Package 'psichomics'

October 16, 2019

Title Graphical Interface for Alternative Splicing Quantification, Analysis and Visualisation

Version 1.10.2

Encoding UTF-8

Description Interactive R package with an intuitive Shiny-based graphical interface for alternative splicing quantification and integrative analyses of alternative splicing and gene expression based on The Cancer Genome Atlas (TCGA), the Genotype-Tissue Expression project (GTEx), Sequence Read Archive (SRA) and user-provided data. The tool interactively performs survival, dimensionality reduction and median- and variance-based differential splicing and gene expression analyses that benefit from the incorporation of clinical and molecular sample-associated features (such as tumour stage or survival). Interactive visual access to genomic mapping and functional annotation of selected alternative splicing events is also included.

Depends R (>= 3.5), shiny (>= 1.0.3), shinyBS

License MIT + file LICENSE

LazyData true

RoxygenNote 6.1.1

Imports AnnotationDbi, AnnotationHub, cluster, colourpicker, data.table, digest, dplyr, DT (>= 0.2), edgeR, fastICA, fastmatch, ggplot2, ggrepel, graphics, grDevices, highcharter (>= 0.5.0), htmltools, httr, jsonlite, limma, miscTools, pairsD3, plyr, Rcpp (>= 0.12.14), recount, R.utils, reshape2, shinyjs, stringr, stats, SummarizedExperiment, survival, tools, utils, XML, xtable, methods, org.Hs.eg.db

Suggests testthat, knitr, parallel, devtools, rmarkdown, gplots, covr, car, rstudioapi

LinkingTo Rcpp

VignetteBuilder knitr

Collate 'RcppExports.R' 'utils.R' 'globalAccess.R' 'app.R' 'analysis.R' 'analysis_correlation.R' 'analysis_diffExpression.R' 'analysis_diffExpression_event.R' 'analysis_diffExpression_table.R' 'analysis_diffSplicing.R' 'analysis_diffSplicing_event.R' 'analysis_diffSplicing_table.R' 'analysis_dimReduction.R' 'analysis_dimReduction_ica.R' 'analysis_dimReduction_pca.R' 'analysis_information.R'

'analysis_survival.R' 'analysis_template.R' 'data.R'
'formats.R' 'data_firebrowse.R'
'data_geNormalisationFiltering.R' 'data_gtex.R'
'data_inclusionLevels.R' 'data_local.R' 'data_recount.R'
'events_suppa.R' 'events_vastTools.R' 'events_miso.R'
'events_mats.R' 'events.R' 'formats_firebrowseGeneExpression.R'
'formats_firebrowseJunctionReads.R'
'formats_firebrowseMergeClinical.R'
'formats_firebrowseNormalizedGeneExpression.R'
'formats_genericClinical.R' 'formats_genericGeneExpression.R'
'formats_genericInclusionLevels.R'
'formats_genericJunctionReads.R' 'formats_genericSampleInfo.R'
'formats_gtexClinical.R' 'formats_gtexGeneReadsFormat.R'
'formats_gtexJunctionReads.R' 'formats_gtexSampleInfo.R'
'formats_gtexV7Clinical.R' 'formats_gtexV7JunctionReads.R'
'formats_psichomicsGeneExpression.R'
'formats_psichomicsInclusionLevels.R'
'formats_recountSampleInfo.R' 'groups.R' 'help.R'

biocViews Sequencing, RNASeq, AlternativeSplicing, DifferentialSplicing, Transcription, GUI, PrincipalComponent, Survival, BiomedicalInformatics, Transcriptomics, ImmunoOncology, Visualization, MultipleComparison, GeneExpression, DifferentialExpression

URL https://github.com/nuno-agostinho/psichomics

BugReports https://github.com/nuno-agostinho/psichomics/issues

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

git_branch RELEASE_3_9

git_last_commit e63d018

git_last_commit_date 2019-10-07

Date/Publication 2019-10-15

Author Nuno Saraiva-Agostinho [aut, cre],
Nuno Luís Barbosa-Morais [aut, led, ths],
André Falcão [ths],
Lina Gallego Paez [ctb],
Marie Bordone [ctb],
Teresa Maia [ctb],
Mariana Ferreira [ctb],
Ana Carolina Leote [ctb],
Bernardo de Almeida [ctb]

Maintainer Nuno Saraiva-Agostinho <nunodanielagostinho@gmail.com>

R topics documented:

`calculateLoadingsContribution` *Calculate the contribution of PCA loadings to the selected principal components*

Description

Total contribution of a variable is calculated as per: ((Cx * Ex) + (Cy * Ey))/(Ex + Ey), where Cx and Cy are the contributions of a variable to principal components (x and y) and Ex and Ey are the eigenvalues of principal components (x and y)

Usage

```
calculateLoadingsContribution(pca, pcX = 1, pcY = 2)
```

Arguments

`pca`	`prcomp` object
`pcX`	Character: name of the X axis of interest from the PCA
`pcY`	Character: name of the Y axis of interest from the PCA

Value

Data frame containing the correlation between variables and selected principal components and the contribution of variables to the selected principal components (both individual and total contribution)

Source

http://www.sthda.com/english/articles/31-principal-component-methods-in-r-practical-guide/112-pca-principal-component-analysis-essentials/

Examples

```
pca <- performPCA(USArrests)
calculateLoadingsContribution(pca)
```

colSums,EList-method *Sum columns using an* EList-class *object*

Description

Sum columns using an EList-class object

Usage

```
## S4 method for signature 'EList'
colSums(x, na.rm = FALSE, dims = 1)
```

Arguments

x	an array of two or more dimensions, containing numeric, complex, integer or logical values, or a numeric data frame. For .colSums() etc, a numeric, integer or logical matrix (or vector of length m * n).
na.rm	logical. Should missing values (including NaN) be omitted from the calculations?
dims	integer: Which dimensions are regarded as 'rows' or 'columns' to sum over. For row*, the sum or mean is over dimensions dims+1, . . .; for col* it is over dimensions 1:dims.

Value

Numeric vector with the sum of the columns

convertGeneIdentifiers *Convert gene identifiers*

Description

Convert gene identifiers

Usage

```
convertGeneIdentifiers(annotation, genes, key = "ENSEMBL",
  target = "SYMBOL", ignoreDuplicatedTargets = TRUE)
```

Arguments

annotation	OrgDb: genome wide annotation for an organism, e.g. org.Hs.eg.db
genes	Character: genes to be converted
key	Character: type of identifier used, e.g. ENSEMBL; read ?AnnotationDbi::columns
target	Character: type of identifier to convert to; read ?AnnotationDbi::columns
ignoreDuplicatedTargets	Boolean: if TRUE, identifiers that share targets with other identifiers will not be converted

Value

Character vector of the respective targets of gene identifiers. The previous identifiers remain other identifiers have the same target (in case `ignoreDuplicatedTargets = TRUE`) or if no target was found.

Examples

```
if ( require("org.Hs.eg.db") ) {
    columns(org.Hs.eg.db)

    genes <- c("ENSG00000012048", "ENSG00000083093", "ENSG00000141510",
               "ENSG00000051180")
    convertGeneIdentifiers(org.Hs.eg.db, genes,
                           key="ENSEMBL", target="SYMBOL")
}
```

`correlateGEandAS` *Correlate gene expression data against alternative splicing quantification*

Description

Test for association between paired samples' gene expression (for any genes of interest) and alternative splicing quantification.

Usage

```
correlateGEandAS(geneExpr, psi, gene, ASevents = NULL, ...)
```

Arguments

`geneExpr`	Matrix or data frame: gene expression data
`psi`	Matrix or data frame: alternative splicing quantification data
`gene`	Character: gene symbol for genes of interest
`ASevents`	Character: alternative splicing events to correlate with gene expression of a gene (if `NULL`, the events will be automatically retrieved from the given gene)
`...`	Arguments passed on to `stats:::cor.test.default`

alternative indicates the alternative hypothesis and must be one of `"two.sided"`, `"greater"` or `"less"`. You can specify just the initial letter. `"greater"` corresponds to positive association, `"less"` to negative association.

method a character string indicating which correlation coefficient is to be used for the test. One of `"pearson"`, `"kendall"`, or `"spearman"`, can be abbreviated.

exact a logical indicating whether an exact p-value should be computed. Used for Kendall's τ and Spearman's ρ. See 'Details' for the meaning of `NULL` (the default).

conf.level confidence level for the returned confidence interval. Currently only used for the Pearson product moment correlation coefficient if there are at least 4 complete pairs of observations.

continuity logical: if true, a continuity correction is used for Kendall's τ and Spearman's ρ when not computed exactly.

Value

List of correlations where each element contains:

eventID	Alternative splicing event identifier
cor	Correlation between gene expression and alternative splicing quantification of one alternative splicing event
geneExpr	Gene expression for the selected gene
psi	Alternative splicing quantification for the alternative splicing event

Examples

```
annot <- readFile("ex_splicing_annotation.RDS")
junctionQuant <- readFile("ex_junctionQuant.RDS")
psi <- quantifySplicing(annot, junctionQuant, eventType=c("SE", "MXE"))

geneExpr <- readFile("ex_gene_expression.RDS")
correlateGEandAS(geneExpr, psi, "ALDOA")
```

---

createGroupByColumn *Split elements into groups based on a given column of a dataset*

Description

Elements are identified by their respective row name.

Usage

```
createGroupByColumn(col, dataset)

createGroupByAttribute(col, dataset)
```

Arguments

col	Character: column name
dataset	Matrix or data frame: dataset

Value

Named list with each unique value from a given column and respective elements

Examples

```
df <- data.frame(gender=c("male", "female"),
                 stage=paste("stage", c(1, 3, 1, 4, 2, 3, 2, 2)))
rownames(df) <- paste0("patient-", LETTERS[1:8])
createGroupByAttribute(col="stage", dataset=df)
```

`diffAnalyses` *Perform statistical analyses*

Description

Perform statistical analyses

Usage

```
diffAnalyses(data, groups = NULL, analyses = c("wilcoxRankSum",
  "ttest", "kruskal", "levene", "fligner"), pvalueAdjust = "BH",
  geneExpr = NULL, psi = NULL)
```

Arguments

`data`	Data frame or matrix: gene expression or alternative splicing quantification
`groups`	Named list of characters (containing elements belonging to each group) or character vector (containing the group of each individual sample); if NULL, sample types are used instead when available, e.g. normal, tumour and metastasis
`analyses`	Character: statistical tests to perform (see Details)
`pvalueAdjust`	Character: method used to adjust p-values (see Details)
`geneExpr`	Character: name of the gene expression dataset (only required for density sparklines available in the interactive mode)
`psi`	Data frame or matrix: alternative splicing quantification (defunct argument, use `data` instead)

Details

The following statistical analyses may be performed by including the respective string in the `analysis` argument:

- `ttest` - Unpaired t-test (2 groups)
- `wilcoxRankSum` - Wilcoxon Rank Sum test (2 groups)
- `kruskal` - Kruskal test (2 or more groups)
- `levene` - Levene's test (2 or more groups)
- `fligner` - Fligner-Killeen test (2 or more groups)
- `density` - Sample distribution per group (only usable through the visual interface)

The following methods for p-value adjustment are supported by using the respective string in the `pvalueAdjust` argument:

- `none`: do not adjust p-values
- `BH`: Benjamini-Hochberg's method (false discovery rate)
- `BY`: Benjamini-Yekutieli's method (false discovery rate)
- `bonferroni`: Bonferroni correction (family-wise error rate)
- `holm`: Holm's method (family-wise error rate)
- `hochberg`: Hochberg's method (family-wise error rate)
- `hommel`: Hommel's method (family-wise error rate)

Value

Table of statistical analyses

Examples

```
# Calculate PSI for skipped exon (SE) and mutually exclusive (MXE) events
eventType <- c("SE", "MXE")
annot <- readFile("ex_splicing_annotation.RDS")
junctionQuant <- readFile("ex_junctionQuant.RDS")

psi <- quantifySplicing(annot, junctionQuant, eventType=c("SE", "MXE"))
group <- c(rep("Normal", 3), rep("Tumour", 3))
diffAnalyses(psi, group)
```

---

ensemblToUniprot *Convert an Ensembl identifier to the respective UniProt identifier*

Description

Convert an Ensembl identifier to the respective UniProt identifier

Usage

```
ensemblToUniprot(protein)
```

Arguments

protein	Character: Ensembl identifier

Value

UniProt protein identifier

Examples

```
gene <- "ENSG00000173262"
ensemblToUniprot(gene)

protein <- "ENSP00000445929"
ensemblToUniprot(protein)
```

`filterGeneExpr` *Filter genes based on their expression*

Description

Filter genes based on their expression

Usage

```
filterGeneExpr(geneExpr, minMean = 0, maxMean = Inf, minVar = 0,
  maxVar = Inf, minCounts = 10, minTotalCounts = 15)
```

Arguments

`geneExpr`	Data frame or matrix: gene expression
`minMean`	Numeric: minimum of read count mean per gene
`maxMean`	Numeric: maximum of read count mean per gene
`minVar`	Numeric: minimum of read count variance per gene
`maxVar`	Numeric: maximum of read count variance per gene
`minCounts`	Numeric: minimum number of read counts per gene for at least some samples
`minTotalCounts`	Numeric: minimum total number of read counts per gene

Value

Boolean vector indicating which genes have sufficiently large counts

Examples

```
geneExpr <- readFile("ex_gene_expression.RDS")

# Add some genes with low expression
geneExpr <- rbind(geneExpr,
                  lowReadGene1=c(rep(4:5, 10)),
                  lowReadGene2=c(rep(5:1, 10)),
                  lowReadGene3=c(rep(10:1, 10)),
                  lowReadGene4=c(rep(7:8, 10)))

# Filter out genes with low reads across samples
geneExpr[filterGeneExpr(geneExpr), ]
```

filterGroups *Filter groups with less data points than the threshold*

Description

Groups containing a number of non-missing values less than the threshold are discarded.

Usage

```
filterGroups(vector, group, threshold = 1)
```

Arguments

vector	Unnamed elements
group	Character: group of the elements
threshold	Integer: number of valid non-missing values by group

Value

Named vector with filtered elements from valid groups. The group of the respective element is given in the name.

Examples

```
# Removes groups with less than two elements
filterGroups(1:4, c("A", "B", "B", "D"), threshold=2)
```

filterPSI *Filter alternative splicing quantification*

Description

Filter alternative splicing quantification

Usage

```
filterPSI(psi, minMedian = -Inf, maxMedian = Inf, minLogVar = -Inf,
  maxLogVar = Inf, minRange = -Inf, maxRange = Inf)
```

Arguments

psi	Data frame or matrix: alternative splicing quantification
minMedian	Numeric: minimum of read count median per splicing event
maxMedian	Numeric: maximum of read count median per splicing event
minLogVar	Numeric: minimum log10(read count variance) per splicing event
maxLogVar	Numeric: maximum log10(read count variance) per splicing event
minRange	Numeric: minimum range of read counts across samples per splicing event
maxRange	Numeric: maximum range of read counts across samples per splicing event

Value

Boolean vector indicating which splicing events pass the thresholds

Examples

```
# Calculate PSI for skipped exon (SE) and mutually exclusive (MXE) events
annot <- readFile("ex_splicing_annotation.RDS")
junctionQuant <- readFile("ex_junctionQuant.RDS")

psi <- quantifySplicing(annot, junctionQuant, eventType=c("SE", "MXE"))
psi[filterPSI(psi, minMedian=0.05, maxMedian=0.95, minRange=0.15), ]
```

---

getAttributesTime *Retrieve the time for given columns in a clinical dataset*

Description

Retrieve the time for given columns in a clinical dataset

Usage

```
getAttributesTime(clinical, event, timeStart, timeStop = NULL,
  followup = "days_to_last_followup")

getColumnsTime(clinical, event, timeStart, timeStop = NULL,
  followup = "days_to_last_followup")
```

Arguments

clinical	Data frame: clinical data
event	Character: name of column containing time of the event of interest
timeStart	Character: name of column containing starting time of the interval or follow up time
timeStop	Character: name of column containing ending time of the interval (only relevant for interval censoring)
followup	Character: name of column containing follow up time

Value

Data frame containing the time for the given columns

Examples

```
df <- data.frame(followup=c(200, 300, 400), death=c(NA, 300, NA))
rownames(df) <- paste("patient", 1:3)
getAttributesTime(df, event="death", timeStart="death", followup="followup")
```

getDownloadsFolder *Get the Downloads folder of the user*

Description

Get the Downloads folder of the user

Usage

```
getDownloadsFolder()
```

Value

Path to Downloads folder

Examples

```
getDownloadsFolder()
```

getFirebrowseDataTypes *Get data types available from Firebrowse*

Description

Get data types available from Firebrowse

Usage

```
getFirebrowseDataTypes()

getFirehoseDataTypes()
```

Value

Named character vector

Examples

```
getFirebrowseDataTypes()
```

getFirebrowseDates *Query the Firebrowse web API*

Description

Query the Firebrowse web API

Usage

```
getFirebrowseDates()

getFirebrowseCohorts(cohort = NULL)
```

Arguments

cohort	Character: filter results by given cohorts (optional)

Value

Parsed response

Examples

```
if (isFirebrowseUp()) getFirebrowseDates()
if (isFirebrowseUp()) getFirebrowseCohorts()
```

getGeneList *Get pre-created, literature-based gene list*

Description

Available gene lists:

- **Sebestyen et al., 2016**: 1350 genes encoding RNA-binding proteins, 167 of which are splicing factors

Usage

```
getGeneList()
```

Value

List of genes

Examples

```
getGeneList()
```

getGtexTissues *Get GTEx tissues from given GTEx sample attributes*

Description

Get GTEx tissues from given GTEx sample attributes

Usage

```
getGtexTissues(dataFolder = getDownloadsFolder())
```

Arguments

dataFolder	Character: folder containing data

Value

Character: available tissues

getMatchingSamples *Get samples matching the given patients*

Description

Get samples matching the given patients

Usage

```
getMatchingSamples(patients, samples, clinical = NULL, rm.NA = TRUE,
  match = NULL, showMatch = FALSE)

getSampleFromPatient(patients, samples, clinical = NULL, rm.NA = TRUE,
  match = NULL, showMatch = FALSE)

getSampleFromSubject(patients, samples, clinical = NULL, rm.NA = TRUE,
  match = NULL, showMatch = FALSE)
```

Arguments

patients	Character or list of characters: patient identifiers
samples	Character: sample identifiers
clinical	Data frame or matrix: clinical dataset
rm.NA	Boolean: remove NAs? TRUE by default
match	Integer: vector of patient index with the sample identifiers as name to save time (optional)
showMatch	Boolean: show matching patient index? FALSE by default

Value

Names of the matching samples (if showMatch is TRUE, a character with the patients as values and their respective samples as names is returned)

Examples

```
patients <- c("GTEX-ABC", "GTEX-DEF", "GTEX-GHI", "GTEX-JKL", "GTEX-MNO")
samples <- paste0(patients, "-sample")
clinical <- data.frame(samples=samples)
rownames(clinical) <- patients
getMatchingSamples(patients[c(1, 4)], samples, clinical)
```

---

getPatientFromSample *Get patients from given samples*

Description

Get patients from given samples

Usage

```
getPatientFromSample(sampleId, patientId = NULL, na = FALSE,
  sampleInfo = NULL)

getSubjectFromSample(sampleId, patientId = NULL, na = FALSE,
  sampleInfo = NULL)
```

Arguments

sampleId	Character: sample identifiers
patientId	Character: patient identifiers to filter by (optional; if a matrix or data frame is given, its rownames will be used to infer the patient identifiers)
na	Boolean: return NA for samples with no matching patients
sampleInfo	Data frame or matrix: sample information containing the sample identifiers as rownames and a column named "Subject ID" with the respective subject identifiers

Value

Character: patient identifiers corresponding to the given samples

Examples

```
samples <- paste0("GTEX-", c("ABC", "DEF", "GHI", "JKL", "MNO"), "-sample")
getPatientFromSample(samples)

# Filter returned samples based on available patients
patients <- paste0("GTEX-", c("DEF", "MNO"))
getPatientFromSample(samples, patients)
```

`getSplicingEventFromGenes` *Get alternative splicing events from genes or vice-versa*

Description

Get alternative splicing events from genes or vice-versa

Usage

```
getSplicingEventFromGenes(genes, ASevents)

getGenesFromSplicingEvents(ASevents)
```

Arguments

`genes`	Character: gene symbols (or TCGA-styled gene symbols)
`ASevents`	Character: alternative splicing events

Details

A list of alternative splicing events is required to run `getSplicingEventFromGenes`

Value

Named character containing alternative splicing events or genes and their respective genes or alternative splicing events as names (depending on the function in use)

Examples

```
ASevents <- c("SE_1_+_201763003_201763300_201763374_201763594_NAV1",
              "SE_1_+_183515472_183516238_183516387_183518343_SMG7",
              "SE_1_+_183441784_183471388_183471526_183481972_SMG7",
              "SE_1_+_181019422_181022709_181022813_181024361_MR1",
              "SE_1_+_181695298_181700311_181700367_181701520_CACNA1E")
genes <- c("NAV1", "SMG7", "MR1", "HELLO")

# Get splicing events from genes
matchedASevents <- getSplicingEventFromGenes(genes, ASevents)

# Names of matched events are the matching input genes
names(matchedASevents)
matchedASevents

# Get genes from splicing events
matchedGenes <- getGenesFromSplicingEvents (ASevents)

# Names of matched genes are the matching input alternative splicing events
names(matchedGenes)
matchedGenes
```

getSplicingEventTypes *Splicing event types available*

Description

Splicing event types available

Usage

```
getSplicingEventTypes(acronymsAsNames = FALSE)
```

Arguments

acronymsAsNames	Boolean: return acronyms as names?

Value

Named character vector with splicing event types

Examples

```
getSplicingEventTypes()
```

getValuePerPatient *Assign average sample values to their corresponding patients*

Description

Assign average sample values to their corresponding patients

Usage

```
getValuePerPatient(data, match, clinical = NULL, patients = NULL,
  samples = NULL)

getValuePerSubject(data, match, clinical = NULL, patients = NULL,
  samples = NULL)

assignValuePerPatient(data, match, clinical = NULL, patients = NULL,
  samples = NULL)

assignValuePerSubject(data, match, clinical = NULL, patients = NULL,
  samples = NULL)

getPSIperPatient(psi, match, clinical = NULL, patients = NULL, ...)
```

Arguments

data	One-row data frame/matrix or vector: values per sample for a single gene
match	Matrix: match between samples and patients
clinical	Data frame or matrix: clinical dataset (only required if the patients argument is not handed)
patients	Character: patient identifiers (only required if the clinical argument is not handed)
samples	Character: samples to use when assigning values per patient (if NULL, all samples will be used)
psi	Data frame or matrix: values per sample
...	Deprecated arguments

Value

Values per patient

Examples

```
# Calculate PSI for skipped exon (SE) and mutually exclusive (MXE) events
annot <- readFile("ex_splicing_annotation.RDS")
junctionQuant <- readFile("ex_junctionQuant.RDS")

psi <- quantifySplicing(annot, junctionQuant, eventType=c("SE", "MXE"))

# Match between subjects and samples
match <- rep(paste("Patient", 1:3), 2)
names(match) <- colnames(psi)

assignValuePerSubject(psi[3, ], match)
```

---

groupByAttribute *Data grouping interface*

Description

Data grouping interface

Usage

```
groupByAttribute(ns, cols, id, example)

groupByPreMadeList(ns, data, id)

groupById(ns, id)

groupByExpression(ns, id)

groupByGrep(ns, cols, id)
```

Arguments

ns	Namespace function
cols	Character or list: name of columns to show
id	Character: identifier
example	Character: text to show as an example
data	List: list of groups with elements

Value

HTML elements

---

groupPerElem *Assign one group to each element*

Description

Assign one group to each element

Usage

```
groupPerElem(groups, elem = NULL, outerGroupName = NA)

groupPerPatient(groups, patients = NULL, includeOuterGroup = FALSE,
  outerGroupName = "(Outer data)")

groupPerSample(groups, samples, includeOuterGroup = FALSE,
  outerGroupName = "(Outer data)")
```

Arguments

groups	List of integers: groups of elements
elem	Character: all elements available (NULL by default)
outerGroupName	Character: name to give to outer group (NA by default; set to NULL to only show elements matched to their respective groups)
patients	Integer: total number of patients
includeOuterGroup	Boolean: join the patients that have no groups?
samples	Character: all available samples

Value

Character vector where each element corresponds to the group of the respective element

Examples

```
groups <- list(1:3, 4:7, 8:10)
names(groups) <- paste("Stage", 1:3)
groupPerElem(groups)
```

isFirebrowseUp *Check whether the Firebrowse web API is running*

Description

The Firebrowse web API is running if it returns the status condition 200; if this is not the status code obtained from the API, the function will raise a warning with the status code and a brief explanation.

Usage

```
isFirebrowseUp()
```

Value

Invisible TRUE if the Firebrowse web API is working; otherwise, raises a warning

Examples

```
isFirebrowseUp()
```

labelBasedOnCutoff *Label groups based on a given cutoff*

Description

Label groups based on a given cutoff

Usage

```
labelBasedOnCutoff(data, cutoff, label = NULL, gte = TRUE)
```

Arguments

data	Numeric: test data
cutoff	Numeric: test cutoff
label	Character: label to prefix group names (NULL by default)
gte	Boolean: test with greater than or equal to cutoff (TRUE) or use less than or equal to cutoff (FALSE)? TRUE by default

Value

Labelled groups

Examples

```
labelBasedOnCutoff(data=c(1, 0, 0, 1, 0, 1), cutoff=0.5)

labelBasedOnCutoff(data=c(1, 0, 0, 1, 0, 1), cutoff=0.5, "Ratio")

# Use "greater than" instead of "greater than or equal to"
labelBasedOnCutoff(data=c(1, 0, 0, 0.5, 0, 1), cutoff=0.5, gte=FALSE)
```

listSplicingAnnotations *List the alternative splicing annotation files available*

Description

List the alternative splicing annotation files available

Usage

```
listSplicingAnnotations()
```

Value

Named character vector with splicing annotation files available

Examples

```
listSplicingAnnotations()
```

loadAnnotation *Load alternative splicing annotation from* AnnotationHub

Description

Load alternative splicing annotation from AnnotationHub

Usage

```
loadAnnotation(annotation)
```

Arguments

annotation	Character: annotation to load

Value

List of data frames containing the alternative splicing annotation per event type

Examples

```
human <- listSplicingAnnotations()[[1]]
## Not run:
annot <- loadAnnotation(human)

## End(Not run)
```

`loadFirebrowseData` *Downloads and processes data from the Firebrowse web API and loads it into R*

Description

Downloads and processes data from the Firebrowse web API and loads it into R

Usage

```
loadFirebrowseData(folder = getDownloadsFolder(), data = NULL,
  exclude = c(".aux.", ".mage-tab.", "MANIFEST.txt"), ...,
  download = TRUE)
```

Arguments

`folder`	Character: directory to store the downloaded archives (by default, it saves in the user's "Downloads" folder)
`data`	Character: data to load
`exclude`	Character: files and folders to exclude from downloading and from loading into R (by default, it excludes `.aux.`, `.mage-tab.` and `MANIFEST.TXT` files)
`...`	Arguments passed on to `queryFirebrowseData` **format** Character: response format as `JSON` (default), `CSV` or `TSV` **date** Character: dates of the data retrieval by Firebrowse (by default, it uses the most recent data available) **cohort** Character: abbreviation of the cohorts (by default, returns data for all cohorts) **data_type** Character: data types (optional) **tool** Character: data produced by the selected Firebrowse tools (optional) **platform** Character: data generation platforms (optional) **center** Character: data generation centres (optional) **level** Integer: data levels (optional) **protocol** Character: sample characterization protocols (optional) **page** Integer: page of the results to return (optional) **page_size** Integer: number of records per page of results; max is 2000 (optional) **sort_by** String: column used to sort the data (by default, it sorts by cohort)
`download`	Boolean: download missing files through the function `download.file` (TRUE by default)

Value

URL of missing files ("missing" class) if files need to be downloaded and if the argument `download` is `FALSE`; else, a list with loaded data

Examples

```
## Not run:
loadFirebrowseData(cohort = "ACC", data_type = "Clinical")

## End(Not run)
```

---

loadGtexData *Load GTEx data*

Description

Load GTEx data

Usage

```
loadGtexData(dataTypes = getGtexDataTypes(),
  dataFolder = getDownloadsFolder(), tissue = NULL)
```

Arguments

dataTypes	Character: data types to load (see getGtexDataTypes)
dataFolder	Character: folder containing data
tissue	Character: tissues to load (if NULL, load all); tissue selection may speed up data loading

Value

List with loaded data

---

loadLocalFiles *Load local files*

Description

Load local files

Usage

```
loadLocalFiles(folder, ignore = c(".aux.", ".mage-tab."),
  name = "Data")
```

Arguments

folder	Character: path to folder containing files of interest
ignore	Character: skip folders and filenames that match the expression
name	Character: name of the category containing all loaded datasets

Value

List of data frames from valid files

Examples

```
## Not run:
folder <- "~/Downloads/ACC 2016"
data <- loadLocalFiles(folder)

ignore <- c(".aux.", ".mage-tab.", "junction quantification")
loadLocalFiles(folder, ignore)

## End(Not run)
```

`loadSRAproject` *Download and load SRA projects*

Description

Download and load SRA projects

Usage

```
loadSRAproject(project, outdir = getDownloadsFolder())
```

Arguments

`project`	Character: SRA project identifiers to download
`outdir`	Character: directory to store the downloaded files

Value

List containing downloaded projects

`normaliseGeneExpression` *Filter and normalise gene expression*

Description

Filter and normalise gene expression

Usage

```
normaliseGeneExpression(geneExpr, geneFilter = NULL, method = "TMM",
  p = 0.75, log2transform = TRUE, priorCount = 0.25,
  performVoom = FALSE)
```

Arguments

geneExpr	Matrix or data frame: gene expression
geneFilter	Boolean: filtered genes
method	Character: normalisation method, including TMM, RLE, upperquartile, none or quantile (see Details)
p	percentile (between 0 and 1) of the counts that is aligned when method="upperquartile"
log2transform	Boolean: perform log2-transformation?
priorCount	Average count to add to each observation to avoid zeroes after log-transformation
performVoom	Boolean: perform mean-variance modelling (voom)?

Details

edgeR::calcNormFactors will be used to normalise gene expression if one of the followin methods is set: TMM, RLE, upperquartile or none. However, limma::voom will be used for normalisation if performVoom = TRUE and the selected method is quantile.

Value

Filtered and normalised gene expression

Examples

```
geneExpr <- readFile("ex_gene_expression.RDS")
normaliseGeneExpression(geneExpr)
```

---

optimalSurvivalCutoff *Calculate optimal data cutoff that best separates survival curves*

Description

Uses stats::optim with the Brent method to test multiple cutoffs and to find the minimum log-rank p-value.

Usage

```
optimalSurvivalCutoff(clinical, data, censoring, event, timeStart,
  timeStop = NULL, followup = "days_to_last_followup",
  session = NULL, filter = TRUE, survTime = NULL, lower = NULL,
  upper = NULL)

optimalPSIcutoff(clinical, psi, censoring, event, timeStart,
  timeStop = NULL, followup = "days_to_last_followup",
  session = NULL, filter = TRUE, survTime = NULL)
```

Arguments

clinical	Data frame: clinical data
data	Numeric: data values
censoring	Character: censor using "left", "right", "interval" or "interval2"
event	Character: name of column containing time of the event of interest
timeStart	Character: name of column containing starting time of the interval or follow up time
timeStop	Character: name of column containing ending time of the interval (only relevant for interval censoring)
followup	Character: name of column containing follow up time
session	Shiny session (only used for the visual interface)
filter	Boolean or numeric: elements to use (all by default)
survTime	survTime object: times to follow up, time start, time stop and event (optional)
lower, upper	Bounds in which to search (if NULL, they will be automatically set to 0 and 1 if all data values are within that interval; otherwise, they will be set to the minimum and maximum values of data)
psi	Numeric: PSI values to test against the cutoff

Value

List containg the optimal cutoff (par) and the corresponding p-value (value)

Examples

```
clinical <- read.table(text = "2549   NA ii  female
                                840   NA i   female
                                 NA 1204 iv    male
                                 NA  383 iv  female
                               1293   NA iii   male
                                 NA 1355 ii    male")
names(clinical) <- c("patient.days_to_last_followup",
                     "patient.days_to_death",
                     "patient.stage_event.pathologic_stage",
                     "patient.gender")
timeStart  <- "days_to_death"
event      <- "days_to_death"

psi <- c(0.1, 0.2, 0.9, 1, 0.2, 0.6)
opt <- optimalSurvivalCutoff(clinical, psi, "right", event, timeStart)
```

---

parseCategoricalGroups

Parse categorical columns in a data frame

Description

Retrieve elements grouped by their unique group based on each categorical column

Usage

```
parseCategoricalGroups(df)
```

Arguments

df	Data frame

Value

List of lists containing values based on rownames of df

See Also

testGroupIndependence and plotGroupIndependence

Examples

```
df <- data.frame("race"=c("caucasian", "caucasian", "asian"),
                 "gender"=c("male", "female", "male"))
rownames(df) <- paste("patient", 1:3)
parseCategoricalGroups(df)
```

---

parseSampleGroups *Return the type of a given sample*

Description

Return the type of a given sample

Usage

```
parseSampleGroups(sample, filename = system.file("extdata",
  "TCGAsampleType.RDS", package = "psichomics"))
```

Arguments

sample	Character: ID of the sample
filename	Character: path to RDS file containing corresponding type

Value

Types of the TCGA samples

Examples

```
parseSampleGroups(c("TCGA-01A-Tumour", "TCGA-10B-Normal"))
```

parseSplicingEvent *Parse an alternative splicing event based on a given identifier*

Description

Parse an alternative splicing event based on a given identifier

Usage

```
parseSplicingEvent(event, char = FALSE, pretty = FALSE, extra = NULL,
  coords = FALSE)
```

Arguments

event	Character: event identifier
char	Boolean: return a single character instead of list with parsed values?
pretty	Boolean: return a prettier name of the event identifier?
extra	Character: extra information to add (such as species and assembly version); only used if pretty and char are TRUE
coords	Boolean: extra coordinates regarding the alternative and constitutive regions of alternative splicing events; only used if char is FALSE

Value

Parsed event

Examples

```
events <- c("SE_1_-_123_456_789_1024_TST",
            "MXE_3_+_473_578_686_736_834_937_HEY/YOU")
parseSplicingEvent(events)
```

parseSuppaAnnotation *Get events from alternative splicing annotation*

Description

Get events from alternative splicing annotation

Usage

```
parseSuppaAnnotation(folder, types = c("SE", "AF", "AL", "MX", "A5",
  "A3", "RI"), genome = "hg19")

parseVastToolsAnnotation(folder, types = c("ALT3", "ALT5", "COMBI", "IR",
  "MERGE3m", "MIC", "EXSK", "MULTI"), genome = "Hsa",
  complexEvents = FALSE)

parseMisoAnnotation(folder, types = c("SE", "AFE", "ALE", "MXE", "A5SS",
```

```
  "A3SS", "RI", "TandemUTR"), genome = "hg19")

parseMatsAnnotation(folder, types = c("SE", "AFE", "ALE", "MXE", "A5SS",
  "A3SS", "RI"), genome = "fromGTF", novelEvents = TRUE)
```

Arguments

folder	Character: path to folder
types	Character: type of events to retrieve (depends on the program of origin; see details)
genome	Character: genome of interest (for instance, hg19; depends on the program of origin)
complexEvents	Boolean: should complex events in A3SS and A5SS be parsed? FALSE by default
novelEvents	Boolean: parse events dedected due to novel splice sites (TRUE by default)

Details

Type of parsable events:

- Alternative 3' splice site
- Alternative 5' splice site
- Alternative first exon
- Alternative last exon
- Skipped exon (may include skipped micro-exons)
- Mutually exclusive exon
- Retained intron
- Tandem UTR

Value

Retrieve data frame with events based on a given alternative splicing annotation

Examples

```
# Load sample files
folder <- "extdata/eventsAnnotSample/suppa_output/suppaEvents"
suppaOutput <- system.file(folder, package="psichomics")

suppa <- parseSuppaAnnotation(suppaOutput)
# Load sample files
folder <- "extdata/eventsAnnotSample/VASTDB/Hsa/TEMPLATES"
vastToolsOutput <- system.file(folder, package="psichomics")

vast <- parseVastToolsAnnotation(vastToolsOutput)
# Load sample files
folder <- "extdata/eventsAnnotSample/miso_annotation"
misoOutput <- system.file(folder, package="psichomics")

miso <- parseMisoAnnotation(misoOutput)
# Load sample files
folder <- "extdata/eventsAnnotSample/mats_output/ASEvents"
```

```
matsOutput <- system.file(folder, package="psichomics")

mats <- parseMatsAnnotation(matsOutput)

# Do not parse novel events
mats <- parseMatsAnnotation(matsOutput, novelEvents=FALSE)
```

`parseTcgaSampleInfo` *Parse sample information from TCGA samples*

Description

Parse sample information from TCGA samples

Usage

```
parseTcgaSampleInfo(samples, match = NULL)

parseTCGAsampleInfo(samples, match = NULL)
```

Arguments

`samples`	Character: sample identifiers
`match`	Integer: match between samples and patients (NULL by default; performs the match)

Value

Data frame containing metadata associated with each TCGA sample

Examples

```
samples <- c("TCGA-3C-AAAU-01A-11R-A41B-07", "TCGA-3C-AALI-01A-11R-A41B-07",
             "TCGA-3C-AALJ-01A-31R-A41B-07", "TCGA-3C-AALK-01A-11R-A41B-07",
             "TCGA-4H-AAAK-01A-12R-A41B-07", "TCGA-5L-AAT0-01A-12R-A41B-07")

parseTcgaSampleInfo(samples)
```

`performICA` *Perform independent component analysis after processing missing values*

Description

Perform independent component analysis after processing missing values

Usage

```
performICA(data, n.comp = min(5, ncol(data)), center = TRUE,
  scale. = FALSE, missingValues = round(0.05 * nrow(data)),
  alg.typ = c("parallel", "defaltion"), fun = c("logcosh", "exp"),
  alpha = 1, ...)
```

Arguments

data	an optional data frame (or similar: see model.frame) containing the variables in the formula formula. By default the variables are taken from environment(formula).
n.comp	number of components to be extracted
center	a logical value indicating whether the variables should be shifted to be zero centered. Alternately, a vector of length equal the number of columns of x can be supplied. The value is passed to scale.
scale.	a logical value indicating whether the variables should be scaled to have unit variance before the analysis takes place. The default is FALSE for consistency with S, but in general scaling is advisable. Alternatively, a vector of length equal the number of columns of x can be supplied. The value is passed to scale.
missingValues	Integer: number of tolerated missing values per column to be replaced with the mean of the values of that same column (5 rows by default)
alg.typ	if alg.typ == "parallel" the components are extracted simultaneously (the default). if alg.typ == "deflation" the components are extracted one at a time.
fun	the functional form of the G function used in the approximation to neg-entropy (see 'details').
alpha	constant in range [1, 2] used in approximation to neg-entropy when fun == "logcosh"
...	Arguments passed on to fastICA::fastICA

Value

ICA result in a prcomp object

See Also

plotICA, performPCA and plotPCA

Examples

```
performICA(USArrests)
```

---

performPCA *Perform principal component analysis after processing missing values*

Description

Perform principal component analysis after processing missing values

Usage

```
performPCA(data, center = TRUE, scale. = FALSE,
  missingValues = round(0.05 * nrow(data)), ...)
```

Arguments

data	an optional data frame (or similar: see model.frame) containing the variables in the formula formula. By default the variables are taken from environment(formula).
center	a logical value indicating whether the variables should be shifted to be zero centered. Alternately, a vector of length equal the number of columns of x can be supplied. The value is passed to scale.
scale.	a logical value indicating whether the variables should be scaled to have unit variance before the analysis takes place. The default is FALSE for consistency with S, but in general scaling is advisable. Alternatively, a vector of length equal the number of columns of x can be supplied. The value is passed to scale.
missingValues	Integer: number of tolerated missing values per column to be replaced with the mean of the values of that same column (5 rows by default)
...	Arguments passed on to stats::prcomp

Value

PCA result in a prcomp object

See Also

plotPCA, performICA and plotICA

Examples

```
performPCA(USArrests)
```

---

plot.GEandAScorrelation

Display results of correlation analyses

Description

Display results of correlation analyses

Usage

```
## S3 method for class 'GEandAScorrelation'
plot(x, autoZoom = FALSE,
  loessSmooth = TRUE, loessFamily = c("gaussian", "symmetric"),
  colour = "black", alpha = 0.2, size = 1.5, loessColour = "red",
  loessAlpha = 1, loessWidth = 0.5, fontSize = 12, ...,
  colourGroups = NULL, legend = FALSE, showAllData = TRUE,
  density = FALSE, densityColour = "blue", densityWidth = 0.5)

plotCorrelation(x, autoZoom = FALSE, loessSmooth = TRUE,
  loessFamily = c("gaussian", "symmetric"), colour = "black",
  alpha = 0.2, size = 1.5, loessColour = "red", loessAlpha = 1,
  loessWidth = 0.5, fontSize = 12, ..., colourGroups = NULL,
  legend = FALSE, showAllData = TRUE, density = FALSE,
  densityColour = "blue", densityWidth = 0.5)
```

```
## S3 method for class 'GEandAScorrelation'
print(x, ...)

## S3 method for class 'GEandAScorrelation'
as.table(x, pvalueAdjust = "BH", ...)
```

Arguments

x	GEandAScorrelation object (obtained after running correlateGEandAS)
autoZoom	Boolean: automatically set the range of PSI values based on available data? If FALSE, the axis relative to PSI values will range from 0 to 1
loessSmooth	Boolean: plot a smooth curve computed by stats::loess.smooth?
loessFamily	Character: if gaussian, loess fitting is by least-squares, and if symmetric, a re-descending M estimator is used
colour	Character: points' colour
alpha	Numeric: points' alpha
size	Numeric: points' size
loessColour	Character: loess line's colour
loessAlpha	Numeric: loess line's opacity
loessWidth	Numeric: loess line's width
fontSize	Numeric: plot font size
...	Arguments passed on to stats::loess.smooth **span** smoothness parameter for loess. **degree** degree of local polynomial used. **evaluation** number of points at which to evaluate the smooth curve.
colourGroups	List of characters: sample colouring by group
legend	Boolean: show legend for sample colouring?
showAllData	Boolean: show data outside selected groups as a single group (coloured based on the colour argument)
density	Boolean: contour plot of a density estimate
densityColour	Character: line colour of contours
densityWidth	Numeric: line width of contours
pvalueAdjust	Character: method used to adjust p-values (see Details)

Details

The following methods for p-value adjustment are supported by using the respective string in the pvalueAdjust argument:

- none: do not adjust p-values
- BH: Benjamini-Hochberg's method (false discovery rate)
- BY: Benjamini-Yekutieli's method (false discovery rate)
- bonferroni: Bonferroni correction (family-wise error rate)
- holm: Holm's method (family-wise error rate)
- hochberg: Hochberg's method (family-wise error rate)
- hommel: Hommel's method (family-wise error rate)

Value

Plots, summary tables or results of correlation analyses

Examples

```
annot <- readFile("ex_splicing_annotation.RDS")
junctionQuant <- readFile("ex_junctionQuant.RDS")
psi <- quantifySplicing(annot, junctionQuant, eventType=c("SE", "MXE"))

geneExpr <- readFile("ex_gene_expression.RDS")
corr <- correlateGEandAS(geneExpr, psi, "ALDOA")

# Quick display of the correlation results per splicing event and gene
print(corr)

# Table summarising the correlation analysis results
as.table(corr)

# Correlation analysis plots
colourGroups <- list(Normal=paste("Normal", 1:3),
                     Tumour=paste("Cancer", 1:3))
attr(colourGroups, "Colour") <- c(Normal="#00C65A", Tumour="#EEE273")
plot(corr, colourGroups=colourGroups, alpha=1)
```

---

plotDistribution *Plot distribution through a density plot*

Description

The tooltip shows the median, variance, max, min and number of non-NA samples of each data series.

Usage

```
plotDistribution(data, groups = NULL, rug = TRUE, vLine = TRUE, ...,
  title = NULL, psi = NULL, rugLabels = FALSE)
```

Arguments

data	Numeric, data frame or matrix: data for one gene or alternative splicing event
groups	List of characters (list of groups containing data identifiers) or character vector (group of each value in data); if NULL or a character vector of length 1, all data points will be considered of the same group
rug	Boolean: include rug plot to better visualise data distribution
vLine	Boolean: include vertical plot lines to display descriptive statistics for each group
...	Extra parameters passed to density to create the kernel density estimates
title	Character: plot title
psi	Boolean: are data composed of PSI values? Automatically set to TRUE if all data values are between 0 and 1
rugLabels	Boolean: plot names or colnames of data in the rug?

Value

Highcharter object with density plot

Examples

```
data   <- sample(20, rep=TRUE)/20
groups <- paste("Group", c(rep("A", 10), rep("B", 10)))
label  <- paste("Sample", 1:20)
plotDistribution(data, groups, label=label)
```

`plotGeneExprPerSample` *Plot distribution of gene expression per sample*

Description

Plot distribution of gene expression per sample

Usage

```
plotGeneExprPerSample(geneExpr, ...)
```

Arguments

`geneExpr`	Data frame or matrix: gene expression
`...`	Arguments passed on to `renderBoxplot` **data** Data frame or matrix **outliers** Boolean: draw outliers? **sortByMedian** Boolean: sort box plots based on ascending median? **showXlabels** Boolean: show labels in X axis?

Value

Gene expression distribution plots

Examples

```
df <- data.frame(geneA=c(2, 4, 5),
                 geneB=c(20, 3, 5),
                 geneC=c(5, 10, 21))
colnames(df) <- paste("Sample", 1:3)
plotGeneExprPerSample(df)
```

plotGroupIndependence *Plot -log10(p-values) of the results obtained after multiple group independence testing*

Description

Plot -log10(p-values) of the results obtained after multiple group independence testing

Usage

```
plotGroupIndependence(groups, top = 50, textSize = 10,
  colourLow = "lightgrey", colourMid = "blue", colourHigh = "orange",
  colourMidpoint = 150)
```

Arguments

groups	multiGroupIndependenceTest object (obtained after running testGroupIndependence)
top	Integer: number of attributes to render
textSize	Integer: size of the text
colourLow	Character: name or HEX code of colour for lower values
colourMid	Character: name or HEX code of colour for middle values
colourHigh	Character: name or HEX code of colour for higher values
colourMidpoint	Numeric: midpoint to identify middle values

Value

ggplot object

See Also

parseCategoricalGroups and testGroupIndependence

Examples

```
elements <- paste("patients", 1:50)
ref      <- elements[10:50]
groups   <- list(race=list(asian=elements[1:3],
                           white=elements[4:7],
                           black=elements[8:10]),
                 region=list(european=elements[c(4, 5, 9)],
                             african=elements[c(6:8, 10:50)]))
groupTesting <- testGroupIndependence(ref, groups, elements)
plotGroupIndependence(groupTesting)
```

plotICA *Create multiple scatterplots from ICA*

Description

Create multiple scatterplots from ICA

Usage

```
plotICA(ica, components = seq(10), groups = NULL, ...)
```

Arguments

ica	Object resulting from performICA
components	Numeric: independent components to plot
groups	Matrix: groups to plot indicating the index of interest of the samples (use clinical or sample groups)
...	Arguments passed on to pairsD3::pairsD3

group a optional vector specifying the group each observation belongs to. Used for tooltips and colouring the observations.

subset an optional vector specifying a subset of observations to be used for plotting. Useful when you have a large number of observations, you can specify a random subset.

labels the names of the variables (column names of x used by default).

cex the magnification of the plotting symbol (default=3)

width the width (and height) of the plot when viewed externally.

col an optional (hex) colour for each of the levels in the group vector.

big a logical parameter. Prevents inadvertent plotting of huge data sets. Default limit is 10 variables, to plot more than 10 set big=TRUE.

theme a character parameter specifying whether the theme should be colour colour (default) or black and white bw.

opacity numeric between 0 and 1. The opacity of the plotting symbols (default 0.9).

tooltip an optional vector with the tool tip to be displayed when hovering over an observation. You can include basic html.

leftmar space on the left margin

topmar space on the bottom margin

Value

Multiple scatterplots as a pairsD3 object

Examples

```
data <- scale(USArrests)
ica  <- fastICA::fastICA(data, n.comp=4)
plotICA(ica)

# Colour by groups
```

```
groups <- NULL
groups$sunny <- c("California", "Hawaii", "Florida")
groups$ozEntrance <- c("Kansas")
groups$novel <- c("New Mexico", "New York", "New Hampshire", "New Jersey")
plotICA(ica, groups=groups)
```

---

plotPCA *Create a scatterplot from a PCA object*

Description

Create a scatterplot from a PCA object

Usage

```
plotPCA(pca, pcX = 1, pcY = 2, groups = NULL, individuals = TRUE,
  loadings = FALSE, nLoadings = NULL)
```

Arguments

pca	prcomp object
pcX	Character: name of the X axis of interest from the PCA
pcY	Character: name of the Y axis of interest from the PCA
groups	Matrix: groups to plot indicating the index of interest of the samples (use clinical or sample groups)
individuals	Boolean: plot PCA individuals
loadings	Boolean: plot PCA loadings/rotations
nLoadings	Integer: Number of variables to plot, ordered by those that most contribute to selected principal components (this allows for faster performance as only the most contributing variables are rendered); if NULL, all variables are plotted

Value

Scatterplot as an highchart object

Examples

```
pca <- prcomp(USArrests, scale=TRUE)
plotPCA(pca)
plotPCA(pca, pcX=2, pcY=3)

# Plot both individuals and loadings
plotPCA(pca, pcX=2, pcY=3, loadings=TRUE)
```

plotProtein *Plot protein features*

Description

Plot protein features

Usage

```
plotProtein(molecule)
```

Arguments

molecule	Character: UniProt protein or Ensembl transcript identifier

Value

highcharter object

Examples

```
protein <- "P38398"
plotProtein(protein)

transcript <- "ENST00000488540"
plotProtein(transcript)
```

plotRowStats *Plot sample statistics per row*

Description

Plot sample statistics per row

Usage

```
plotRowStats(data, x, y, xmin = NULL, xmax = NULL, ymin = NULL,
  ymax = NULL, xlim = NULL, ylim = NULL)
```

Arguments

data	Data frame or matrix
x, y	Character: statistic to calculate and display in the plot per row; choose between mean, median, var or range (or transformations of those variables, e.g. log10(var))
xmin, xmax, ymin, ymax	Numeric: minimum and maximum X and Y values to draw in the plot
xlim, ylim	Numeric: X and Y axis range

Value

Plot of data

Examples

```
library(ggplot2)

# Plotting gene expression data
geneExpr <- readFile("ex_gene_expression.RDS")
plotRowStats(geneExpr, "mean", "var^(1/4)") +
    ggtitle("Mean-variance plot") +
    labs(y="Square Root of the Standard Deviation")

# Plotting alternative splicing quantification
annot <- readFile("ex_splicing_annotation.RDS")
junctionQuant <- readFile("ex_junctionQuant.RDS")
psi <- quantifySplicing(annot, junctionQuant, eventType=c("SE", "MXE"))

medianVar <- plotRowStats(psi, x="median", y="var", xlim=c(0, 1)) +
    labs(x="Median PSI", y="PSI variance")
medianVar

rangeVar  <- plotRowStats(psi, x="range", y="log10(var)", xlim=c(0, 1)) +
    labs(x="PSI range", y="log10(PSI variance)")
rangeVar
```

---

plotSurvivalCurves *Plot survival curves*

Description

Plot survival curves

Usage

```
plotSurvivalCurves(surv, mark = TRUE, interval = FALSE,
  pvalue = NULL, title = "Survival analysis", scale = NULL,
  auto = TRUE)
```

Arguments

surv	Survival object
mark	Boolean: mark times? TRUE by default
interval	Boolean: show interval ranges? FALSE by default
pvalue	Numeric: p-value of the survival curves
title	Character: plot title
scale	Character: time scale; default is "days"
auto	Boolean: return the plot automatically prepared (TRUE) or only the bare minimum (FALSE)? TRUE by default

Value

Plot of survival curves

Examples

```
require("survival")
fit <- survfit(Surv(time, status) ~ x, data = aml)
plotSurvivalCurves(fit)
```

`plotSurvivalPvaluesByCutoff` *Plot p-values of survival difference between groups based on multiple cutoffs*

Description

Plot p-values of survival difference between groups based on multiple cutoffs

Usage

```
plotSurvivalPvaluesByCutoff(clinical, data, censoring, event, timeStart,
  timeStop = NULL, followup = "days_to_last_followup",
  significance = 0.05, cutoffs = seq(0, 0.99, 0.01))
```

Arguments

`clinical`	Data frame: clinical data
`data`	Numeric: elements of interest to test against the cutoff
`censoring`	Character: censor using "left", "right", "interval" or "interval2"
`event`	Character: name of column containing time of the event of interest
`timeStart`	Character: name of column containing starting time of the interval or follow up time
`timeStop`	Character: name of column containing ending time of the interval (only relevant for interval censoring)
`followup`	Character: name of column containing follow up time
`significance`	Numeric: significance threshold
`cutoffs`	Numeric: cutoffs to test

Value

p-value plot

plotTranscripts *Plot transcripts*

Description

Plot transcripts

Usage

```
plotTranscripts(info, eventPosition = NULL, event = NULL,
  shiny = FALSE)
```

Arguments

info	Information retrieved from Ensembl
eventPosition	Numeric: coordinates of the alternative splicing event (ignored if event is set)
event	Character: identifier of the alternative splicing event to plot
shiny	Boolean: is the function running in a Shiny session? FALSE by default

Value

NULL (this function is used to modify the Shiny session's state)

Examples

```
event <- "SE_12_-_7985318_7984360_7984200_7982602_SLC2A14"
info  <- queryEnsemblByEvent(event, species="human", assembly="hg19")
## Not run:
plotTranscripts(info, event=event)

## End(Not run)
```

plotVariance *Create the explained variance plot from a PCA*

Description

Create the explained variance plot from a PCA

Usage

```
plotVariance(pca)
```

Arguments

pca	prcomp object

Value

Plot variance as an highchart object

Examples

```
pca <- prcomp(USArrests)
plotVariance(pca)
```

`prepareAnnotationFromEvents` *Prepare annotation from alternative splicing events*

Description

In case more than one data frame with alternative splicing events is given, the events are cross-referenced according to the chromosome, strand and relevant coordinates per event type (see details).

Usage

```
prepareAnnotationFromEvents(...)
```

Arguments

...	Data frame(s) of alternative splicing events to include in the annotation

Details

Events from two or more data frames are cross-referenced based on each event's chromosome, strand and specific coordinates relevant for each event type:

- Skipped exon: constitutive exon 1 end, alternative exon (start and end) and constitutive exon 2 start
- Mutually exclusive exon: constitutive exon 1 end, alternative exon 1 and 2 (start and end) and constitutive exon 2 start
- Alternative 5' splice site: constitutive exon 1 end, alternative exon 1 end and constitutive exon 2 start
- Alternative first exon: same as alternative 5' splice site
- Alternative 3' splice site: constitutive exon 1 end, alternative exon 1 start and constitutive exon 2 start
- Alternative last exon: same as alternative 3' splice site

Value

List of data frames with the annotation from different data frames joined by event type

Note

When cross-referencing events, gene information is discarded.

Examples

```
# Load sample files (SUPPA annotation)
folder <- "extdata/eventsAnnotSample/suppa_output/suppaEvents"
suppaOutput <- system.file(folder, package="psichomics")

# Parse and prepare SUPPA annotation
suppa <- parseSuppaAnnotation(suppaOutput)
annot <- prepareAnnotationFromEvents(suppa)

# Load sample files (rMATS annotation)
folder <- "extdata/eventsAnnotSample/mats_output/ASEvents/"
matsOutput <- system.file(folder, package="psichomics")

# Parse rMATS annotation and prepare combined annotation from rMATS and SUPPA
mats <- parseMatsAnnotation(matsOutput)
annot <- prepareAnnotationFromEvents(suppa, mats)
```

---

prepareJunctionQuantSTAR

Prepare files to be loaded into psichomics

Description

Prepare files to be loaded into psichomics

Usage

```
prepareJunctionQuantSTAR(..., startOffset = -1, endOffset = +1)

prepareGeneQuantSTAR(..., strandedness = c("unstranded", "stranded",
  "stranded (reverse)"))
```

Arguments

`...`	Character: path to file(s) to read
`startOffset`	Numeric: value to offset start position
`endOffset`	Numeric: value to offset end position
`strandedness`	Character: strandedness of RNA-seq protocol; may be one of the following: `unstraded`, `stranded` or `stranded (reverse)`

Value

Prepared file

Examples

```
## Not run:
prepareJunctionQuant("Control rep1"=junctionFile1,
                     "Control rep2"=junctionFile2,
                     "KD rep1"=junctionFile3,
                     "KD rep2"=junctionFile4)
```

```
## End(Not run)
## Not run:
prepareGeneQuant("Control rep1"=geneCountFile1,
                 "Control rep2"=geneCountFile2,
                 "KD rep1"=geneCountFile3,
                 "KD rep2"=geneCountFile4)

## End(Not run)
```

`prepareSRAmetadata` *Prepare files to be loaded into psichomics*

Description

Prepare files to be loaded into psichomics

Usage

```
prepareSRAmetadata(file, output = "psichomics_metadata.txt")

prepareJunctionQuant(..., output = "psichomics_junctions.txt",
  startOffset = NULL, endOffset = NULL)

prepareGeneQuant(..., output = "psichomics_gene_counts.txt",
  strandedness = c("unstranded", "stranded", "stranded (reverse)"))
```

Arguments

`file`	Character: path to file
`output`	Character: path of output file (if `NULL`, only returns the data without saving it to a file)
`...`	Character: path to file(s) to read
`startOffset`	Numeric: value to offset start position
`endOffset`	Numeric: value to offset end position
`strandedness`	Character: strandedness of RNA-seq protocol; may be one of the following: `unstraded`, `stranded` or `stranded (reverse)`

Value

Prepared file

Examples

```
## Not run:
prepareJunctionQuant("Control rep1"=junctionFile1,
                     "Control rep2"=junctionFile2,
                     "KD rep1"=junctionFile3,
                     "KD rep2"=junctionFile4)

## End(Not run)
## Not run:
```

```
prepareGeneQuant("Control rep1"=geneCountFile1,
                 "Control rep2"=geneCountFile2,
                 "KD rep1"=geneCountFile3,
                 "KD rep2"=geneCountFile4)

## End(Not run)
```

`processSurvTerms` *Process survival curves terms to calculate survival curves*

Description

Process survival curves terms to calculate survival curves

Usage

```
processSurvTerms(clinical, censoring, event, timeStart, timeStop = NULL,
  group = NULL, formulaStr = NULL, coxph = FALSE, scale = "days",
  followup = "days_to_last_followup", survTime = NULL)
```

Arguments

`clinical`	Data frame: clinical data
`censoring`	Character: censor using "left", "right", "interval" or "interval2"
`event`	Character: name of column containing time of the event of interest
`timeStart`	Character: name of column containing starting time of the interval or follow up time
`timeStop`	Character: name of column containing ending time of the interval (only relevant for interval censoring)
`group`	Character: group relative to each patient
`formulaStr`	Character: formula to use
`coxph`	Boolean: fit a Cox proportional hazards regression model? FALSE by default
`scale`	Character: rescale the survival time to "days", "weeks", "months" or "years"
`followup`	Character: name of column containing follow up time
`survTime`	survTime object: times to follow up, time start, time stop and event (optional)

Details

If `survTime` is NULL, the survival times will be fetch from the clinical dataset according to the names given in `timeStart`, `timeStop`, `event` and `followup`. This can became quite slow when using the function in a for loop. If these variables are constant, consider running the function `getAttributesTime` to retrieve the time of such columns once and hand the result to the `survTime` argument of this function.

Value

A list with a `formula` object and a data frame with terms needed to calculate survival curves

Examples

```
clinical <- read.table(text = "2549   NA ii  female
                               840   NA i   female
                                NA 1204 iv    male
                                NA  383 iv  female
                              1293   NA iii   male
                                NA 1355 ii    male")
names(clinical) <- c("patient.days_to_last_followup",
                     "patient.days_to_death",
                     "patient.stage_event.pathologic_stage",
                     "patient.gender")
timeStart  <- "days_to_death"
event      <- "days_to_death"
formulaStr <- "patient.stage_event.pathologic_stage + patient.gender"
survTerms  <- processSurvTerms(clinical, censoring="right", event, timeStart,
                               formulaStr=formulaStr)
```

`psichomics` *Start graphical interface of psichomics*

Description

Start graphical interface of psichomics

Usage

```
psichomics(..., reset = FALSE, testData = FALSE)
```

Arguments

`...`	Arguments passed on to `shiny::runApp` **port** The TCP port that the application should listen on. If the `port` is not specified, and the `shiny.port` option is set (with `options(shiny.port = XX)`), then that port will be used. Otherwise, use a random port. **host** The IPv4 address that the application should listen on. Defaults to the `shiny.host` option, if set, or `"127.0.0.1"` if not. See Details. **workerId** Can generally be ignored. Exists to help some editions of Shiny Server Pro route requests to the correct process. **quiet** Should Shiny status messages be shown? Defaults to FALSE. **display.mode** The mode in which to display the application. If set to the value `"showcase"`, shows application code and metadata from a `DESCRIPTION` file in the application directory alongside the application. If set to `"normal"`, displays the application normally. Defaults to `"auto"`, which displays the application in the mode given in its `DESCRIPTION` file, if any. **test.mode** Should the application be launched in test mode? This is only used for recording or running automated tests. Defaults to the `shiny.testmode` option, or FALSE if the option is not set.
`reset`	Boolean: reset Shiny session? Requires package `devtools`
`testData`	Boolean: auto-start with test data

Value

NULL (this function is used to modify the Shiny session's state)

Examples

```
## Not run:
psichomics()

## End(Not run)
```

`quantifySplicing` *Quantify alternative splicing events*

Description

Quantify alternative splicing events

Usage

```
quantifySplicing(annotation, junctionQuant, eventType = c("SE", "MXE",
  "ALE", "AFE", "A3SS", "A5SS"), minReads = 10, genes = NULL)
```

Arguments

`annotation`	List of data frames: annotation for each alternative splicing event type
`junctionQuant`	Data frame: junction quantification
`eventType`	Character: splicing event types to quantify
`minReads`	Integer: discard alternative splicing quantified using a number of reads below this threshold
`genes`	Character: gene symbols for which the splicing quantification of associated splicing events is performed (by default, all splicing events undergo splicing quantification)

Value

Data frame with the quantification of the alternative splicing events

Examples

```
# Calculate PSI for skipped exon (SE) and mutually exclusive (MXE) events
annot <- readFile("ex_splicing_annotation.RDS")
junctionQuant <- readFile("ex_junctionQuant.RDS")

psi <- quantifySplicing(annot, junctionQuant, eventType=c("SE", "MXE"))
```

queryEnsemblByGene *Query information from Ensembl*

Description

Query information from Ensembl

Usage

```
queryEnsemblByGene(gene, species = NULL, assembly = NULL)

queryEnsemblByEvent(event, species, assembly)
```

Arguments

gene	Character: gene
species	Character: species (may be NULL for an Ensembl identifier)
assembly	Character: assembly version (may be NULL for an Ensembl identifier)
event	Character: alternative splicing event

Value

Information from Ensembl

Examples

```
queryEnsemblByGene("BRCA1", "human", "hg19")
queryEnsemblByGene("ENSG00000139618")
event <- "SE_17_-_41251792_41249306_41249261_41246877_BRCA1"
queryEnsemblByEvent(event, species="human", assembly="hg19")
```

readFile *Load local file*

Description

Load local file

Usage

```
readFile(file)
```

Arguments

file	Character: path to the file

Value

Loaded file

Examples

```
junctionQuant <- readFile("ex_junctionQuant.RDS")
```

`rowMeans` *Calculate mean or variance for each row of a matrix*

Description

Calculate mean or variance for each row of a matrix

Usage

```
rowMeans(mat, na.rm = FALSE)

rowVars(mat, na.rm = FALSE)
```

Arguments

`mat`	Matrix
`na.rm`	Boolean: remove missing values (`NA`)?

Value

Vector of means or variances

Examples

```
df <- rbind("Gene 1"=c(3, 5, 7), "Gene 2"=c(8, 2, 4), "Gene 3"=c(9:11))
rowMeans(df)
rowVars(df)
```

`survdiffTerms` *Test Survival Curve Differences*

Description

Tests if there is a difference between two or more survival curves using the G^ρ family of tests, or for a single curve against a known alternative.

Usage

```
survdiffTerms(survTerms, ...)
```

Arguments

survTerms	survTerms object: survival terms obtained after running processSurvTerms (see examples)
...	Arguments passed on to survival::survdiff

subset expression indicating which subset of the rows of data should be used in the fit. This can be a logical vector (which is replicated to have length equal to the number of observations), a numeric vector indicating which observation numbers are to be included (or excluded if negative), or a character vector of row names to be included. All observations are included by default.

na.action a missing-data filter function. This is applied to the model.frame after any subset argument has been used. Default is options()$na.action.

rho a scalar parameter that controls the type of test.

timefix process times through the aeqSurv function to eliminate potential round-off issues.

Value

a list with components:

n	the number of subjects in each group.
obs	the weighted observed number of events in each group. If there are strata, this will be a matrix with one column per stratum.
exp	the weighted expected number of events in each group. If there are strata, this will be a matrix with one column per stratum.
chisq	the chisquare statistic for a test of equality.
var	the variance matrix of the test.
strata	optionally, the number of subjects contained in each stratum.

METHOD

This function implements the G-rho family of Harrington and Fleming (1982), with weights on each death of $S(t)^\rho$, where $S(t)$ is the Kaplan-Meier estimate of survival. With rho = 0 this is the log-rank or Mantel-Haenszel test, and with rho = 1 it is equivalent to the Peto & Peto modification of the Gehan-Wilcoxon test.

If the right hand side of the formula consists only of an offset term, then a one sample test is done. To cause missing values in the predictors to be treated as a separate group, rather than being omitted, use the factor function with its exclude argument.

References

Harrington, D. P. and Fleming, T. R. (1982). A class of rank test procedures for censored survival data. *Biometrika* **69**, 553-566.

Examples

```
clinical <- read.table(text = "2549   NA ii  female
                                840   NA i   female
                                 NA 1204 iv    male
                                 NA  383 iv  female
```

```
                                1293   NA iii   male
                                  NA 1355 ii    male")
names(clinical) <- c("patient.days_to_last_followup",
                     "patient.days_to_death",
                     "patient.stage_event.pathologic_stage",
                     "patient.gender")
timeStart  <- "days_to_death"
event      <- "days_to_death"
formulaStr <- "patient.stage_event.pathologic_stage + patient.gender"
survTerms  <- processSurvTerms(clinical, censoring="right", event, timeStart,
                               formulaStr=formulaStr)
survdiffTerms(survTerms)
```

`survfit.survTerms` *Create survival curves*

Description

Create survival curves

Usage

```
## S3 method for class 'survTerms'
survfit(survTerms, ...)
```

Arguments

`survTerms`	`survTerms` object: survival terms obtained after running `processSurvTerms` (see examples)
`...`	Arguments passed on to `survival::survdiff`

subset expression indicating which subset of the rows of data should be used in the fit. This can be a logical vector (which is replicated to have length equal to the number of observations), a numeric vector indicating which observation numbers are to be included (or excluded if negative), or a character vector of row names to be included. All observations are included by default.

na.action a missing-data filter function. This is applied to the `model.frame` after any subset argument has been used. Default is `options()$na.action`.

rho a scalar parameter that controls the type of test.

timefix process times through the `aeqSurv` function to eliminate potential round-off issues.

Details

A survival curve is based on a tabulation of the number at risk and number of events at each unique death time. When time is a floating point number the definition of "unique" is subject to interpretation. The code uses factor() to define the set. For further details see the documentation for the appropriate method, i.e., `?survfit.formula` or `?survfit.coxph`.

A survfit object may contain a single curve, a set of curves, or a matrix curves. Predicted curves from a `coxph` model have one row for each stratum in the Cox model fit and one column for each specified covariate set. Curves from a multi-state model have one row for each stratum and a column

for each state, the strata correspond to predictors on the right hand side of the equation. The default printing and plotting order for curves is by column, as with other matrices.

Curves can be subscripted using either a single or double subscript. If the set of curves is a matrix, as in the above, and one of the dimensions is 1 then the code allows a single subscript to be used. (That is, it is not quite as general as using a single subscript for a numeric matrix.)

Value

An object of class `survfit` containing one or more survival curves.

Examples

```
library("survival")
clinical <- read.table(text = "2549   NA ii  female
                                840   NA i   female
                                 NA 1204 iv    male
                                 NA  383 iv  female
                               1293   NA iii   male
                                 NA 1355 ii    male")
names(clinical) <- c("patient.days_to_last_followup",
                     "patient.days_to_death",
                     "patient.stage_event.pathologic_stage",
                     "patient.gender")
timeStart  <- "days_to_death"
event      <- "days_to_death"
formulaStr <- "patient.stage_event.pathologic_stage + patient.gender"
survTerms  <- processSurvTerms(clinical, censoring="right", event, timeStart,
                               formulaStr=formulaStr)
survfit(survTerms)
```

---

testGroupIndependence *Multiple independence tests between reference groups and list of groups*

Description

Test multiple contigency tables comprised by two groups (one reference group and another containing remaing elements) and provided groups.

Usage

```
testGroupIndependence(ref, groups, elements, pvalueAdjust = "BH")
```

Arguments

`ref`	List of character: list of groups where each element contains the identifiers of respective elements
`groups`	List of characters: list of groups where each element contains the identifiers of respective elements
`elements`	Character: all available elements (if a data frame is given, its rownames will be used)
`pvalueAdjust`	Character: method used to adjust p-values (see Details)

Details

The following methods for p-value adjustment are supported by using the respective string in the `pvalueAdjust` argument:

- `none`: Do not adjust p-values
- `BH`: Benjamini-Hochberg's method (false discovery rate)
- `BY`: Benjamini-Yekutieli's method (false discovery rate)
- `bonferroni`: Bonferroni correction (family-wise error rate)
- `holm`: Holm's method (family-wise error rate)
- `hochberg`: Hochberg's method (family-wise error rate)
- `hommel`: Hommel's method (family-wise error rate)

Value

`multiGroupIndependenceTest` object, a data frame containing:

`attribute`	Name of the original groups compared against the reference groups
`table`	Contigency table used for testing
`pvalue`	Fisher's exact test's p-value

See Also

`parseCategoricalGroups` and `plotGroupIndependence`

Examples

```
elements <- paste("patients", 1:10)
ref      <- elements[5:10]
groups   <- list(race=list(asian=elements[1:3],
                           white=elements[4:7],
                           black=elements[8:10]),
                 region=list(european=elements[c(4, 5, 9)],
                             african=elements[c(6:8, 10)]))
groupTesting <- testGroupIndependence(ref, groups, elements)
# View(groupTesting)
```

`testSurvival` *Test the survival difference between groups of patients*

Description

Test the survival difference between groups of patients

Usage

```
testSurvival(survTerms, ...)
```

Arguments

survTerms	survTerms object: survival terms obtained after running processSurvTerms (see examples)
...	Arguments passed on to survival::survdiff
	subset expression indicating which subset of the rows of data should be used in the fit. This can be a logical vector (which is replicated to have length equal to the number of observations), a numeric vector indicating which observation numbers are to be included (or excluded if negative), or a character vector of row names to be included. All observations are included by default.
	na.action a missing-data filter function. This is applied to the model.frame after any subset argument has been used. Default is options()$na.action.
	rho a scalar parameter that controls the type of test.
	timefix process times through the aeqSurv function to eliminate potential round-off issues.

Value

p-value of the survival difference or NA

Note

Instead of raising errors, an NA is returned

Examples

```
require("survival")
data <- aml
timeStart  <- "event"
event      <- "event"
followup   <- "time"
data$event <- NA
data$event[aml$status == 1] <- aml$time[aml$status == 1]
censoring  <- "right"
formulaStr <- "x"
survTerms <- processSurvTerms(data, censoring=censoring, event=event,
                              timeStart=timeStart, followup=followup,
                              formulaStr=formulaStr)
testSurvival(survTerms)
```

[.GEandAScorrelation *Subset correlation results between gene expression and splicing quantification*

Description

Subset correlation results between gene expression and splicing quantification

Usage

```
## S3 method for class 'GEandAScorrelation'
x[genes = NULL, ASevents = NULL]
```

Arguments

x	GEandAScorrelation object to subset
genes	Character: genes
ASevents	Character: ASevents

Value

GEandAScorrelation object subset

Index