Package 'hiAnnotator'

September 19, 2024

Title Functions for annotating GRanges objects

Version 1.39.0

Date 2021-07-27

Author Nirav V Malani <malnirav@gmail.com>

Maintainer Nirav V Malani <malnirav@gmail.com>

Description hiAnnotator contains set of functions which allow users to annotate a GRanges object with custom set of annotations. The basic philosophy of this package is to take two GRanges objects (query & subject) with common set of seqnames (i.e. chromosomes) and return associated annotation per seqnames and rows from the query matching seqnames and rows from the subject (i.e. genes or cpg islands). The package comes with three types of annotation functions which calculates if a position from query is: within a feature, near a feature, or count features in defined window sizes. Moreover, each function is equipped with parallel backend to utilize the foreach package. In addition, the package is equipped with wrapper functions, which finds appropriate columns needed to make a GRanges object from a common data frame.

Depends GenomicRanges, R (>= 2.10)

Imports foreach, iterators, rtracklayer, dplyr, BSgenome, ggplot2, scales, methods

License GPL (>= 2)

VignetteBuilder knitr

Suggests knitr, doParallel, testthat, BiocGenerics, markdown

biocViews Software, Annotation

LazyLoad yes

RoxygenNote 7.1.1

git_url https://git.bioconductor.org/packages/hiAnnotator

git_branch devel

git_last_commit 946eaa3

git_last_commit_date 2024-04-30

Repository Bioconductor 3.20

Date/Publication 2024-09-18

Contents

`.checkArgsSetDefaults` *Check args and set defaults.*

Description

This function checks all the arguments passed to an annotation function and set default values for later use. Evaluation of this function happens in the parent function.

Usage

```
.checkArgsSetDefaults()
```

`.mergeAndReturn` *Merge results back to the query object and perform additional post processing steps.*

Description

This function merges all the calculation results back to the query object. Additionally, if any flags were set, the function does the necessary checks and processing to format the return object as required. Evaluation of this function happens in the parent function.

Usage

```
.mergeAndReturn()
```

cleanColname *Clean the supplied string from punctuations and spaces.*

Description

Function to clean the supplied string from punctuations and spaces so it can be used as column headings.

Usage

```
cleanColname(x, description = NULL)
```

Arguments

x	string or a vector to be cleaned.
description	OPTIONAL string identifying the purpose of the supplied string in x to be displayed in the cleaning message. This triggers a message.

Value

cleaned string or a vector.

See Also

getFeatureCounts, makeGRanges, getNearestFeature, getSitesInFeature.

Examples

```
cleanColname("HIV-test")
cleanColname("HIV*test")
cleanColname("HIV-test", "myAlias")
```

doAnnotation *Annotate a GRanges object using one of annotation functions.*

Description

This is a wrapper function which calls one of following functions depending on annotType parameter: getFeatureCounts, getFeatureCountsBig, getNearestFeature, get2NearestFeature, getSitesInFeature

Usage

```
doAnnotation(
  annotType = NULL,
  ...,
  postProcessFun = NULL,
  postProcessFunArgs = list()
)
```

Arguments

annotType	one of following: within, nearest, twoNearest, counts, countsBig.
...	Additional parameters to be passed to the respective annotation function.
postProcessFun	function to call on the resulting object for any post processing steps.
postProcessFunArgs	additional arguments for postProcessFun as a list.

Value

a GRanges object with new annotation columns appended at the end of sites.rd.

See Also

makeGRanges, getFeatureCounts, getFeatureCountsBig, getNearestFeature, get2NearestFeature, getSitesInFeature.

Examples

```
# Convert a dataframe to GRanges object
data(sites)
alldata.rd <- makeGRanges(sites, soloStart = TRUE)

data(genes)
genes.rd <- makeGRanges(genes)

doAnnotation(annotType = "within", alldata.rd, genes.rd, "InGene", asBool = TRUE)
## Not run:
doAnnotation(annotType = "counts", alldata.rd, genes.rd, "NumOfGene")
doAnnotation(annotType = "nearest", alldata.rd, genes.rd, "NearestGene")
doAnnotation(annotType = "countsBig", alldata.rd, genes.rd, "ChipSeqCounts")
geneCheck <- function(x,wanted) { x$isWantedGene <- x$InGene %in% wanted;
return(x) }
doAnnotation(annotType = "within", alldata.rd, genes.rd, "InGene",
postProcessFun = geneCheck,
postProcessFunArgs = list("wanted" = c("FOXJ3", "SEPT9", "RPTOR")) )

## End(Not run)
```

---

genes *Sample RefSeq genes annotation*

Description

A sample annotation containing collection of genes from RefSeq database in the human genome mapped to UCSC freeze hg18. See UCSC table description page for the details regarding the column headings.

Format

A data frame with 33965 rows and 9 variables

Source

http://genome.ucsc.edu/cgi-bin/hgTables?db=hg18&hgta_table=refGene&hgta_doSchema=describe+table+schema

---

`get2NearestFeature` *Get two nearest upstream and downstream annotation boundary for a position range.*

Description

Given a query object, the function retrieves the two nearest feature upstream and downstream along with their properties from a subject and then appends them as new columns within the query object. When used in genomic context, the function can be used to retrieve two nearest gene upstream and downstream of the genomic position of interest.

Usage

```
get2NearestFeature(
  sites.rd,
  features.rd,
  colnam = NULL,
  side = "either",
  feature.colnam = NULL,
  relativeTo = "subject"
)
```

Arguments

`sites.rd`	GRanges object to be used as the query.
`features.rd`	GRanges object to be used as the subject or the annotation table.
`colnam`	column name to be added to sites.rd for the newly calculated annotation...serves a core!
`side`	boundary of annotation to use to calculate the nearest distance. Options are '5p','3p', 'either'(default), or 'midpoint'.
`feature.colnam`	column name from features.rd to be used for retrieving the nearest feature name. By default this is NULL assuming that features.rd has a column that includes the word 'name' somewhere in it.
`relativeTo`	calculate distance relative to query or subject. Default is 'subject'. See documentation of `getNearestFeature` for more information.

Value

a GRanges object with new annotation columns appended at the end of sites.rd.

Note

- When side='midpoint', the distance to nearest feature is calculated by (start+stop)/2.
- For cases where a position is at the edge and there are no feature up/down stream since it would fall off the chromosome, the function simply returns 'NA'.
- If there are multiple locations where a query falls into, the function arbitrarily chooses one to serve as the nearest feature, then reports 2 upstream & downstream feature. That may occasionally yield features which are the same upstream and downstream, which is commonly encountered when studying spliced genes or phenomena related to it.
- If strand information doesn't exist, then everything is defaults to '+' orientation (5' -> 3')
- If parallel=TRUE, then be sure to have a parallel backend registered before running the function. One can use any of the following libraries compatible with foreach: doMC, doSMP, doSNOW, doMPI, doParallel. For example: library(doMC); registerDoMC(2)

See Also

getNearestFeature, makeGRanges, getFeatureCounts, getSitesInFeature.

Examples

```
# Convert a dataframe to GRanges object
data(sites)
alldata.rd <- makeGRanges(sites, soloStart = TRUE)

data(genes)
genes.rd <- makeGRanges(genes)

nearestGenes <- get2NearestFeature(alldata.rd, genes.rd, "NearestGene")
nearestGenes
## Not run:
nearestGenes <- get2NearestFeature(alldata.rd, genes.rd, "NearestGene",
side = "5p")
nearestGenes
nearestGenes <- get2NearestFeature(alldata.rd, genes.rd, "NearestGene",
side = "3p")
nearestGenes

## End(Not run)
```

---

getFeatureCounts *Get counts of annotation within a defined window around each query range or positions.*

Description

Given a query object and window size(s), the function finds all the rows in subject which are <= window size/2 distance away. If weights are assigned to each positions in the subject, then tallied counts are multiplied accordingly. For large annotations, use getFeatureCountsBig.

Usage

```
getFeatureCounts(
  sites.rd,
  features.rd,
  colnam = NULL,
  chromSizes = NULL,
  widths = c(1000, 10000, 1e+06),
  weightsColname = NULL,
  doInChunks = FALSE,
  chunkSize = 10000,
  parallel = FALSE
)
```

Arguments

sites.rd	GRanges object to be used as the query.
features.rd	GRanges object to be used as the subject or the annotation table.
colnam	column name to be added to sites.rd for the newly calculated annotation...serves as a prefix to windows sizes!
chromSizes	named vector of chromosome/seqnames sizes to be used for testing if a position is off the mappable region. DEPRECATED and will be removed in future release.
widths	a named/numeric vector of window sizes to be used for casting a net around each position. Default: c(1000,10000,1000000).
weightsColname	if defined, weigh each row from features.rd when tallying up the counts.
doInChunks	break up sites.rd into small pieces of chunkSize to perform the calculations. Default is FALSE. Useful if you are expecting to find great deal of overlap between sites.rd and features.rd.
chunkSize	number of rows to use per chunk of sites.rd. Default to 10000. Only used if doInChunks=TRUE.
parallel	use parallel backend to perform calculation with foreach. Defaults to FALSE. If no parallel backend is registered, then a serial version of foreach is ran using registerDoSEQ.

Value

a GRanges object with new annotation columns appended at the end of sites.rd. There will be a column for each width defined in widths parameter. If widths was a named vector i.e. c("100bp"=100,"1K"=1000), then the colname parameter will be pasted together with width name else default name will be generated by the function.

Note

- If parallel=TRUE, then be sure to have a parallel backend registered before running the function. One can use any of the following libraries compatible with foreach: doMC, doSMP, doSNOW, doMPI. For example: library(doMC); registerDoMC(2)

See Also

makeGRanges, getNearestFeature, getSitesInFeature, getFeatureCountsBig.

Examples

```
# Convert a dataframe to GRanges object
data(sites)
alldata.rd <- makeGRanges(sites, soloStart = TRUE)

data(genes)
genes.rd <- makeGRanges(genes)

geneCounts <- getFeatureCounts(alldata.rd, genes.rd, "NumOfGene")
## Not run:
geneCounts <- getFeatureCounts(alldata.rd, genes.rd, "NumOfGene",
doInChunks = TRUE, chunkSize = 200)
geneCounts
## Parallel version of getFeatureCounts
# geneCounts <- getFeatureCounts(alldata.rd, genes.rd, "NumOfGene",
parallel = TRUE)
# geneCounts

## End(Not run)
```

`getFeatureCountsBig` *Get counts of annotation within a defined window around each query range/position for large annotation objects spanning greater than 1 billion rows.*

Description

Given a query object and window size(s), the function finds all the rows in subject which are <= window size/2 distance away. Note that here counting is done using midpoint of the ranges in query instead of start-stop boundaries. The counts will differ slightly when compared to `getFeatureCounts`.

Usage

```
getFeatureCountsBig(
  sites.rd,
  features.rd,
  colnam = NULL,
  widths = c(1000, 10000, 1e+06)
)
```

Arguments

`sites.rd`	GRanges object to be used as the query.
`features.rd`	GRanges object to be used as the subject or the annotation table.
`colnam`	column name to be added to sites.rd for the newly calculated annotation...serves as a prefix to windows sizes!
`widths`	a named/numeric vector of window sizes to be used for casting a net around each position. Default: `c(1000,10000,1000000)`

Value

a GRanges object with new annotation columns appended at the end of sites.rd. There will be a column for each width defined in widths parameter. If widths was a named vector i.e. c("100bp"=100,"1K"=1000), then the colname parameter will be pasted together with width name else default name will be generated by the function.

See Also

makeGRanges, getNearestFeature, getSitesInFeature, getFeatureCounts.

Examples

```
# Convert a dataframe to GRanges object
data(sites)
alldata.rd <- makeGRanges(sites, soloStart = TRUE)

data(genes)
genes.rd <- makeGRanges(genes)

geneCounts1 <- getFeatureCounts(alldata.rd, genes.rd, "NumOfGene")
## Not run:
geneCounts2 <- getFeatureCountsBig(alldata.rd, genes.rd, "NumOfGene")
identical(geneCounts1, geneCounts2)

## End(Not run)
```

---

getLowestDists *Get the lowest biological distance from the 5' or 3' boundaries of query and subject.*

Description

Given a query and subject with indicies from nearest, calculate the shortest biological distance to either boundaries of the query and subject. This is a helper function utilized in getNearestFeature, get2NearestFeature

Usage

```
getLowestDists(
  query = NULL,
  subject = NULL,
  res.nrst = NULL,
  side = "either",
  relativeTo = "subject"
)
```

Arguments

query	GRanges object to be used as the query which holds data for 'queryHits' attribute of res.nrst.
subject	GRanges object to be used as the subject which holds data for 'subjectHits' attribute of res.nrst.

res.nrst	a dataframe of nearest indices as returned by nearest.
side	boundary of subject/annotation to use to calculate the nearest distance. Options are '5p','3p', or the default 'either'.
relativeTo	calculate distance relative to query or subject. Default is 'subject'. See documentation of getNearestFeature for more information.

Value

res.nrst with lowest distances appended at the end.

Note

for cases where a query has multiple nearest neighbors or overlaps with >1 subjects, the function will choose the subject with the lowest absolute distance.

See Also

getNearestFeature, get2NearestFeature.

Examples

```
query <- GRanges("A", IRanges(c(1, 5, 12, 20), width = 1),
strand = c("-", "+", "-", "+"))
subject <- GRanges("A", IRanges(c(1, 5, 10, 15, 21), width = 8:4),
strand = c("+", "+", "-", "-", "-"))
res <- as.data.frame(nearest(query, subject, select = "all",
ignore.strand = TRUE))
res <- getLowestDists(query, subject, res, "either", "query")
```

---

getNearestFeature *Get nearest annotation boundary for a position range.*

Description

Given a query object, the function retrieves the nearest feature and its properties from a subject and then appends them as new columns within the query object. When used in genomic context, the function can be used to retrieve the nearest gene 5' or 3' end relative to genomic position of interest.

Usage

```
getNearestFeature(
  sites.rd,
  features.rd,
  colnam = NULL,
  side = "either",
  feature.colnam = NULL,
  dists.only = FALSE,
  parallel = FALSE,
  relativeTo = "subject"
)
```

Arguments

sites.rd	GRanges object to be used as the query.
features.rd	GRanges object to be used as the subject or the annotation table.
colnam	column name to be added to sites.rd for the newly calculated annotation...serves a core!
side	boundary of annotation to use to calculate the nearest distance. Options are '5p','3p', 'either'(default), or 'midpoint'.
feature.colnam	column name from features.rd to be used for retrieving the nearest feature name. By default this is NULL assuming that features.rd has a column that includes the word 'name' somewhere in it.
dists.only	flag to return distances only. If this is TRUE, then 'feature.colnam' is not required and only distance to the nearest feature will be returned. By default this is FALSE.
parallel	use parallel backend to perform calculation with foreach. Defaults to FALSE. If no parallel backend is registered, then a serial version of foreach is ran using registerDoSEQ.
relativeTo	calculate distance relative to query or subject. Default is 'subject'. This essentially means whether to use query or subject as the anchor point to get distance from!

Value

a GRanges object with new annotation columns appended at the end of sites.rd.

Note

- When side='midpoint', the distance to nearest feature is calculated by (start+stop)/2.
- If strand information doesn't exist, then everything is defaulted to '+' orientation (5' -> 3')
- If parallel=TRUE, then be sure to have a parallel backend registered before running the function. One can use any of the following libraries compatible with foreach: doMC, doSMP, doSNOW, doMPI, doParallel. For example: library(doMC); registerDoMC(2)
- When relativeTo="subject", the biological distance is relative to subject, meaning, the function reports the distance to query from subject (i.e. an integration site is upstream or downstream from a gene). When relativeTo="query", the distance is from the point of view of query or an integration site (i.e. gene is upstream or downstream from an integration site).

See Also

makeGRanges, getFeatureCounts, getSitesInFeature, get2NearestFeature.

Examples

```
# Convert a dataframe to GRanges object
data(sites)
alldata.rd <- makeGRanges(sites, soloStart = TRUE)

data(genes)
genes.rd <- makeGRanges(genes)

nearestGenes <- getNearestFeature(alldata.rd, genes.rd, "NearestGene")
```

```
nearestGenes
nearestGenes <- getNearestFeature(alldata.rd, genes.rd, "NearestGene",
side = "5p")
nearestGenes
## Not run:
nearestGenes <- getNearestFeature(alldata.rd, genes.rd, "NearestGene",
side = "3p")
nearestGenes
nearestGenes <- getNearestFeature(alldata.rd, genes.rd, "NearestGene",
side = "midpoint")
## Parallel version of getNearestFeature
nearestGenes <- getNearestFeature(alldata.rd, genes.rd, "NearestGene",
parallel = TRUE)
nearestGenes

## End(Not run)
```

`getRelevantCol` *Find the column index of interest given the potential choices.*

Description

The function finds relevant column(s) of interest from a vector of column names derived from a dataframe. If no usable column is found, the function spits out a relevant error or returns the index of the usable column(s). This is an assistant function called by functions listed in the see also section.

Usage

```
getRelevantCol(col.names, col.options, col.type = NULL, multiple.ok = FALSE)
```

Arguments

`col.names`	column names from a dataframe
`col.options`	potential column names or partial names that may exist in col.names
`col.type`	type of column information the function is searching for, used in construction of error messages. Default is NULL.
`multiple.ok`	if multiple matches are found then return indices, else spit an error out. Default is TRUE.

Value

the index of usable column(s) or an error if no applicable column is found.

See Also

`makeGRanges`, `getNearestFeature`, `getSitesInFeature`.

Examples

```
data(sites)
names(sites)
getRelevantCol(names(sites), c("chr", "chromosome", "tname", "seqnames",
"chrom","contig"),"seqnames")
getRelevantCol(names(sites), c("ort", "orientation", "strand"), "strand")
```

`getSitesInFeature` *Find overlapping positions/ranges that match between the query and subject.*

Description

When used in genomic context, the function annotates genomic positions of interest with information like if they were in a gene or cpg island or whatever annotation that was supplied in the subject.

Usage

```
getSitesInFeature(
  sites.rd,
  features.rd,
  colnam = NULL,
  asBool = FALSE,
  feature.colnam = NULL,
  parallel = FALSE,
  allSubjectCols = FALSE,
  overlapType = "any"
)
```

Arguments

`sites.rd`	GRanges object to be used as the query.
`features.rd`	GRanges object to be used as the subject or the annotation table.
`colnam`	column name to be added to sites.rd for the newly calculated annotation...serves a core! If allSubjectCols=TRUE, then this is used as a prefix to all metadata column.
`asBool`	Flag indicating whether to return results as TRUE/FALSE or the property of an overlapping feature..namely feature name and orientation if available. Defaults to FALSE.
`feature.colnam`	column name from features.rd to be used for retrieving the feature name. By default this is NULL assuming that features.rd has a column that includes the word 'name' somewhere in it. Not required if asBool=TRUE or allSubjectCols=TRUE
`parallel`	use parallel backend to perform calculation with `foreach`. Defaults to FALSE. Not applicable when asBool=T. If no parallel backend is registered, then a serial version of foreach is ran using `registerDoSEQ`.
`allSubjectCols`	Flag indicating whether to return all annotations or metadata columns from features.rd. Defaults to FALSE.
`overlapType`	see `findOverlaps`. Defaults to 'any'

Value

a GRanges object with new annotation columns appended at the end of sites.rd.

Note

- If parallel=TRUE, then be sure to have a parallel backend registered before running the function. One can use any of the following libraries compatible with foreach: doMC, doSMP, doSNOW, doMPI. For example: library(doMC); registerDoMC(2)

See Also

makeGRanges, getFeatureCounts, getNearestFeature.

Examples

```
# Convert a dataframe to GRanges object
data(sites)
alldata.rd <- makeGRanges(sites, soloStart = TRUE)

data(genes)
genes.rd <- makeGRanges(genes)

InGenes <- getSitesInFeature(alldata.rd, genes.rd, "InGene")
InGenes
## Not run:
InGenes <- getSitesInFeature(alldata.rd, genes.rd, "InGene", asBool = TRUE)
InGenes
## Parallel version of getSitesInFeature
InGenes <- getSitesInFeature(alldata.rd, genes.rd, "InGene", asBool = TRUE,
parallel = TRUE)
InGenes
InGenes <- getSitesInFeature(alldata.rd, genes.rd, "InGene",
allSubjectCols = TRUE, parallel = TRUE)
InGenes

## End(Not run)
```

---

getUCSCtable *Obtain a UCSC annotation table given the table & track name.*

Description

Obtain a UCSC annotation table given the table & track name.

Usage

```
getUCSCtable(tableName, trackName, bsession = NULL, freeze = "hg19", ...)
```

Arguments

tableName	Name of the annotation table as it appears on UCSC browser.
trackName	Name of the track annotation table as it appears in on UCSC browser.
bsession	UCSC session object returned by makeUCSCsession or browserSession. If left NULL the function will call makeUCSCsession with the provided freeze to initiate a session.
freeze	one of following: hg19, mm8, rheM, etc. Default is hg19.
...	Arguments to be passed to ucscTableQuery.

Value

a dataframe containing the annotation data.

See Also

makeUCSCsession, getNearestFeature, getSitesInFeature.

Examples

```
## Not run:
refflat <- getUCSCtable("refFlat", "RefSeq Genes")
## same as above ##
refflat <- getUCSCtable("refFlat", "RefSeq Genes",
bsession=session,freeze="hg19")

## End(Not run)
```

---

getWindowLabel *Generate a window size label.*

Description

Function to generate aesthetically pleasing window size label given an integer. This is one of the helper function used in getFeatureCounts & getFeatureCountsBig.

Usage

```
getWindowLabel(x)
```

Arguments

x	vector of integers to generate the labels for.

Value

a character vector of length(x) which has x normalized and suffixed by bp, Kb, Mb, or Gb depending on respective interval sizes.

See Also

getFeatureCounts, makeGRanges, getNearestFeature, getSitesInFeature.

Examples

```
getWindowLabel(c(0, 1e7, 1e3, 1e6, 2e9))
```

`hiAnnotator` *Annotating GRanges objects with hiAnnotator.*

Description

hiAnnotator contains set of functions which allow users to annotate a GRanges object with custom set of annotations. The basic philosophy of this package is to take two GRanges objects (query & subject) with common set of seqnames (i.e. chromosomes) and return associated annotation per seqnames and rows from the query matching seqnames and rows from the subject (i.e. genes or cpg islands). The package comes with three types of annotation functions which calculates if a position from query is: within a feature, near a feature, or count features in defined window sizes. Moreover, one can utilize parallel backend for each annotation function to utilize the foreach package. In addition, the package is equipped with wrapper functions, which finds appropriate columns needed to make a GRanges object from a common dataframe.

Author(s)

Nirav V Malani

`makeChunks` *Breaks two GRanges objects into chunks of N size.*

Description

Given a query and subject GRanges objects, the function breaks query into chunks of N size where each chunk has a respective subject object filtered by seqnames present in the query chunk. This is a helper function used by one of the annotation function in 'See Also' section where each chunk is sent to a parallel node for processing.

Usage

```
makeChunks(sites.rd, features.rd, chunkSize = NULL)
```

Arguments

`sites.rd`	a GRanges object.
`features.rd`	a GRanges object.
`chunkSize`	number of rows to use per chunk of query. Default to length(sites.rd)/detectCores() or length(query)/getDoParWorkers() depending on parallel backend registered.

Value

a list of GRanges objects where each element is of length 2 representing query & subject chunks.

See Also

`makeGRanges`, `doAnnotation`, `getNearestFeature`, `getSitesInFeature`, `getFeatureCounts`.

Examples

```
data(sites)
data(genes)
sites <- makeGRanges(sites, soloStart = TRUE)
genes <- makeGRanges(genes)
makeChunks(sites, genes)
```

`makeGRanges` *Make a sorted GRanges object from a dataframe.*

Description

The function converts a dataframe into a GRanges object without too much hassle of renaming column names. The function finds column names that sound like seqname, chromosome, start, stop, position, etc and puts them in respective slots to facilitate the conversion of a dataframe to a GRanges object. If more than one column that sounds like start, stop, or position is present, the function will use the first match as the representative. It is recommended to run this function before utilizing any other annotation functions since it will sort the object by chromosome and position for copying annotations back to their respective rows confidently.

Usage

```
makeGRanges(
  x,
  freeze = NULL,
  positionsOnly = FALSE,
  soloStart = FALSE,
  chromCol = NULL,
  strandCol = NULL,
  startCol = NULL,
  stopCol = NULL,
  keepFactors = FALSE
)
```

Arguments

`x`	dataframe to be converted into a GRanges object
`freeze`	UCSC genome version of the data in x. Default is NULL. This parameter is generally used to populate seqinfo slot of GRanges objects.
`positionsOnly`	boolean flag indicating to return only position based data or everything from the dataframe. Defaults to FALSE.
`soloStart`	flag denoting whether only one position based column is available. In other words, only starts are present and no stops. Default=FALSE.
`chromCol`	use the defined column name for seqname/chromosome based data from the dataframe. Defaults to NULL.
`strandCol`	use the defined column name for strand or orientation from the dataframe. Defaults to NULL.
`startCol`	use the defined column name for start coordinate from the dataframe. Defaults to NULL.

stopCol	use the defined column name for stop coordinate from the dataframe. Defaults to NULL and not required if soloStart=TRUE.
keepFactors	keep vectors/columns stored as factors? Defaults to FALSE

Value

a GRanges object converted from x.

See Also

getNearestFeature, getFeatureCounts, getSitesInFeature.

Examples

```
# Convert a dataframe to GRanges object
data(genes)

makeGRanges(genes, soloStart = TRUE)
makeGRanges(genes)
#makeGRanges(genes, freeze = "hg19", soloStart = TRUE)
#makeGRanges(genes, freeze = "hg19")
```

---

makeUCSCsession *Initiate UCSC genome browser session given the freeze argument.*

Description

Initiate UCSC genome browser session given the freeze argument.

Usage

```
makeUCSCsession(freeze = "hg19")
```

Arguments

freeze	one of following: hg19, mm8, rheM, etc. Default is hg19.

Value

browser session object compatible with rtracklayer functions.

See Also

getUCSCtable, makeGRanges, getNearestFeature, getSitesInFeature

Examples

```
## Not run:
session <- makeUCSCsession()
genome(session)
session <- makeUCSCsession("mm8")
genome(session)

## End(Not run)
```

plotdisFeature *Plot distance distribution to a feature boundary.*

Description

Given a dataframe of samples and distance based annotation, the function calculates the distribution of data in or around the given annotation. From genomic point of view, the function can be used to identify distribution of data around genomic features like gene TSS, CpG island, etc.

Usage

```
plotdisFeature(
  dat = NULL,
  grouping = NULL,
  annotCol = NULL,
  breaks = NULL,
  discreteBins = TRUE,
  geom = "bar",
  stacked = FALSE,
  typeRatio = FALSE,
  printPlotData = FALSE
)
```

Arguments

dat	a dataframe/GRanges with required columns to make the plot.
grouping	name of the column grouping the data or denoting the samples
annotCol	name of the column holding the distance to feature data. This can also be boolean data in which case plot will be in/out of feature.
breaks	intervals by which to break up the distance data. Default is seq(-1e5,1e5,5e3). Not required if 'annotCol' is of type boolean.
discreteBins	whether to plot continuous variable supplied in annotCol as a discrete axis. This conserves plotting area, thus default is TRUE.
geom	plot distribution using bars or lines? Default is 'bar'. One can use 'line' as well when there are many groups.
stacked	make a stacked plot? Only applies when geom is 'bar'. Default is FALSE.
typeRatio	whether to plot data as ratio of experimental to controls. Default is FALSE. Enabling this requires a column in 'dat' called "type" with two values "expr" for experimental and "ctrl" for control. This column subdivides data within each group. Enabling this transforms the data into plotting distribution of ratios of experimental/controls around feature of interest.
printPlotData	return summarized plot data? Default is FALSE.

Value

ggplot2 plot and/or table of summarized plot data.

See Also

makeGRanges, getNearestFeature, getSitesInFeature, getFeatureCounts

Examples

```
# Convert a dataframe to GRanges object
data(sites)
data(sites.ctrl)
sites$type <- "expr"
sites <- rbind(sites,sites.ctrl)
alldata.rd <- makeGRanges(sites,soloStart=TRUE)

data(genes)
genes.rd <- makeGRanges(genes)

res <- doAnnotation(annotType="within", alldata.rd, genes.rd, "InGene",
asBool=TRUE)
plotdisFeature(res, "virus", "InGene")
plotdisFeature(res, "virus", "InGene", typeRatio=TRUE)
## Not run:
res <- doAnnotation(annotType="nearest", res, genes.rd, "NearestGene",
side='5p')
plotdisFeature(res, "virus", "X5pNearestGeneDist")
plotdisFeature(res, "virus", "X5pNearestGeneDist", typeRatio=TRUE)

## End(Not run)
```

---

sites *Sample Retrovirus Integration Sites data*

Description

A sample dataset containing collection of unique HIV & MLV integration sites in the human genome mapped to UCSC freeze hg18 from PMID: 12805549.

Format

A data frame with 1303 rows and 5 variables

Details

- Sequence. Name of the DNA sequence which was aligned to the host genome. This is also a unique ID.
- Position. The genomic coordinate of the integration site.
- Chr. The chromosome of the integration site.
- Ort. The orientation or strand of the integration site.
- virus. Name of the virus used for the experiment and a given sequencing clone.

Source

http://www.ncbi.nlm.nih.gov/pubmed/?term=12805549

`sites.ctrl` *Controls for Sample Retrovirus Integration Sites data*

Description

Controls for a sample dataset containing collection of unique HIV & MLV integration sites in the human genome mapped to UCSC freeze hg18 from PMID: 12805549. Each row represents three controls per integration site in sites object.

Format

A data frame with 3909 rows and 6 variables

Details

- Sequence. Name of the DNA sequence which was aligned to the host genome. There should be three control sites per experimental site from the "sites" dataset.
- Position. The genomic coordinate of the integration site.
- Chr. The chromosome of the integration site.
- Ort. The orientation or strand of the integration site.
- virus. Name of the virus used for the experiment and a given sequencing clone.
- type. Column denoting whether the data is control

Index