DSI Summer Workshops Series

June 21, 2018

Peggy Lindner
Center for Advanced Computing & Data Science (CACDS)
Data Science Institute (DSI)
University of Houston
plindner@uh.edu

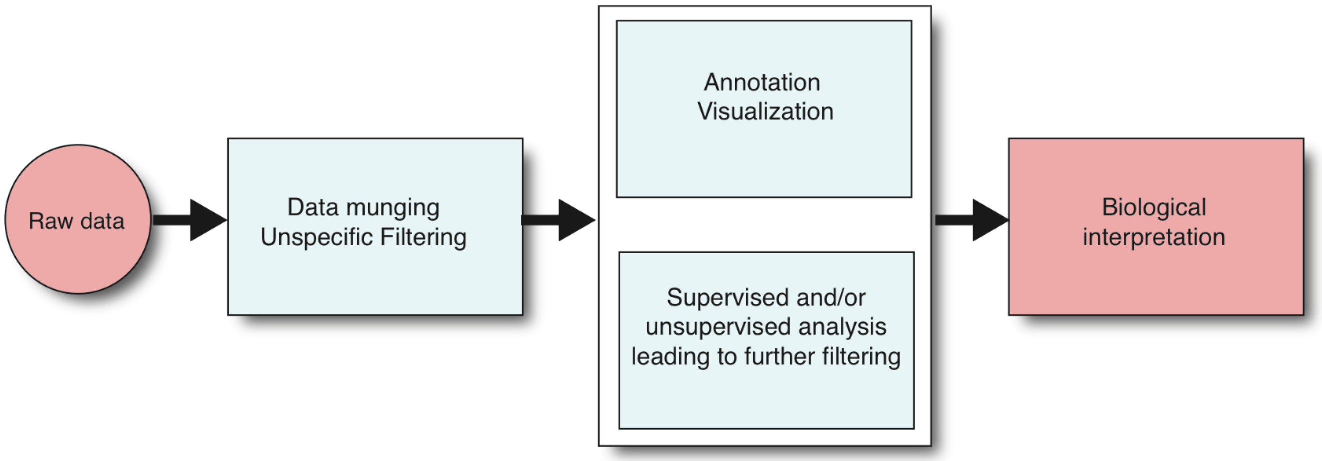
Please make sure you have Jupyterhub running with support for R and all the required packages installed. Data for this and other tutorials can be found in the github repsoitory for the Summer 2018 DSI Workshops https://github.com/peggylind/Materials_Summer2018 (https://github.com/peggylind/Materials_Summer2018)

Computational Genomics with R

Basis understanding of Genomic Data Analysis using R

Goals

- If you are not familiar with R, you will get the basics of R and divide right in to specialized uses of R for computational genomics.
- You will understand genomic intervals and operations on them, such as overlap
- You will be able to retrieve data and explore it



Some R Basics

Packages and functions

In [1]:

```
library(stats)
ls("package:stats") # functions in the package
ls() # objects in your R enviroment
```

```
'acf'   'acf2AR'   'add.scope'   'add1'   'addmargins'
'aggregate'   'aggregate.data.frame'   'aggregate.ts'
'AIC'   'alias'   'anova'   'ansari.test'   'aov'   'approx'
'approxfun'   'ar'   'ar.burg'   'ar.mle'   'ar.ols'   'ar.yw'
'arima'   'arima.sim'   'arima0'   'arima0.diag'
'ARMAacf'   'ARMAtoMA'   'as.dendrogram'   'as.dist'
'as.formula'   'as.hclust'   'as.stepfun'   'as.ts'
'asOneSidedFormula'   'ave'   'bandwidth.kernel'
'bartlett.test'   'BIC'   'binom.test'   'binomial'   'biplot'
'Box.test'   'bw.bcv'   'bw.nrd'   'bw.nrd0'   'bw.SJ'
'bw.ucv'   'C'   'cancor'   'case.names'   'ccf'
'chisq.test'   'cmdscale'   'coef'   'coefficients'
'complete.cases'   'confint'   'confint.default'
'confint.lm'   'constrOptim'   'contr.helmert'   'contr.poly'
'contr.SAS'   'contr.sum'   'contr.treatment'   'contrasts'
'contrasts<-'   'convolve'   'cooks.distance'
'cophenetic'   'cor'   'cor.test'   'cov'   'cov.wt'
'cov2cor'   'covratio'   'cpgram'   'cutree'   'cycle'   'D'
'dbeta'   'dbinom'   'dcauchy'   'dchisq'   'decompose'
'delete.response'   'deltat'   'dendrapply'   'density'
'density.default'   'deriv'   'deriv3'   'deviance'   'dexp'
'df'   'df.kernel'   'df.residual'   'dfbeta'   'dfbetas'
'dffits'   'dgamma'   'dgeom'   'dhyper'   'diffinv'   'dist'
'dlnorm'   'dlogis'   'dmultinom'   'dnbinom'   'dnorm'
'dpois'   'drop.scope'   'drop.terms'   'drop1'
'dsignrank'   'dt'   'dummy.coef'   'dummy.coef.lm'
'dunif'   'dweibull'   'dwilcox'   'ecdf'   'eff.aovlist'
'effects'   'embed'   'end'   'estVar'
'expand.model.frame'   'extractAIC'   'factanal'
'factor.scope'   'family'   'fft'   'filter'   'fisher.test'   'fitted'
'fitted.values'   'fivenum'   'fligner.test'   'formula'
'frequency'   'friedman.test'   'ftable'   'Gamma'
'gaussian'   'get_all_vars'   'getCall'   'getInitial'   'glm'
'glm.control'   'glm.fit'   'hasTsp'   'hat'   'hatvalues'
'hclust'   'heatmap'   'HoltWinters'   'influence'
'influence.measures'   'integrate'   'interaction.plot'
'inverse.gaussian'   'IQR'   'is.empty.model'   'is.leaf'
```

'is.mts' 'is.stepfun' 'is.ts' 'is.tskernel' 'isoreg'
'KalmanForecast' 'KalmanLike' 'KalmanRun'
'KalmanSmooth' 'kernapply' 'kernel' 'kmeans'
'knots' 'kruskal.test' 'ks.test' 'ksmooth' 'lag'
'lag.plot' 'line' 'lm' 'lm.fit' 'lm.influence' 'lm.wfit'
'loadings' 'loess' 'loess.control' 'loess.smooth'
'logLik' 'loglin' 'lowess' 'ls.diag' 'ls.print' 'lsfit'
'mad' 'mahalanobis' 'make.link' 'makeARIMA'
'makepredictcall' 'manova' 'mantelhaen.test'
'mauchly.test' 'mcnemar.test' 'median'
'median.default' 'medpolish' 'model.extract'
'model.frame' 'model.frame.default' 'model.matrix'
'model.matrix.default' 'model.matrix.lm' 'model.offset'
'model.response' 'model.tables' 'model.weights'
'monthplot' 'mood.test' 'mvfft' 'na.action'
'na.contiguous' 'na.exclude' 'na.fail' 'na.omit'
'na.pass' 'napredict' 'naprint' 'naresid' 'nextn'
'nlm' 'nlminb' 'nls' 'nls.control' 'NLSstAsymptotic'
'NLSstClosestX' 'NLSstLfAsymptote'
'NLSstRtAsymptote' 'nobs' 'numericDeriv' 'offset'
'oneway.test' 'optim' 'optimHess' 'optimise'
'optimize' 'order.dendrogram' 'p.adjust'
'p.adjust.methods' 'pacf' 'pairwise.prop.test'
'pairwise.t.test' 'pairwise.table' 'pairwise.wilcox.test'
'pbeta' 'pbinom' 'pbirthday' 'pcauchy' 'pchisq'
'pexp' 'pf' 'pgamma' 'pgeom' 'phyper' 'plclust'
'plnorm' 'plogis' 'plot.ecdf' 'plot.spec.coherency'
'plot.spec.phase' 'plot.stepfun' 'plot.ts' 'pnbinom'
'pnorm' 'poisson' 'poisson.test' 'poly' 'polym'
'power' 'power.anova.test' 'power.prop.test'
'power.t.test' 'PP.test' 'ppoints' 'ppois' 'ppr'
'prcomp' 'predict' 'predict.glm' 'predict.lm'
'preplot' 'princomp' 'printCoefmat' 'profile' 'proj'
'promax' 'prop.test' 'prop.trend.test' 'psignrank'
'pt' 'ptukey' 'punif' 'pweibull' 'pwilcox' 'qbeta'
'qbinom' 'qbirthday' 'qcauchy' 'qchisq' 'qexp' 'qf'
'qgamma' 'qgeom' 'qhyper' 'qlnorm' 'qlogis'

```
'qnbinom'   'qnorm'   'qpois'   'qqline'   'qqnorm'
'qqplot'   'qsignrank'   'qt'   'qtukey'   'quade.test'
'quantile'   'quasi'   'quasibinomial'   'quasipoisson'
'qunif'   'qweibull'   'qwilcox'   'r2dtable'   'rbeta'
'rbinom'   'rcauchy'   'rchisq'   'read.ftable'   'rect.hclust'
'reformulate'   'relevel'   'reorder'   'replications'
'reshape'   'resid'   'residuals'   'residuals.glm'
'residuals.lm'   'rexp'   'rf'   'rgamma'   'rgeom'   'rhyper'
'rlnorm'   'rlogis'   'rmultinom'   'rnbinom'   'rnorm'
'rpois'   'rsignrank'   'rstandard'   'rstudent'   'rt'   'runif'
'runmed'   'rweibull'   'rwilcox'   'rWishart'
'scatter.smooth'   'screeplot'   'sd'   'se.contrast'
'selfStart'   'setNames'   'shapiro.test'   'sigma'
'simulate'   'smooth'   'smooth.spline'   'smoothEnds'
'sortedXyData'   'spec.ar'   'spec.pgram'   'spec.taper'
'spectrum'   'spline'   'splinefun'   'splinefunH'
'SSasymp'   'SSasympOff'   'SSasympOrig'   'SSbiexp'
'SSD'   'SSfol'   'SSfpl'   'SSgompertz'   'SSlogis'
'SSmicmen'   'SSweibull'   'start'   'stat.anova'   'step'
'stepfun'   'stl'   'StructTS'   'summary.aov'
'summary.glm'   'summary.lm'   'summary.manova'
'summary.stepfun'   'supsmu'   'symnum'   't.test'
'termplot'   'terms'   'terms.formula'   'time'   'toeplitz'
'ts'   'ts.intersect'   'ts.plot'   'ts.union'   'tsdiag'   'tsp'
'tsp<-'   'tsSmooth'   'TukeyHSD'   'uniroot'   'update'
'update.default'   'update.formula'   'var'   'var.test'
'variable.names'   'varimax'   'vcov'   'weighted.mean'
'weighted.residuals'   'weights'   'wilcox.test'   'window'
'window<-'   'write.ftable'   'xtabs'
```

In [2]:

```
# get help on hist() function
?hist
help("hist")
# search the word "hist" in help pages
help.search("hist")
??hist
```

Basic Computations in R

In [3]:

```
2 + 3 * 5       # Note the order of operations.
log(10)        # Natural logarithm with base e
5^2            # 5 raised to the second power
3/2            # Division
sqrt(16)      # Square root
abs(3-7)      # Absolute value of 3-7
pi             # The number
exp(2)        # exponential function
# This is a comment line
```

```
17

2.30258509299405

25

1.5

4

4

3.14159265358979

7.38905609893065
```

Data Structures

Vectors

In [4]:

```
x <- c(1, 3, 2, 10, 5)  #create a vector x with 5 components
x
## [1]  1  3  2 10  5
y <- 1:5  #create a vector of consecutive integers y
y + 2  #scalar addition
## [1] 3 4 5 6 7
2 * y  #scalar multiplication
## [1]  2  4  6  8 10
y^2  #raise each component to the second power
## [1]  1  4  9 16 25
2^y  #raise 2 to the first through fifth power
## [1]  2  4  8 16 32
y  #y itself has not been unchanged
## [1] 1 2 3 4 5
y <- y * 2
y  #it is now changed
## [1]  2  4  6  8 10
r1 <- rep(1, 3)  # create a vector of 1s, length 3
length(r1)  #length of the vector
## [1] 3
class(r1)  # class of the vector
## [1] "numeric"
a <- 1  # this is actually a vector length one
```

```
1  3  2  10  5

3  4  5  6  7

2  4  6  8  10

1  4  9  16  25

2  4  8  16  32

1  2  3  4  5

2  4  6  8  10
```

```
3

'numeric'
```

Matrix

In [5]:

```
x <- c(1, 2, 3, 4)
y <- c(4, 5, 6, 7)
m1 <- cbind(x, y)
m1
##      x y
## [1,] 1 4
## [2,] 2 5
## [3,] 3 6
## [4,] 4 7
t(m1)  # transpose of m1
##   [,1] [,2] [,3] [,4]
## x    1    2    3    4
## y    4    5    6    7
dim(m1)  # 2 by 5 matrix
## [1] 4 2
```

x	y
1	4
2	5
3	6
4	7

x	1	2	3	4
y	4	5	6	7

```
4  2
```

Data Frames

```
In [6]:
```

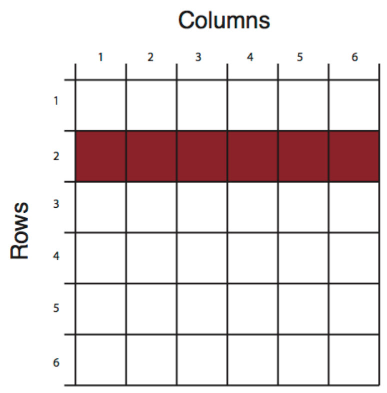
```
chr <- c("chr1", "chr1", "chr2", "chr2")
strand <- c("-","-","+","+")
start<- c(200,4000,100,400)
end<-c(250,410,200,450)
mydata <- data.frame(chr,start,end,strand)
#change column names
names(mydata) <- c("chr","start","end","strand")
mydata # OR this will work too
mydata <- data.frame(chr=chr,start=start,end=end,strand=strand
)
mydata
```

chr	start	end	strand
chr1	200	250	-
chr1	4000	410	-
chr2	100	200	+
chr2	400	450	+

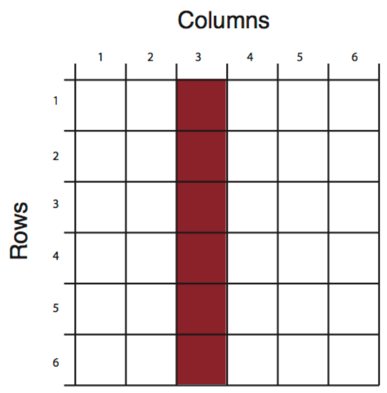
chr	start	end	strand
chr1	200	250	-
chr1	4000	410	-
chr2	100	200	+
chr2	400	450	+

Slicing and Dicing

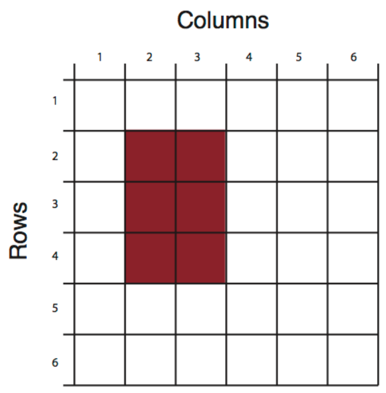
`mat[2,]`



`mat[,3]`



`mat[2:4,2:3]`



In [7]:

```
mydata[,2:4] # columns 2,3,4 of data frame
mydata[,c("chr","start")] # columns chr and start from data fr
ame
mydata$start # variable start in the data frame
mydata[c(1,3),] # get 1st and 3rd rows
mydata[mydata$start>400,] # get all rows where start>400
```

start	end	strand
200	250	-
4000	410	-
100	200	+
400	450	+

chr	start
chr1	200
chr1	4000
chr2	100
chr2	400

```
200   4000   100   400
```

	chr	start	end	strand
1	chr1	200	250	-
3	chr2	100	200	+

	chr	start	end	strand
2	chr1	4000	410	-

List

In [8]:

```
# example of a list with 4 components
# a string, a numeric vector, a matrix, and a scalar
w <- list(name="Fred",
       mynumbers=c(1,2,3),
       mymatrix=matrix(1:4,ncol=2),
       age=5.3)
w
```

$name

'Fred'

$mynumbers

```
    1  2  3
```

$mymatrix

1	3
2	4

$age

5.3

In [9]:

```
w[[3]] # 3rd component of the list
w[["mynumbers"]] # component named mynumbers in list
w$age
```

1	3
2	4

```
    1  2  3
```

5.3

Factors

In [10]:

```
features=c("promoter","exon","intron")
f.feat=factor(features)
```

Data types

- numeric
- logical
- character
- integer

```
In [11]:
```

```r
#create a numeric vector x with 5 components
x<-c(1,3,2,10,5)
x
#create a logical vector x
x<-c(TRUE,FALSE,TRUE)
x
# create a character vector
x<-c("sds","sd","as")
x
class(x)
# create an integer vector
x<-c(1L,2L,3L)
x
class(x)
```

```
    1   3   2   10   5

    TRUE   FALSE   TRUE

    'sds'   'sd'   'as'

'character'

    1   2   3

'integer'
```

Reading and Writing Data

Most of the genomics data sets are in the form of genomic intervals associated with a score. That means mostly the data will be in table format with columns denoting chromosome, start positions, end positions, strand and score. One of the popular formats is BED format used primarily by UCSC genome browser but most other genome browsers and tools will support BED format. We have all the annotation data in BED format. In R, you can easily read tabular format data with read.table() function.

In [12]:

```
enh.df <- read.table("dataJune21th/subset.enhancers.hg18.bed",
 header = FALSE)  # read enhancer marker BED file
cpgi.df <- read.table("dataJune21th/subset.cpgi.hg18.bed", hea
der = FALSE) # read CpG island BED file
# check first lines to see how the data looks like
head(enh.df)
head(cpgi.df)
```

V1	V2	V3	V4	V5	V6	V7	V8	V9
chr20	266275	267925	.	1000	.	9.11	13.1693	-1
chr20	287400	294500	.	1000	.	10.53	13.0231	-1
chr20	300500	302500	.	1000	.	9.10	13.3935	-1
chr20	330400	331800	.	1000	.	6.39	13.5105	-1
chr20	341425	343400	.	1000	.	6.20	12.9852	-1
chr20	437975	439900	.	1000	.	6.31	13.5184	-1

V1	V2	V3	V4
chr20	195575	195851	CpG:_28
chr20	207789	208148	CpG:_32
chr20	219055	219437	CpG:_33
chr20	225831	227155	CpG:_135
chr20	252826	256323	CpG:_286
chr20	275376	276977	CpG:_116

In [13]:

```
write.table(cpgi.df,file="cpgi.txt",quote=FALSE,
            row.names=FALSE,col.names=FALSE,sep="\t")
```

```
In [14]:
```

```
save(cpgi.df,enh.df,file="mydata.RData")
load("mydata.RData")
# saveRDS() can save one object at a type
saveRDS(cpgi.df,file="cpgi.rds")
x=readRDS("cpgi.rds")
head(x)
```

V1	V2	V3	V4
chr20	195575	195851	CpG:_28
chr20	207789	208148	CpG:_32
chr20	219055	219437	CpG:_33
chr20	225831	227155	CpG:_135
chr20	252826	256323	CpG:_286
chr20	275376	276977	CpG:_116

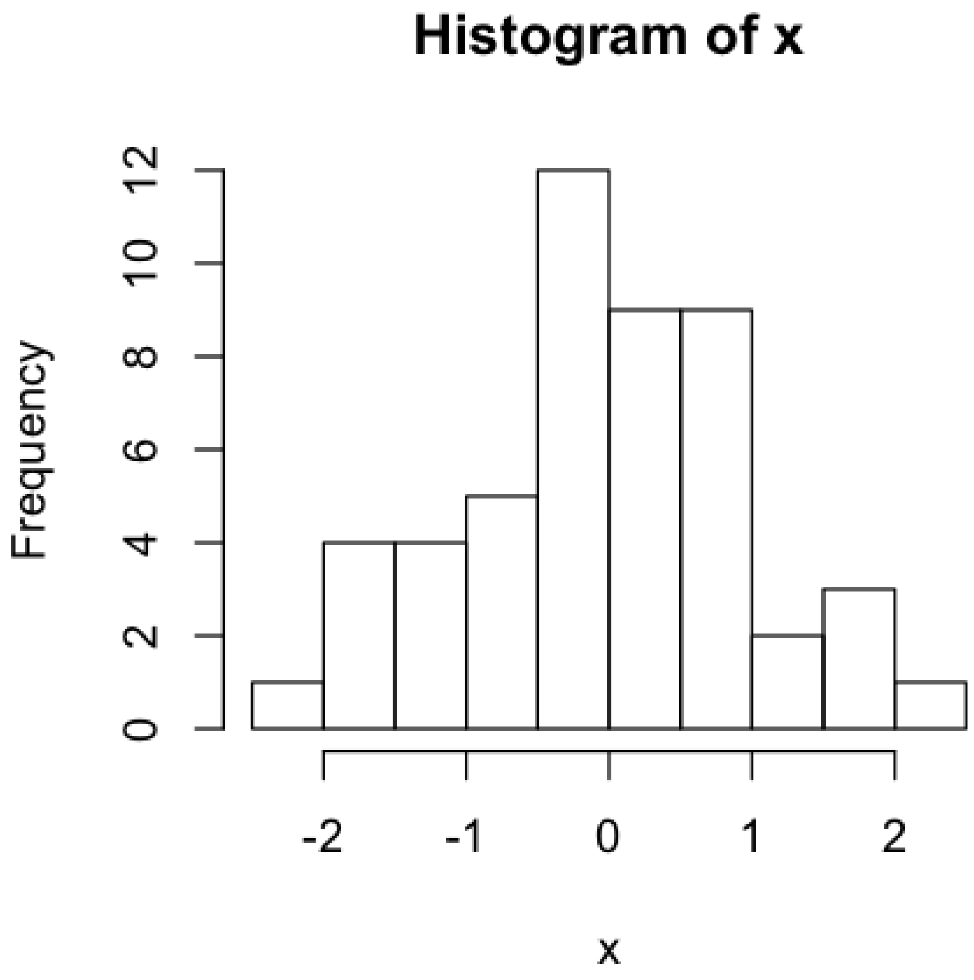
One important thing is that with save() you can save many objects at a time and when they are loaded into memory with load() they retain their variable names. For example, in the above code when you use load("mydata.RData") in a fresh R session, an object names “cpg.df” will be created. That means you have to figure out what name you gave it to the objects before saving them. On the contrary to that, when you save an object by saveRDS() and read by readRDS() the name of the object is not retained, you need to assign the output of readRDS() to a new variable (“x” in the above code chunk).

Plotting in R

Let us sample 50 values from normal distribution and do some plots.

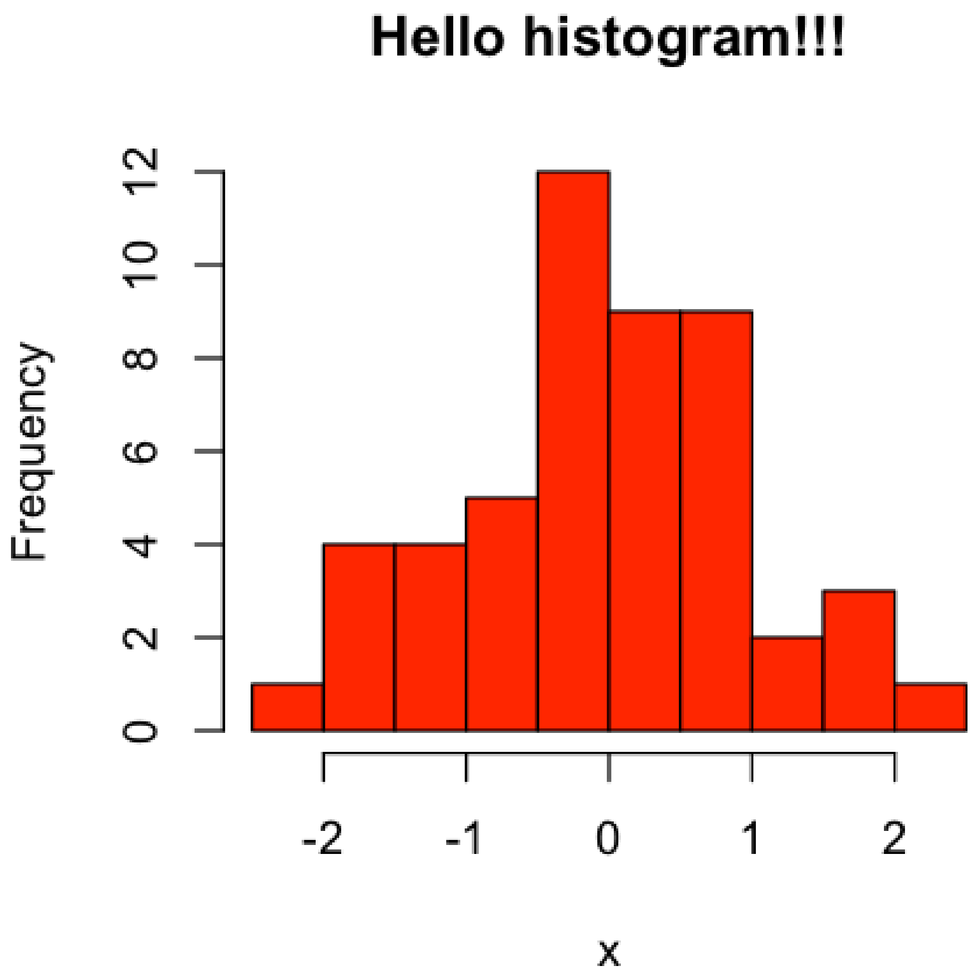
In [15]:

```
# setting figure size in notebook
options(repr.plot.width = 4, repr.plot.height = 4)
# sample 50 values from normal distribution
# and store them in vector x
x<-rnorm(50)
hist(x) # plot the histogram of those values
```



In [16]:

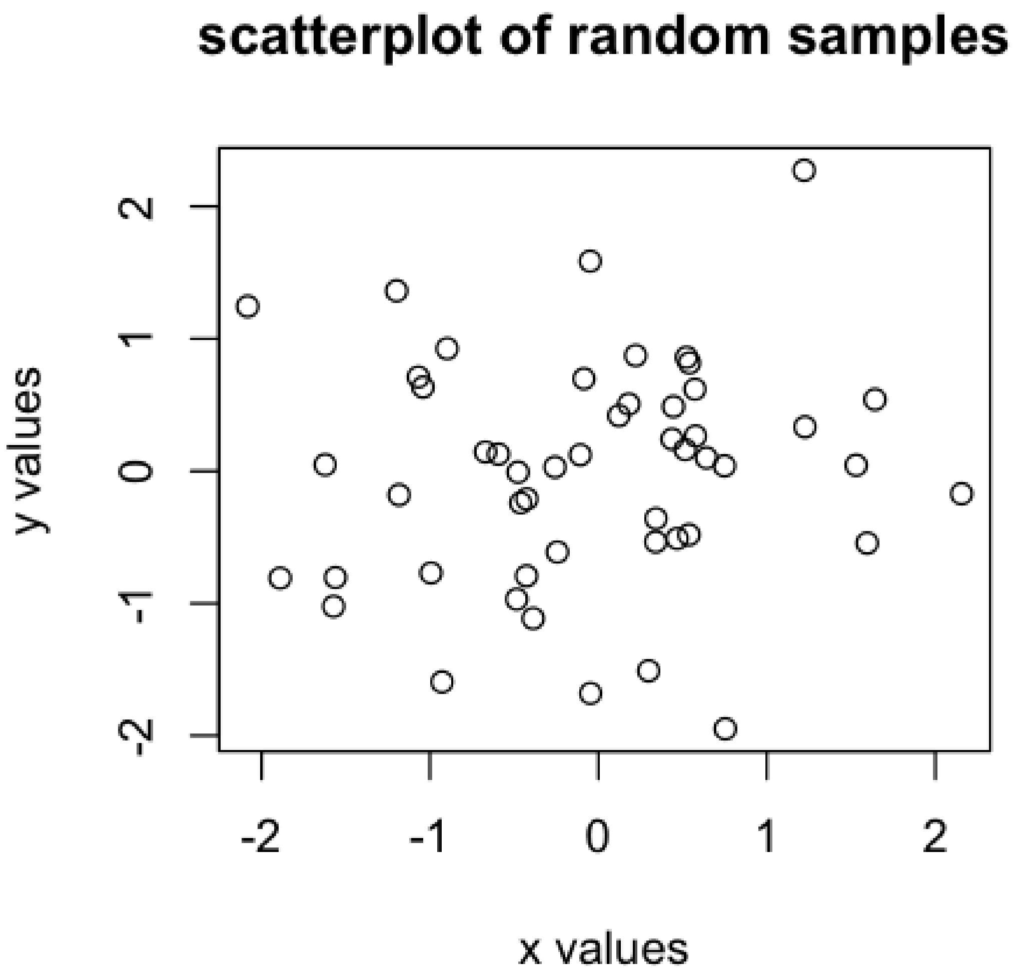
```
#let's add a title and change the color
hist(x,main="Hello histogram!!!",col="red")
```



Scatterplot

In [17]:

```
# randomly sample 50 points from normal distribution
y<-rnorm(50)
#plot a scatter plot
# control x-axis and y-axis labels
plot(x,y,main="scatterplot of random samples",
        ylab="y values",xlab="x values")
```



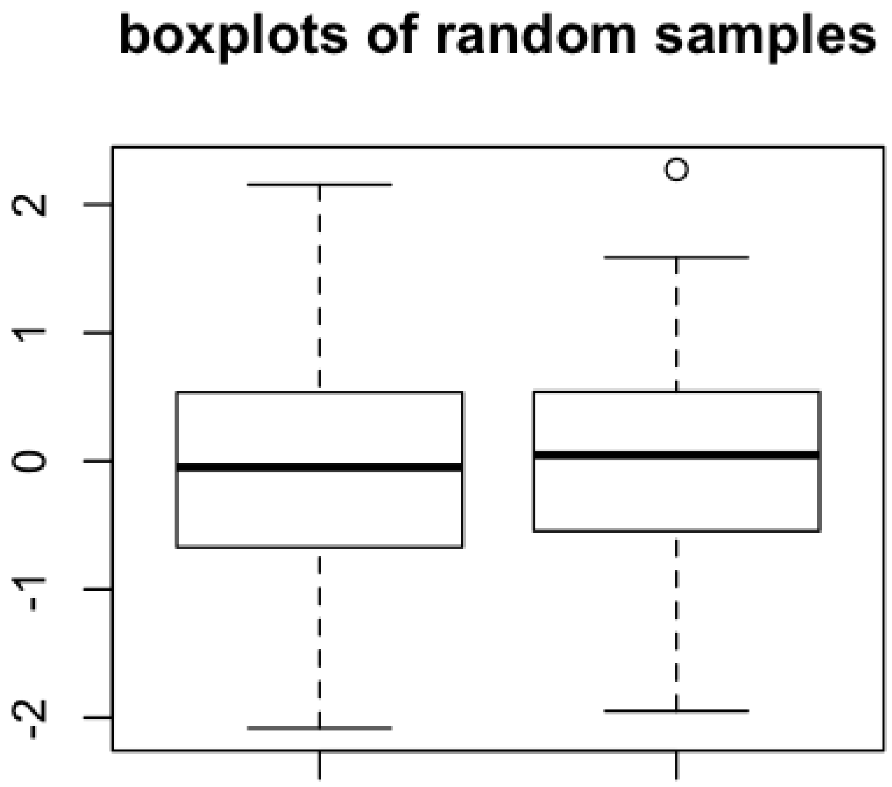
Boxplot

lowerWhisker=Q1-1.5[IQR] and upperWhisker=Q1+1.5*[IQR]

In addition, outliers can be depicted as dots. In this case, outliers are the values that remain outside the whiskers.

In [18]:

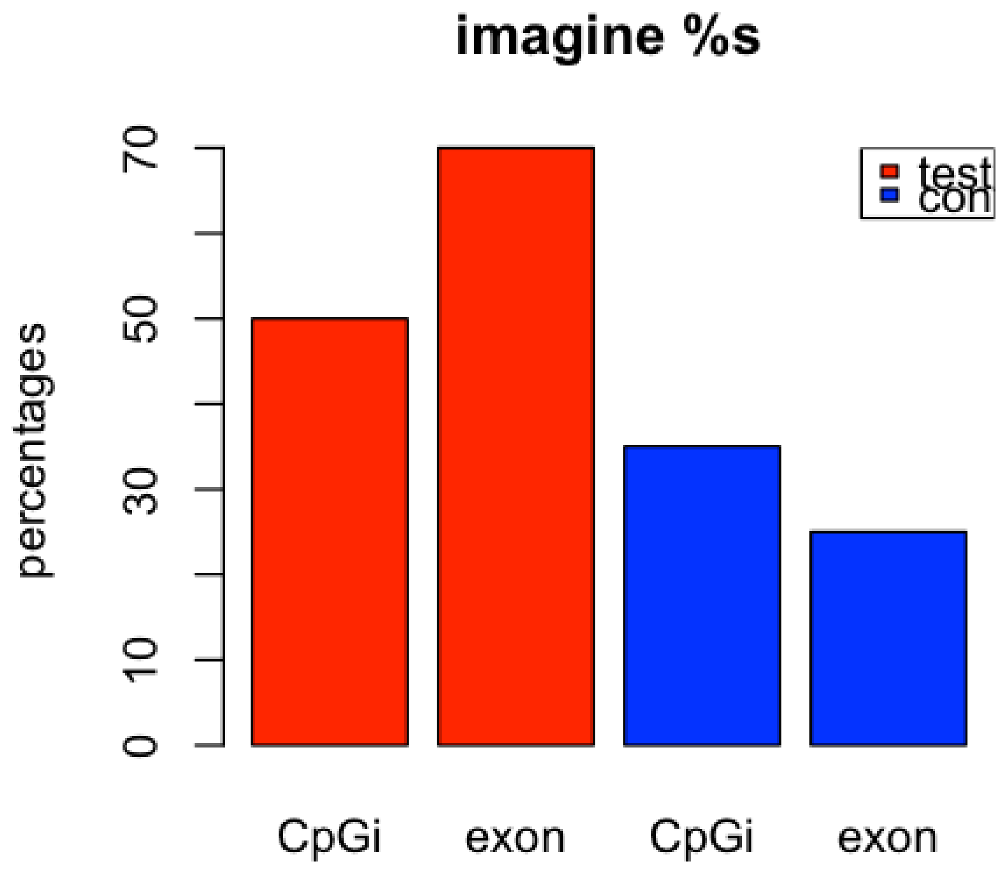
```
boxplot(x,y,main="boxplots of random samples")
```



Barplot

In [19]:

```
perc=c(50,70,35,25)
barplot(height=perc,names.arg=c("CpGi","exon","CpGi","exon"),
        ylab="percentages",main="imagine %s",
        col=c("red","red","blue","blue"))
legend("topright",legend=c("test","control"),fill=c("red","blu
e"))
```



Saving plots

If you want to save your plots to an image file there are couple of ways of doing that. Normally, you will have to do the following:

1. Open a graphics device
2. Create the plot
3. Close the graphics device

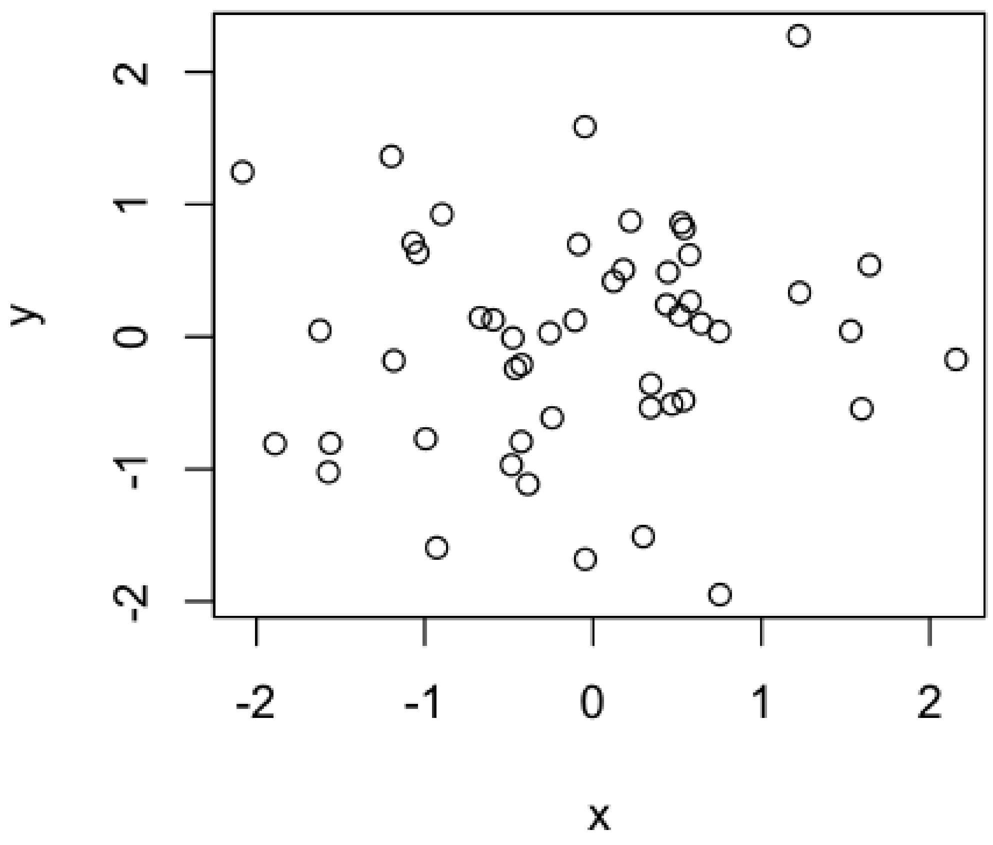
In [20]:

```
pdf("myplot.pdf",width=5,height=5)
plot(x,y)
dev.off()

 #Alternatively, you can first create the plot then copy the p
lot to a graphic device.

plot(x,y)
dev.copy(pdf,"myplot.pdf",width=7,height=5)
dev.off()
```

pdf: 2

pdf: 3

pdf: 2


y
2
1
0
-1
-2
-2
-1
0
1
2
x

Working with sequences, primarily DNA sequences, and genomic features.

We will be using Bioconductor packages for this.

Bioconductor represents a different strand of current development in R, separate from the Hadley Wickham tidyverse. Where Hadley emphasizes the data frame above all else, Bioconductor uses a great variety of data types. It's the very opposite of tidy!

Nevertheless, Bioconductor is overwhelmingly *comprehensive*, and represents the most complete environment available for working with bioinformatic data currently available.

Bioconductor packages usually have useful documentation in the form of "vignettes". These are readable on the Bioconductor website, or within R:

```
In [21]:
```

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

```
Bioconductor version 3.7 (BiocInstaller 1.30.0), ?b
iocLite for help
```

In [22]:

```
# Install a basic set of packages
biocLite()

# Install further packages used in this tutorial
biocLite(c(
    "Biostrings",
    "GenomicRanges",
    "rtracklayer",
    "motifRG",
    "AnnotationHub",
    "ggbio"
))
```

```
BioC_mirror: https://bioconductor.org
Using Bioconductor 3.7 (BiocInstaller 1.30.0), R 3.
5.0 (2018-04-23).
Old packages: 'BiocParallel', 'biovizBase', 'broo
m', 'caTools', 'dbplyr',
  'DelayedArray', 'devtools', 'dplyr', 'evaluate',
'foreign', 'ggthemes',
  'git2r', 'glue', 'gmodels', 'gtools', 'haven', 'h
ighr', 'httpuv',
  'iterators', 'MASS', 'matrixStats', 'mgcv', 'mode
ltools', 'munsell',
  'nycflights13', 'pillar', 'plotly', 'purrr', 'Rcp
p', 'RCurl',
  'recommenderlab', 'reprex', 'rlang', 'rmarkdown',
'Rsamtools', 'Rttf2pt1',
  'stringi', 'survival', 'tm', 'VariantAnnotation',
'xlsx', 'XML', 'xts',
  'yaml', 'zoo'
BioC_mirror: https://bioconductor.org
Using Bioconductor 3.7 (BiocInstaller 1.30.0), R 3.
5.0 (2018-04-23).
Installing package(s) ‘Biostrings’, ‘GenomicRange
s’, ‘rtracklayer’, ‘motifRG’,
  ‘AnnotationHub’, ‘ggbio’

The downloaded binary packages are in
        /var/folders/jw/knt_b30n31xgtwmrfn00sctm000
0gn/T//Rtmp9YAq3L/downloaded_packages

Old packages: 'BiocParallel', 'biovizBase', 'broo
m', 'caTools', 'dbplyr',
  'DelayedArray', 'devtools', 'dplyr', 'evaluate',
'foreign', 'ggthemes',
  'git2r', 'glue', 'gmodels', 'gtools', 'haven', 'h
ighr', 'httpuv',
  'iterators', 'MASS', 'matrixStats', 'mgcv', 'mode
ltools', 'munsell',
  'nycflights13', 'pillar', 'plotly', 'purrr', 'Rcp
p', 'RCurl',
  'recommenderlab', 'reprex', 'rlang', 'rmarkdown',
'Rsamtools', 'Rttf2pt1',
  'stringi', 'survival', 'tm', 'VariantAnnotation',
'xlsx', 'XML', 'xts',
  'yaml', 'zoo'
```

In [23]:

```
library(Biostrings)     # Provides DNAString, DNAStringSet, et
c
library(GenomicRanges)  # Provides GRanges, etc
library(rtracklayer)    # Provides import() and export()
```

```
Loading required package: BiocGenerics
Loading required package: parallel

Attaching package: ‘BiocGenerics’

The following objects are masked from ‘package:para
llel’:

    clusterApply, clusterApplyLB, clusterCall, clus
terEvalQ,
    clusterExport, clusterMap, parApply, parCapply,
parLapply,
    parLapplyLB, parRapply, parSapply, parSapplyLB

The following objects are masked from ‘package:stat
s’:

    IQR, mad, sd, var, xtabs

The following objects are masked from ‘package:bas
e’:

    anyDuplicated, append, as.data.frame, basename,
cbind, colMeans,
    colnames, colSums, dirname, do.call, duplicate
d, eval, evalq,
    Filter, Find, get, grep, grepl, intersect, is.u
nsorted, lapply,
    lengths, Map, mapply, match, mget, order, past
e, pmax, pmax.int,
    pmin, pmin.int, Position, rank, rbind, Reduce,
rowMeans, rownames,
    rowSums, sapply, setdiff, sort, table, tapply,
union, unique,
    unsplit, which, which.max, which.min

Loading required package: S4Vectors
Loading required package: stats4

Attaching package: ‘S4Vectors’

The following object is masked from ‘package:base’:
    expand.grid

Loading required package: IRanges
Loading required package: XVector

Attaching package: ‘Biostrings’

The following object is masked from ‘package:base’:

    strsplit

Warning message:
“package ‘GenomicRanges’ was built under R version
3.5.1”Loading required package: GenomeInfoDb
```

DNAString

Package `Biostrings` offers classes for storing DNA strings, `DNAString`, amino acid sequences, `AAString`, or anything else in a `BString`. These are very like character strings, but a variety of biologically meaningful functions can be applied to them.

In [24]:

```
myseq <- DNAString("ACCATTGATTAT")
myseq
```

```
  12-letter "DNAString" instance
seq: ACCATTGATTAT
```

In [25]:

```
class(myseq)
```

'DNAString'

In [26]:

```
reverseComplement(myseq)
translate(myseq)
```

```
  12-letter "DNAString" instance
seq: ATAATCAATGGT


  4-letter "AAString" instance
seq: TIDY
```

In [27]:

```
subseq(myseq, 3,5)
myseq[3:5]
```

```
  3-letter "DNAString" instance
seq: CAT

  3-letter "DNAString" instance
seq: CAT
```

In [28]:

```
as.character(myseq)
```

```
'ACCATTGATTAT'
```

In [29]:

```
methods(class="DNAString")
```

```
 [1] !=                               [

 [3] [<-                              %in%

 [5] <                                <=

 [7] ==                               >

 [9] >=                               aggregate

[11] alphabetFrequency                anyNA

[13] append                           as.character

[15] as.complex                       as.data.frame

[17] as.env                           as.integer

[19] as.list                          as.logical

[21] as.matrix                        as.numeric

[23] as.raw                           as.vector

[25] by                               c

[27] chartr                           codons

[29] coerce                           compact

[31] compareStrings                   complement

[33] concatenateObjects               countOverlaps

[35] countPattern                     countPDict

[37] countPWM                         duplicated

[39] elementMetadata                  elementMetada
ta<-
[41] eval                             expand

[43] expand.grid                      extractAt
```

```
 [45] extractList                        extractROWS

 [47] findOverlaps                       findPalindrom
es
 [49] hasOnlyBaseLetters                 head

 [51] intersect                          is.na

 [53] isMatchingEndingAt                 isMatchingSta
rtingAt
 [55] lcprefix                           lcsubstr

 [57] lcsuffix                           length

 [59] lengths                            letter

 [61] letterFrequency                    letterFrequen
cyInSlidingView
 [63] maskMotif                          masks

 [65] masks<-                            match

 [67] matchLRPatterns                    matchPattern

 [69] matchPDict                         matchProbePai
r
 [71] matchPWM                           mcols

 [73] mcols<-                            merge

 [75] metadata                           metadata<-

 [77] mstack                             nchar

 [79] neditEndingAt                      neditStarting
At
 [81] needwunsQS                         NROW

 [83] oligonucleotideFrequency           overlapsAny

 [85] PairwiseAlignments                 PairwiseAlign
mentsSingleSubject
 [87] palindromeArmLength                palindromeLef
```

```
tArm
 [89] palindromeRightArm               parallelSlotN
ames
 [91] pcompare                         pmatchPattern

 [93] rank                             relist

 [95] relistToClass                    rename

 [97] rep                              rep.int

 [99] replaceAt                        replaceLetter
At
[101] replaceROWS                      rev

[103] reverse                          reverseComple
ment
[105] ROWNAMES                         seqlevelsInUs
e
[107] seqtype                          seqtype<-

[109] setdiff                          setequal

[111] shiftApply                       show

[113] showAsCell                       sort

[115] split                            split<-

[117] subseq                           subseq<-

[119] subset                           subsetByOverl
aps
[121] substr                           substring

[123] table                            tail

[125] tapply                           toComplex

[127] toString                         transform

[129] translate                        trimLRPattern
s
[131] twoWayAlphabetFrequency          union
```

```
[133] unique                          uniqueLetters

[135] unmasked                        updateObject

[137] values                          values<-

[139] vcountPattern                   vcountPDict

[141] Views                           vmatchPattern

[143] vmatchPDict                     vwhichPDict

[145] which.isMatchingEndingAt        which.isMatch
ingStartingAt
[147] whichPDict                      window

[149] window<-                        with

[151] xtabs                           xvcopy

see '?methods' for accessing help and source code
```

In [30]:

```
?"DNAString-class"
```

DNAStringSet

Often we want to work with a list of sequences, such as chromosomes.

In [31]:

```
myset <- DNAStringSet( list(chrI=myseq, chrII=DNAString("ACGTA
CGT")) )
myset

# A DNAStringSet is list-like
myset$chrII
# or myset[["chrII"]]
# or myset[[2]]
```

```
  A DNAStringSet instance of length 2
    width seq
         names
[1]    12 ACCATTGATTAT
         chrI
[2]     8 ACGTACGT
         chrII
```

```
  8-letter "DNAString" instance
seq: ACGTACGT
```

Loading files

Loading sequences

DNA sequences are generally stored in FASTA format, a simple text format. These can be loaded with `readDNAStringSet` from `Biostrings`. Let's load the genome of E. coli strain K-12, obtained from the Ensembl FTP site.

```
### The start of the .fa file looks like this:
# >Chromosome dna:chromosome chromosome:GCA_000800765.1:
Chromosome:1:4558660:1
# AGCTTTTCATTCTGACTGCAACGGGCAATATGTCTCTGTGTGGATTAAAAAAAG
AGTGTC
# TGATAGCAGCTTCTGAACTGGTTACCTGCCGTGAGTAAATTAAAATTTTATTGA
CTTAGG
# TCACTAAATACTTTAACCAATATAGGCATAGCGCACAGACAGATAAAAATTACA
GAGTAC
# ACAACATCCATGAAACGCATTAGCACCACCATTACCACCACCATCACCATTACC
ACAGGT
# AACGGTGCGGGCTGACGCGTACAGGAAACACAGAAAAAAGCCCGCACCTGACAG
TGCGGG
# CTTTTTTTTTCGACCAAAGGTAACGAGGTAACAACCATGCGAGTGTTGAAGTTC
GGCGGT
# ...
```

In [32]:

```
seqs <- readDNAStringSet("dataJune21th/gendata/Escherichia_col
i_k_12.GCA_000800765.1.29.dna.genome.fa")
seqs
```

```
  A DNAStringSet instance of length 1
      width seq
         names
[1] 4558660 AGCTTTTCATTCTGACTGCAAC...AACGCCTTAGTAAG
TATTTTTC Chromosome dna:ch...
```

In [33]:

```
# Our chromosome name is too verbose.
# Remove everything from the name after the first space.
names(seqs)
names(seqs) <- sub(" .*","",names(seqs))
names(seqs)
```

'Chromosome dna:chromosome chromosome:GCA_000800765.1:Chromosome:1:4558660:1'

'Chromosome'

Genomic Intervals

Bioconductor (http://bioconductor.org) project has a dedicated package called **GenomicRanges** to deal with genomic intervals. In this section, we will provide use cases involving operations on genomic intervals. The main reason we will stick to this package is that it provides tools to do overlap operations. However package requires that users operate on specific data types that are conceptually similar to a tabular data structure implemented in a way that makes overlapping and related operations easier. The main object we will be using is called GRanges object and we will also see some other related objects from the GenomicRanges package.

How to create and manipulate a GRanges object

In [34]:

```
gr=GRanges(seqnames=c("chr1","chr2","chr2"),
           ranges=IRanges(start=c(50,150,200),end=c(100,200,30
0)),
           strand=c("+","-","-")
)
gr
```

```
GRanges object with 3 ranges and 0 metadata column
s:
      seqnames    ranges strand
         <Rle> <IRanges>  <Rle>
  [1]     chr1    50-100      +
  [2]     chr2   150-200      -
  [3]     chr2   200-300      -
  -------
  seqinfo: 2 sequences from an unspecified genome;
no seqlengths
```

In [35]:

```
# subset like a data frame
gr[1:2,]
```

```
GRanges object with 2 ranges and 0 metadata column
s:
      seqnames    ranges strand
         <Rle> <IRanges>  <Rle>
  [1]     chr1    50-100      +
  [2]     chr2   150-200      -
  -------
  seqinfo: 2 sequences from an unspecified genome;
no seqlengths
```

In [36]:

```
gr=GRanges(seqnames=c("chr1","chr2","chr2"),
           ranges=IRanges(start=c(50,150,200),end=c(100,200,30
0)),
           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(100,200,30
0)),
           names=c("id1","id3","id2"),
           scores=c(100,90,50)
)

# or appends to existing meta data
mcols(gr)=cbind(mcols(gr),
                          DataFrame(name2=c("pax6","meis1","zi
c4")) )
gr
```

```
GRanges object with 3 ranges and 3 metadata column
s:
      seqnames    ranges strand |       names    sc
ores       name2
         <Rle> <IRanges>  <Rle> | <character> <nume
ric> <character>
  [1]     chr1    50-100      * |         id1
100        pax6
  [2]     chr2   150-200      * |         id3
  90       meis1
  [3]     chr2   200-300      * |         id2
  50        zic4
  -------
  seqinfo: 2 sequences from an unspecified genome;
no seqlengths
```

In [37]:

```
# elementMetadata() and values() do the same things
elementMetadata(gr)
```

```
DataFrame with 3 rows and 3 columns
        names    scores       name2
  <character> <numeric> <character>
1         id1       100        pax6
2         id3        90       meis1
3         id2        50        zic4
```

In [38]:

```
values(gr)
```

```
DataFrame with 3 rows and 3 columns
        names    scores       name2
  <character> <numeric> <character>
1         id1       100        pax6
2         id3        90       meis1
3         id2        50        zic4
```

Getting genomic regions into R as GRanges objects

There are multiple ways you can read in your genomic features into R and create a GRanges object. Most genomic interval data comes as a tabular format that has the basic information about the location of the interval and some other information. We already showed how to read BED files as data frame. Now we will show how to convert it to GRanges object.

In [39]:

```
# read CpGi data set
cpgi.df = read.table("dataJune21th/cpgi.hg19.chr21.bed", heade
r = FALSE,
                     stringsAsFactors=FALSE)
# remove chr names with "_"
cpgi.df =cpgi.df [grep("_",cpgi.df[,1],invert=TRUE),]

cpgi.gr=GRanges(seqnames=cpgi.df[,1],
                ranges=IRanges(start=cpgi.df[,2],
                               end=cpgi.df[,3]))

cpgi.gr
```

```
GRanges object with 205 ranges and 0 metadata colum
ns:
        seqnames              ranges strand
           <Rle>           <IRanges>  <Rle>
    [1]    chr21   9825442-9826296      *
    [2]    chr21   9909011-9909218      *
    [3]    chr21   9968264-9968620      *
    [4]    chr21 10989913-10991413      *
    [5]    chr21 14409412-14410501      *
    ...      ...                 ...    ...
  [201]    chr21 47918497-47918728      *
  [202]    chr21 48018542-48018791      *
  [203]    chr21 48055199-48056060      *
  [204]    chr21 48068517-48068808      *
  [205]    chr21 48081241-48081849      *
  -------
  seqinfo: 1 sequence from an unspecified genome; n
o seqlengths
```

Sometimes pre-processing is necessary

In [40]:

```
# read refseq file
ref.df = read.table("dataJune21th/refseq.hg19.chr21.bed", head
er = FALSE,
                     stringsAsFactors=FALSE)
ref.gr=GRanges(seqnames=ref.df[,1],
               ranges=IRanges(start=ref.df[,2],
                              end=ref.df[,3]),
               strand=ref.df[,6],name=ref.df[,4])
# get TSS
tss.gr=ref.gr
# end of the + strand genes must be equalized to start pos
end(tss.gr[strand(tss.gr)=="+",])  =start(tss.gr[strand(tss.gr
)=="+",])
# startof the - strand genes must be equalized to end pos
start(tss.gr[strand(tss.gr)=="-",])=end(tss.gr[strand(tss.gr)=
="-",])
# remove duplicated TSSes ie alternative transcripts
# this keeps the first instance and removes duplicates
tss.gr=tss.gr[!duplicated(tss.gr),]
```

Reading the genomic features as text files and converting to GRanges is not the only way to create GRanges object. With the help of rtracklayer package we can directly import.

In [41]:

```
import.bed("dataJune21th/refseq.hg19.chr21.bed")
```

```
GRanges object with 571 ranges and 5 metadata colum
ns:
        seqnames           ranges strand |
name     score     itemRgb
           <Rle>        <IRanges>  <Rle> |  <chara
cter> <numeric> <character>
    [1]    chr21 41384343-42219039      - |    NR_0
73202         0        <NA>
    [2]    chr21 41384343-42219039      - |    NM_0
01389         0        <NA>
    [3]    chr21 41384343-42219039      - | NM_0012
71534         0        <NA>
    [4]    chr21 17442842-17982094      + |    NR_0
27790         0        <NA>
    [5]    chr21 17566699-17982094      + |    NR_0
27791         0        <NA>
    ...      ...              ...    ... .
  ...       ...         ...
  [567]    chr21 48055507-48075276      + | NM_0012
42865         0        <NA>
  [568]    chr21 48055507-48085155      + | NM_0012
42864         0        <NA>
  [569]    chr21 48018531-48025035      - |    NM_0
06272         0        <NA>
  [570]    chr21 48055507-48085155      + |    NM_0
01535         0        <NA>
  [571]    chr21 48055507-48085155      + |    NM_2
06962         0        <NA>
                    thick
    blocks
                <IRanges>                       <IR
angesList>
    [1] 42219040-42219039   1-971,29956-30258,31663
-31860,...
    [2] 41384961-42218587   1-971,29956-30258,31663
-31860,...
    [3] 41384961-42218587   1-917,29956-30258,31663
-31860,...
    [4] 17982095-17982094      1-27,593-877,111070-
111166,...
    [5] 17982095-17982094 1-255,36678-36737,197236-
197307,...
    ...               ...
```

```
       ...
  [567] 48056864-48071918       1-169,1302-1396,794
1-8045,...
  [568] 48056864-48084239       1-169,1302-1396,794
1-8045,...
  [569] 48019276-48022328           1-886,3661-379
9,6396-6505
  [570] 48056864-48084239       1-169,1302-1396,794
1-8045,...
  [571] 48056864-48084239         1-169,845-953,130
2-1396,...
  -------
  seqinfo: 1 sequence from an unspecified genome; n
o seqlengths
```

Now we will show how to use other packages to automatically obtain the data in GRanges format. But you will not be able to use these methods for every data set so it is good to now how to read data from flat files as well. First, we will use rtracklayer package to download data from UCSC browser. We will download CpG islands as GRanges objects.

In [42]:

```
session <- browserSession()
genome(session) <- "mm9"
## choose CpG island track on chr12
query <- ucscTableQuery(session, track="CpG Islands",table="cp
gIslandExt",
        range=GRangesForUCSCGenome("mm9", "chr12"))
## get the GRanges object for the track
track(query)
```

```
UCSC track 'cpgIslandExt'
UCSCData object with 627 ranges and 1 metadata colu
mn:
        seqnames              ranges strand |
  name
           <Rle>           <IRanges>  <Rle> | <char
acter>
    [1]    chr12     3235441-3235920      * |      C
pG:_55
    [2]    chr12     3309325-3310176      * |     Cp
G:_112
    [3]    chr12     3365112-3365428      * |      C
pG:_33
    [4]    chr12     3426606-3427706      * |     Cp
G:_112
    [5]    chr12     3572056-3572883      * |      C
pG:_87
    ...      ...                 ...    ... .
   ...
  [623]    chr12 120074998-120075659      * |      C
pG:_64
  [624]    chr12 120081568-120081824      * |      C
pG:_22
  [625]    chr12 120085202-120085696      * |      C
pG:_45
  [626]    chr12 120086987-120088377      * |     Cp
G:_147
  [627]    chr12 120476260-120476575      * |      C
pG:_23
  -------
  seqinfo: 35 sequences from mm9 genome
```

Finding regions that (does/does not) overlap with another set of regions

This is one of the most common tasks in genomics. Usually, you have a set of regions that you are interested in and you want to see if they overlap with another set of regions or see how many of them overlap. A good example is transcription factor binding sites determined by ChIP-seq experiments. In these types of experiments and followed analysis, one usually ends up with genomic regions that are bound by transcription factors. One of the standard next questions would be to annotate binding sites with genomic annotations such as promoter,exon,intron and/or CpG islands. Below is a demonstration of how transcription factor binding sites can be annotated using CpG islands. First, we will get the subset of binding sites that overlap with the CpG islands. In this case, binding sites are ChIP-seq peaks.

We can find the subset of peaks that overlap with the CpG islands using subsetByoverlaps() function. You will also see another way of converting data frames to GRanges.

In [43]:

```
pk1=read.table("dataJune21th/wgEncodeHaibTfbsGm12878Sp1Pcr1xPk
Rep1.broadPeak.gz")
head(pk1)
```

V1	V2	V3	V4	V5	V6	V7	V8	V9
chr1	9990	10480	peak1	143	.	464.20	-1	-1
chr1	11020	12230	peak2	347	.	1123.56	-1	-1
chr1	13882	14300	peak3	68	.	221.82	-1	-1
chr1	22720	23011	peak4	33	.	109.62	-1	-1
chr1	23479	23786	peak5	40	.	129.51	-1	-1
chr1	25950	26457	peak6	69	.	225.58	-1	-1

In [44]:

```
# convert data frame to GRanges
pk1.gr=makeGRangesFromDataFrame(pk1,
        seqnames.field=c("V1"),
        start.field=c("V2"),
        end.field=c("V3"))
# only peaks on chr21
pk1.gr=pk1.gr[seqnames(pk1.gr)=="chr21",]
# get the peaks that overlap with CpG
# islands
subsetByOverlaps(pk1.gr,cpgi.gr)
```

```
GRanges object with 44 ranges and 0 metadata column
s:
       seqnames            ranges strand
          <Rle>         <IRanges>  <Rle>
   [1]    chr21   9825359-9826582      *
   [2]    chr21   9968468-9968984      *
   [3]    chr21 15755367-15755956      *
   [4]    chr21 19191578-19192525      *
   [5]    chr21 26979618-26980048      *
   ...      ...               ...    ...
  [40]    chr21 46237463-46237809      *
  [41]    chr21 46707701-46708084      *
  [42]    chr21 46961551-46961875      *
  [43]    chr21 47743586-47744125      *
  [44]    chr21 47878411-47878891      *
  -------
  seqinfo: 23 sequences from an unspecified genome;
no seqlengths
```

For each CpG island, we can count the number of peaks that overlap with a given CpG island with countOverlaps().

In [45]:

```
#count the peaks that
# overlap with CpG islands
counts=countOverlaps(pk1.gr,cpgi.gr)
head(counts)
```

```
    0  0  0  0  0  0
```

findOverlaps() function can be used to see one-to-one overlaps between peaks and CpG islands. It returns a matrix showing which peak overlaps with which CpGi island.

In [46]:

```
findOverlaps(pk1.gr,cpgi.gr)
```

```
Hits object with 45 hits and 0 metadata columns:
       queryHits subjectHits
       <integer>   <integer>
   [1]       123           1
   [2]       154           3
   [3]       389           8
   [4]       401          13
   [5]       415          16
   ...       ...         ...
  [41]       595         155
  [42]       598         166
  [43]       600         176
  [44]       611         192
  [45]       613         200
  -------
  queryLength: 620 / subjectLength: 205
```

Another interesting thing would be to look at the distances to nearest CpG islands for each peak. In addition, just finding the nearest CpG island could also be interesting. Often times, you will need to find nearest TSS or gene to your regions of interest, and the code below is handy for doing that.

In [47]:

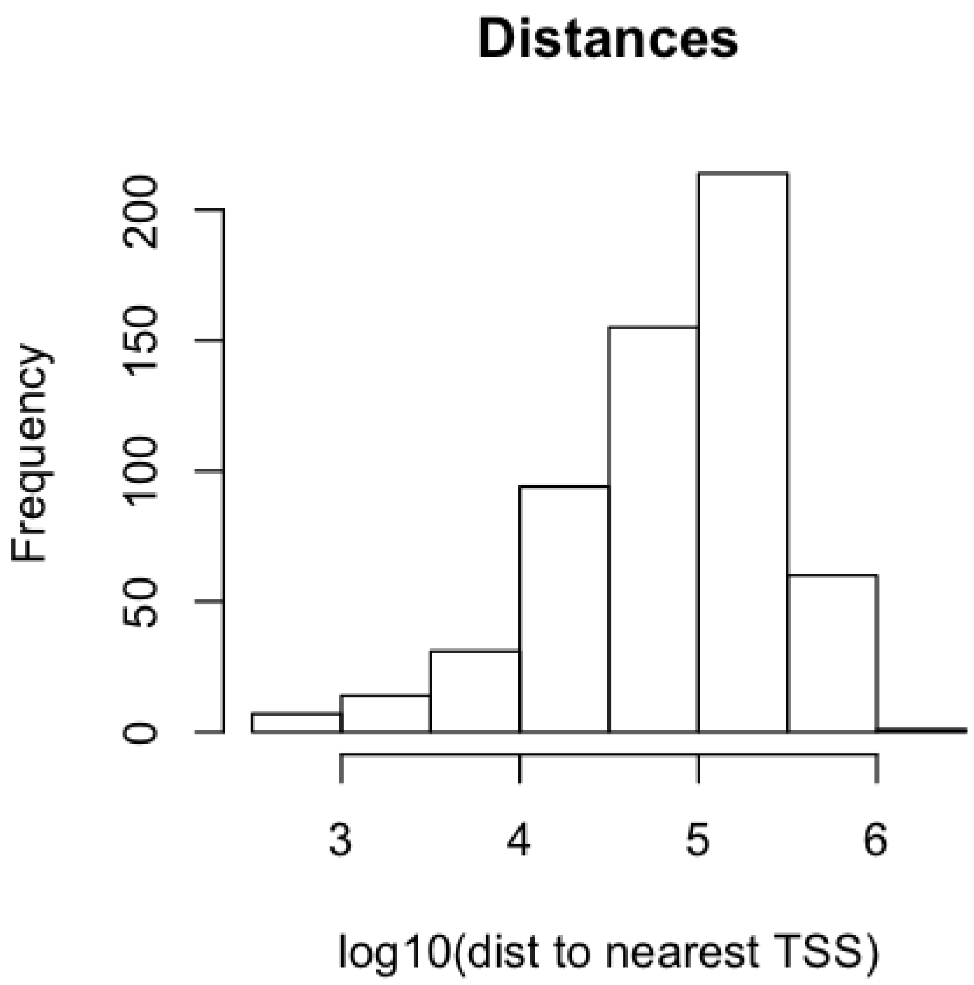
```
# find nearest CpGi to each TSS
n.ind=nearest(pk1.gr,cpgi.gr)
# get distance to nearest
dists=distanceToNearest(pk1.gr,cpgi.gr,select="arbitrary")
dists
```

```
Hits object with 620 hits and 1 metadata column:
        queryHits subjectHits |  distance
        <integer>   <integer> | <integer>
    [1]         1           1 |    384188
    [2]         2           1 |    382968
    [3]         3           1 |    381052
    [4]         4           1 |    379311
    [5]         5           1 |    376978
    ...       ...         ... .       ...
  [616]       616         205 |     26211
  [617]       617         205 |     27401
  [618]       618         205 |     30467
  [619]       619         205 |     31610
  [620]       620         205 |     34089
  -------
  queryLength: 620 / subjectLength: 205
```

Some Visualizations:

In [48]:

```
# histogram of the distances to nearest TSS
dist2plot=mcols(dists)[,1]
hist(log10(dist2plot),xlab="log10(dist to nearest TSS)",
     main="Distances")
```

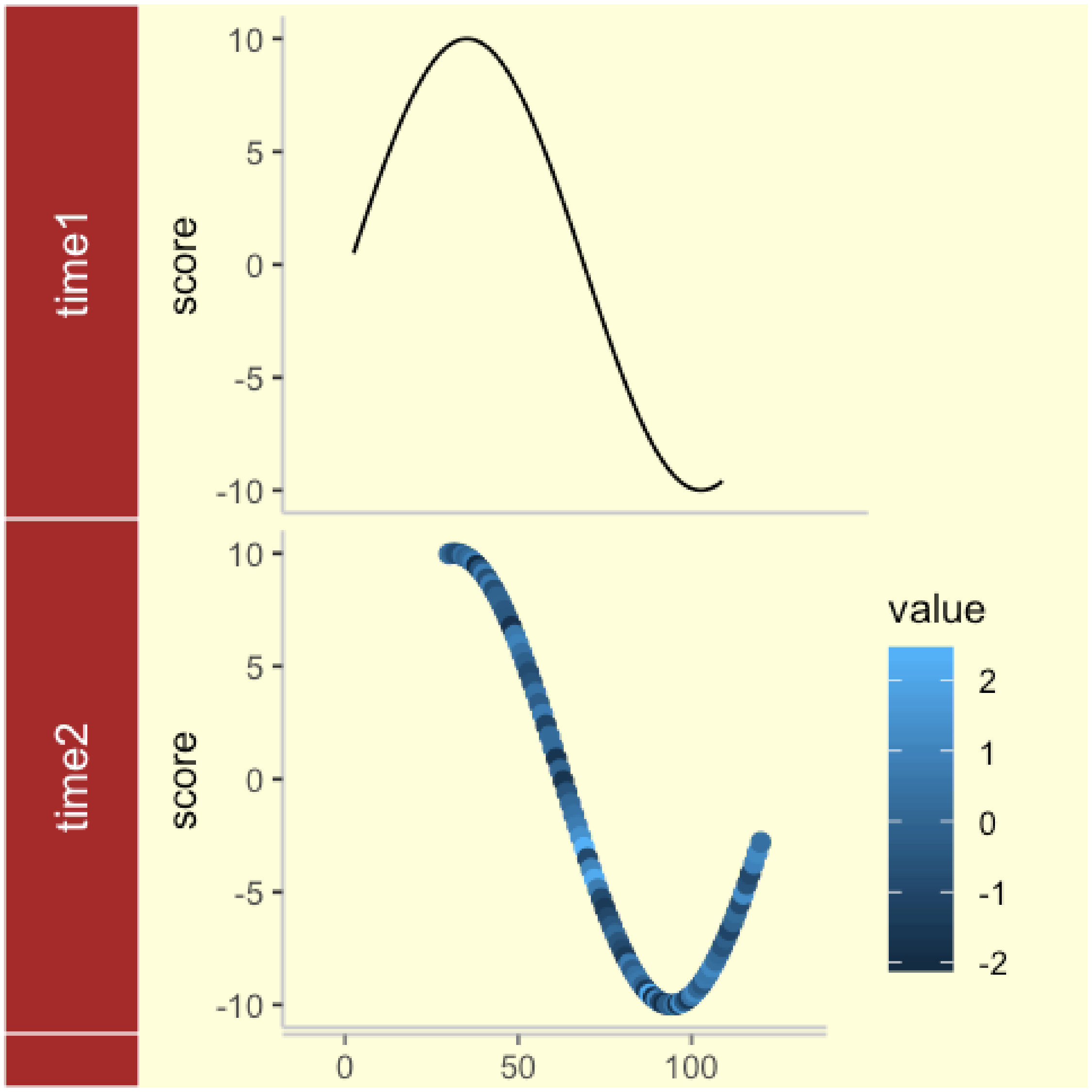


Tracks - aligning plots along chromosomes

In [49]:

```
library(ggbio)

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 = 2, aes(color = value))
tracks(time1 = p1, time2 = p2) + xlim(1, 40) + theme_tracks_su
nset()
```

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

Attaching package: 'ggbio'

The following objects are masked from 'package:ggpl
ot2':

    geom_bar, geom_rect, geom_segment, ggsave, stat
_bin, stat_identity,
    xlim

Coordinate system already present. Adding new coord
inate system, which will replace the existing one.
Coordinate system already present. Adding new coord
inate system, which will replace the existing one.
Coordinate system already present. Adding new coord
inate system, which will replace the existing one.
```

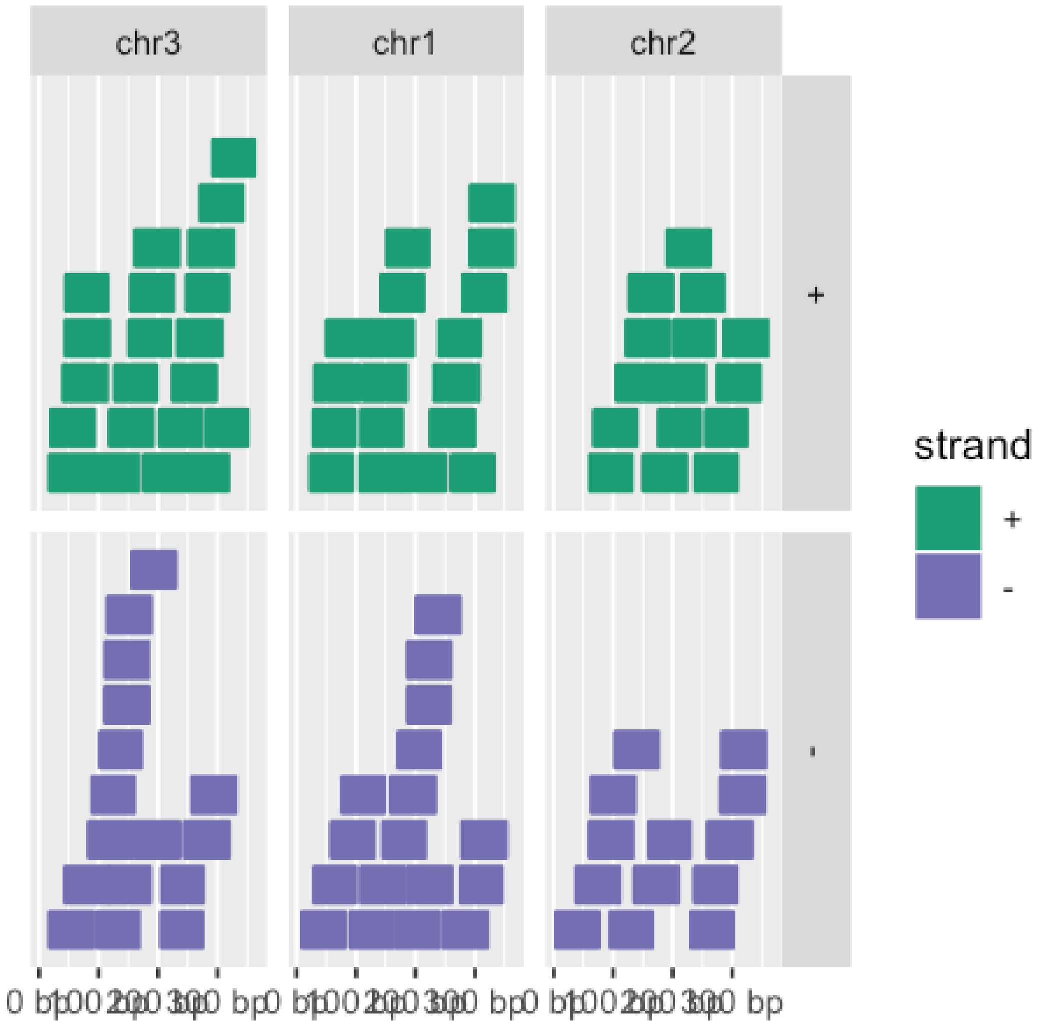


Plotting genomic ranges

GRanges objects are essential for storing alignment or annotation ranges in R/Bioconductor. The following creates a sample GRanges object and plots its content.

In [50]:

```
set.seed(1); N <- 100; gr <- GRanges(seqnames = sample(c("chr
1", "chr2", "chr3"), size = N, replace = TRUE), IRanges(start
= sample(1:300, size = N, replace = TRUE), width = sample(70:7
5, size = N,replace = TRUE)), strand = sample(c("+", "-"), siz
e = 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 = T
RUE))
autoplot(gr, aes(color = strand, fill = strand), facets = stra
nd ~ seqnames)
```



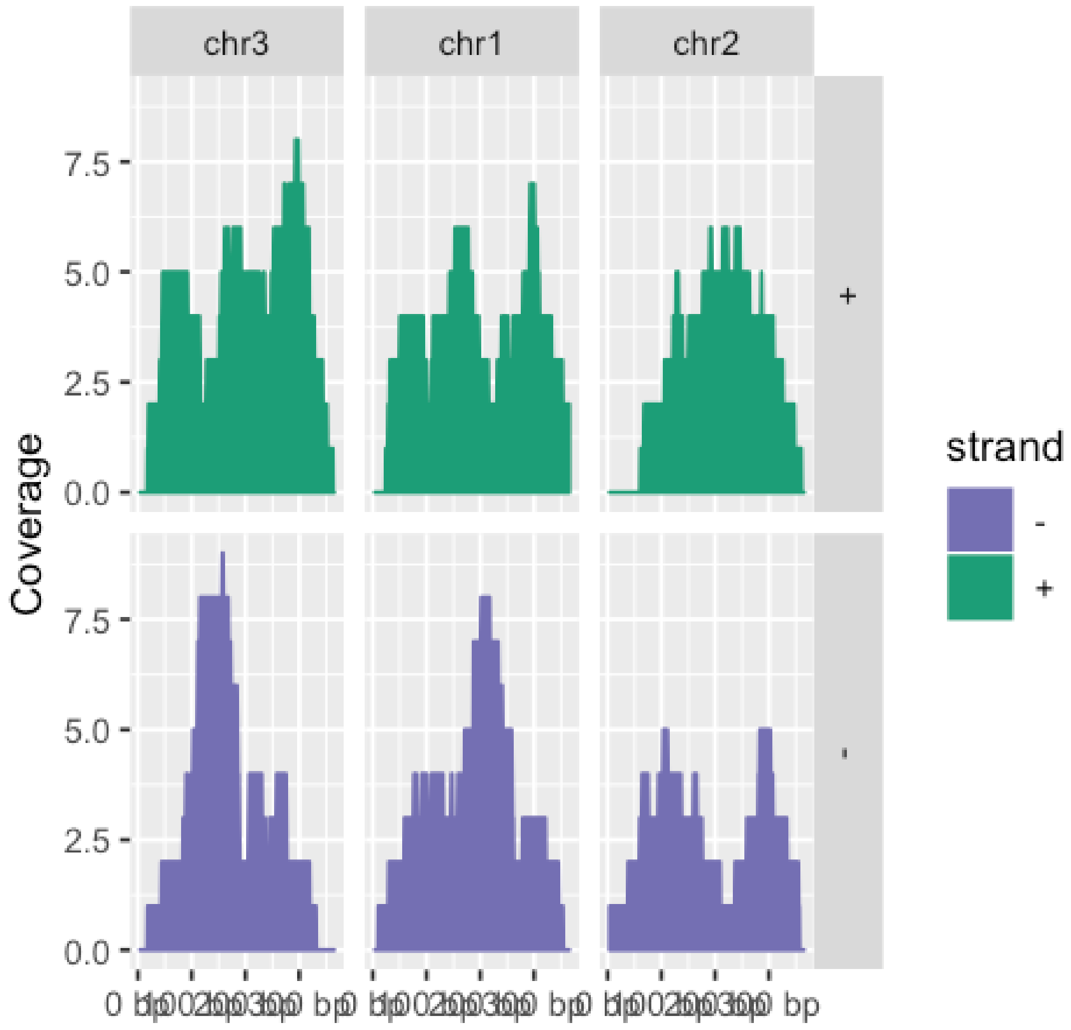
Plotting coverage

Mirrored coverage

```
In [51]:
```

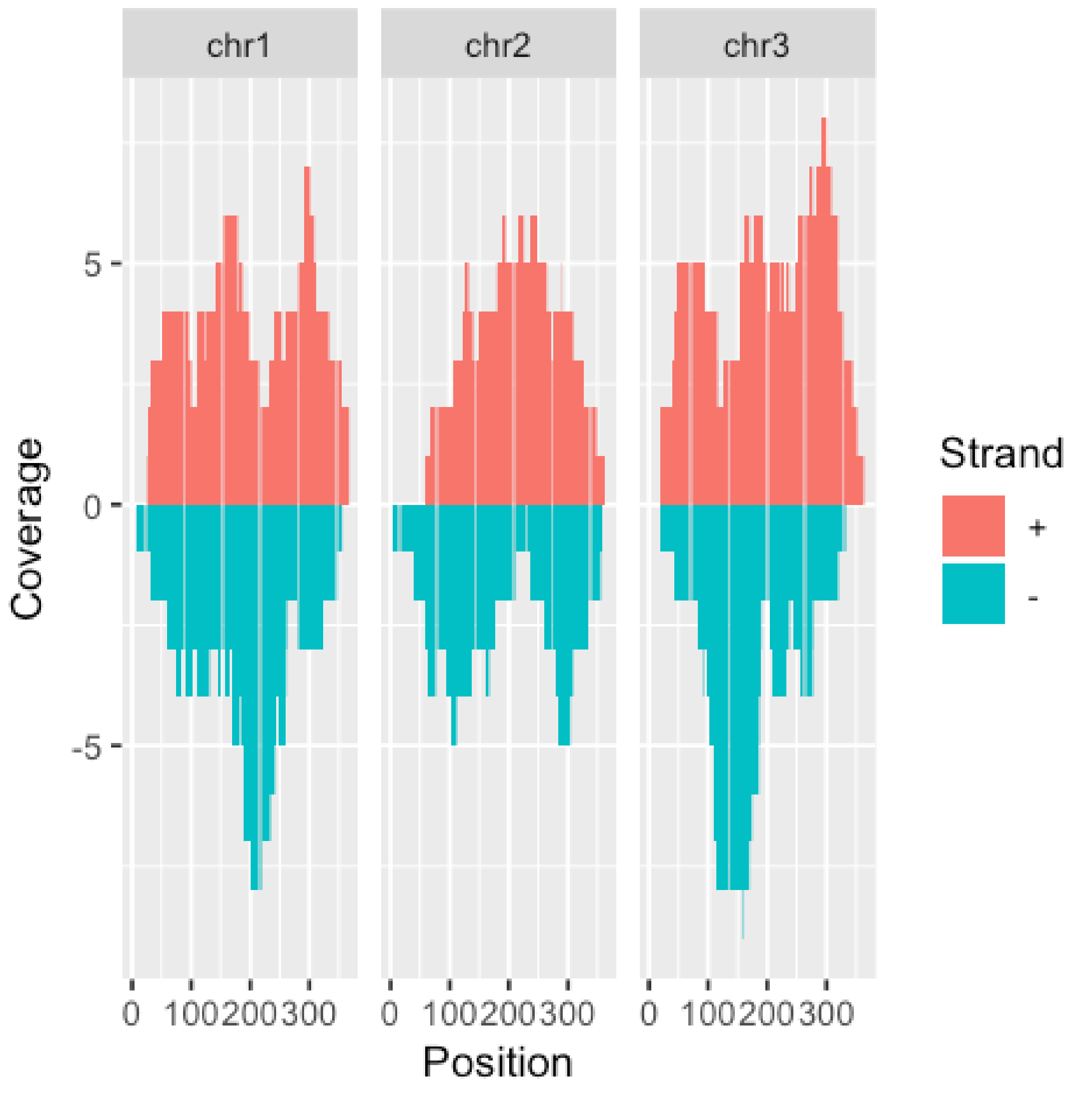
```
autoplot(gr, aes(color = strand, fill = strand), facets = stra
nd ~ seqnames, stat = "coverage")
```

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



In [52]:

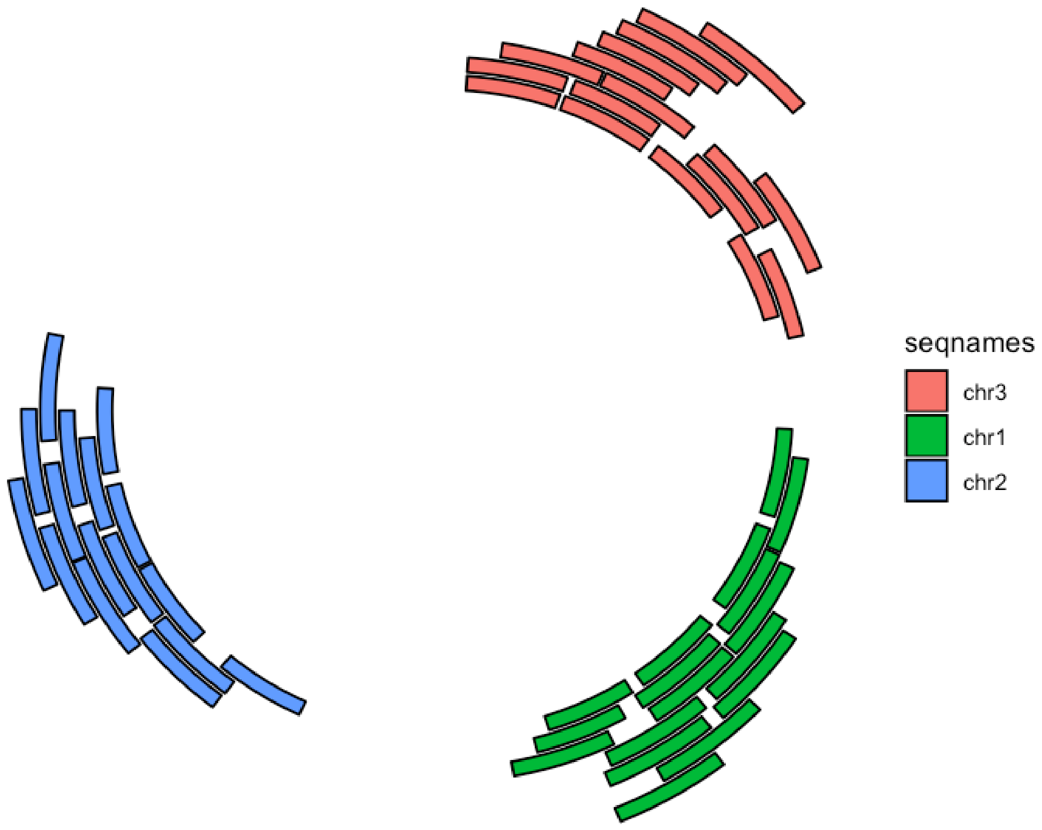
```
pos <- sapply(coverage(gr[strand(gr)=="+"]), as.numeric)
pos <- data.frame(Chr=rep(names(pos), sapply(pos, length)), St
rand=rep("+", length(unlist(pos))), Position=unlist(sapply(pos
, function(x) 1:length(x))), Coverage=as.numeric(unlist(pos)))
neg <- sapply(coverage(gr[strand(gr)=="-"]), as.numeric)
neg <- data.frame(Chr=rep(names(neg), sapply(neg, length)), St
rand=rep("-", length(unlist(neg))), Position=unlist(sapply(neg
, function(x) 1:length(x))), Coverage=-as.numeric(unlist(neg
)))
covdf <- rbind(pos, neg)
p <- ggplot(covdf, aes(Position, Coverage, fill=Strand)) +
            geom_bar(stat="identity", position="identity") + f
acet_wrap(~Chr)
p
```



Circular genome plots

In [57]:

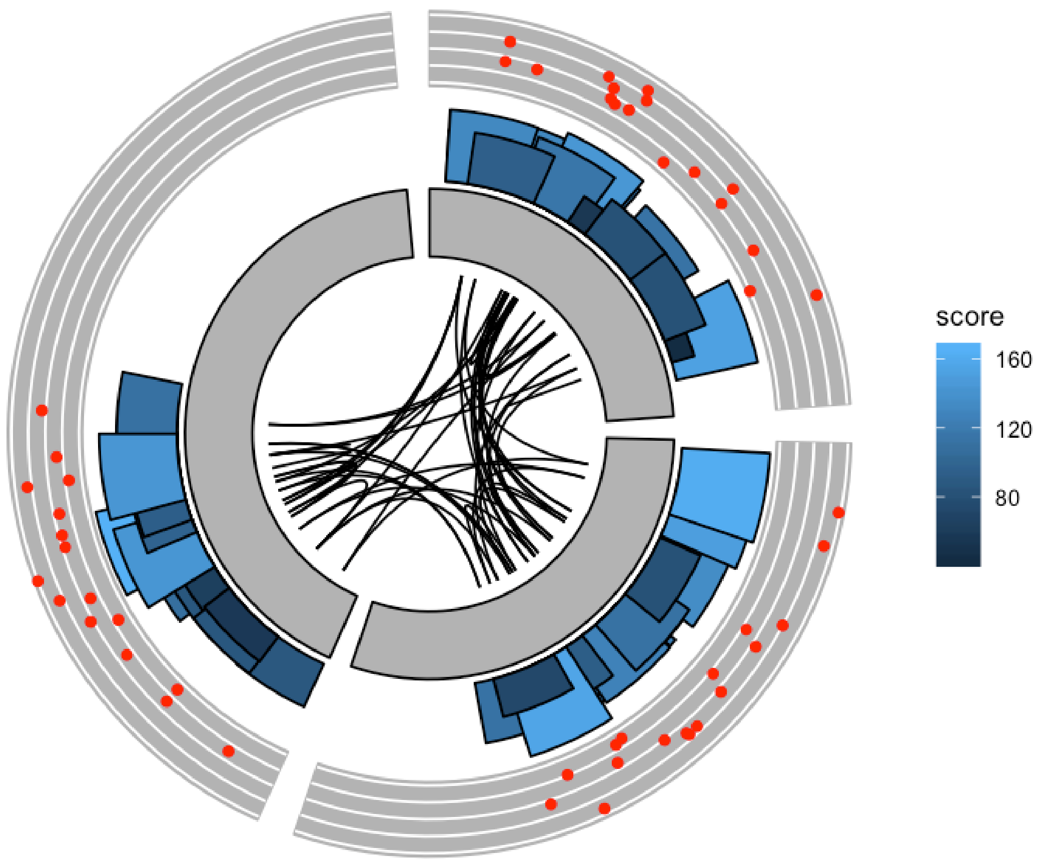
```
ggplot(gr) + layout_circle(aes(fill = seqnames), geom = "rect"
)
```



More complex circular example

```
In [58]:
```

```
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", r
adius = 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)
```

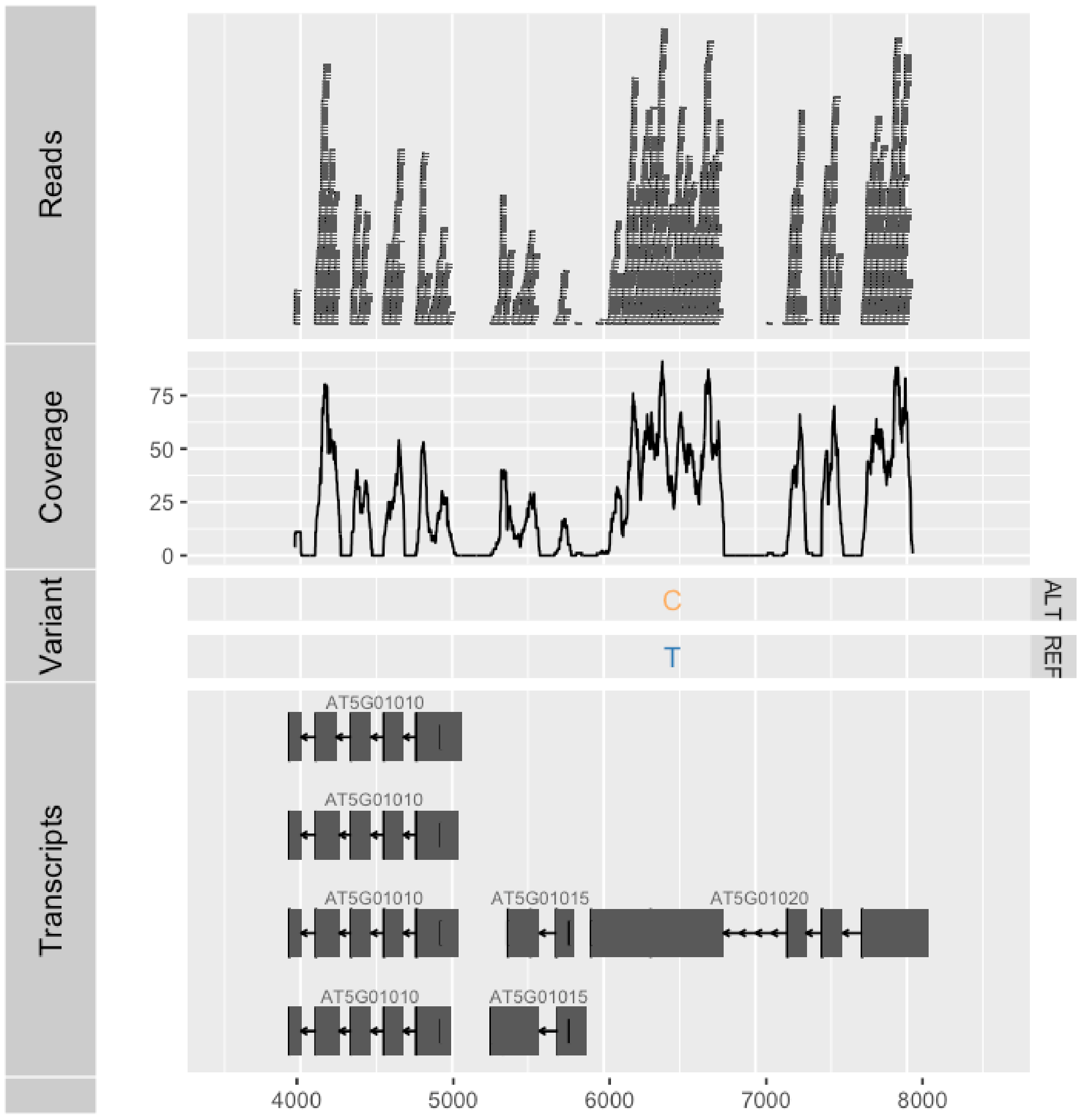


Alignments and variants plot

In [59]:

```
library(rtracklayer); library(GenomicFeatures); library(Rsamto
ols); library(GenomicAlignments); library(VariantAnnotation)

options(repr.plot.width = 6, repr.plot.height = 6)
ga <- readGAlignments("dataJune21th/plotdata/SRR064167.fastq.b
am", use.names=TRUE, param=ScanBamParam(which=GRanges("Chr5",
 IRanges(4000, 8000))))
p1 <- autoplot(ga, geom = "rect")
p2 <- autoplot(ga, geom = "line", stat = "coverage")
vcf <- readVcf(file="dataJune21th/plotdata/varianttools_gnsap.
vcf", genome="ATH1")
p3 <- autoplot(vcf[seqnames(vcf)=="Chr5"], type = "fixed") + x
lim(4000, 8000) + theme(legend.position = "none", axis.text.y
= element_blank(), axis.ticks.y=element_blank())
txdb <- makeTxDbFromGFF(file="dataJune21th/plotdata/TAIR10_GFF
3_trunc.gff", format="gff3")
p4 <- autoplot(txdb, which=GRanges("Chr5", IRanges(4000, 8000
)), names.expr = "gene_id")
tracks(Reads=p1, Coverage=p2, Variant=p3, Transcripts=p4, heig
hts = c(0.3, 0.2, 0.1, 0.35)) + ylab("")
```

```
extracting information...
extracting information...
Scale for 'x' is already present. Adding another sc
ale for 'x', which will
replace the existing scale.
Import genomic features from the file as a GRanges
object ... Warning message in .local(con, format, t
ext, ...):
"gff-version directive indicates version is 1, not
3"OK
Prepare the 'metadata' data frame ... OK
Make the TxDb object ... Warning message in .extrac
t_exons_from_GRanges(exon_IDX, gr, ID, Name, Paren
t, feature = "exon", :
"The following orphan exon were dropped (showing on
ly the 6 first):
  seqid start   end strand   ID       Parent Name
1  Chr2 10478 12861      - <NA> AT2G01022.1 <NA>
2  Chr2 14395 16377      - <NA> AT2G01024.1 <NA>
3  Chr2 17624 22540      - <NA> AT2G01026.1 <NA>
4  Chr2 23971 26923      - <NA> AT2G01028.1 <NA>
5  Chr2 28465 38652      + <NA> AT2G01029.1 <NA>
6  Chr2 39867 40358      - <NA> AT2G01031.1 <NA>"Wa
rning message in .extract_exons_from_GRanges(cds_ID
X, gr, ID, Name, Parent, feature = "cds", :
"The following orphan CDS were dropped (showing onl
y the 6 first):
  seqid start  end strand   ID               Parent
Name
1  Chr1  3760 3913      + <NA> AT1G01010.1-Protein
<NA>
2  Chr1  3996 4276      + <NA> AT1G01010.1-Protein
<NA>
3  Chr1  4486 4605      + <NA> AT1G01010.1-Protein
<NA>
4  Chr1  4706 5095      + <NA> AT1G01010.1-Protein
<NA>
5  Chr1  5174 5326      + <NA> AT1G01010.1-Protein
<NA>
6  Chr1  5439 5630      + <NA> AT1G01010.1-Protein
<NA>"OK
Parsing transcripts...
Parsing exons...
Parsing cds...
Parsing utrs...
------exons...
------cdss...
------introns...
------utr...
aggregating...
Done
Constructing graphics...
Parsing transcripts...
Parsing exons...
Parsing cds...
Parsing utrs...
------exons...
------cdss...
------introns...
------utr...
aggregating...
Done
Constructing graphics...
Coordinate system already present. Adding new coord
inate system, which will replace the existing one.
```



Reads
Coverage
Variant
Transcripts
75
50
25
0
C
ALT
T
REF
AT5G01010
AT5G01010
AT5G01010
AT5G01015
AT5G01020
AT5G01010
AT5G01015
4000
5000
6000
7000
8000

Loading features

Genome annotations are available in a variety of text formats such as GFF3 and GTF. They can be loaded with the `import` function from `rtracklayer`. This GTF file is also from Ensembl, and gives the locations of the genes in the genome, and features within them.

```
### The start of the .gtf file looks like this:
# #!genome-build ASM80076v1
# #!genome-version GCA_000800765.1
# #!genome-date 2014-12
# #!genome-build-accession GCA_000800765.1
# #!genebuild-last-updated 2014-12
# Chromosome      ena     gene    190     255     .
  +       .       gene_id "ER3413_4519"; gene_version
 "1"; gene_name "thrL"; gene_source "ena"; gene_biotype
 "protein_coding";
```

In [61]:

```
features <- import("dataJune21th/gendata/Escherichia_coli_k_1
2.GCA_000800765.1.29.gtf")

# Optional: just retain the columns of metadata we need
mcols(features) <- mcols(features)[,c("type","gene_name","gene
_id")]

features
```

```
GRanges object with 24926 ranges and 3 metadata col
umns:
            seqnames          ranges strand |
  type   gene_name
               <Rle>       <IRanges>  <Rle> |    <f
actor> <character>
      [1] Chromosome         190-255      + |
  gene        thrL
      [2] Chromosome         190-255      + |  tran
script        thrL
      [3] Chromosome         190-255      + |
  exon        thrL
      [4] Chromosome         190-252      + |
   CDS        thrL
      [5] Chromosome         190-192      + | start
_codon        thrL
      ...        ...             ...    ... .
   ...         ...
  [24922] Chromosome 4557950-4558636      + |  tran
script        yjtD
  [24923] Chromosome 4557950-4558636      + |
  exon        yjtD
  [24924] Chromosome 4557950-4558633      + |
   CDS        yjtD
  [24925] Chromosome 4557950-4557952      + | start
_codon        yjtD
  [24926] Chromosome 4558634-4558636      + |  stop
_codon        yjtD
                  gene_id
              <character>
      [1] ER3413_4519
      [2] ER3413_4519
      [3] ER3413_4519
      [4] ER3413_4519
      [5] ER3413_4519
      ...         ...
  [24922] ER3413_4514
  [24923] ER3413_4514
  [24924] ER3413_4514
  [24925] ER3413_4514
  [24926] ER3413_4514
  -------
  seqinfo: 1 sequence from an unspecified genome; n
o seqlengths
```

We can use these annotations to grab sequences from the genome.

In [62]:

```
feat <- features[4,]
feat
```

```
GRanges object with 1 range and 3 metadata columns:
        seqnames    ranges strand |     type   gene
_name     gene_id
           <Rle> <IRanges>  <Rle> | <factor> <chara
cter> <character>
  [1] Chromosome   190-252      + |      CDS
thrL ER3413_4519
  -------
  seqinfo: 1 sequence from an unspecified genome; n
o seqlengths
```

The metadata columns let us query the GRanges, for example for a particular gene.

In [63]:

```
subset(features, gene_name == "lacA")
# Equivalently:
#   features[features$gene_name == "lacA" & !is.na(features$ge
ne_name),]
```

```
GRanges object with 6 ranges and 3 metadata column
s:
        seqnames        ranges strand |        type
   gene_name     gene_id
           <Rle>     <IRanges>  <Rle> |    <factor>
<character> <character>
  [1] Chromosome 363147-363758      - |        gene
        lacA  ER3413_350
  [2] Chromosome 363147-363758      - |  transcript
        lacA  ER3413_350
  [3] Chromosome 363147-363758      - |        exon
        lacA  ER3413_350
  [4] Chromosome 363150-363758      - |         CDS
        lacA  ER3413_350
  [5] Chromosome 363756-363758      - | start_codon
        lacA  ER3413_350
  [6] Chromosome 363147-363149      - |  stop_codon
        lacA  ER3413_350
  -------
  seqinfo: 1 sequence from an unspecified genome; n
o seqlengths
```

Note: `subset` is a generic R function. It is also similar to dplyr's `filter`. The second argument is special, in it you can refer to columns of the GRanges directly.

We could also get all features of a particular type.

In [64]:

```
cds <- subset(features, type == "CDS")
cds
# Equivalently:
#   features[features$type == "CDS",]
```

```
GRanges object with 4052 ranges and 3 metadata colu
mns:
           seqnames         ranges strand |     ty
pe   gene_name     gene_id
              <Rle>      <IRanges>  <Rle> | <facto
r> <character> <character>
     [1] Chromosome        190-252      + |      C
DS        thrL ER3413_4519
     [2] Chromosome       337-2796      + |      C
DS        thrA    ER3413_1
     [3] Chromosome      2801-3730      + |      C
DS        thrB    ER3413_2
     [4] Chromosome      3734-5017      + |      C
DS        thrC    ER3413_3
     [5] Chromosome      5234-5527      + |      C
DS        yaaX    ER3413_4
     ...        ...            ...    ... .
...         ...         ...
  [4048] Chromosome 4553704-4555125      + |      C
DS        creC ER3413_4511
  [4049] Chromosome 4555186-4556535      + |      C
DS        creD ER3413_4512
  [4050] Chromosome 4556601-4557314      - |      C
DS        arcA ER3413_4513
  [4051] Chromosome 4557410-4557547      + |      C
DS        yjjY ER3413_4541
  [4052] Chromosome 4557950-4558633      + |      C
DS        yjtD ER3413_4514
  -------
  seqinfo: 1 sequence from an unspecified genome; n
o seqlengths
```

Further data types to explore

GRangesList, etc: Many Bioconductor types have a List version -- `GRangesList`, `DNAStringSetList`, etc. For example the exons of a collection of genes could be naturally stored in a `GRangesList`. Most functions that work with `GRanges` will also worked with `GRangesList`, and operate on each list element separately.

TxDb: `TxDb` objects represent the hierarchy of genes which contain transcripts which contain exons and CDS (CoDing Sequence) ranges. `TxDb` objects are provided by the `GenomicFeatures` package.

Seqinfo: `GRanges` (and various other types) may have associated sequence information accessed with `seqinfo()`. This contains the names and lengths of the sequences the ranges may refer to, and whether they are circular. It allows for some error checking if present.

Finding a known motif

AGGAGGU is the Shine-Dalgarno sequence, which assists binding of the ribosome to a transcript.

In [65]:

```
vmatchPattern("AGGAGGT", seqs)
```

```
MIndex object of length 1
$Chromosome
IRanges object with 63 ranges and 0 metadata column
s:
           start       end     width
       <integer> <integer> <integer>
   [1]     56593     56599         7
   [2]     67347     67353         7
   [3]    226876    226882         7
   [4]    229408    229414         7
   [5]    241665    241671         7
   ...       ...       ...       ...
  [59]   4312631   4312637         7
  [60]   4371930   4371936         7
  [61]   4410503   4410509         7
  [62]   4420666   4420672         7
  [63]   4484025   4484031         7
```

`vmatchPattern` is strand specific. If we want matches on the reverse strand we need to also:

In [66]:

```
vmatchPattern(reverseComplement(DNAString("AGGAGGT")), seqs)
```

```
MIndex object of length 1
$Chromosome
IRanges object with 76 ranges and 0 metadata column
s:
           start       end     width
       <integer> <integer> <integer>
   [1]     59133     59139         7
   [2]    125294    125300         7
   [3]    136473    136479         7
   [4]    226640    226646         7
   [5]    266770    266776         7
   ...       ...       ...       ...
  [72]   4139844   4139850         7
  [73]   4181244   4181250         7
  [74]   4241083   4241089         7
  [75]   4397026   4397032         7
  [76]   4473495   4473501         7
```

Demanding an exact match here is overly strict. `vmatchPattern` has arguments allowing inexact matches. Alternatively, there is a similar function for searching for a Position Weight Matrix pattern, `matchPWM`.

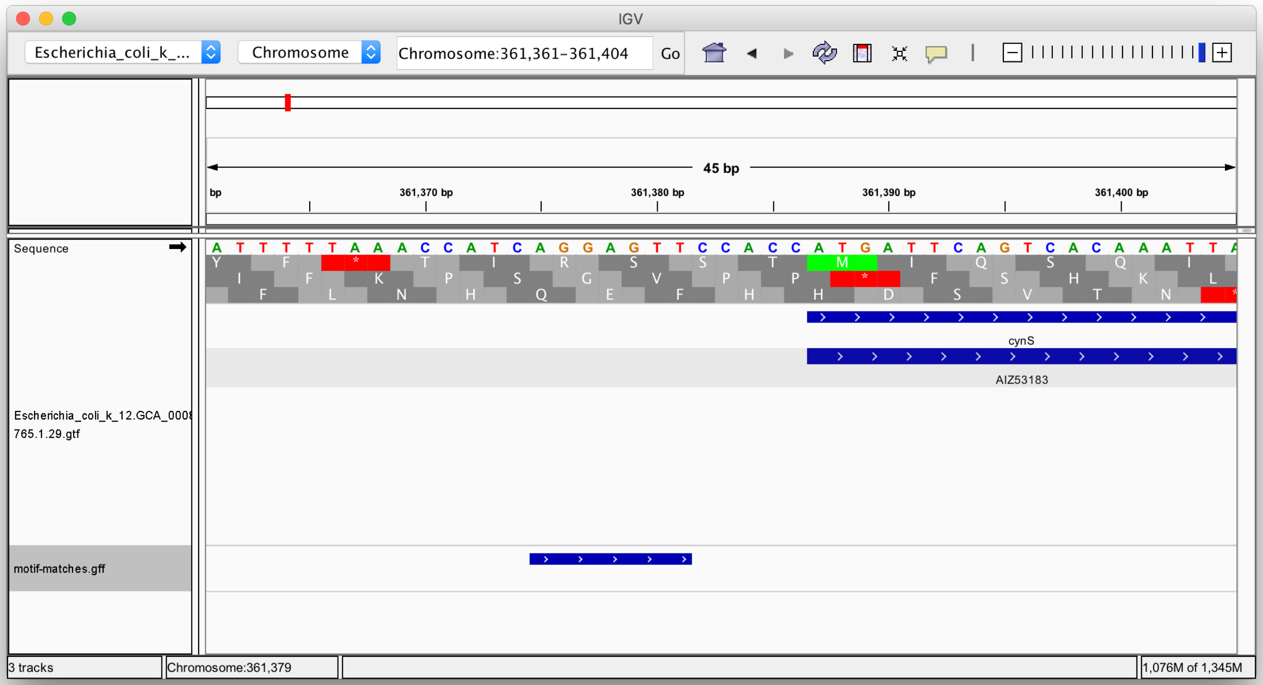
The following will search both strands, allowing one mismatch, and produce the result in convenient GRanges form:

In [67]:

```
query <- DNAString("AGGAGGT")
max.mismatch <- 1

fwd <- vmatchPattern(query, seqs, max.mismatch=max.mismatch)
fwd <- as(fwd, "GRanges")
strand(fwd) <- "+"
rev <- vmatchPattern(reverseComplement(query), seqs, max.misma
tch=max.mismatch)
rev <- as(rev, "GRanges")
strand(rev) <- "-"

complete <- c(fwd, rev)
complete

# Write to GFF file
export(complete, "motif-matches.gff")
```

```
GRanges object with 7534 ranges and 0 metadata colu
mns:
           seqnames           ranges strand
              <Rle>        <IRanges>  <Rle>
     [1] Chromosome          323-329      +
     [2] Chromosome        3540-3546      +
     [3] Chromosome        3765-3771      +
     [4] Chromosome        5374-5380      +
     [5] Chromosome        7641-7647      +
     ...        ...              ...    ...
  [7530] Chromosome 4550281-4550287      -
  [7531] Chromosome 4551603-4551609      -
  [7532] Chromosome 4551732-4551738      -
  [7533] Chromosome 4552223-4552229      -
  [7534] Chromosome 4552751-4552757      -
  -------
  seqinfo: 1 sequence from an unspecified genome; n
o seqlengths
```

We might then view this in the IGV genome browser:



[http://software.broadinstitute.org/software/igv/home (http://software.broadinstitute.org/software/igv/home)]

De novo motif finding

Let's try to "discover" the Shine-Dalgarno sequence for ourselves.

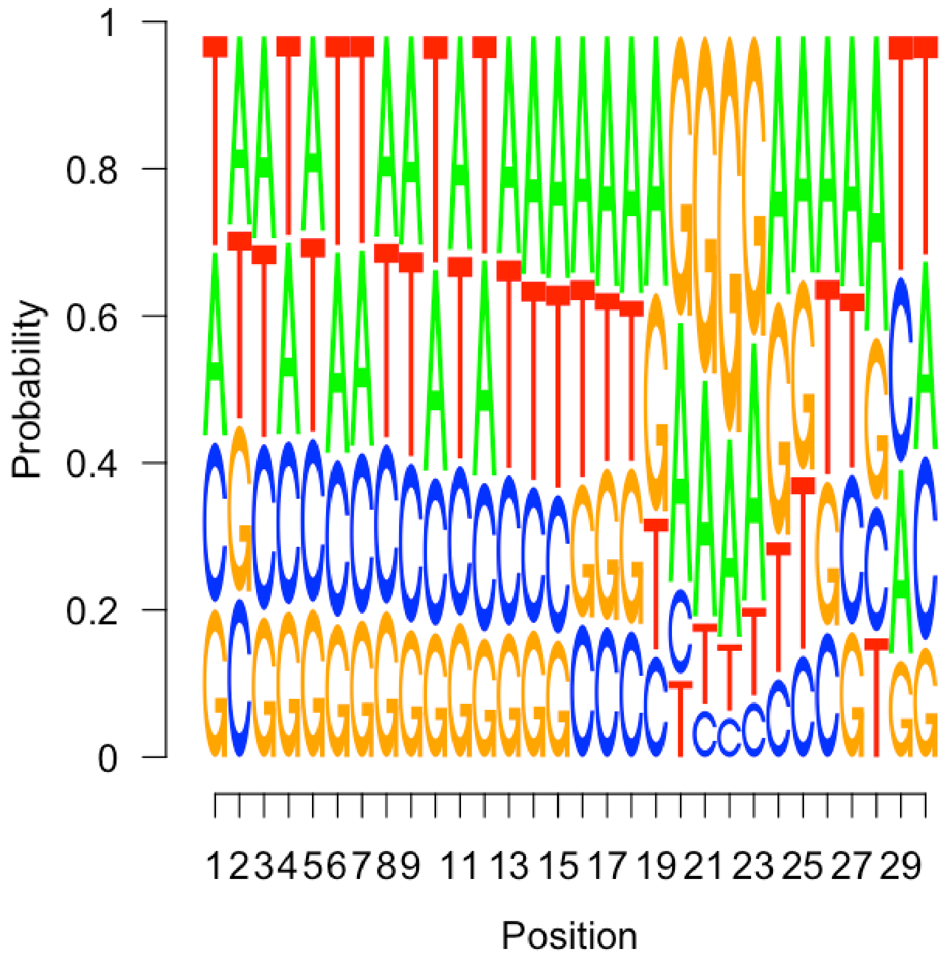
In [68]:

```
# Note: bacteria do not have introns
# In a eukaryote, you would need to merge CDS by transcript

size <- 30

initiation_regions <- flank(cds, size, start=TRUE)
initiation_seqs <- getSeq(seqs, initiation_regions)
names(initiation_seqs) <- initiation_regions$gene_id
```

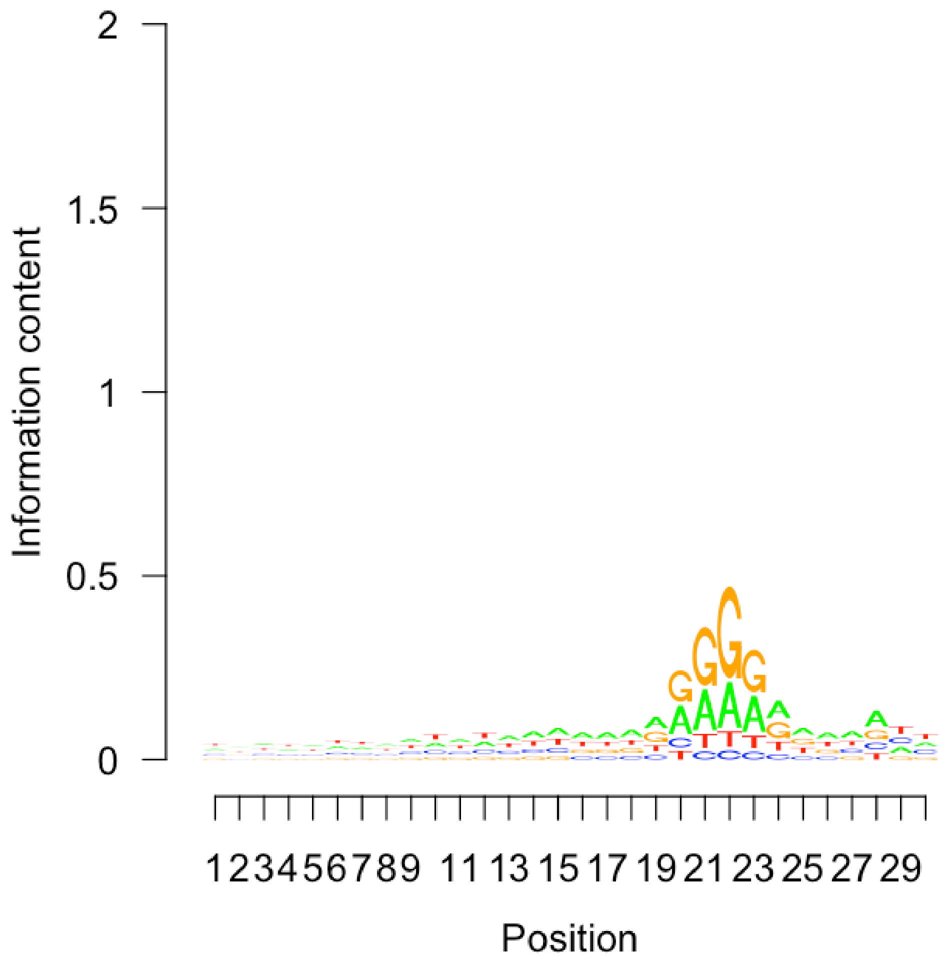
In [69]:

```
# Look for any composition bias
library(seqLogo)
letter_counts <- consensusMatrix(initiation_seqs)
probs <- prop.table(letter_counts[1:4,], 2)
seqLogo(probs, ic.scale=FALSE)
seqLogo(probs)
```

```
Loading required package: grid
```



Probability
0
0.2
0.4
0.6
0.8
1
1 2 3 4 5 6 7 8 9 11 13 15 17 19 21 23 25 27 29
Position


Information content
2
1.5
1
0.5
0
1 2 3 4 5 6 7 8 9 11 13 15 17 19 21 23 25 27 29
Position

Next steps

We've seen just the smallest part of what Bioconductor (http://bioconductor.org/) has to offer in this space.

- Most downloaded Bioconctor packages (http://bioconductor.org/packages/stats/)
- Bioconductor cheat sheet (https://github.com/mikelove/bioc-refcard/blob/master/README.Rmd)
- COMBINE Bioconductor course from May 2017 (https://combine-australia.github.io/2017-05-19-bioconductor-melbourne/)
- Bioconductor's Stack Overflow-style support site (https://support.bioconductor.org/)

Besides software, Bioconductor includes packages with data for model organisms, for example. The data is generally from these central repositories:

- NCBI's Entrez Gene gene database and Refseq reference sequences
- The EBI's Ensembl genome browser
- The UCSC genome browser

These organizations will generally obtain genome assemblies from the same ultimate sources. For example, all of the above use the Genome Reference Consortium's GRCh38 DNA sequence for homo sapiens. UCSC likes to call this "hg38" but it is the same DNA sequence. These DNA sequences serve as a common frame of reference. However the three organizations above will differ on their exact set of gene and transcript annotations, and all use a different gene and transcript ID system. These annotations are also revised more often than the underlying DNA sequences.

This mess is partly due to American/European rivalry, and partly due to differing goals. The UCSC genome browser has always been about practicality and showing many lines of evidence. The others are more concerned with careful curation and standardization.

Some example packages:

BSgenome.Hsapiens.UCSC.hg38

Biostrings genome, Homo sapiens, from the UCSC browser, version hg38.

DNA for chromosomes, usable in the same way as the DNAStringSet used above.

TxDb.Hsapiens.UCSC.hg38.knownGene

Transcript database, Homo sapiens, from UCSC browser, genome verison hg38, "knownGene" gene annotations.

GRanges information for genes and transcripts, much as we loaded from a GTF file above.

org.Hs.eg.db

Organism Homo sapiens, primary key is Entrez Gene, database.

Translation of gene ids from various databases, assignment to GO terms, KEGG pathways, etc. Entrez Gene ids are used as the primary key.

biomaRt

Access to BioMart data, on the internet -- translation of gene ids, gene sets, gene information, etc.

AnnotationHub

AnnotationHub is a way to retrieve data from a more comprehensive set of organisms and data providers than the above styles of package. The retrieved data is returned in an appropriate Bioconductor type. If data is being updated over time (eg improved annotation of a genome), each version receives a unique ID in AnnotationHub, making it much easier to write reproducable analyses.

AnnotationHub also provides access to experimental data which maps to locations on a genome, similar to the sorts of tracks you would load in the UCSC browser.

Files are cached, so they will only be downloaded once.

In [70]:

```r
library(AnnotationHub)
ah <- AnnotationHub()
```

```
Attaching package: 'AnnotationHub'

The following object is masked from 'package:Biobas
e':

    cache

updating metadata: retrieving 1 resource
snapshotDate(): 2018-04-30
```

```
In [71]:
```

```
# ah contains a large collection of records that can be retrie
ved
ah
length(ah)
colnames( mcols(ah) )
table( ah$rdataclass )
```

```
AnnotationHub with 44923 records
# snapshotDate(): 2018-04-30
# $dataprovider: BroadInstitute, Ensembl, UCSC, ft
p://ftp.ncbi.nlm.nih.gov/g...
# $species: Homo sapiens, Mus musculus, Drosophila
melanogaster, Bos taurus,...
# $rdataclass: GRanges, BigWigFile, FaFile, TwoBitF
ile, Rle, OrgDb, ChainFil...
# additional mcols(): taxonomyid, genome, descripti
on,
#   coordinate_1_based, maintainer, rdatadateadded,
preparerclass, tags,
#   rdatapath, sourceurl, sourcetype
# retrieve records with, e.g., 'object[["AH2"]]'

            title

  AH2     | Ailuropoda_melanoleuca.ailMel1.69.dna.t
oplevel.fa
  AH3     | Ailuropoda_melanoleuca.ailMel1.69.dna_r
m.toplevel.fa
  AH4     | Ailuropoda_melanoleuca.ailMel1.69.dna_s
m.toplevel.fa
  AH5     | Ailuropoda_melanoleuca.ailMel1.69.ncrn
a.fa
  AH6     | Ailuropoda_melanoleuca.ailMel1.69.pep.a
ll.fa
  ...       ...

  AH63653 | phastCons46wayPrimates.UCSC.hg19.chrUn_
gl000248.rds
  AH63654 | phastCons46wayPrimates.UCSC.hg19.chrUn_
gl000249.rds
  AH63655 | phastCons46wayPrimates.UCSC.hg19.chrX.r
ds
  AH63656 | phastCons46wayPrimates.UCSC.hg19.chrY.r
ds
  AH63657 | Alternative Splicing Annotation for Hom
o sapiens (Human)
```

44923

'title'   'dataprovider'   'species'   'taxonomyid'
'genome'   'description'   'coordinate_1_based'
'maintainer'   'rdatadateadded'   'preparerclass'   'tags'
'rdataclass'   'rdatapath'   'sourceurl'   'sourcetype'

```
      AAStringSet        BigWigFile            biopax
        ChainFile
                1             10247                 9
             1113
       data.frame             EnsDb            FaFile
          GRanges
               40               460              5122
            19550
           igraph     Inparanoid8Db              list
           MSnSet
                1               268                18
                1
         mzRident           mzRpwiz            OrgDb
              Rle
                1                 1              1691
             1852
SQLiteConnection        TwoBitFile              TxDb
          VcfFile
                1              4480                59
                8
```

In [72]:

```
# query() searches for terms in an unstructured way
records <- query(ah, c("Ensembl", "85", "Saccharomyces cerevis
iae"))
records
```

```
AnnotationHub with 7 records
# snapshotDate(): 2018-04-30
# $dataprovider: Ensembl
# $species: Saccharomyces cerevisiae
# $rdataclass: TwoBitFile, GRanges
# additional mcols(): taxonomyid, genome, descripti
on,
#   coordinate_1_based, maintainer, rdatadateadded,
preparerclass, tags,
#   rdatapath, sourceurl, sourcetype
# retrieve records with, e.g., 'object[["AH5108
7"]]'

            title

  AH51087 | Saccharomyces_cerevisiae.R64-1-1.85.abi
nitio.gtf
  AH51088 | Saccharomyces_cerevisiae.R64-1-1.85.gtf

  AH51396 | Saccharomyces_cerevisiae.R64-1-1.cdna.a
ll.2bit
  AH51397 | Saccharomyces_cerevisiae.R64-1-1.dna_r
m.toplevel.2bit
  AH51398 | Saccharomyces_cerevisiae.R64-1-1.dna_s
m.toplevel.2bit
  AH51399 | Saccharomyces_cerevisiae.R64-1-1.dna.to
plevel.2bit
  AH51400 | Saccharomyces_cerevisiae.R64-1-1.ncrna.
2bit
```

In [73]:

```
mcols(records)
mcols(records)[,c("title","rdataclass")]
```

```
DataFrame with 7 rows and 15 columns

     title dataprovider
                                                       <
character>  <character>
AH51087      Saccharomyces_cerevisiae.R64-1-1.85.ab
initio.gtf      Ensembl
AH51088               Saccharomyces_cerevisiae.R64-
1-1.85.gtf      Ensembl
AH51396        Saccharomyces_cerevisiae.R64-1-1.cdn
a.all.2bit      Ensembl
AH51397 Saccharomyces_cerevisiae.R64-1-1.dna_rm.top
level.2bit      Ensembl
AH51398 Saccharomyces_cerevisiae.R64-1-1.dna_sm.top
level.2bit      Ensembl
AH51399    Saccharomyces_cerevisiae.R64-1-1.dna.top
level.2bit      Ensembl
AH51400           Saccharomyces_cerevisiae.R64-1-1.
ncrna.2bit      Ensembl
                         species taxonomyid
                     <character>  <integer>
AH51087 Saccharomyces cerevisiae       4932
AH51088 Saccharomyces cerevisiae       4932
AH51396 Saccharomyces cerevisiae       4932
AH51397 Saccharomyces cerevisiae       4932
AH51398 Saccharomyces cerevisiae       4932
AH51399 Saccharomyces cerevisiae       4932
AH51400 Saccharomyces cerevisiae       4932

    genome
                                                       <
character>
AH51087      Saccharomyces_cerevisiae.R64-1-1.85.ab
initio.gtf
AH51088               Saccharomyces_cerevisiae.R64-
1-1.85.gtf
AH51396        Saccharomyces_cerevisiae.R64-1-1.cdn
a.all.2bit
AH51397 Saccharomyces_cerevisiae.R64-1-1.dna_rm.top
level.2bit
AH51398 Saccharomyces_cerevisiae.R64-1-1.dna_sm.top
level.2bit
AH51399    Saccharomyces_cerevisiae.R64-1-1.dna.top
```

```
level.2bit
AH51400           Saccharomyces_cerevisiae.R64-1-1.
ncrna.2bit
                                                desc
ription coordinate_1_based
                                                <cha
racter>          <integer>
AH51087       Gene Annotation for Saccharomyces cer
evisiae                 1
AH51088       Gene Annotation for Saccharomyces cer
evisiae                 1
AH51396  TwoBit cDNA sequence for Saccharomyces cer
evisiae                 1
AH51397   TwoBit DNA sequence for Saccharomyces cer
evisiae                 1
AH51398   TwoBit DNA sequence for Saccharomyces cer
evisiae                 1
AH51399   TwoBit DNA sequence for Saccharomyces cer
evisiae                 1
AH51400 TwoBit ncRNA sequence for Saccharomyces cer
evisiae                 1

maintainer rdatadateadded
                                                   <
character>    <character>
AH51087 Bioconductor Maintainer <maintainer@biocond
uctor.org>     2016-07-20
AH51088 Bioconductor Maintainer <maintainer@biocond
uctor.org>     2016-07-20
AH51396 Bioconductor Maintainer <maintainer@biocond
uctor.org>     2016-08-15
AH51397 Bioconductor Maintainer <maintainer@biocond
uctor.org>     2016-08-15
AH51398 Bioconductor Maintainer <maintainer@biocond
uctor.org>     2016-08-15
AH51399 Bioconductor Maintainer <maintainer@biocond
uctor.org>     2016-08-15
AH51400 Bioconductor Maintainer <maintainer@biocond
uctor.org>     2016-08-15
                   preparerclass
                     <character>
AH51087 EnsemblGtfImportPreparer
AH51088 EnsemblGtfImportPreparer
AH51396    EnsemblTwoBitPreparer
```

```
AH51397    EnsemblTwoBitPreparer
AH51398    EnsemblTwoBitPreparer
AH51399    EnsemblTwoBitPreparer
AH51400    EnsemblTwoBitPreparer

        tags  rdataclass

      <list> <character>
AH51087 c("GTF", "ensembl", "Gene", "Transcript",
"Annotation")     GRanges
AH51088 c("GTF", "ensembl", "Gene", "Transcript",
"Annotation")     GRanges
AH51396     c("TwoBit", "ensembl", "sequence", "2bi
t", "FASTA")  TwoBitFile
AH51397     c("TwoBit", "ensembl", "sequence", "2bi
t", "FASTA")  TwoBitFile
AH51398     c("TwoBit", "ensembl", "sequence", "2bi
t", "FASTA")  TwoBitFile
AH51399     c("TwoBit", "ensembl", "sequence", "2bi
t", "FASTA")  TwoBitFile
AH51400     c("TwoBit", "ensembl", "sequence", "2bi
t", "FASTA")  TwoBitFile


    rdatapath



  <character>
AH51087                 release-85/gtf/saccharomyce
s_cerevisiae/Saccharomyces_cerevisiae.R64-1-1.85.ab
initio.gtf.gz
AH51088                          release-85/gtf/sac
charomyces_cerevisiae/Saccharomyces_cerevisiae.R64-
1-1.85.gtf.gz
AH51396       ensembl/release-85/fasta/saccharomyce
s_cerevisiae/cdna/Saccharomyces_cerevisiae.R64-1-1.
cdna.all.2bit
AH51397 ensembl/release-85/fasta/saccharomyces_cere
visiae/dna/Saccharomyces_cerevisiae.R64-1-1.dna_rm.
toplevel.2bit
AH51398 ensembl/release-85/fasta/saccharomyces_cere
visiae/dna/Saccharomyces_cerevisiae.R64-1-1.dna_sm.
toplevel.2bit
AH51399    ensembl/release-85/fasta/saccharomyces_c
```

```
erevisiae/dna/Saccharomyces_cerevisiae.R64-1-1.dna.
toplevel.2bit
AH51400          ensembl/release-85/fasta/saccharomy
ces_cerevisiae/ncrna/Saccharomyces_cerevisiae.R64-1
-1.ncrna.2bit


                       sourceurl


                     <character>
AH51087          ftp://ftp.ensembl.org/pub/release-
85/gtf/saccharomyces_cerevisiae/Saccharomyces_cerev
isiae.R64-1-1.85.abinitio.gtf.gz
AH51088                   ftp://ftp.ensembl.org/pu
b/release-85/gtf/saccharomyces_cerevisiae/Saccharom
yces_cerevisiae.R64-1-1.85.gtf.gz
AH51396       ftp://ftp.ensembl.org/pub/release-85/
fasta/saccharomyces_cerevisiae/cdna/Saccharomyces_c
erevisiae.R64-1-1.cdna.all.fa.gz
AH51397 ftp://ftp.ensembl.org/pub/release-85/fasta/
saccharomyces_cerevisiae/dna/Saccharomyces_cerevisi
ae.R64-1-1.dna_rm.toplevel.fa.gz
AH51398 ftp://ftp.ensembl.org/pub/release-85/fasta/
saccharomyces_cerevisiae/dna/Saccharomyces_cerevisi
ae.R64-1-1.dna_sm.toplevel.fa.gz
AH51399    ftp://ftp.ensembl.org/pub/release-85/fas
ta/saccharomyces_cerevisiae/dna/Saccharomyces_cerev
isiae.R64-1-1.dna.toplevel.fa.gz
AH51400          ftp://ftp.ensembl.org/pub/release-8
5/fasta/saccharomyces_cerevisiae/ncrna/Saccharomyce
s_cerevisiae.R64-1-1.ncrna.fa.gz
         sourcetype
        <character>
AH51087         GTF
AH51088         GTF
AH51396       FASTA
AH51397       FASTA
AH51398       FASTA
AH51399       FASTA
AH51400       FASTA
```

```
DataFrame with 7 rows and 2 columns

     title  rdataclass
                                                       <
character> <character>
AH51087      Saccharomyces_cerevisiae.R64-1-1.85.ab
initio.gtf     GRanges
AH51088               Saccharomyces_cerevisiae.R64-
1-1.85.gtf     GRanges
AH51396        Saccharomyces_cerevisiae.R64-1-1.cdn
a.all.2bit  TwoBitFile
AH51397 Saccharomyces_cerevisiae.R64-1-1.dna_rm.top
level.2bit  TwoBitFile
AH51398 Saccharomyces_cerevisiae.R64-1-1.dna_sm.top
level.2bit  TwoBitFile
AH51399    Saccharomyces_cerevisiae.R64-1-1.dna.top
level.2bit  TwoBitFile
AH51400            Saccharomyces_cerevisiae.R64-1-1.
ncrna.2bit  TwoBitFile
```

In [ ]:

```
# Having located records of interest,
# your R script can refer to the specific AH... record,
# so it always uses the same version of the data.
ah[["AH51399"]]
sc_genome <- import( ah[["AH51399"]] )
sc_granges <- ah[["AH51088"]]
```

```
downloading 1 resources
retrieving 1 resource
```

In [ ]:

```
# More recent versions of Bioconductor also allow you to
# retrieve TxDb (and similar EnsDb) objects.



query(ah, c("OrgDb", "Saccharomyces cerevisiae"))
sc_orgdb <- ah[["AH49589"]]
```

Tutorial based on input from:

https://al2na.github.io/compgenr/ (https://al2na.github.io/compgenr/)

https://monashbioinformaticsplatform.github.io/r-more/topics/sequences_and_features.html (https://monashbioinformaticsplatform.github.io/r-more/topics/sequences_and_features.html)