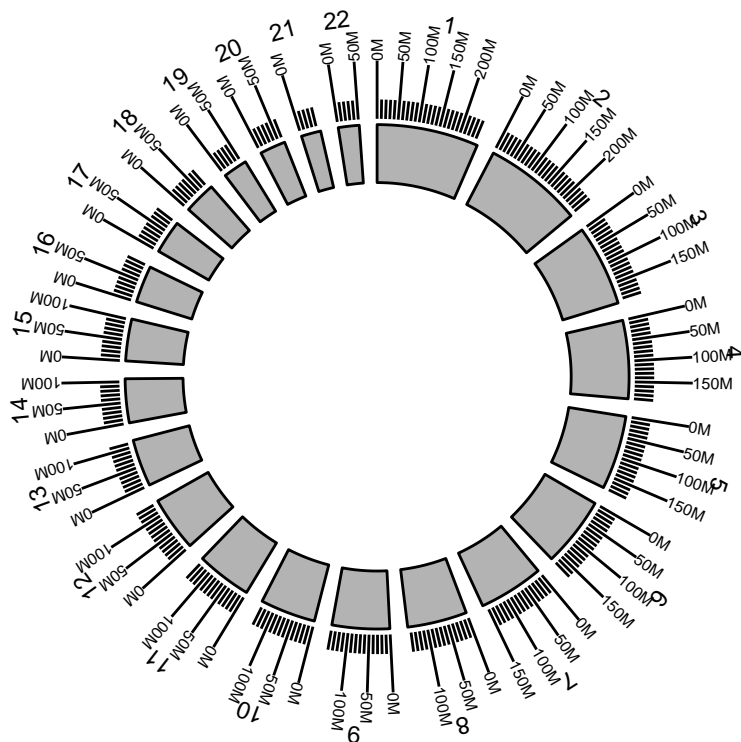
```
p <- ggbio() + circle(hg19sub, geom = "ideo", fill = "gray70") +
  circle(hg19sub, geom = "scale", size = 2) +
  circle(hg19sub, geom = "text", aes(label = seqnames), vjust = 0, size = 3)
```

p



To simply override the setting, you can do it globally in `ggbio` function or individually `circle` function by specifying parameters `trackWidth` and `radius`, you can also specify the global setting for `buffer` in between in `ggbio` like example below.

```
p <- ggbio(trackWidth = 10, buffer = 0, radius = 10) + circle(hg19sub, geom = "ideo", fill = "gray70") +
  circle(hg19sub, geom = "scale", size = 2) +
  circle(hg19sub, geom = "text", aes(label = seqnames), vjust = 0, size = 3)
p
```



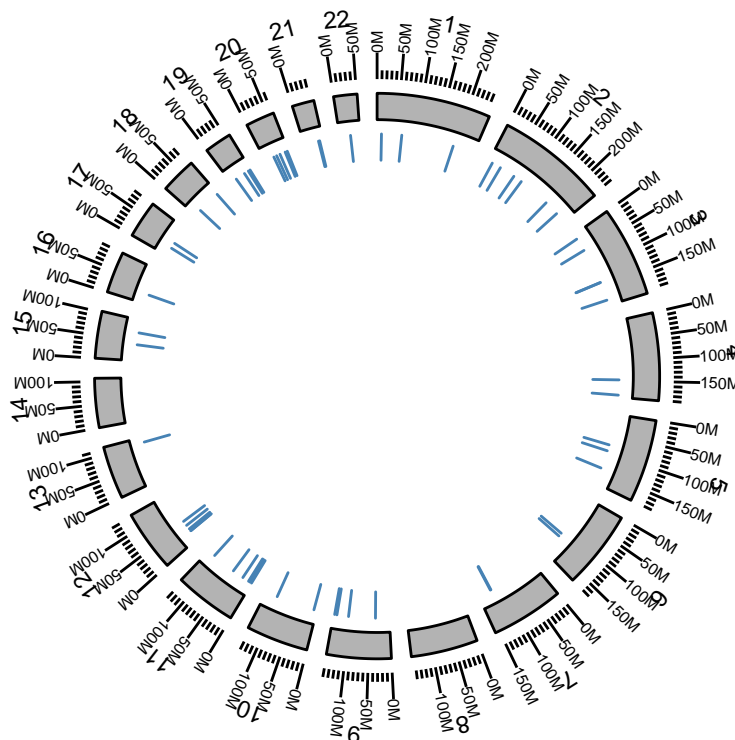
Then we add a "rectangle" track to show somatic mutation, this will look like vertical segments.

```
head(mut.gr)

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

```
## [5] C C T
## [6] T T C
## ---
## seqlengths:
## 1 2 3 4 ... 20 21 22
## 249250621 243199373 198022430 191154276 ... 63025520 48129895 51304566

p <- ggbio() + circle(mut.gr, geom = "rect", color = "steelblue") +
  circle(hg19sub, geom = "ideo", fill = "gray70") +
  circle(hg19sub, geom = "scale", size = 2) +
  circle(hg19sub, geom = "text", aes(label = seqnames), vjust = 0, size = 3)
p
```



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. 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.

```
head(crc.gr)
## GRanges with 6 ranges and 17 metadata columns:
## seqnames ranges strand | individual str1 class
## <Rle> <IRanges> <Rle> | <factor> <integer> <factor>
## [1] 18 [56258628, 56258628] * | CRC-4 1 long_range
## [2] 18 [44496014, 44496014] * | CRC-4 1 long_range
## [3] 18 [45023683, 45023683] * | CRC-4 0 long_range
## [4] 8 [52186319, 52186319] * | CRC-4 0 deletion
## [5] 8 [37328910, 37328910] * | CRC-4 0 inter_chr
```

```
## [6] 8 [35575394, 35575394] * | CRC-4 0 inter_chr
## span tumreads normreads gene1 gene2
## <numeric> <integer> <integer> <factor> <factor>
## [1] 2104165 491 2 MC4R ZCCHC2
## [2] 12947165 265 0 KIAA0427 CDH20
## [3] 13356670 238 0 DYM ZCCHC2
## [4] 268 94 0 PXDNL PXDNL
## [5] <NA> 56 0 ZNF703 PAK7
## [6] <NA> 53 0 UNC5D RALGAPB
## site1
## <factor>
## [1] IGR: 69Kb before MC4R(-)
## [2] Intron of KIAA0427(+): 4Kb after exon 8
## [3] Intron of DYM(-): 14Kb after exon 13
## [4] IGR: 208Kb before PXDNL(-)
## [5] IGR: 344Kb before ZNF703(+)
## [6] Intron of UNC5D(+): 3Kb after exon 4
## site2
## <factor>
## [1] Intron of ZCCHC2(+): 222bp before exon 4
## [2] IGR: 134Kb before CDH20(+)
## [3] Intron of ZCCHC2(+): 854bp before exon 9
## [4] IGR: 208Kb before PXDNL(-)
## [5] Intron of PAK7(-): 11Kb after exon 4
## [6] Intron of RALGAPB(+): 839bp after exon 15
## fusion quality score BPresult
## <factor> <numeric> <numeric> <integer>
## [1] - 1.0000 491.00 -1
## [2] - 0.9944 263.52 1
## [3] Protein fusion: in frame (ZCCHC2-DYM) 1.0000 238.00 1
## [4] - 1.0000 94.00 -1
## [5] - 0.9740 54.55 1
## [6] Antisense fusion 1.0000 53.00 1
## validation_result to.gr rearrangements
## <factor> <GRanges> <character>
## [1] not_subjected_to_validation 18*:[58362793, 58362793] intrachromosomal
## [2] not_subjected_to_validation 18*:[57443167, 57443167] intrachromosomal
## [3] somatic 18*:[58380361, 58380361] intrachromosomal
## [4] not_subjected_to_validation 8*:[52186587, 52186587] intrachromosomal
## [5] somatic 20*:[ 9561906, 9561906] interchromosomal
## [6] not_subjected_to_validation 20*:[36595752, 36595752] interchromosomal
## ---
## seqlengths:
## 1 2 3 4 ... 20 21 22
## 249250621 243199373 198022430 191154276 ... 63025520 48129895 51304566
```

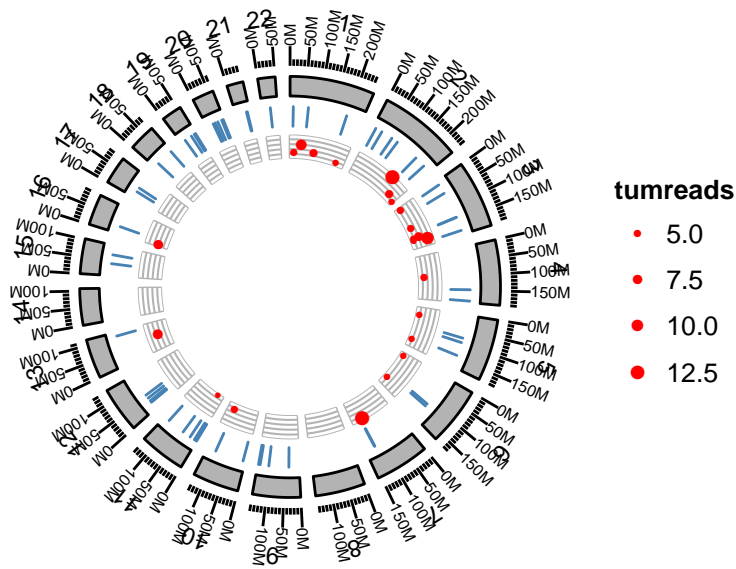
Here in this example, we use "intrachromosomal" to label rearrangement within the same chromosomes and use "interchromosomal" to label rearrangement in different chromosomes.

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

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

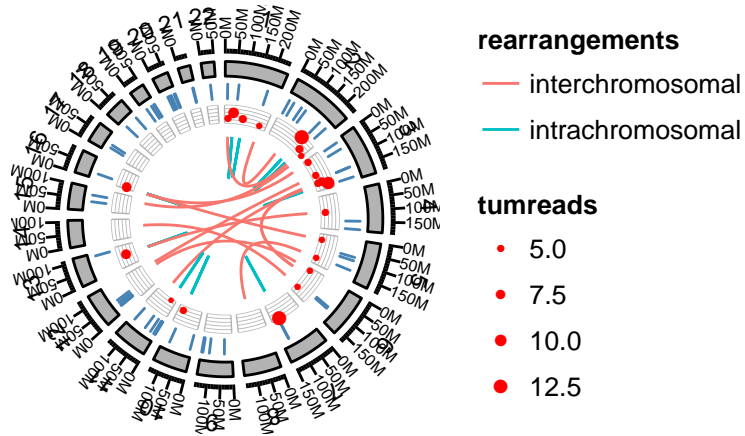
Ok, add a "point" track with grid background for rearrangement data and map 'y' to variable "score", map 'size' to variable "tumreads", rescale the size to a proper size range.


```
## manually specify radius
p <- p + circle(gr.crc1, geom = "point", aes(y = score, size = tumreads),
              color = "red", grid = TRUE, radius = 30) + scale_size(range = c(1, 2.5))
p
```



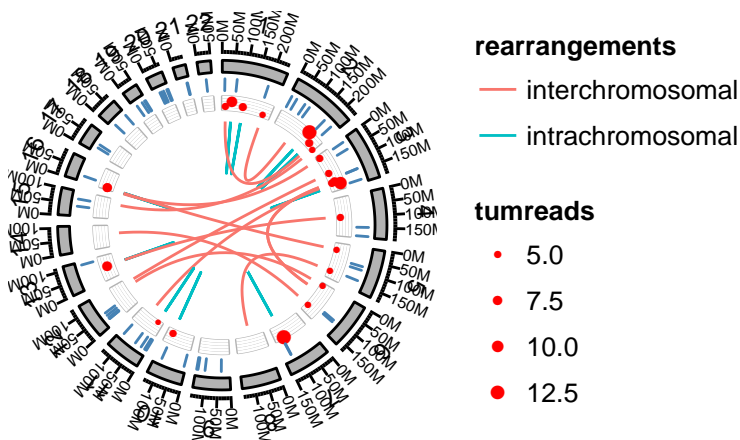
Finally, let's add links and map color to rearrangement types. Remember you need to specify 'linked.to' parameter to the column that contain end point of the data.

```
## specify radius manually
p <- p + circle(gr.crc1, geom = "link", linked.to = "to.gr", aes(color = rearrangements),
              radius = 23)
p
```



All those code could be simply constructed by following code

```
p <- ggbio() +
  circle(gr.crc1, geom = "link", linked.to = "to.gr", aes(color = rearrangements)) +
  circle(gr.crc1, geom = "point", aes(y = score, size = tumreads),
        color = "red", grid = TRUE) + scale_size(range = c(1, 2.5)) +
  circle(mut.gr, geom = "rect", color = "steelblue") +
  circle(hg19sub, geom = "ideo", fill = "gray70") +
  circle(hg19sub, geom = "scale", size = 2) +
  circle(hg19sub, geom = "text", aes(label = seqnames), vjust = 0, size = 3)
p
```



4.1.3 Complex arrangement 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.

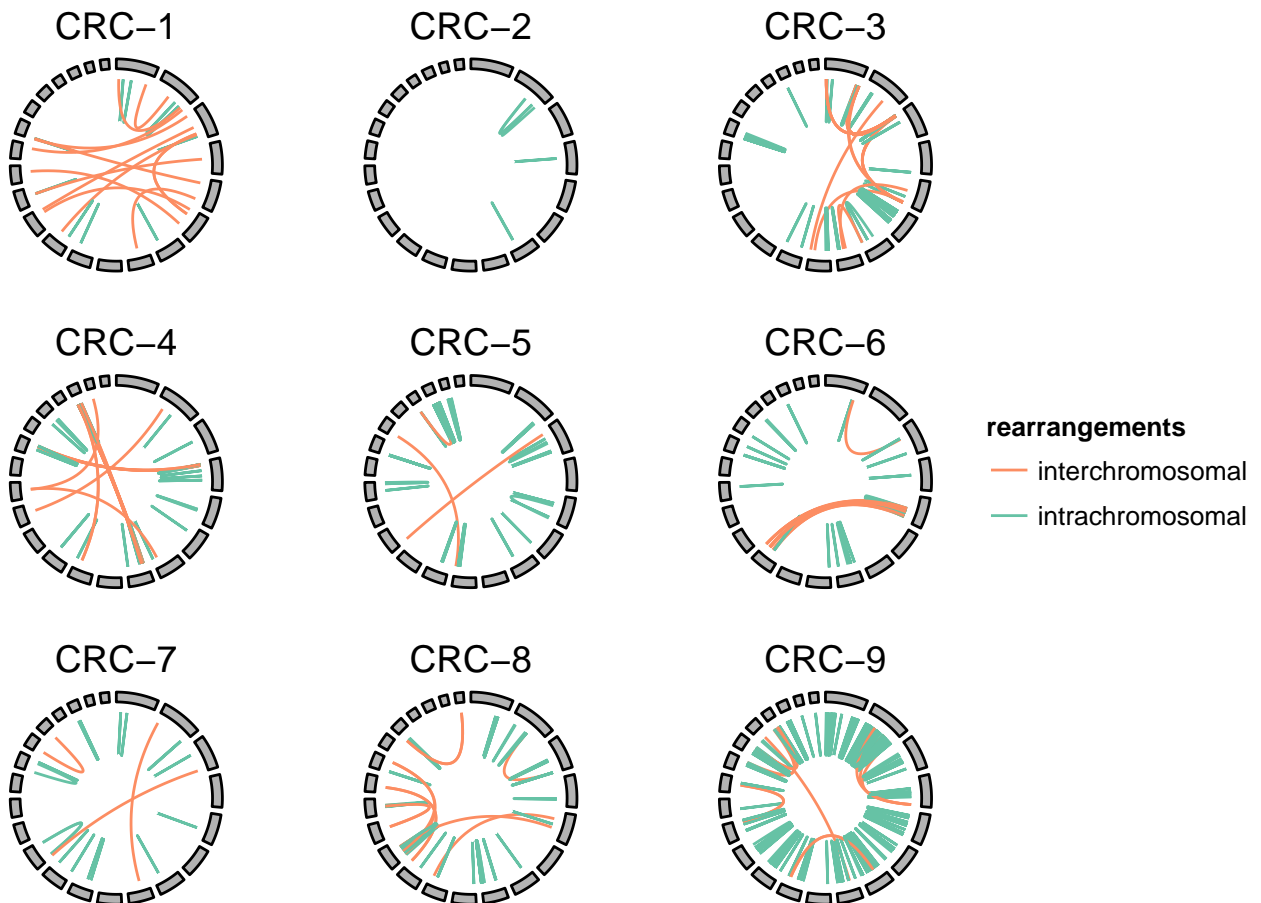
We just want 9 single circular plots put together in one page, since we cannot keep too many tracks, we only keep ideogram and links. Here is one sample.

```
grl <- split(crc.gr, values(crc.gr)$individual)
## need "unit", load grid
library(grid)
crc.lst <- lapply(grl, function(gr.cur){
  print(unique(as.character(values(gr.cur)$individual)))
  cols <- RColorBrewer::brewer.pal(3, "Set2")[2:1]
  names(cols) <- c("interchromosomal", "intrachromosomal")
  p <- ggbio() + circle(gr.cur, geom = "link", linked.to = "to.gr",
    aes(color = rearrangements)) +
    circle(hg19sub, geom = "ideo",
      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(crc.lst, widths = c(4, 1), legend.idx = 1, ncol = 3)
```



```
## NULL
```

4.2 How to make grandlinear plots

4.2.1 Introduction

Let's use a subset of *PLINK* output (<https://github.com/stephenturner/qqman/blob/master/plink.assoc.txt.gz>) as our example test data.

```
snp <- read.table(system.file("extdata", "plink.assoc.sub.txt", package = "biovizBase"),
                 header = TRUE)
require(biovizBase)
gr.snp <- transformDfToGr(snp, seqnames = "CHR", start = "BP", width = 1)
head(gr.snp)

## GRanges with 6 ranges and 10 metadata columns:
##      seqnames      ranges strand |      CHR      SNP      BP
##      <Rle>        <IRanges> <Rle> | <integer> <factor> <integer>
## [1]      4 [ 10794096, 10794099] * |      4 rs9291494 10794096
## [2]     14 [ 55853742, 55853755] * |     14 rs1152481 55853742
## [3]      6 [ 55188853, 55188858] * |      6 rs3134708 55188853
## [4]     17 [ 4146033, 4146049] * |     17 rs2325988 4146033
## [5]     19 [ 46089501, 46089519] * |     19 rs8103444 46089501
## [6]      1 [107051695, 107051695] * |      1 rs12072065 107051695
##      A1      F_A      F_U      A2      CHISQ      P      OR
##      <factor> <numeric> <numeric> <factor> <numeric> <numeric> <numeric>
## [1]      G    0.3061    0.1341      A    7.5070 0.006147    2.8480
## [2]      G    0.3542    0.2805      A    1.1030 0.293600    1.4070
## [3]      C    0.2500    0.2875      A    0.3135 0.575500    0.8261
## [4]      G    0.2551    0.2317      A    0.1323 0.716100    1.1360
## [5]      C    0.3980    0.2927      A    2.1750 0.140300    1.5970
## [6]      0    0.0000    0.0000      C      <NA>      <NA>      <NA>
## ---
##      seqlengths:
##      4 14 6 17 19 1 9 7 20 11 10 2 18 5 12 22 8 13 3 16 15 21
##      NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA
##
## change the seqname order
require(GenomicRanges)
gr.snp <- keepSeqlevels(gr.snp, as.character(1:22))
seqlengths(gr.snp)

## 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22
## NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA
##
## need to assign seqlengths
data(ideoCyto, package = "biovizBase")
seqlengths(gr.snp) <- as.numeric(seqlengths(ideoCyto$hg18)[1:22])
## remove missing
gr.snp <- gr.snp[!is.na(gr.snp$P)]
## transform pvalue
values(gr.snp)$pvalue <- -log10(values(gr.snp)$P)
head(gr.snp)

## GRanges with 6 ranges and 11 metadata columns:
##      seqnames      ranges strand |      CHR      SNP      BP
##      <Rle>        <IRanges> <Rle> | <integer> <factor> <integer>
## [1]      4 [10794096, 10794099] * |      4 rs9291494 10794096
## [2]     14 [55853742, 55853755] * |     14 rs1152481 55853742
## [3]      6 [55188853, 55188858] * |      6 rs3134708 55188853
## [4]     17 [ 4146033, 4146049] * |     17 rs2325988 4146033
```

```
## [5] 19 [46089501, 46089519] * | 19 rs8103444 46089501
## [6] 9 [81517907, 81517915] * | 9 rs2591 81517907
## A1 F_A F_U A2 CHISQ P OR
## <factor> <numeric> <numeric> <factor> <numeric> <numeric> <numeric>
## [1] G 0.3061 0.13410 A 7.5070 0.006147 2.8480
## [2] G 0.3542 0.28050 A 1.1030 0.293600 1.4070
## [3] C 0.2500 0.28750 A 0.3135 0.575500 0.8261
## [4] G 0.2551 0.23170 A 0.1323 0.716100 1.1360
## [5] C 0.3980 0.29270 A 2.1750 0.140300 1.5970
## [6] C 0.1042 0.04878 T 1.8720 0.171200 2.2670
## pvalue
## <numeric>
## [1] 2.2113
## [2] 0.5322
## [3] 0.2400
## [4] 0.1450
## [5] 0.8529
## [6] 0.7665
## ---
## seqlengths:
## 1 2 3 4 ... 20 21 22
## 247249719 242951149 199501827 191273063 ... 62435964 46944323 49691432
## done
```

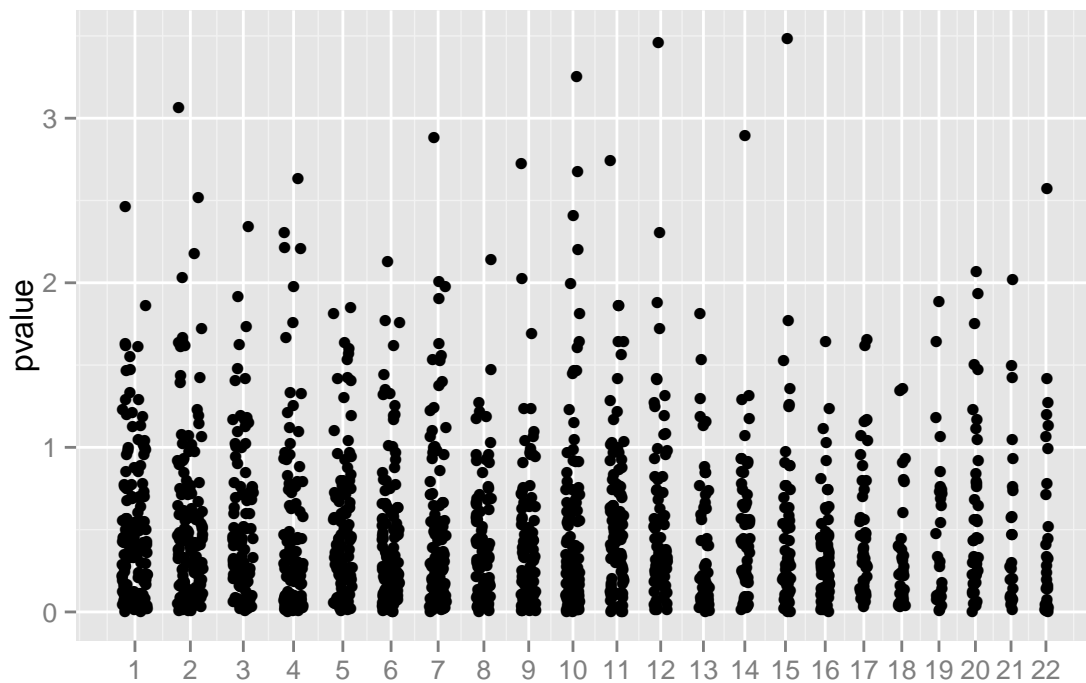
The data is ready, we need to pay attention

- if seqlengths is missing, we use data range, so the chromosome length is not accurate
- use seqlevel to control order of chromosome

4.2.2 Corrdinate genome

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.

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



However, we recommend you to use more powerful function `plotGrandLinear` to generate manhattan plot introduced in next section.

4.2.3 Convenient `plotGrandLinear` function

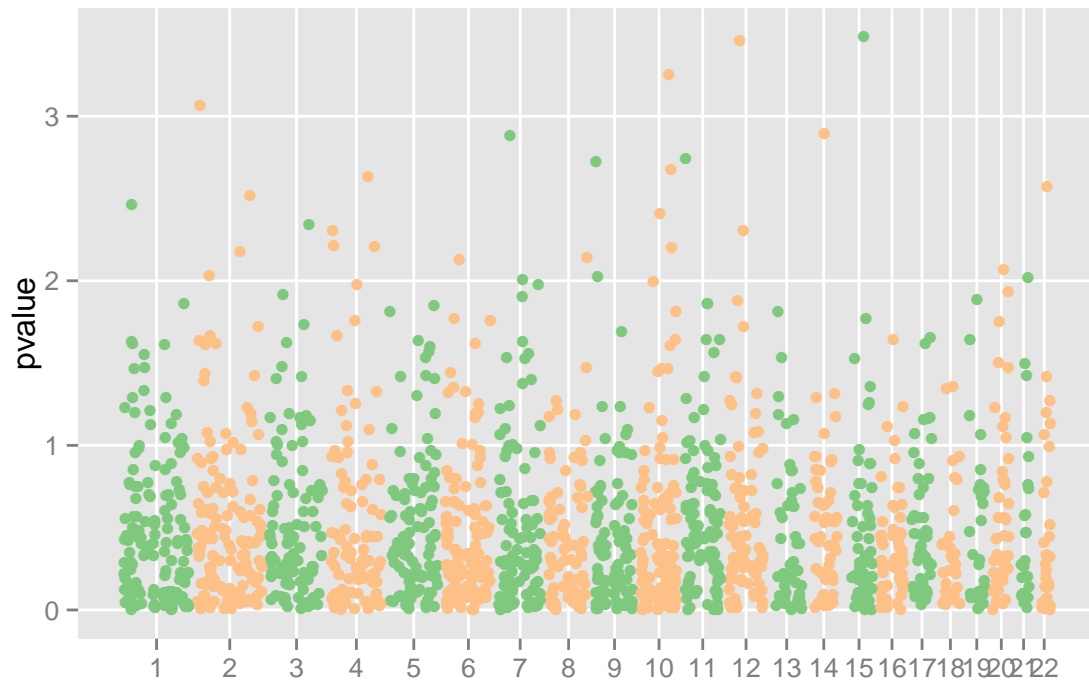
For *Manhattan plot*, we have a function called `plotGrandLinear`. `aes(y =)` is required to indicate the y value, e.g. p-value.

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 **recycle** them to all chromosomes.
- if `color` is single character representing color, then just use one arbitrary color.

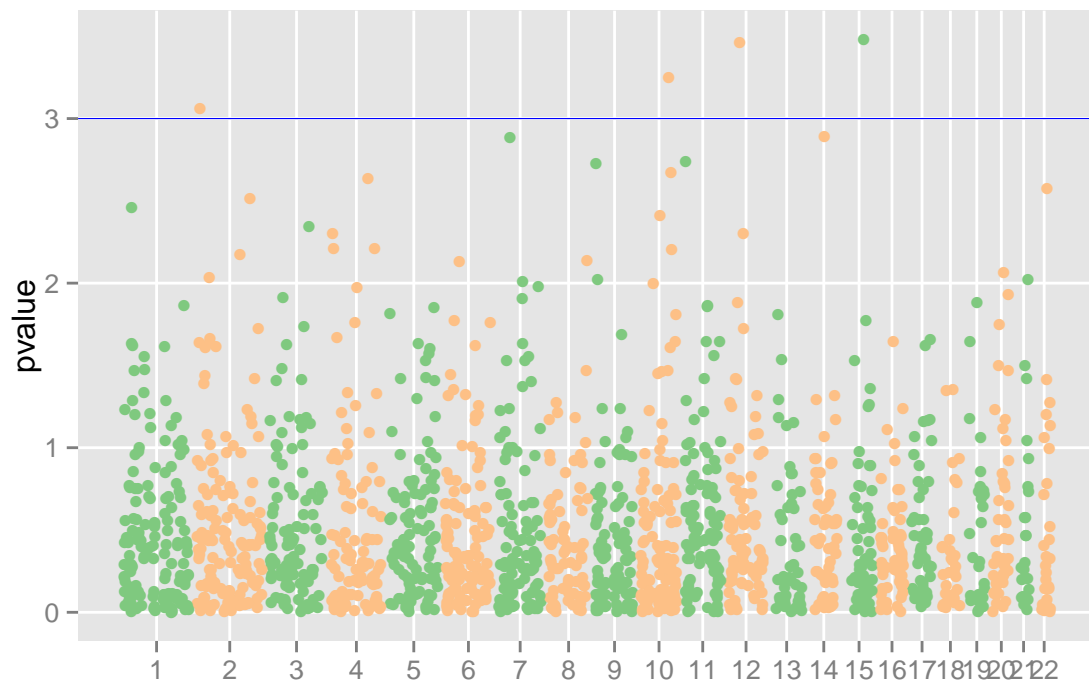
Let's test some examples for controlling colors.

```
plotGrandLinear(gr.snp, aes(y = pvalue), color = c("#7fc97f", "#fdc086"))
```



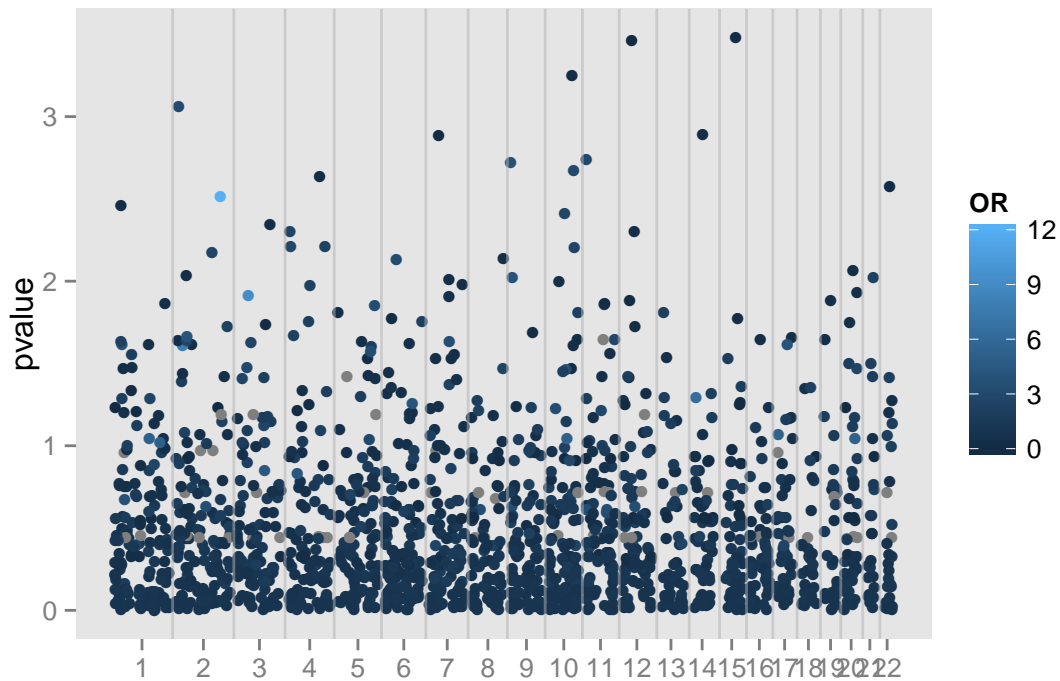
Let's add a cutoff line

```
plotGrandLinear(gr.snp, aes(y = pvalue), color = c("#7fc97f", "#fdc086"),  
               cutoff = 3, cutoff.color = "blue", cutoff.size = 0.2)
```



Sometimes you use color to mapping other variables so you may need a different to separate chromosomes.

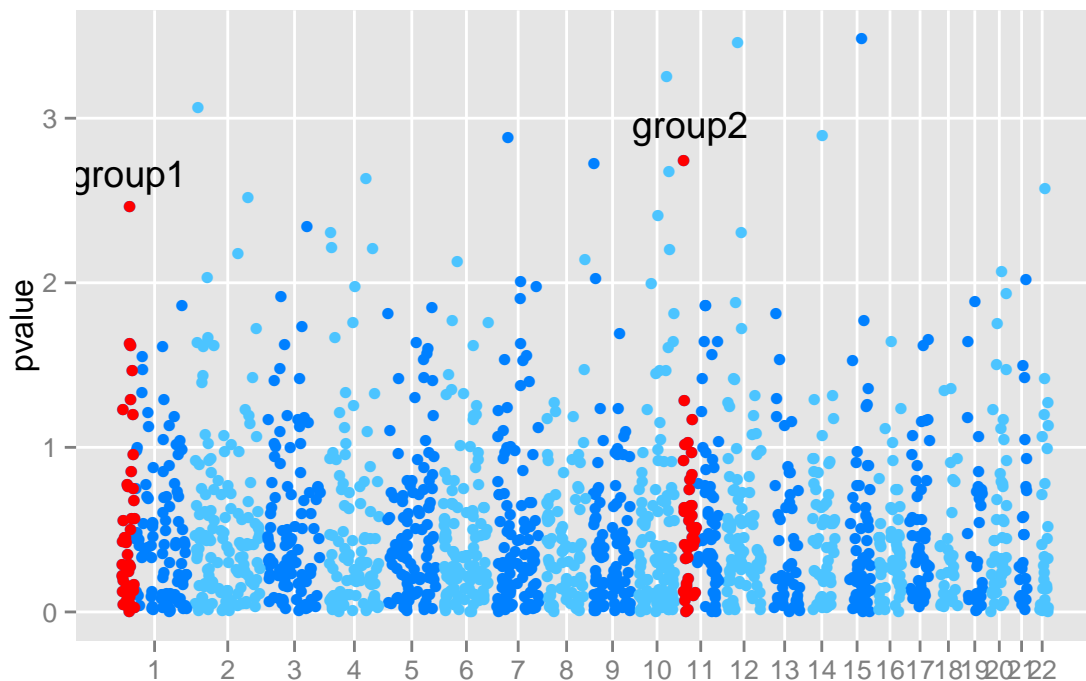
```
plotGrandLinear(gr.snp, aes(y = pvalue, color = OR), spaceline = TRUE, legend = TRUE)
```



4.2.4 How to highlight some points?

You can provide a highlight *GRanges*, and each row highlights a set of overlapped snps, and labeled by rownames or certain columns, there is more control in the function as parameters, with prefix `highlight.*`, so you could control color, label size and color, etc.

```
gro <- GRanges(c("1", "11"), IRanges(c(100, 2e6), width = 5e7))
names(gro) <- c("group1", "group2")
plotGrandLinear(gr.snp, aes(y = pvalue), highlight.gr = gro)
```



4.3 How to make stacked karyogram overview plots

4.3.1 Introduction

A karyotype is the number and appearance of chromosomes in the nucleus of a eukaryotic cell². It's one kind of overview when we want to show distribution of certain events on the genome, for example, binding sites for certain protein, even compare them across samples as example shows in this section.

`GRanges` and `Seqinfo` objects are an ideal container for storing data needed for karyogram plot. Here is the strategy we used for generating ideogram templates.

- Although `seqlengths` is not required, it's 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.

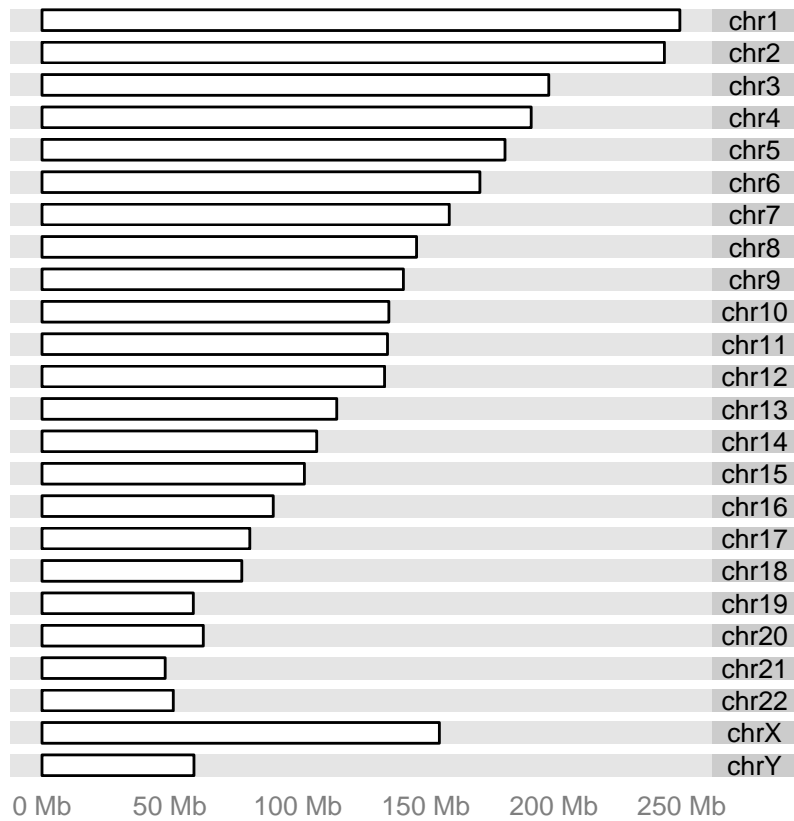
4.3.2 Create karyogram template

Let's first introduce how to use `autoplot` to generate karyogram graphic.

The most easy one is to just plot `Seqinfo` by using `autoplot`, if your `GRanges` object has `seqinfo` with `seqlengths` information. Then you add data layer later.

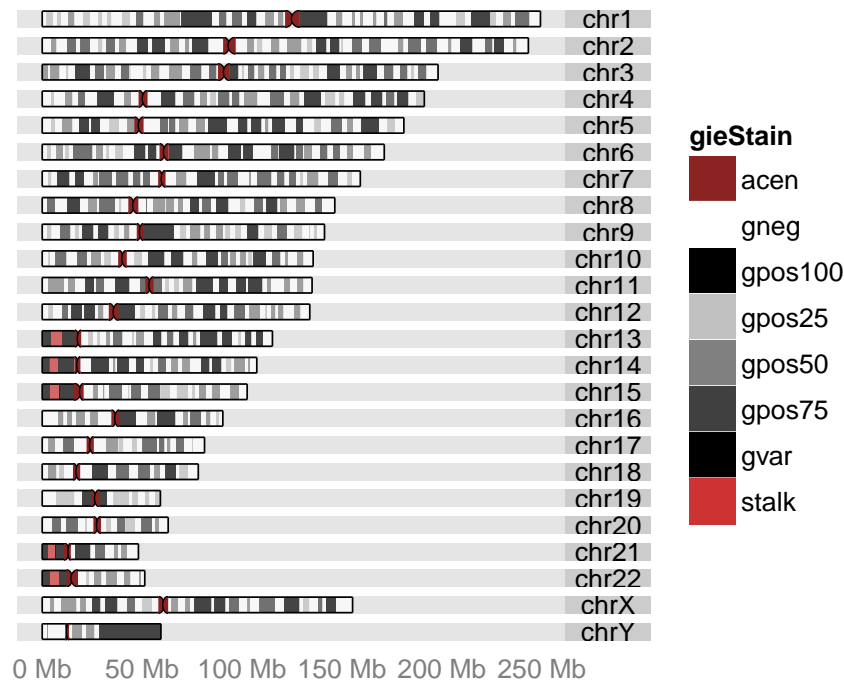
²<http://en.wikipedia.org/wiki/Karyotype>

```
data(ideoCyto, package = "biovizBase")
autoplot(seqinfo(ideoCyto$hg19), layout = "karyogram")
```



To show cytoband, your data need to have cytoband information, we stored some data for you, including *hg19*, *hg18*, *mm10*, *mm9*.

```
## turn on cytoband if it exists
biovizBase::isIdeogram(ideoCyto$hg19)
## [1] TRUE
autoplot(ideoCyto$hg19, layout = "karyogram", cytoband = TRUE)
```



To change order or only show a subset of the karyogram, you have to manipulate `seqlevels`, please check out manual for `keepSeqlevels`, `seqlevels` in [GenomicRanges](#) package for more information. Or you could read the example below.

4.3.3 Add data on karyogram layout

If you have single data set stored as *GRanges* to show on a karyogram layout, `autoplot` function is enough for you to plot the data on it.

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 template, otherwise, the estimated chromosome lengths will be very rough and inaccurate. So what we need to do first is to *add seqlength information to this object*.

```
data(darned_hg19_subset500, package = "biovizBase")
dn <- darned_hg19_subset500
library(GenomicRanges)
seqlengths(dn)

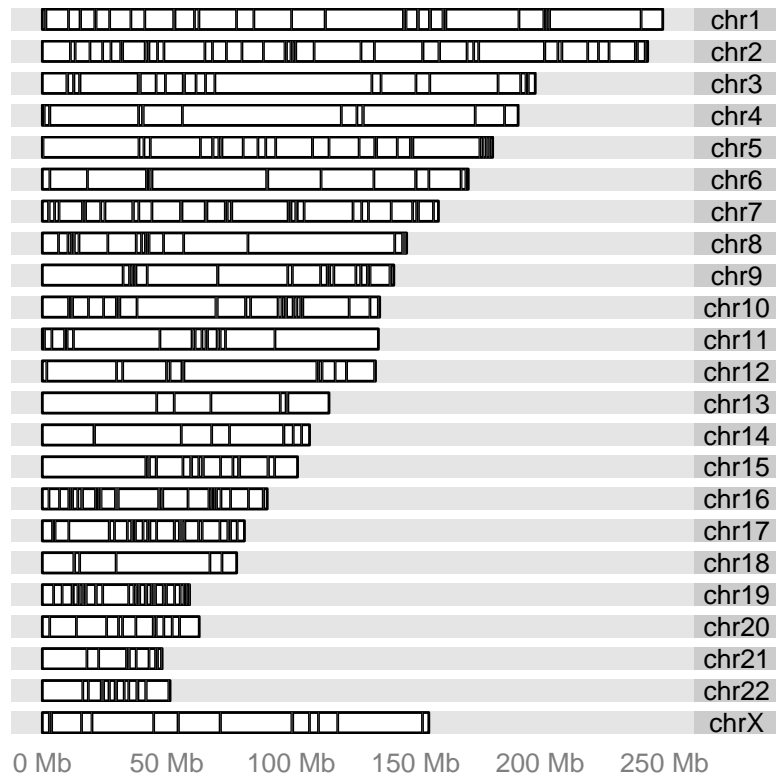
## chr1 chr10 chr11 chr12 chr13 chr14 chr15 chr16 chr17 chr18 chr19 chr2 chr20
## NA NA NA NA NA NA NA NA NA NA NA NA NA NA
## chr21 chr22 chr3 chr4 chr5 chr6 chr7 chr8 chr9 chrX
## NA NA NA NA NA NA NA NA NA NA

## add seqlengths
## we have seqlegnths information in another data set
seqlengths(dn) <- seqlengths(ideoCyto$hg19)[names(seqlengths(dn))]
## then we change order
dn <- keepSeqlevels(dn, paste0("chr", c(1:22, "X")))
seqlengths(dn)

## chr1 chr2 chr3 chr4 chr5 chr6 chr7 chr8
## 249250621 243199373 198022430 191154276 180915260 171115067 159138663 146364022
```

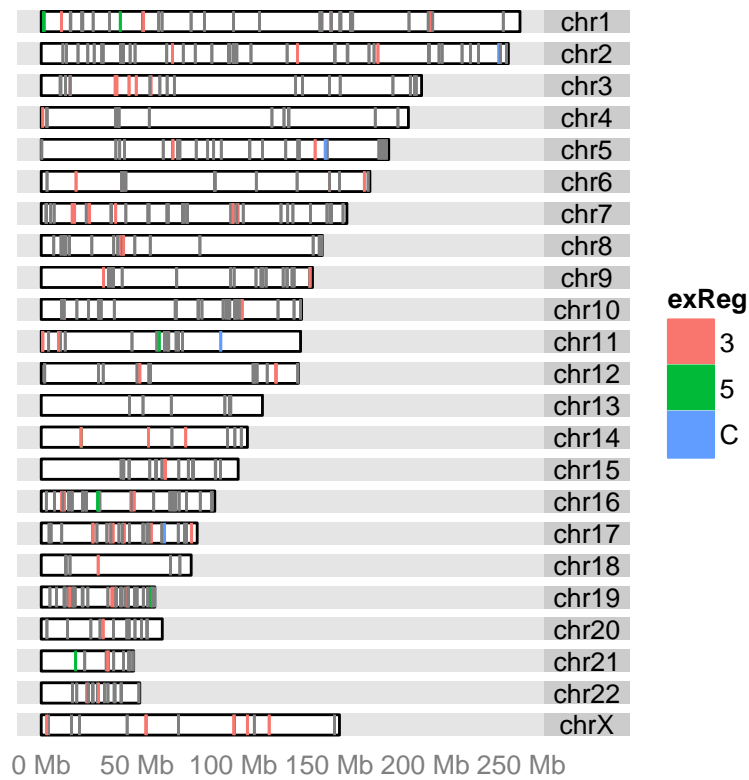
```
##      chr9      chr10      chr11      chr12      chr13      chr14      chr15      chr16
## 141213431 135534747 135006516 133851895 115169878 107349540 102531392 90354753
##      chr17      chr18      chr19      chr20      chr21      chr22      chrX
## 81195210 78077248 59128983 63025520 48129895 51304566 155270560
```

```
autoplot(dn, layout = "karyogram")
```



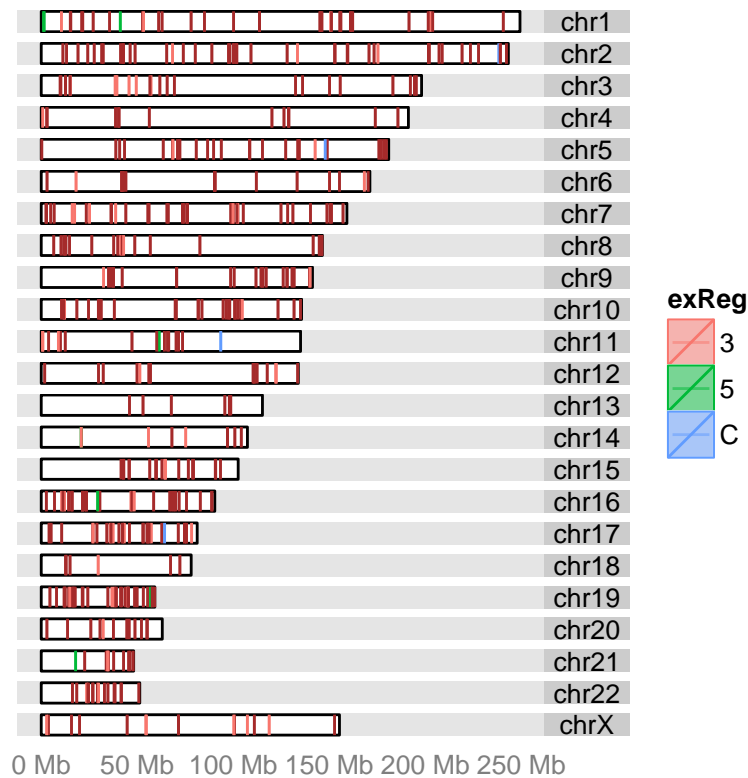
Then we take one step further, the power of *ggplot2* or *ggbio* is the flexible multivariate data mapping ability in graphics, make data exploration much more convenient. In the following example, we are trying to map a categorical variable 'exReg' to color, this variable is included in the data, and 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.

```
## 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))
```



Or you can set the missing value to particular color you want (NA values is not shown on the legend).

```
## 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), alpha = 0.5) +
  scale_color_discrete(na.value = "brown")
```



Well, sometimes we have too many values, we want to separate them by groups and show them at different height, below is a hack for that purpose and in next section, we will introduce a more flexible and general way to add data layer by layer.

Template chromosome y limits is $[0, 10]$, that's why this hack works

```
## let's remove the NA value
dn.nona <- dn[!is.na(dn$exReg)]
## compute levels based on categories
dn.nona$levels <- as.numeric(factor(dn.nona$exReg))
## do a trick show them at different height
p.ylim <- autoplot(dn.nona, layout = "karyogram", aes(color = exReg, fill = exReg,
  ymin = (levels - 1) * 10/3,
  ymax = levels * 10 / 3))
```

4.3.4 Add more data using layout_karyogram function

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

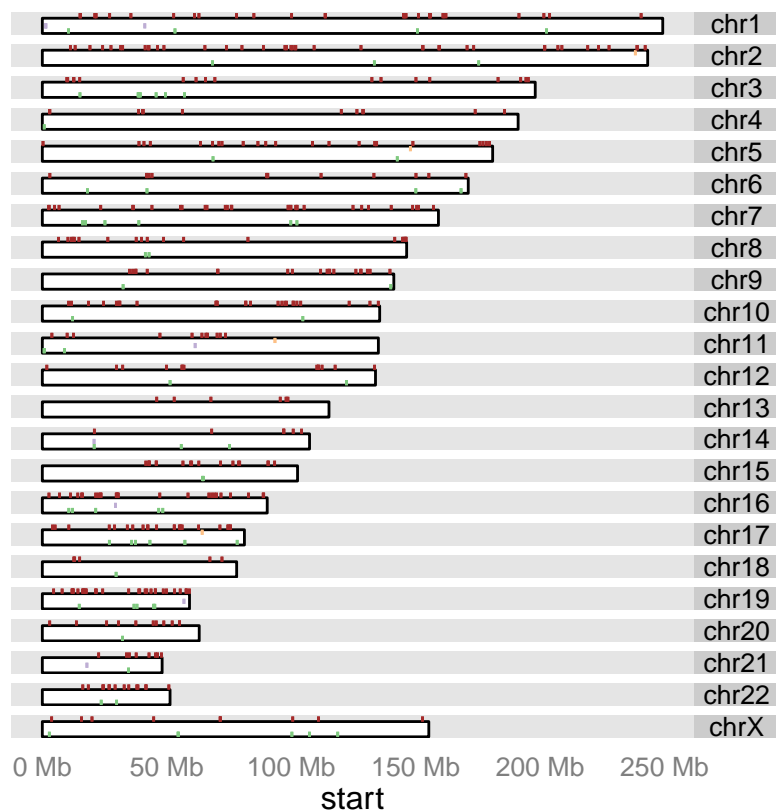
- when you add plots layer by layer, seqnames of different data must be the same to make sure the data are mapped to the same chromosome. For example, if you name chromosome following schema like `chr1` and use just number `1` to name other data, they will be treated as different chromosomes.
- cannot 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.

- 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.

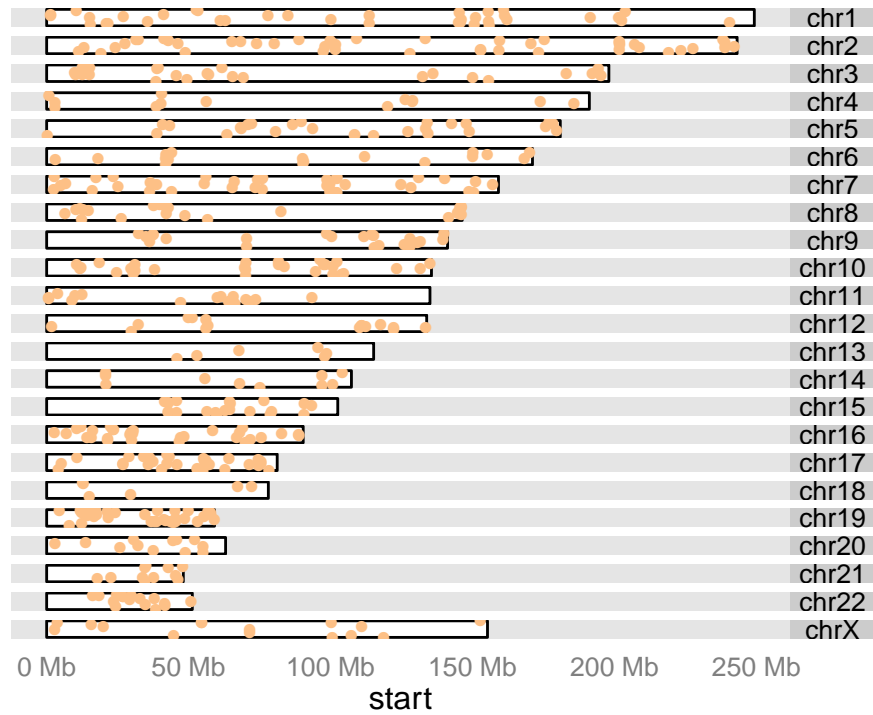
Many times we overlay different datas sets, so let's break down the previous samples into 4 groups and treat them as different data and build them layer by layer, assign the color by hand. You could use `ylim` to control where they are plotted.

```
## prepare the data
dn3 <- dn.nona[dn.nona$exReg == '3']
dn5 <- dn.nona[dn.nona$exReg == '5']
dnC <- dn.nona[dn.nona$exReg == 'C']
dn.na <- dn[is.na(dn$exReg)]
## now we have 4 different data sets
autoplot(seqinfo(dn3), layout = "karyogram") +
  layout_karyogram(data = dn3, geom = "rect", ylim = c(0, 10/3), color = "#7fc97f") +
  layout_karyogram(data = dn5, geom = "rect", ylim = c(10/3, 10/3*2), color = "#beaed4") +
  layout_karyogram(data = dnC, geom = "rect", ylim = c(10/3*2, 10), color = "#fdc086") +
  layout_karyogram(data = dn.na, geom = "rect", ylim = c(10, 10/3*4), color = "brown")
```



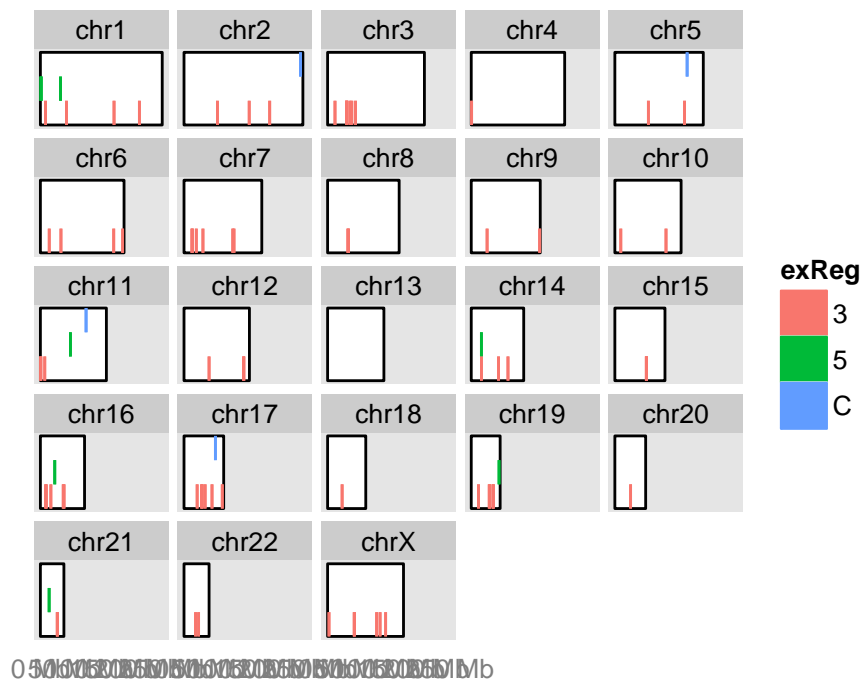
What's more, you could even change the geom for those data


```
dn$pvalue <- runif(length(dn)) * 10  
p <- autoplot(seqinfo(dn)) + layout_karyogram(dn, aes(x = start, y = pvalue),  
  geom = "point", color = "#fdc086")  
p
```



4.3.5 More flexible layout of karyogram

```
p.ylim + facet_wrap(~seqnames)
```



Chapter 5

Link ranges to your data

Plot GRanges object structure and linked to a even spaced parallel coordinates plot which representing the data in elementMetadata.

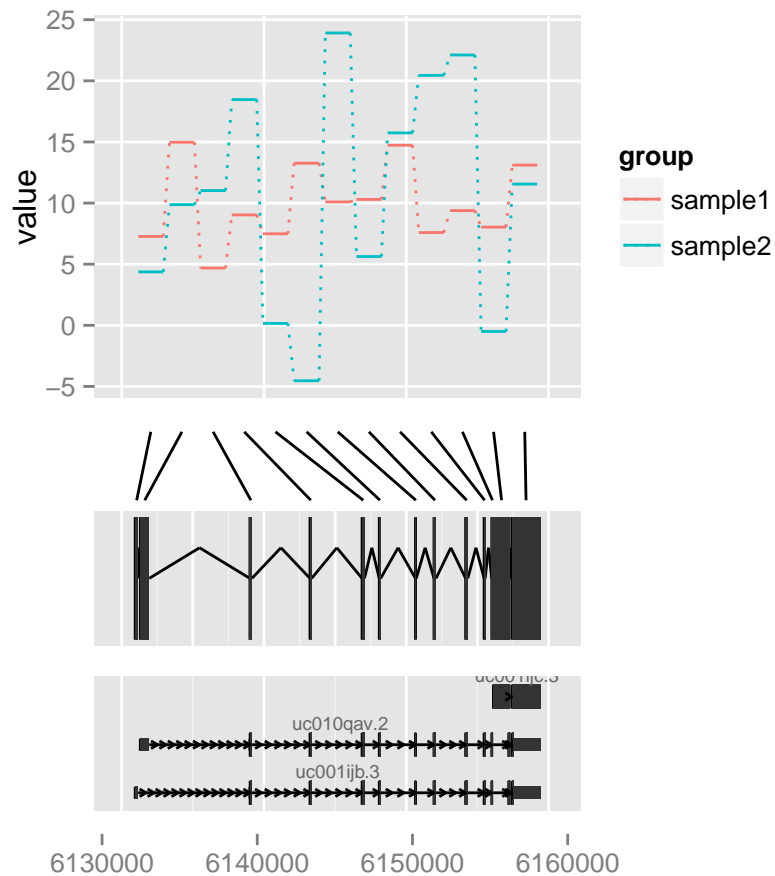
```
library(TxDb.Hsapiens.UCSC.hg19.knownGene)
library(ggbio)
data(genesymbol, package = "biovizBase")
txdb <- TxDb.Hsapiens.UCSC.hg19.knownGene
model <- exonsBy(txdb, by = "tx")
model17 <- subsetByOverlaps(model, genesymbol["RBM17"])
exons <- exons(txdb)
exon17 <- subsetByOverlaps(exons, genesymbol["RBM17"])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

## GRanges with 13 ranges and 4 metadata columns:
##      seqnames      ranges strand |      sample1
##      <Rle>        <IRanges> <Rle> | <numeric>
## [1] chr10 [6130949, 6131156] + | 7.26895725483784
## [2] chr10 [6131309, 6131934] + | 14.9766691947961
## [3] chr10 [6139011, 6139151] + | 4.6851558464563
## [4] chr10 [6143234, 6143350] + | 9.02997654442108
## [5] chr10 [6146894, 6147060] + | 7.48731547091016
## ...      ...      ...      ... ..
## [9] chr10 [6154173, 6154324] + | 14.7316906767153
## [10] chr10 [6155471, 6155544] + | 7.58742745452715
## [11] chr10 [6156012, 6156110] + | 9.38550268352287
## [12] chr10 [6156126, 6157274] + | 8.03439235975778
## [13] chr10 [6157416, 6159422] + | 13.1087718583769
##      sample2      score significant
##      <numeric>      <numeric> <logical>
## [1] 4.36852305658921 -2.96791738482802 0
```

```
## [2] 9.87036450259352 0.101747387012454 0
## [3] 11.0197811445135 -0.597429436326523 0
## [4] 18.4654156968838 1.11616833300151 1
## [5] 0.155813603844239 0.487526566098763 1
## ...
## [9] 15.7429658816936 0.7217784444658207 1
## [10] 20.4460170390669 -1.65781546925978 1
## [11] 22.1135443543151 -1.81953389443168 0
## [12] -0.503535023101866 1.91602272974415 1
## [13] 11.5534416861042 0.468713318813882 1
## ---
## seqlengths:
##          chr1          chr2 ... chrUn_gl000249
## 249250621 243199373 ... 38502
```

Make the plots, you can pass a list of annotation tracks too.

```
p17 <- autoplot(txdb, genesymbol["RBM17"])
plotRangesLinkedToData(exon.new, stat.y = c("sample1", "sample2"), annotation = list(p17))
```



For more information, check the manual.

Chapter 6

Miscellaneous

Every plot object produced by *ggplot2* is essentially a *ggplot2* object, so you could use all the tricks you know with *ggplot2* on *ggbio* plots too, including scales, colors, themes, etc.

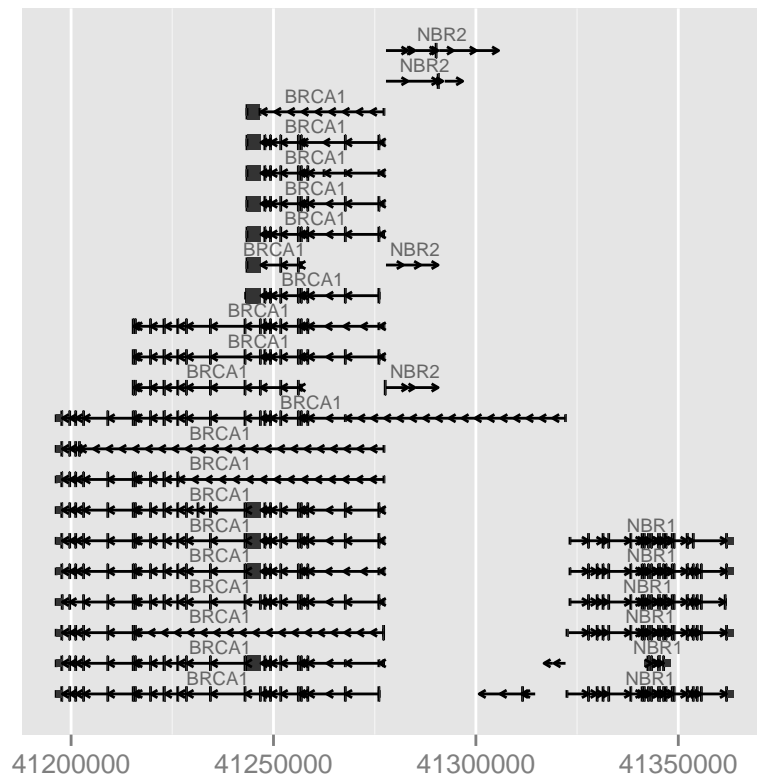
6.1 Themes

In *ggbio*, we developed some more themes to make things easier.

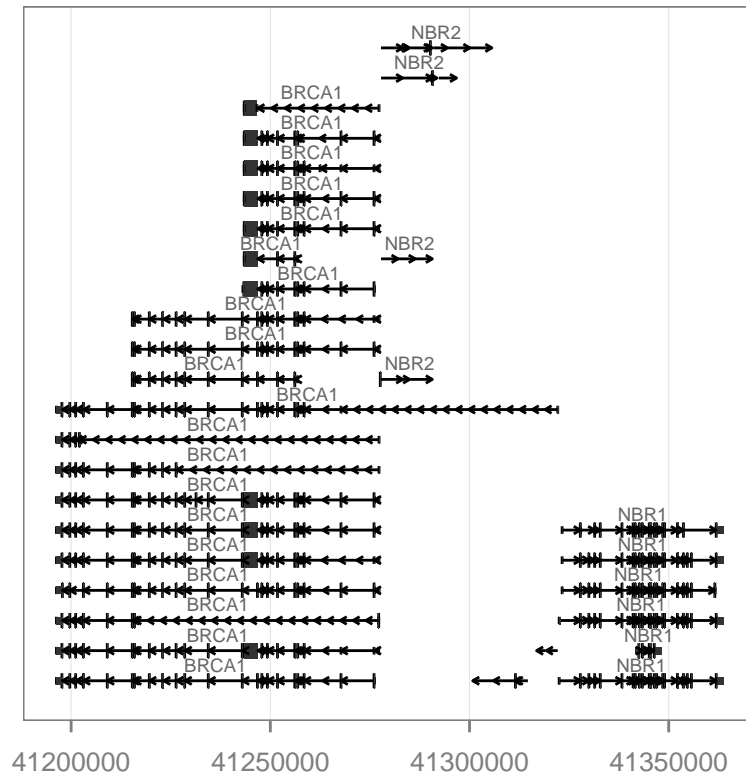
6.1.1 Plot theme

Plot level themes are like any other themes defined in *ggplot2*, simply apply it to a plot.

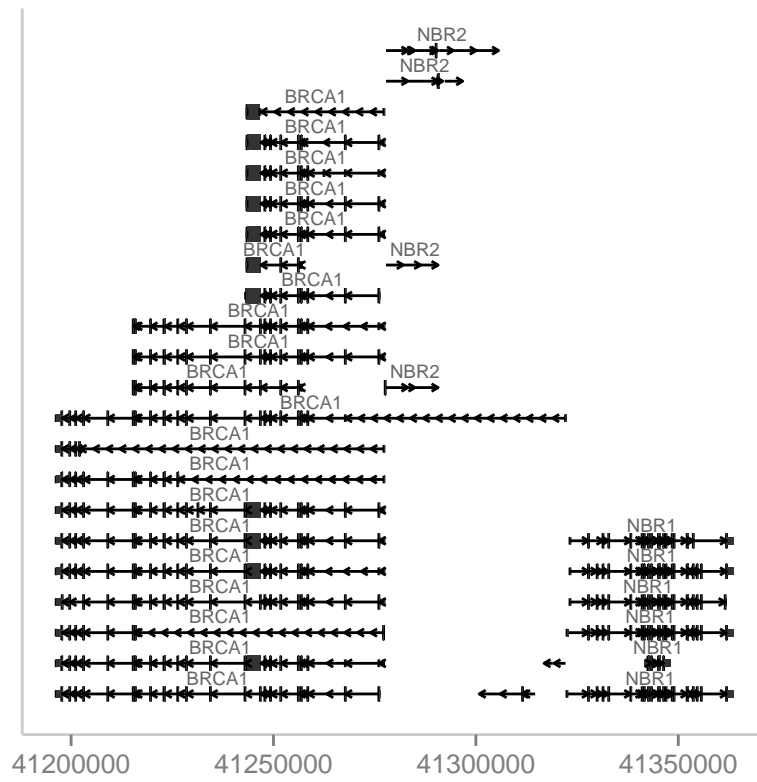
```
p.txdb
```



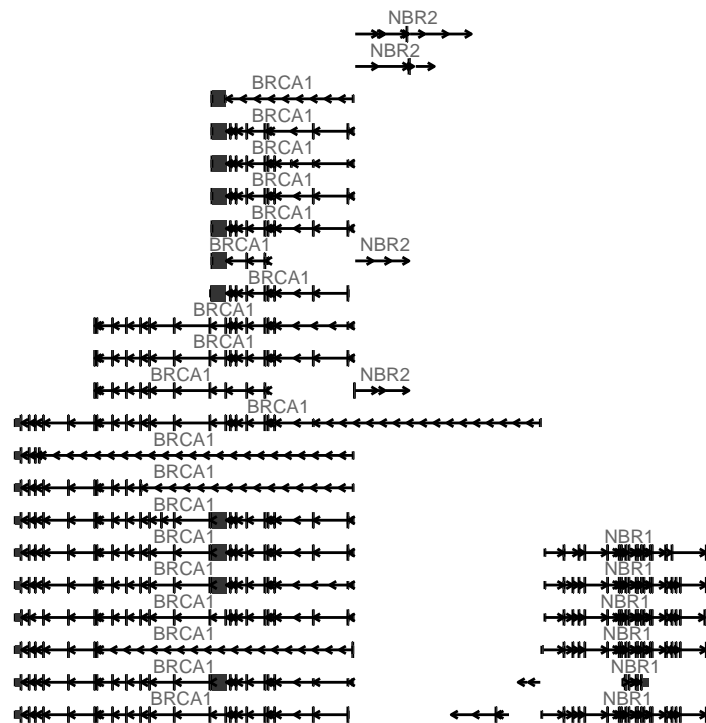
```
p.txdb + theme_alignment()
```



```
p.txdb + theme_clear()
```

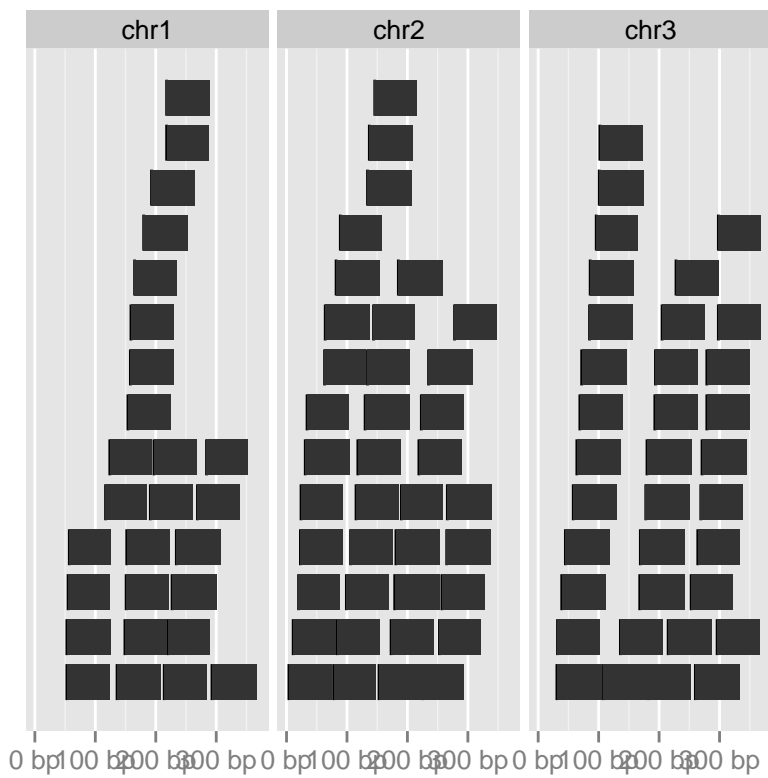


```
p.txdb + theme_null()
```

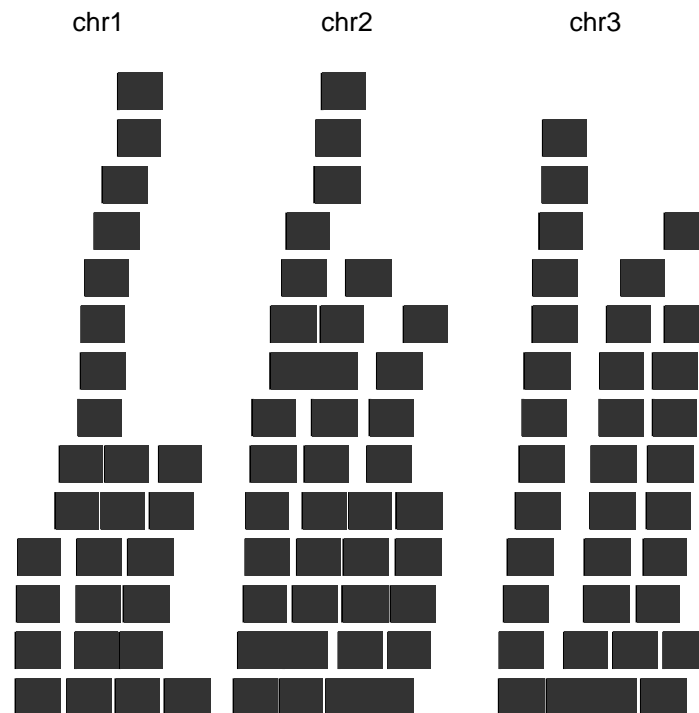



When you have multiple chromosomes encoded in seqnames, you could use `theme_genome` to make a 'fake' linear view of genome coordinates quickly by applying this theme, because it's not equal to chromosome lengths, it's simply

```
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))
seqlengths(gr) <- c(400, 1000, 500)
autoplot(gr)
```



```
autoplot(gr) + theme_genome()
```



6.1.2 Track theme

Track level themes are more complex, it controls whole looking of the tracks, it's essentially a theme object with some attributes controlling the tracks appearance.

See how we make a template, you could customize in the same way

```
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
## }
## <environment: namespace:ggbio>
```

The attributes you could control is basically passed to `tracks()` constructor, including

label.bg.color	character
label.bg.fill	character
label.text.color	character
label.text.cex	numeric
track.plot.color	characterORNULL
track.bg.color	characterORNULL
label.width	unit

Table 6.1: tracks attributes

Chapter 7

Session Information

```
sessionInfo()

## R version 3.1.1 (2014-07-10)
## 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=en_US.UTF-8     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      parallel stats      graphics grDevices utils      datasets
## [8] methods  base
##
## other attached packages:
## [1] VariantAnnotation_1.10.5
## [2] Rsamtools_1.16.1
## [3] BSgenome.Hsapiens.UCSC.hg19_1.3.1000
## [4] BSgenome_1.32.0
## [5] Biostrings_2.32.1
## [6] biovizBase_1.12.3
## [7] XVector_0.4.0
## [8] Homo.sapiens_1.1.2
## [9] TxDb.Hsapiens.UCSC.hg19.knownGene_2.14.0
## [10] org.Hs.eg.db_2.14.0
## [11] GO.db_2.14.0
## [12] RSQLite_0.11.4
## [13] DBI_0.2-7
## [14] OrganismDbi_1.6.0
## [15] GenomicFeatures_1.16.2
## [16] AnnotationDbi_1.26.0
## [17] Biobase_2.24.0
## [18] GenomicRanges_1.16.4
## [19] GenomeInfoDb_1.0.2
## [20] IRanges_1.22.10
```

```
## [21] ggbio_1.12.10
## [22] ggplot2_1.0.0
## [23] BiocGenerics_0.10.0
## [24] knitr_1.6
##
## loaded via a namespace (and not attached):
## [1] BBmisc_1.7          BatchJobs_1.3      BiocParallel_0.6.1
## [4] BiocStyle_1.2.0    Formula_1.1-2      GenomicAlignments_1.0.6
## [7] Hmisc_3.14-4       MASS_7.3-34        RBGL_1.40.1
## [10] RColorBrewer_1.0-5 RCurl_1.95-4.3     Rcpp_0.11.2
## [13] XML_3.98-1.1       biomaRt_2.20.0     bitops_1.0-6
## [16] brew_1.0-6         checkmate_1.3      cluster_1.15.2
## [19] codetools_0.2-9   colorspace_1.2-4   dichromat_2.0-0
## [22] digest_0.6.4      evaluate_0.5.5     fail_1.2
## [25] foreach_1.4.2     formatR_1.0        graph_1.42.0
## [28] gridExtra_0.9.1   gtable_0.1.2       highr_0.3
## [31] iterators_1.0.7   labeling_0.3        lattice_0.20-29
## [34] latticeExtra_0.6-26 munsell_0.4.2      plyr_1.8.1
## [37] proto_0.3-10      reshape2_1.4       rtracklayer_1.24.2
## [40] scales_0.2.4      sendmailR_1.1-2    splines_3.1.1
## [43] stats4_3.1.1     stringr_0.6.2      survival_2.37-7
## [46] tools_3.1.1      zlibbioc_1.10.0
```