CHAPTER 6

OPERATIONS ON GENOMIC INTERVALS AND GENOMIC ARITHMETIC

- What analyses do we use after overlapping intervals of interest with other features of the genome?
- In this chapter we explore ways to deal with genomic intervals

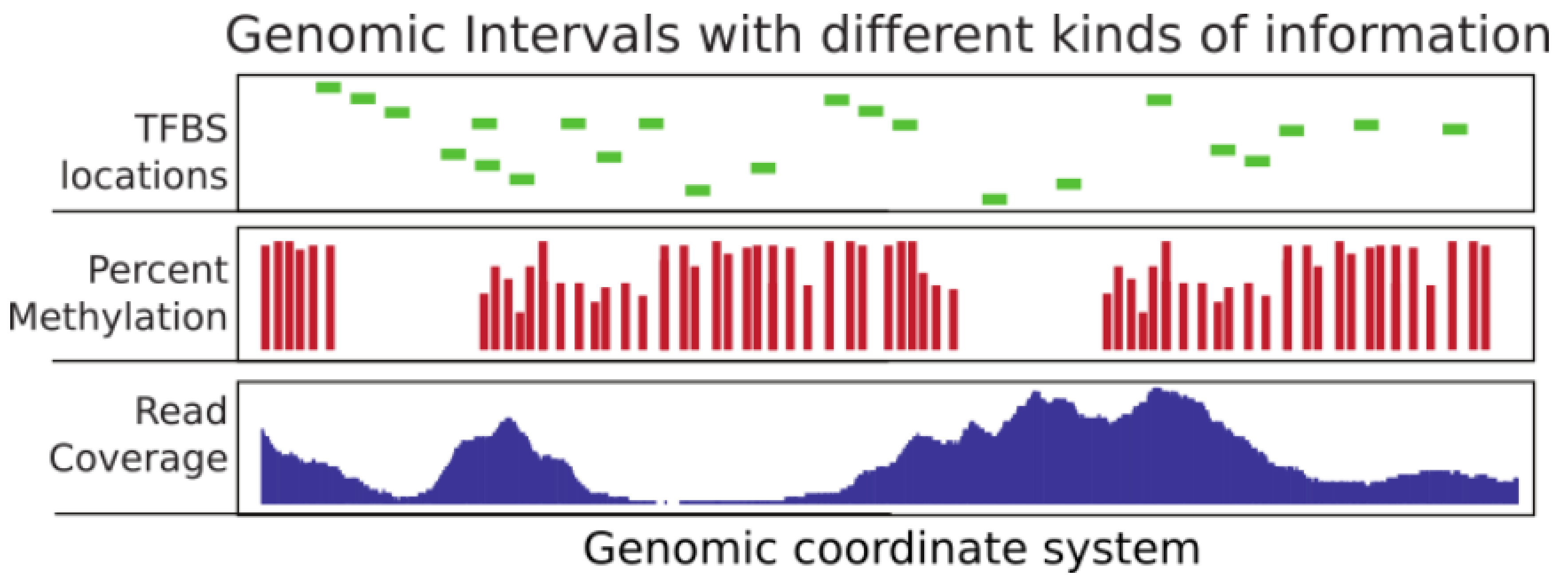
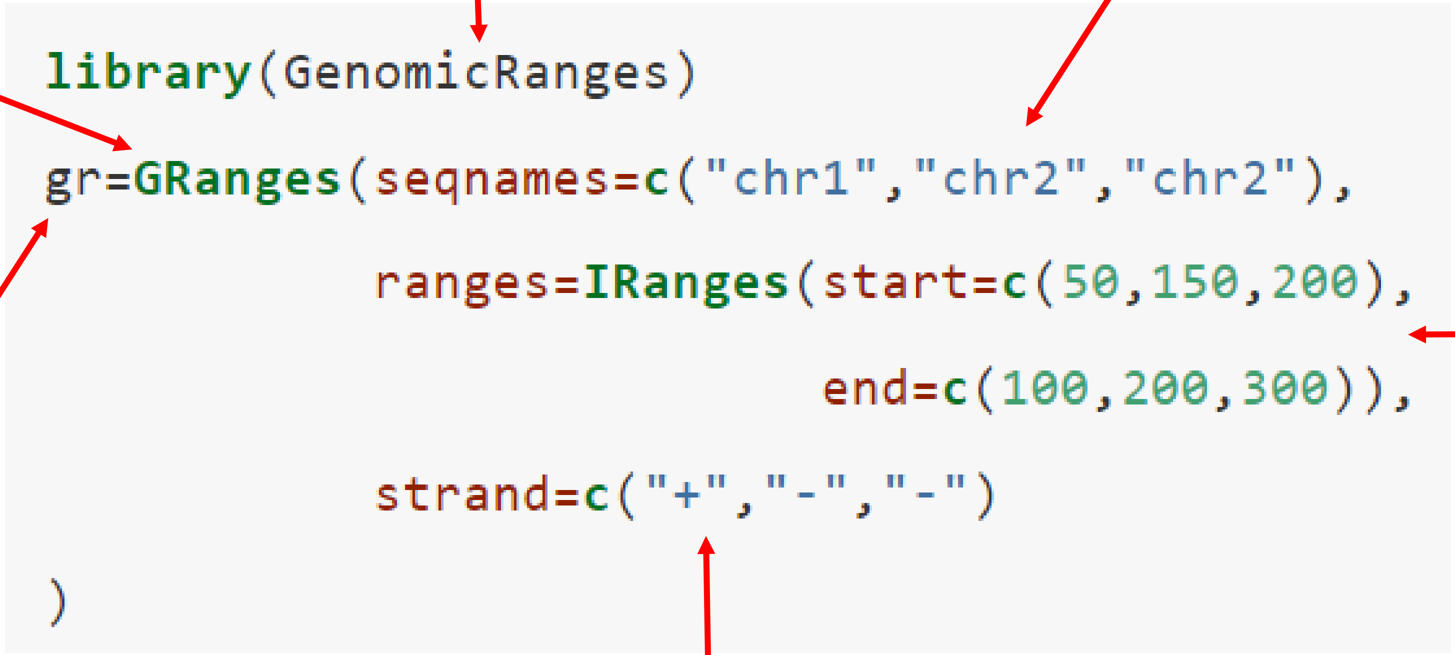

Genomic Intervals with different kinds of information
TFBS locations
Percent Methylation
Read Coverage
Genomic coordinate system

FIGURE 6.1: Summary of genomic intervals with different kinds of information.

6.1 GenomicRanges Package

- We will use this package because it provides tools to do overlap operations
- The package requires specific data types that make overlapping and related operations easier
- These data types are conceptually like a data frame
- Not everything that works for data frames will work on GRanges object

6.1 GenomicRanges Package



```r
library(GenomicRanges)

gr=GRanges(seqnames=c("chr1","chr2","chr2"),

           ranges=IRanges(start=c(50,150,200),

                          end=c(100,200,300)),

           strand=c("+","-","-")

)
```

package

sequence names

argument

main object that holds the genomic intervals and extra information about those intervals

start and end positions of each genomic range

directionality of each genomic range

6.1 GenomicRanges Package

We can include more information about the genomic interval such as scores, names, etc. when we create the object...

```r
gr=GRanges(seqnames=c("chr1","chr2","chr2"),
           ranges=IRanges(start=c(50,150,200),
                          end=c(100,200,300)),
           names=c("id1","id3","id2"),
           scores=c(100,90,50)
)
# or add it later (replaces the existing meta data)
mcols(gr)=DataFrame(name2=c("pax6","meis1","zic4"),
                    score2=c(1,2,3))


gr=GRanges(seqnames=c("chr1","chr2","chr2"),
           ranges=IRanges(start=c(50,150,200),
                          end=c(100,200,300)),
           names=c("id1","id3","id2"),
           scores=c(100,90,50)
)
```

…
or retroactively add it to the gr object

We can also add data using the $ operator:

```r
gr$name3 = c("A","C", "B")
```

6 . 1 GenomicRanges Package

- Ways to convert BED files into a GRanges object:

1. Use the read.table() function to read data into a data frame
2. Follow with the grep() function to remove names with underscores
3. Create a GRanges object

OR

1. Use the readTranscriptfeatures() function

- The rtracklayer package can directly import BED files and obtain the data in the `GRanges` format from online databases
- Many other packages as well

6 . 1 GenomicRanges Package

Frequently used file formats:

BED	The chromosome name, the start position and end position for a genomic feature of interest	Used by the UC Santa Cruz Genome Browser (genome sequence data)
GFF	a tabular text format for genomic features like-- but more flexible than-- BED	Harder to separate into more easily processed components. Many gene annotation files are in this format.
BAM	A compressed and indexed tabular file format designed for aligned sequencing reads	-
SAM	The uncompressed version of the BAM file	Contains basic chromosomal location information, columns related to the quality of alignment, and other relevant information
bigWig (Big Wiggle)	Scores associated with genomic intervals. It is an indexed format.	Easier to query and only necessary portions of the file can be in memory
Generic Text files	Any text file with the minimal information, start and end coordinates	-
Tabix/Bcf	Tabular file formats indexed and compressed like BAM	Mostly used to store genomic variation data such as SNPs and indels

6.1 GenomicRanges Package

- One of the most common tasks in genomics: finding regions that do/do not overlap with another set of regions

- The goal is to understand how different parts of the genome interact and the mechanisms that control genes

- In section 6.1.3, ChIP-seq analysis output files are used to show how to annotate binding sites with CpG islands. Code is provided to find the following:
  - Subset of peaks that overlap
  - Counts of the number of peaks that overlap with a given CpG island
  - One-to-one overlaps between peaks and CpG islands

- CpG islands - regions in the genome with a high density of CpG dinucleotides (a cytosine (C) followed by a guanine (G) nucleotide that are connected by a phosphate bond).

Distances to the nearest transcription start site (TSS) for each peak using the nearest() and distanceToNearest() functions

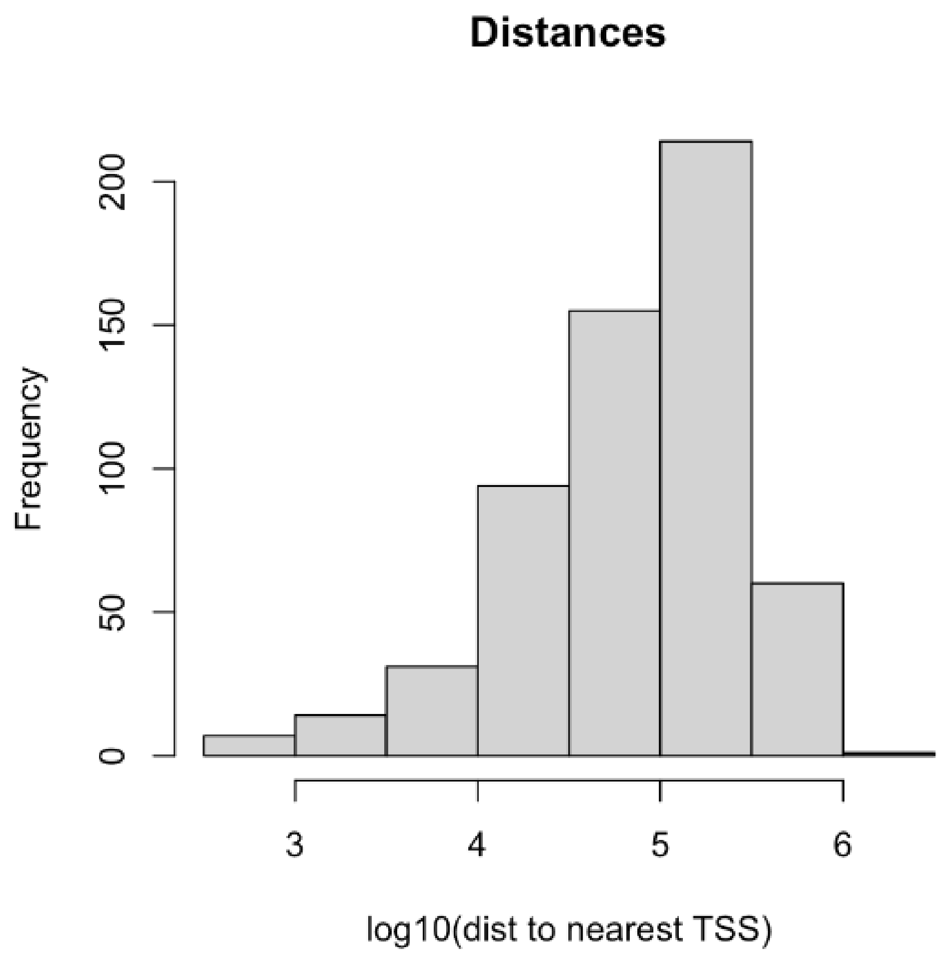


FIGURE 6.2: Histogram of distances of CpG islands to the nearest TSSes.

6 . 2 Mapped high-throughput sequencing reads

- Mapped reads are like genomic intervals stored in a file

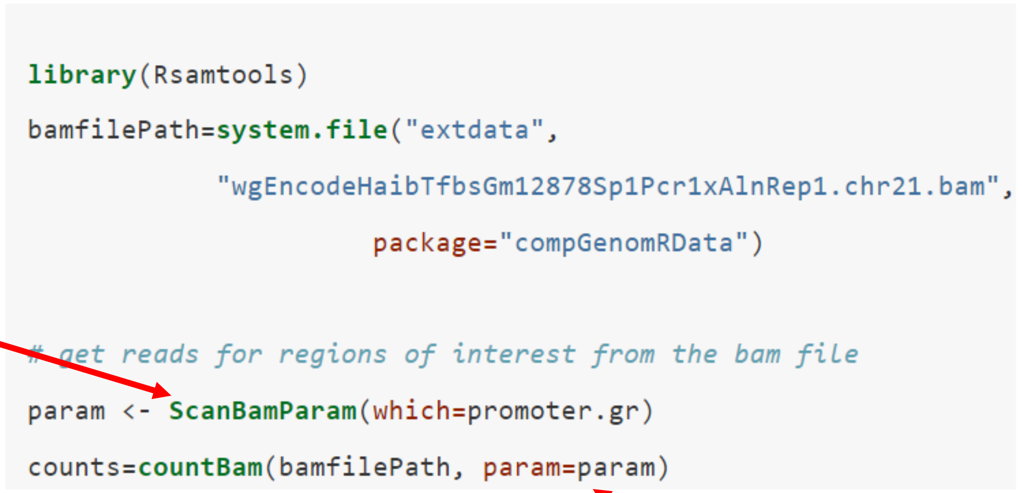
- After mapping, we quantify the enrichment of those aligned reads in the regions of interest (similar to operations on genomic intervals)
    - Enrichment --> the frequency of reads in regions of interest

- We often don't have the memory for all mapped reads (often BAM/SAM), so we use tools to query and quantify alignments on a given set of regions

6.2 Mapped high-throughput sequencing reads

This is one way to count mapped reads for a set of regions using Rsamtools:

Creates object for the param argument with the parameters for scanning the BAM file



```
library(Rsamtools)
bamfilePath=system.file("extdata",
            "wgEncodeHaibTfbsGm12878Sp1Pcr1xAlnRep1.chr21.bam",
                      package="compGenomRData")

# get reads for regions of interest from the bam file
param <- ScanBamParam(which=promoter.gr)
counts=countBam(bamfilePath, param=param)
```

Function that queries the BAM file

ScanBamParam object

6.3 Dealing with continuous scores over the genome

- Most high-throughput data can be viewed as a continuous score/ signal over the bases of the genome
- This sort of data can be stored as a text file or can have special formats like Wig or bigWig (bigWig is good for large amounts of the genome with varying scores)
- In R/Bioconductor, continuous data can also be represented in a compressed format, called Rle (run-length encoded) vector

6.3 Dealing with continuous scores over the genome

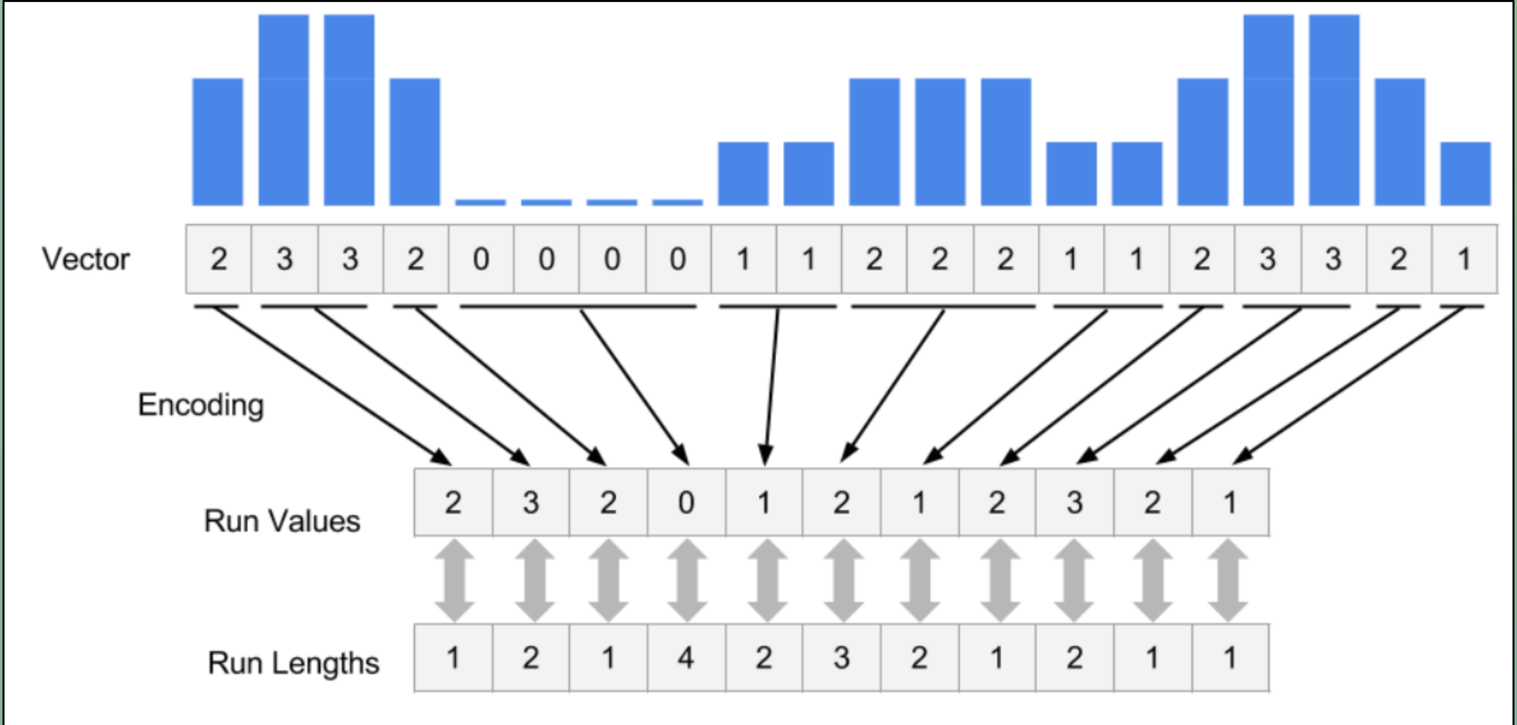


FIGURE 6.3: Rle encoding explained.

Rle vectors have superior memory performance over regular vectors because repeating consecutive values are represented as one value

6.3 Dealing with continuous scores over the genome

- For genome-wide data you sometimes have an RleList object (a list of Rle vectors per chromosome). You can obtain these vectors by getting the coverage:

Use GenomicRanges to get a list of Rle vectors:

```r
covs=coverage(alns) # get coverage vectors
covs
```

Or

Get the coverage directly from a BAM file:

```r
covs=coverage(bamfilePath, param=param) # get coverage vectors
```

- The wig or bigWig format is one of the most common ways of storing score data
- We can extract only the needed information from a bigWig file and read it into R using the rtracklayer package

GRanges object with scores per genomic region

```r
library(rtracklayer)

# File from ENCODE ChIP-seq tracks
bwFile=system.file("extdata",
                      "wgEncodeHaibTfbsA549.chr21.bw",
                      package="compGenomRData")
bw.gr=import(bwFile, which=promoter.gr) # get coverage vectors
```

6.3 Dealing with continuous scores over the genome

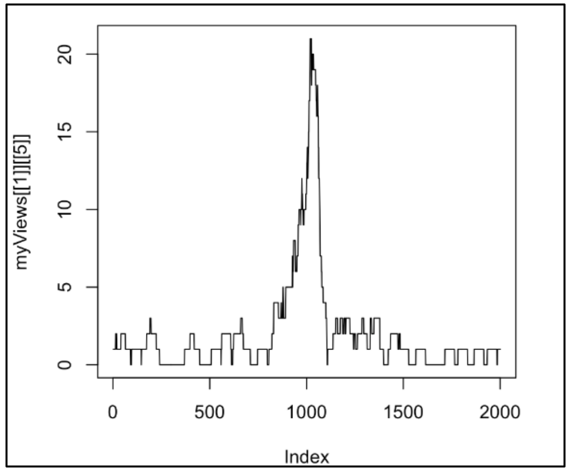
- How do we extract subsections of Rle and RleList objects to visualize or to do stats?
- The book demonstrates by extracting promoter regions from ChIP-seq read coverage:

```
myViews=Views(cov.bw,as(promoter.gr,"IRangesList")) # get subsets of coverage
# there is a views object for each chromosome
```

6.3 Dealing with continuous scores over the genome

- How to plot one of the promoter's coverage values:

```r
# get the coverage vector from the 5th view and plot
plot(myViews[[1]][[5]],type="l")
```



- Next, we see how to apply summary statistics:

```r
# get the mean of the views
head(
  viewMeans(myViews[[1]])
)
```

```r
# get the max of the views
head(
  viewMaxs(myViews[[1]])
)
```

6 . 4 Genomic intervals with more information

- Genomic data often have many layers
    - Sometimes with many tables and each table may have some metadata associated with it
    - Rows and columns might have additional annotations that cannot be contained by their names

- For these cases we can use the SummarizedExperiment class, which can hold multi-layered tabular data

6.4 Genomic intervals with more information

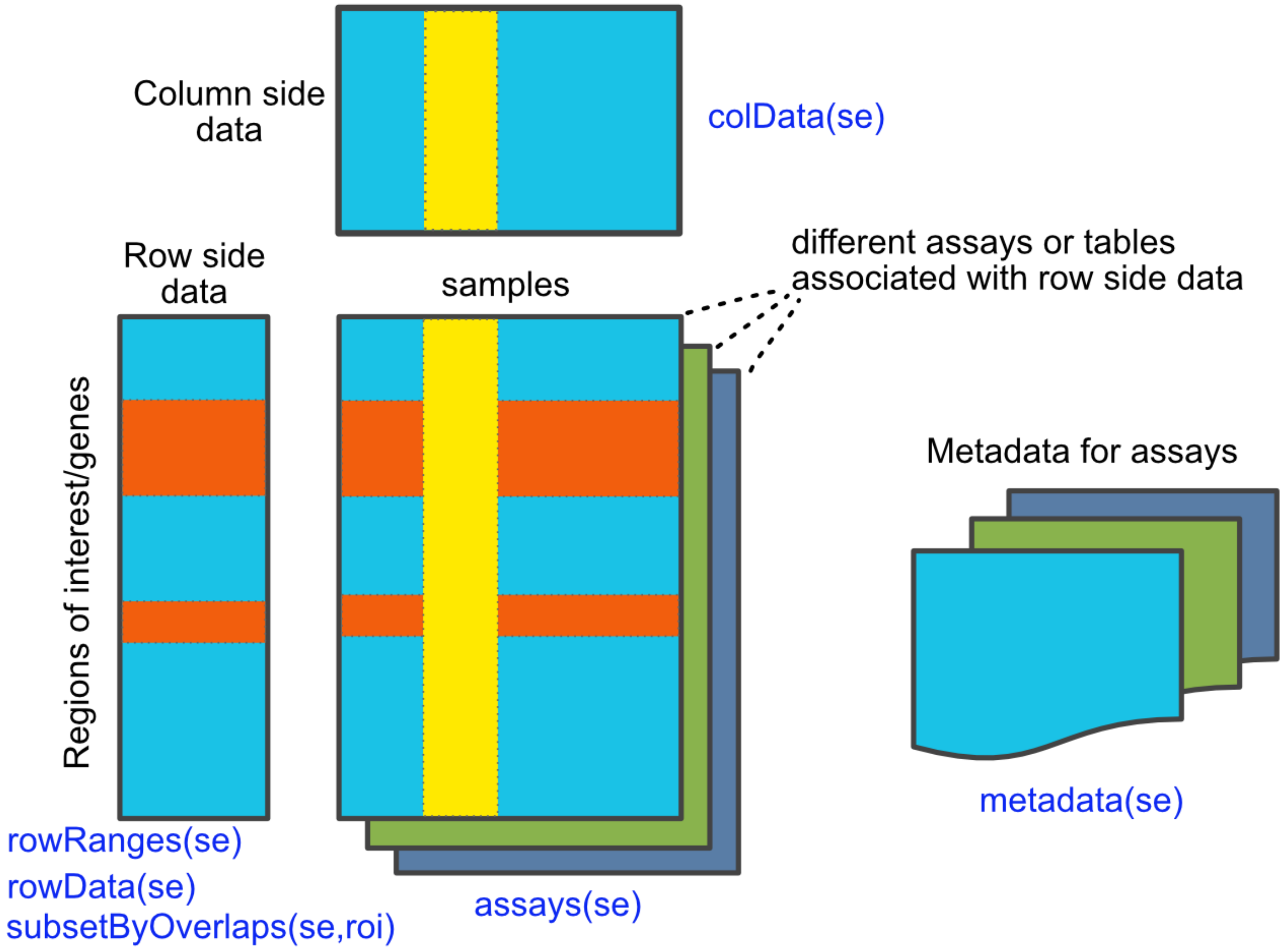


FIGURE 6.5: Overview of SummarizedExperiment class and functions. Adapted from the SummarizedExperiment package vignette.

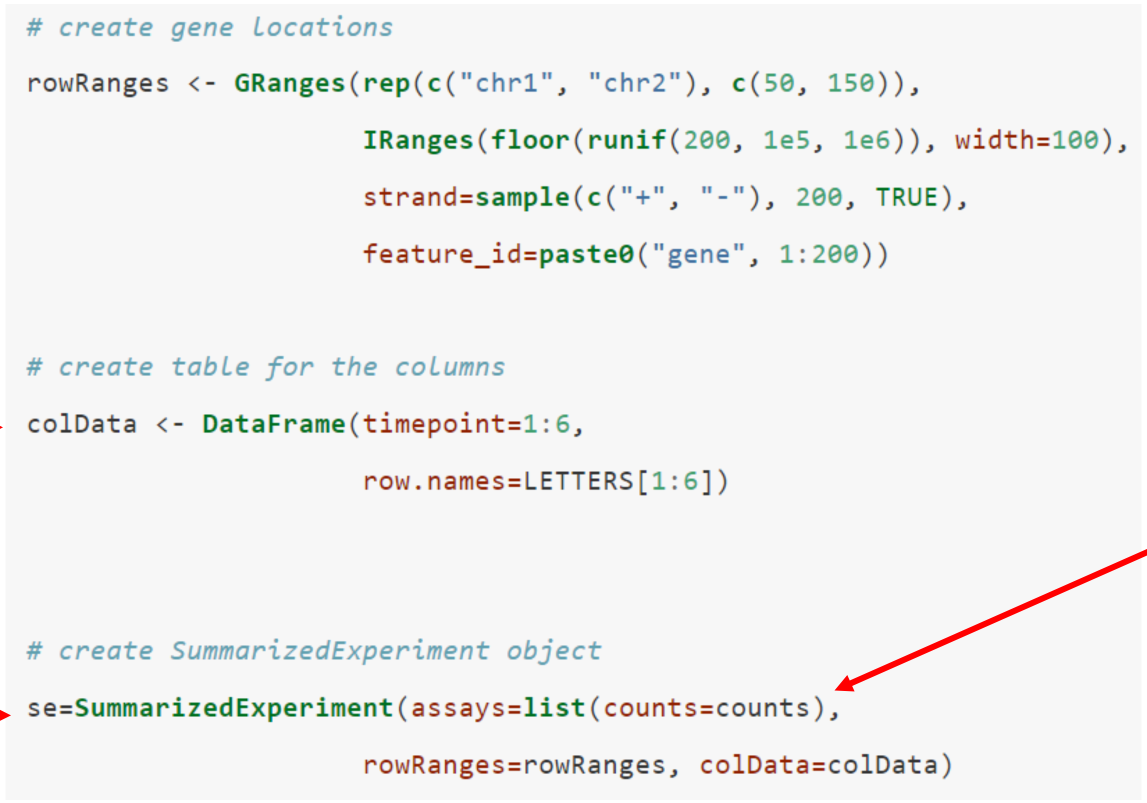
6.4 Genomic intervals with more information

How to create a basic SummarizedExperiment object:

GRanges object representing the locations of the genes

Table for column annotation

SummarizedExperiment object made by combining all pieces



```
# create gene locations
rowRanges <- GRanges(rep(c("chr1", "chr2"), c(50, 150)),
                     IRanges(floor(runif(200, 1e5, 1e6)), width=100),
                     strand=sample(c("+", "-"), 200, TRUE),
                     feature_id=paste0("gene", 1:200))


# create table for the columns
colData <- DataFrame(timepoint=1:6,
                     row.names=LETTERS[1:6])



# create SummarizedExperiment object
se=SummarizedExperiment(assays=list(counts=counts),
                     rowRanges=rowRanges, colData=colData)
```

Matrix of read counts from a series of RNA-seq experiments from different time points

6.4 Genomic intervals with more information

How to subset it and extract/change parts of a SummarizedExperiment object:

1. extract the column-associated and row-associated tables

```
colData(se) # extract column associated data
```

```
rowData(se) # extrac row associated data
```

2. Extract the main table or tables that contain the values of interest

```
assays(se) # extract list of assays
```

```
assays(se)$counts # get the table named "counts"

assays(se)[[1]] # get the first table
```

3. Subset using [ ] notation (similar to the way we subset data frames or matrices)

```
se[1:5, 1:3]
```

4. These objects can also support all of the findOverlaps() methods and associated functions that work on GRanges objects

```
# Subset for only rows which are in chr1:100,000-1,100,000

roi <- GRanges(seqnames="chr1", ranges=100000:1100000)

subsetByOverlaps(se, roi)
```

6.5 Visualizing and summarizing genomic intervals

We are shown some ways to integrate and visualize genomic intervals:

1. Visualizing intervals on a locus of interest
2. Summaries of genomic intervals on multiple loci
3. Making karyograms and circos plots

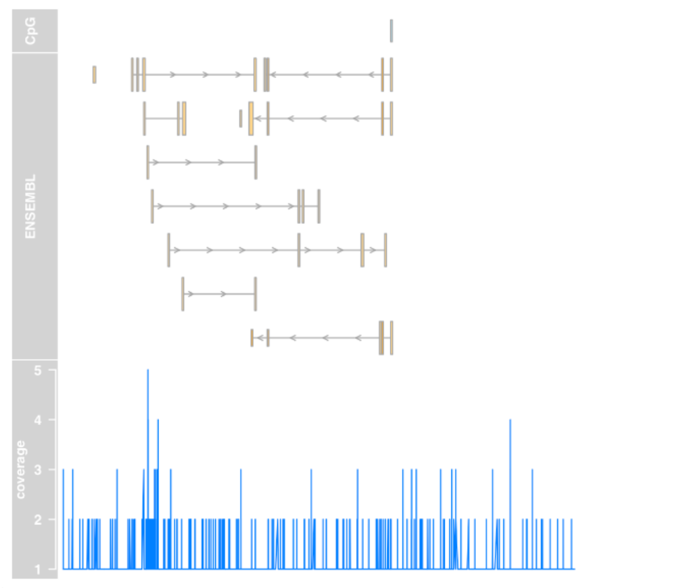


FIGURE 6.6: Genomic data tracks visualized using the Gviz functions.

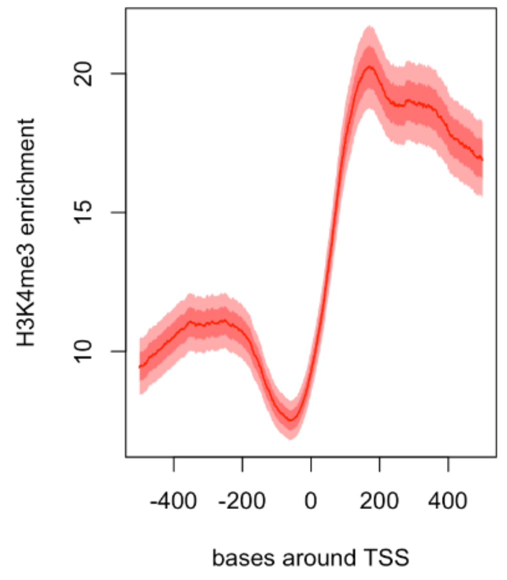


FIGURE 6.7: Meta-region plot using genomation.

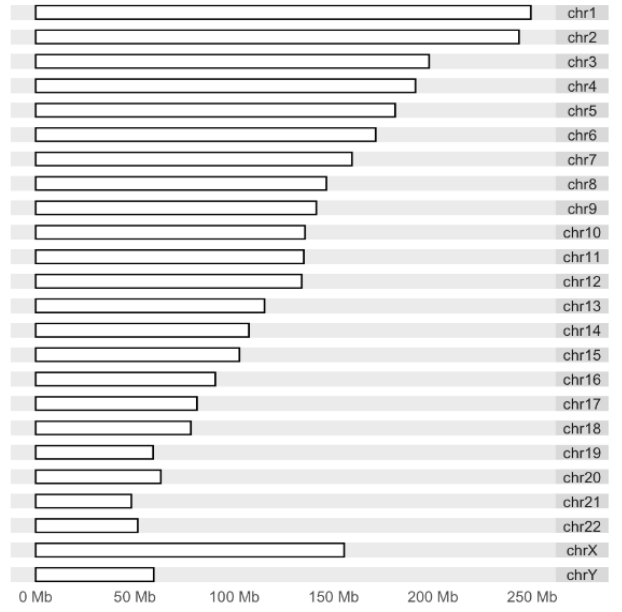
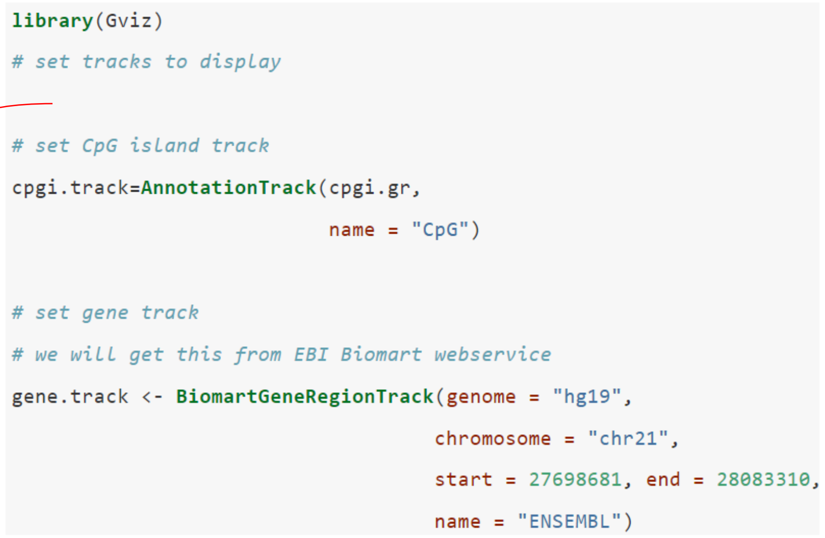


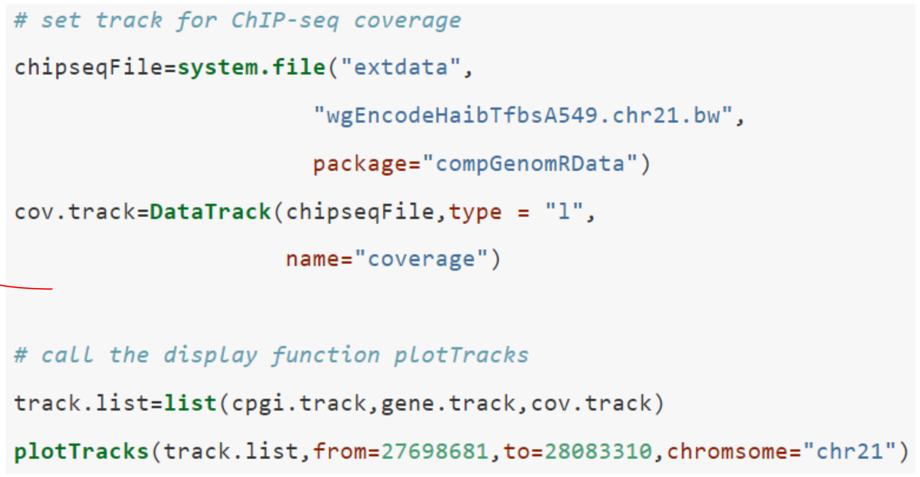
FIGURE 6.11: Karyogram example.

6 . 5 Visualizing and summarizing genomic intervals

How to visualize different genomic datasets over a particular genomic locus using the Gviz package:



```r
library(Gviz)
# set tracks to display

# set CpG island track
cpgi.track=AnnotationTrack(cpgi.gr,
                           name = "CpG")

# set gene track
# we will get this from EBI Biomart webservice
gene.track <- BiomartGeneRegionTrack(genome = "hg19",
                                     chromosome = "chr21",
                                     start = 27698681, end = 28083310,
                                     name = "ENSEMBL")
```



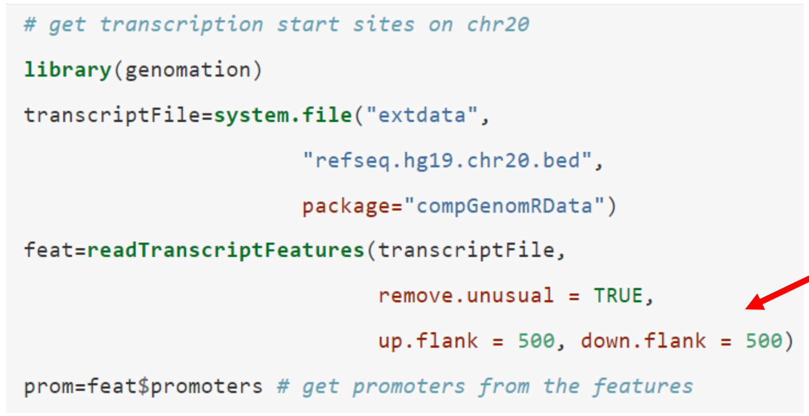
```r
# set track for ChIP-seq coverage
chipseqFile=system.file("extdata",
                      "wgEncodeHaibTfbsA549.chr21.bw",
                      package="compGenomRData")
cov.track=DataTrack(chipseqFile,type = "l",
                    name="coverage")

# call the display function plotTracks
track.list=list(cpgi.track,gene.track,cov.track)
plotTracks(track.list,from=27698681,to=28083310,chromsome="chr21")
```

Tracks (different types of genomic data) to display

Function that displays tracks

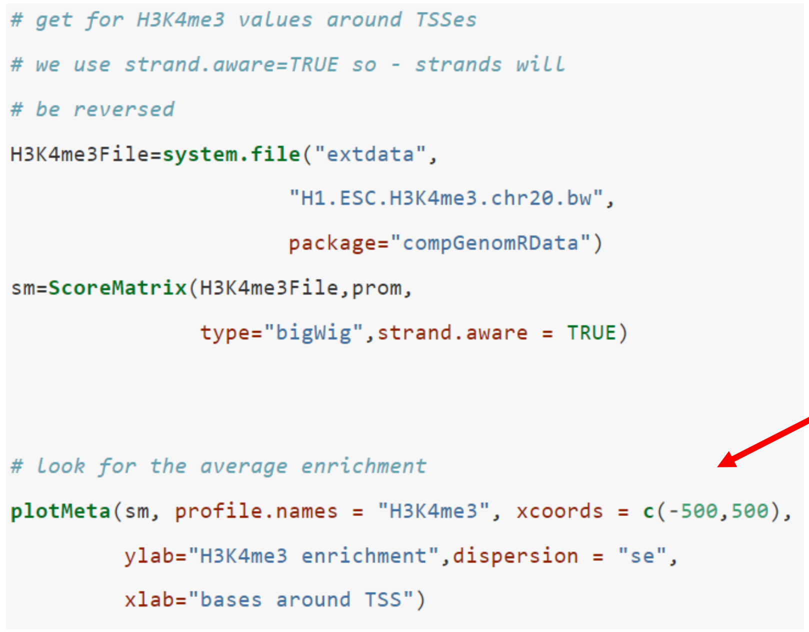
6.5 Visualizing and summarizing genomic intervals

How to visualize & summarize different data sets over many regions of interest & identify patterns using the genomation package:



```r
# get transcription start sites on chr20
library(genomation)
transcriptFile=system.file("extdata",
                           "refseq.hg19.chr20.bed",
                           package="compGenomRData")
feat=readTranscriptFeatures(transcriptFile,
                                 remove.unusual = TRUE,
                                 up.flank = 500, down.flank = 500)
prom=feat$promoters # get promoters from the features
```

Extract region around transcription start sites, 500bp upstream and downstream



```r
# get for H3K4me3 values around TSSes
# we use strand.aware=TRUE so - strands will
# be reversed
H3K4me3File=system.file("extdata",
                        "H1.ESC.H3K4me3.chr20.bw",
                        package="compGenomRData")
sm=ScoreMatrix(H3K4me3File,prom,
               type="bigWig",strand.aware = TRUE)


# look for the average enrichment
plotMeta(sm, profile.names = "H3K4me3", xcoords = c(-500,500),
         ylab="H3K4me3 enrichment",dispersion = "se",
         xlab="bases around TSS")
```

Create a matrix of ChIP-seq scores

Each row represents a region around a specific TSS. Columns are scores per base.

Plot average enrichment values around the TSS

6.5 Visualizing and summarizing genomic intervals

Heatmap where each row is a region around the TSS and color coded by enrichment:

```
heatMatrix(sm,order=TRUE,xcoords = c(-500,500),
             xlab="bases around TSS")
```

~ half of the regions don't have any signal

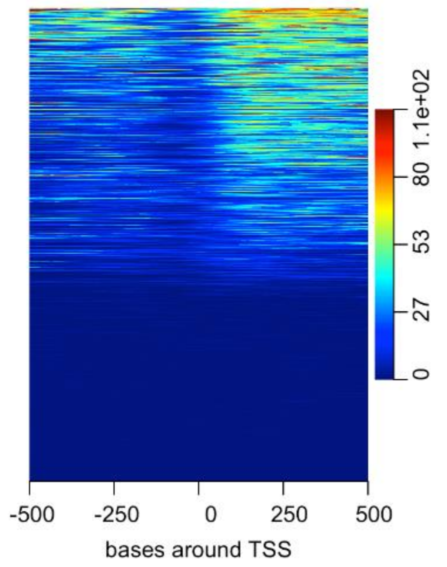


FIGURE 6.8: Heatmap of enrichment of H3K4me2 around the TSS.

Some regions have signal on both sides of the TSS while others have signal mostly on the downstream side.

6.5 Visualizing and summarizing genomic intervals

Making a meta-region plot:

- DNAse-seq data is used to create a list of matrices with the datasets
- The average profile of the signals from both datasets are plotted

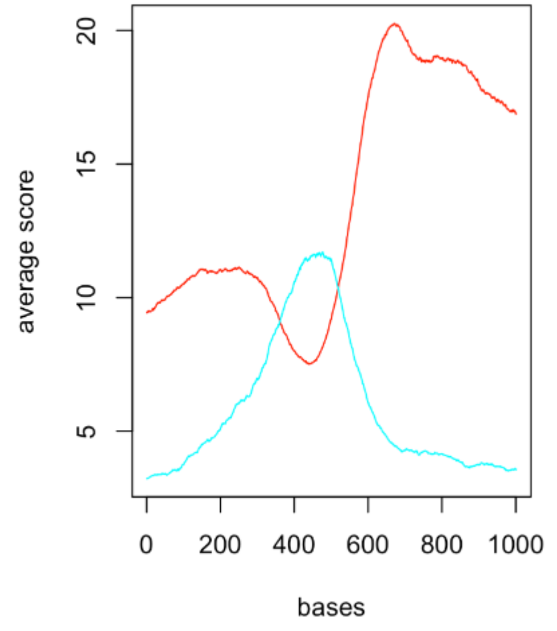


FIGURE 6.9: Average profiles of DNAse and H3K4me3 ChIP-seq.

6.5 Visualizing and summarizing genomic intervals

- We can view the heatmaps for both datasets and cluster the rows based on their similarity (multiHeatMatrix function)
- Next, limit extreme values (winsorize argument) by equalizing scores above the 95th percentile to that value
- About 50% of the promoters lack signal for either dataset, suggesting that the associated genes are not expressed

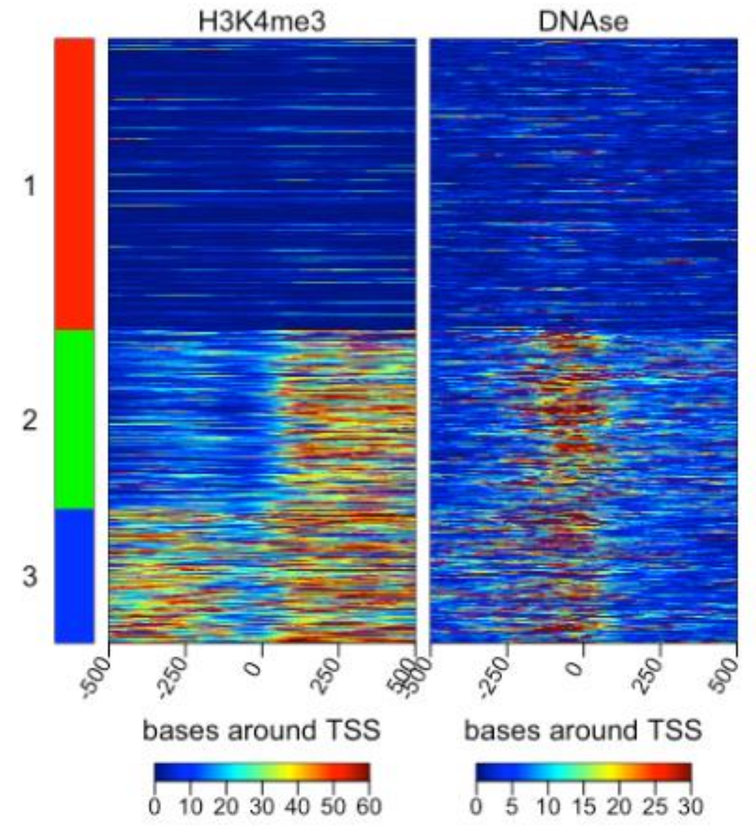


FIGURE 6.10: Heatmaps of H3K4me3 and DNAse data.

6.5 Visualizing and summarizing genomic intervals

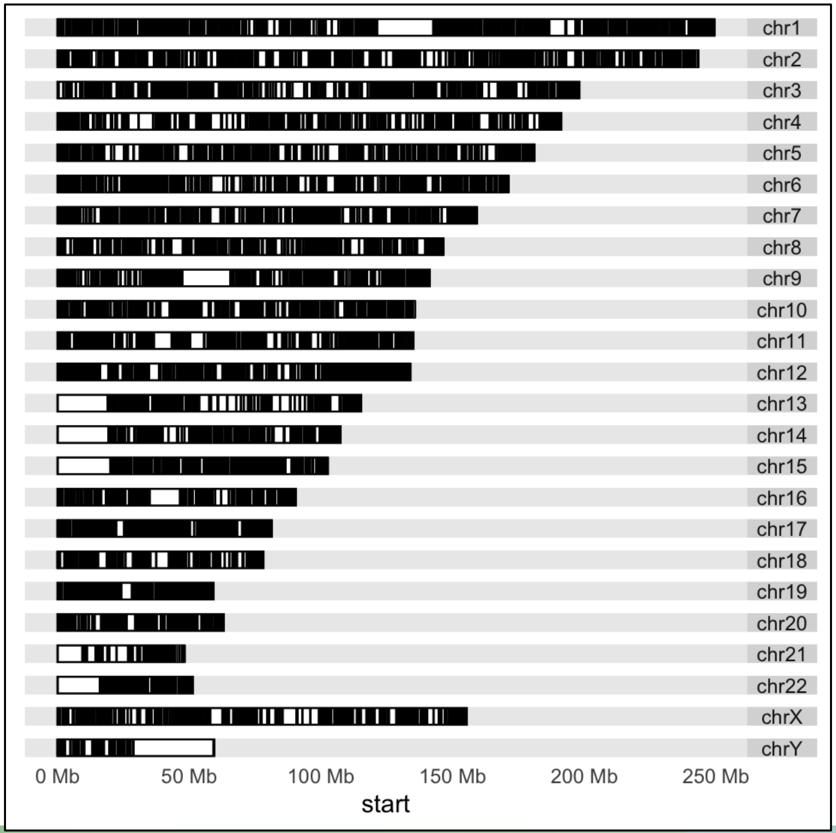
- Chromosomal karyograms and circos plots are good for displaying data over the whole genome of chromosomes of interest
- They are best for showing large trends
- ggbio package for plotting (syntax follows "grammar of graphics" logic, and depends on ggplot2)
- The code on this slide uses the sizes of chromosomes to make a karyogram template

```r
library(ggbio)
data(ideoCyto, package = "biovizBase")
p <- autoplot(seqinfo(ideoCyto$hg19), layout = "karyogram")
```

```r
# read CpG islands from a generic text file

CpGiFile=filePath=system.file("extdata",
                              "CpGi.hg19.table.txt",
                              package="compGenomRData")
cpgi.gr=genomation::readGeneric(CpGiFile,
                  chr = 1, start = 2, end = 3,header=TRUE,
                keep.all.metadata =TRUE,remove.unusual=TRUE )

p + layout_karyogram(cpgi.gr)
```

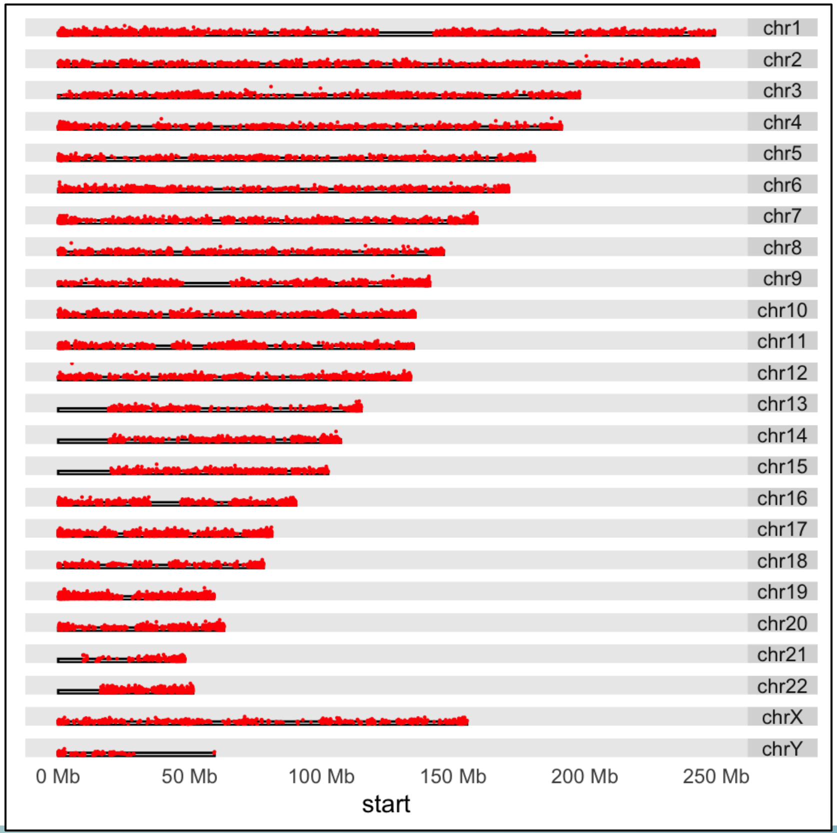
6.5 Visualizing and summarizing genomic intervals

Adding CpG islands to the karyogram using the layout_karyogram() function:



Adding the CpG island scores:

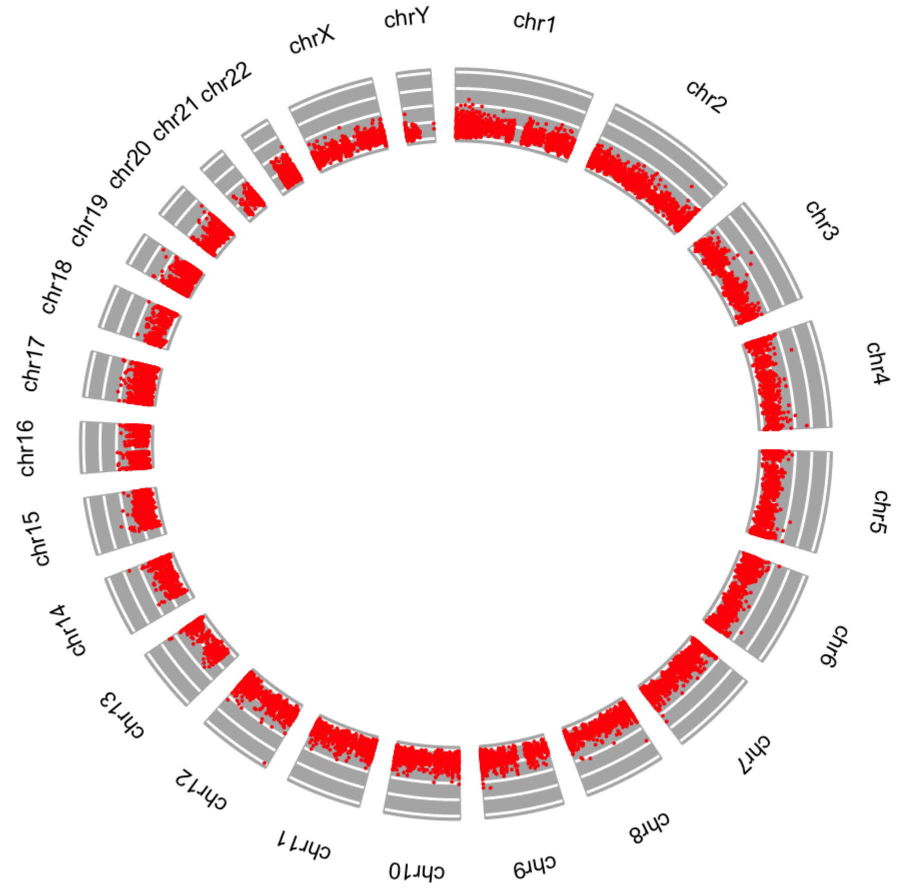
- Plot a point proportional to "obsExp" column from the data
- Squish the chromosomal rectangles and plot on top
- Use aes to map the geometry



6.5 Visualizing and summarizing genomic intervals

- These circular plots are for showing chromosomal rearrangements, but can also be used for depicting signals

- Circos plot using the CpG island score example:


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