Package 'psichomics'

October 14, 2021

Title Graphical Interface for Alternative Splicing Quantification, Analysis and Visualisation

Version 1.18.6

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 (>= 4.0), shiny (>= 1.7.0), shinyBS

License MIT + file LICENSE

LazyData true

RoxygenNote 7.1.2

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

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

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_inclusionLevelsFilter.R'
'data_local.R' 'data_recount.R' 'events_suppa.R'
'events_vastTools.R' 'events_miso.R' 'events_mats.R' 'events.R'
'formats_SraRunTableSampleInfo.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_gtexV8JunctionReads.R'
'formats_psichomicsGeneExpression.R'
'formats_psichomicsInclusionLevels.R'
'formats_recountSampleInfo.R'
'formats_vasttoolsGeneExpression.R'
'formats_vasttoolsInclusionLevels.R'
'formats_vasttoolsInclusionLevelsTidy.R' 'groups.R' 'help.R'
'utils_drawSplicingEvent.R' 'utils_eventParsing.R'
'utils_fileBrowserDialog.R' 'utils_interactiveGgplot.R'
'utils_interface.R'

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

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

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

Language en-GB

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

git_branch RELEASE_3_13

git_last_commit 2ffd219

git_last_commit_date 2021-10-04

Date/Publication 2021-10-14

Author Nuno Saraiva-Agostinho [aut, cre]
(<https://orcid.org/0000-0002-5549-105X>),
Nuno Luís Barbosa-Morais [aut, led, ths]
(<https://orcid.org/0000-0002-1215-0538>),
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:

.onAttach 4
assignValuePerSubject 5
calculateLoadingsContribution 6
colSums,EList-method 7
convertGeneIdentifiers 8
correlateGEandAS 9
createGroupByAttribute 10
diffAnalyses 11
discardLowCoveragePSIvalues 12
ensemblToUniprot 13
filterGeneExpr 14
filterGroups 15
filterPSI 16
getAttributesTime 17
getDownloadsFolder 18
getGeneList 19
getGtexDataTypes 19
getGtexTissues 20
getSampleFromSubject 21
getSplicingEventData 22
getSplicingEventFromGenes 22
getSplicingEventTypes 23
getSubjectFromSample 24
getTCGAdataTypes 25
groupPerElem 25
isFirebrowseUp 26
labelBasedOnCutoff 27
listSplicingAnnotations 28
loadAnnotation 28
loadGtexData 29
loadLocalFiles 30

loadSRAproject . . . 31
loadTCGAdata . . . 32
normaliseGeneExpression . . . 33
optimalSurvivalCutoff . . . 35
parseCategoricalGroups . . . 36
parseSplicingEvent . . . 37
parseSuppaAnnotation . . . 38
parseTCGAsampleTypes . . . 40
performICA . . . 41
performPCA . . . 42
plotDistribution . . . 43
plotGeneExprPerSample . . . 46
plotGroupIndependence . . . 47
plotICA . . . 48
plotLibrarySize . . . 49
plotPCA . . . 50
plotPCAvariance . . . 51
plotProtein . . . 52
plotRowStats . . . 53
plotSplicingEvent . . . 54
plotSurvivalCurves . . . 56
plotSurvivalPvaluesByCutoff . . . 57
plotTranscripts . . . 59
prepareAnnotationFromEvents . . . 60
prepareSRAmetadata . . . 61
processSurvTerms . . . 62
psichomics . . . 64
quantifySplicing . . . 66
queryEnsemblByGene . . . 67
readFile . . . 68
survdiffTerms . . . 68
survfit.survTerms . . . 70
t.sticky . . . 71
testGroupIndependence . . . 72
testSurvival . . . 73
[.GEandAScorrelation . . . 74

Index **78**

`.onAttach` *Print startup message*

Description

Print startup message

Usage

```
.onAttach(libname, pkgname)
```

Arguments

libname	Character: library name
pkgname	Character: package name

Value

Startup message

---

assignValuePerSubject *Assign average sample values to their corresponding subjects*

Description

Assign average sample values to their corresponding subjects

Usage

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

Arguments

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

Value

Values per subject

See Also

Other functions to analyse survival: `getAttributesTime()`, `labelBasedOnCutoff()`, `optimalSurvivalCutoff()`, `plotSurvivalCurves()`, `plotSurvivalPvaluesByCutoff()`, `processSurvTerms()`, `survdiffTerms()`, `survfit.survTerms()`, `testSurvival()`

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("Subject", 1:3), 2)
names(match) <- colnames(psi)

# Assign PSI values to each subject based on the PSI of their samples
assignValuePerSubject(psi[3, ], match)
```

---

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`
- `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/

See Also

Other functions to analyse principal components: performPCA(), plotPCAvariance(), plotPCA()

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 with genome wide annotation for an organism or character with species name to query OrgDb, e.g. "Homo sapiens"
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.

See Also

Other functions for gene expression pre-processing: filterGeneExpr(), normaliseGeneExpression(), plotGeneExprPerSample(), plotLibrarySize(), plotRowStats()

Examples

```
# Use species name to automatically look for a OrgDb database
sp <- "Homo sapiens"
genes <- c("ENSG00000012048", "ENSG00000083093", "ENSG00000141510",
           "ENSG00000051180")
convertGeneIdentifiers(sp, genes)
```

```
convertGeneIdentifiers(sp, genes, key="ENSEMBL", target="UNIPROT")

# Alternatively, set the annotation database directly
ah <- AnnotationHub::AnnotationHub()
sp <- AnnotationHub::query(ah, c("OrgDb", "Homo sapiens"))[[1]]
columns(sp) # these attributes can be used to change the attributes

convertGeneIdentifiers(sp, genes)
convertGeneIdentifiers(sp, genes, key="ENSEMBL", target="UNIPROT")
```

`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)
`...`	Extra parameters passed to cor.test

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

See Also

Other functions to correlate gene expression and alternative splicing: `[.GEandAScorrelation()`

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")
```

---

createGroupByAttribute

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

---

Description

Elements are identified by their respective row name.

Usage

```
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

See Also

Other functions for data grouping: getGeneList(), getSampleFromSubject(), getSubjectFromSample(), groupPerElem(), plotGroupIndependence(), testGroupIndependence()

Examples

```
df <- data.frame(gender=c("male", "female"),
                 stage=paste("stage", c(1, 3, 1, 4, 2, 3, 2, 2)))
rownames(df) <- paste0("subject-", 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,
  inputID = "sparklineInput"
)
```

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)
`inputID`	Character: identifier of input to get attributes of clicked event (Shiny only)

Details

The following statistical analyses may be performed simultaneously via 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 p-value adjustment methods are supported via 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

See Also

Other functions to perform and plot differential analyses: plotDistribution()

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)
```

---

discardLowCoveragePSIvalues

Remove alternative splicing quantification values based on coverage

---

Description

Remove alternative splicing quantification values based on coverage

Usage

```
discardLowCoveragePSIvalues(
  psi,
  minReads = 10,
  vasttoolsScoresToDiscard = c("VLOW", "N")
)
```

Arguments

psi	Data frame or matrix: alternative splicing quantification
minReads	Currently this argument does nothing
vasttoolsScoresToDiscard	Character: if you are using inclusion levels from VAST-TOOLS, filter the data based on quality scores for read coverage, e.g. use vasttoolsScoresToDiscard = c("SOK","OK","LOW") to only keep events with good read coverage (by default, events are not filtered based on quality scores); read https://github.com/vastgroup/vast-tools for more information on VAST-TOOLS quality scores

Value

Alternative splicing quantification data with missing values for any values with insufficient coverage

---

ensemblToUniprot *Convert from Ensembl to UniProt identifier*

Description

Convert from Ensembl to UniProt identifier

Usage

```
ensemblToUniprot(protein)
```

Arguments

protein	Character: Ensembl identifier

Value

UniProt protein identifier

See Also

Other functions to retrieve external information: plotProtein(), plotTranscripts(), queryEnsemblByGene()

Examples

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

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

`filterGeneExpr` *Filter genes based on their expression*

Description

Uses `filterByExpr` to determine genes with sufficiently large counts to retain for statistical analysis.

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 a worthwhile number of samples (check `filterByExpr` for more information)
`minTotalCounts`	Numeric: minimum total number of read counts per gene

Value

Boolean vector indicating which genes have sufficiently large counts

See Also

Other functions for gene expression pre-processing: `convertGeneIdentifiers()`, `normaliseGeneExpression()`, `plotGeneExprPerSample()`, `plotLibrarySize()`, `plotRowStats()`

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	Character: elements
group	Character: respective group of each 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 as an attribute.

Examples

```
# Removes groups with less than two elements
vec <- 1:6
names(vec) <- paste("sample", letters[1:6])
filterGroups(vec, c("A", "B", "B", "C", "D", "D"), threshold=2)
```

`filterPSI` *Filter alternative splicing quantification*

Description

Filter alternative splicing quantification

Usage

```
filterPSI(
  psi,
  eventType = NULL,
  eventSubtype = NULL,
  minPSI = -Inf,
  maxPSI = Inf,
  minMedian = -Inf,
  maxMedian = Inf,
  minLogVar = -Inf,
  maxLogVar = Inf,
  minRange = -Inf,
  maxRange = Inf
)
```

Arguments

`psi`	Data frame or matrix: alternative splicing quantification
`eventType`	Character: filter data based on event type; check all event types available by using `getSplicingEventTypes(psi)`, where `psi` is the alternative splicing quantification data; if `eventType = NULL`, events are not filtered by event type
`eventSubtype`	Character: filter data based on event subtype; check all event subtypes available in your data by using `unique(getSplicingEventData(psi)$subtype)`, where `psi` is the alternative splicing quantification data; if `eventSubtype = NULL`, events are not filtered by event subtype
`minPSI`	Numeric: minimum PSI value
`maxPSI`	Numeric: maximum PSI value
`minMedian`	Numeric: minimum median PSI per splicing event
`maxMedian`	Numeric: maximum median PSI per splicing event
`minLogVar`	Numeric: minimum log10(PSI variance) per splicing event
`maxLogVar`	Numeric: maximum log10(PSI variance) per splicing event
`minRange`	Numeric: minimum PSI range across samples per splicing event
`maxRange`	Numeric: maximum PSI range across samples per splicing event

Value

Boolean vector indicating which splicing events pass the thresholds

See Also

Other functions for PSI quantification: `getSplicingEventTypes()`, `listSplicingAnnotations()`, `loadAnnotation()`, `plotRowStats()`, `quantifySplicing()`

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"))
# Filter PSI
psi[filterPSI(psi, minMedian=0.05, maxMedian=0.95, minRange=0.15), ]
```

---

getAttributesTime *Get time values for given columns in a clinical dataset*

Description

Get time values for given columns in a clinical dataset

Usage

```
getAttributesTime(
  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

See Also

Other functions to analyse survival: assignValuePerSubject(), labelBasedOnCutoff(), optimalSurvivalCutoff(), plotSurvivalCurves(), plotSurvivalPvaluesByCutoff(), processSurvTerms(), survdiffTerms(), survfit.survTerms(), testSurvival()

Examples

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

---

getDownloadsFolder *Get the path to the Downloads folder*

Description

Get the path to the Downloads folder

Usage

```
getDownloadsFolder()
```

Value

Path to Downloads folder

See Also

Other functions associated with TCGA data retrieval: getTCGAdataTypes(), isFirebrowseUp(), loadTCGAdata(), parseTCGAsampleTypes()

Other functions associated with GTEx data retrieval: getGtexDataTypes(), getGtexTissues(), loadGtexData()

Other functions associated with SRA data retrieval: loadSRAproject()

Examples

```
getDownloadsFolder()
```

getGeneList *Get curated, literature-based gene lists*

Description

Available gene lists:

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

Usage

```
getGeneList(genes = NULL)
```

Arguments

genes	Vector of characters: intersect lists with given genes (lists with no matching genes will not be returned)

Value

List of genes

See Also

Other functions for data grouping: `createGroupByAttribute()`, `getSampleFromSubject()`, `getSubjectFromSample()`, `groupPerElem()`, `plotGroupIndependence()`, `testGroupIndependence()`

Examples

```
getGeneList()
```

getGtexDataTypes *Get GTEx data information*

Description

Get GTEx data information

Usage

```
getGtexDataTypes()

getGtexReleases()
```

Value

GTEx data information

See Also

Other functions associated with GTEx data retrieval: `getDownloadsFolder()`, `getGtexTissues()`, `loadGtexData()`

Examples

```
getGtexDataTypes()
getGtexReleases()
```

`getGtexTissues` *Get GTEx tissues from given GTEx sample attributes*

Description

Get GTEx tissues from given GTEx sample attributes

Usage

```
getGtexTissues(folder = getDownloadsFolder(), release = getGtexReleases()[[1]])
```

Arguments

`folder`	Character: folder containing data
`release`	Numeric: GTEx data release to load

Value

Character: available tissues

See Also

Other functions associated with GTEx data retrieval: `getDownloadsFolder()`, `getGtexDataTypes()`, `loadGtexData()`

Examples

```
## Not run:
getGtexTissues()

## End(Not run)
```

`getSampleFromSubject` *Get samples matching the given subjects*

Description

Get samples matching the given subjects

Usage

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

Arguments

`patients`	Character or list of characters: subject identifiers
`samples`	Character: sample identifiers
`clinical`	Data frame or matrix: clinical dataset
`rm.NA`	Boolean: remove missing values?
`match`	Integer: vector of subject index with the sample identifiers as name to save time (optional)
`showMatch`	Boolean: show matching subject index?

Value

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

See Also

Other functions for data grouping: `createGroupByAttribute()`, `getGeneList()`, `getSubjectFromSample()`, `groupPerElem()`, `plotGroupIndependence()`, `testGroupIndependence()`

Examples

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

getSplicingEventData *Get splicing event information for given alternative splicing quantification data*

Description

Get splicing event information for given alternative splicing quantification data

Usage

```
getSplicingEventData(psi)
```

Arguments

psi	Matrix or data frame: alternative splicing quantification data

Value

Matrix or data frame containing splicing event information for alternative splicing events in psi (if available)

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

Description

Get alternative splicing events from genes or vice-versa

Usage

```
getSplicingEventFromGenes(genes, ASevents, data = NULL)

getGenesFromSplicingEvents(ASevents, data = NULL)
```

Arguments

genes	Character: gene symbols (or TCGA-styled gene symbols)
ASevents	Character: alternative splicing events
data	Matrix or data frame: alternative splicing information

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 *Get supported splicing event types*

Description

Get supported splicing event types

Usage

```
getSplicingEventTypes(psi = NULL, acronymsAsNames = FALSE)
```

Arguments

psi	Data frame or matrix: alternative splicing quantification data
acronymsAsNames	Boolean: return acronyms as names?

Value

Named character vector with splicing event types

See Also

Other functions for PSI quantification: filterPSI(), listSplicingAnnotations(), loadAnnotation(), plotRowStats(), quantifySplicing()

Examples

```
getSplicingEventTypes()
```

---

getSubjectFromSample *Get subjects from given samples*

Description

Get subjects from given samples

Usage

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

Arguments

sampleId	Character: sample identifiers
patientId	Character: subject identifiers to filter by (optional; if a matrix or data frame is given, its rownames will be used to infer the subject identifiers)
na	Boolean: return NA for samples with no matching subjects
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: subject identifiers corresponding to the given samples

See Also

Other functions for data grouping: createGroupByAttribute(), getGeneList(), getSampleFromSubject(), groupPerElem(), plotGroupIndependence(), testGroupIndependence()

Examples

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

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

getTCGAdataTypes *Get available parameters for TCGA data*

Description

Parameters obtained via FireBrowse

Usage

```
getTCGAdataTypes()

getTCGAdates()

getTCGAcohorts(cohort = NULL)
```

Arguments

cohort	Character: filter results by cohorts (optional)

Value

Parsed response

See Also

Other functions associated with TCGA data retrieval: `getDownloadsFolder()`, `isFirebrowseUp()`, `loadTCGAdata()`, `parseTCGAsampleTypes()`

Examples

```
getTCGAdataTypes()
if (isFirebrowseUp()) getTCGAdates()
if (isFirebrowseUp()) getTCGAcohorts()
```

groupPerElem *Assign one group to each element*

Description

Assign one group to each element

Usage

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

Arguments

groups	List of integers: groups of elements
elem	Character: all elements available
outerGroupName	Character: name to give to outer group (if NULL, only show elements matched to their respective groups)

Value

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

See Also

Other functions for data grouping: createGroupByAttribute(), getGeneList(), getSampleFromSubject(), getSubjectFromSample(), plotGroupIndependence(), testGroupIndependence()

Examples

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

---

isFirebrowseUp *Check if Rhrefhttp://firebrowse.org/api-docs/FireBrowse API is running*

Description

Check if FireBrowse API is running

Usage

```
isFirebrowseUp()
```

Value

Invisible TRUE if the FireBrowse API is working; otherwise, raises a warning with the status code and a brief explanation.

See Also

Other functions associated with TCGA data retrieval: getDownloadsFolder(), getTCGAdataTypes(), loadTCGAdata(), parseTCGAsampleTypes()

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
`gte`	Boolean: test using greater than or equal than cutoff (`TRUE`) or less than or equal than cutoff (`FALSE`)?

Value

Labelled groups

See Also

Other functions to analyse survival: `assignValuePerSubject()`, `getAttributesTime()`, `optimalSurvivalCutoff()`, `plotSurvivalCurves()`, `plotSurvivalPvaluesByCutoff()`, `processSurvTerms()`, `survdiffTerms()`, `survfit.survTerms()`, `testSurvival()`

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 alternative splicing annotations*

Description

List alternative splicing annotations

Usage

```
listSplicingAnnotations(species = NULL, assembly = NULL, date = NULL)
```

Arguments

species	Character: filter results by species (regular expression)
assembly	Character: filter results by assembly (regular expression)
date	Character: filter results by date (regular expression)

Value

Named character vector with splicing annotation names

See Also

Other functions for PSI quantification: filterPSI(), getSplicingEventTypes(), loadAnnotation(), plotRowStats(), quantifySplicing()

Examples

```
listSplicingAnnotations() # Return all alternative splicing annotations
listSplicingAnnotations(assembly="hg19") # Search for hg19 annotation
listSplicingAnnotations(assembly="hg38") # Search for hg38 annotation
listSplicingAnnotations(date="201(7|8)") # Search for 2017 or 2018 annotation
```

loadAnnotation *Load alternative splicing annotation from* AnnotationHub

Description

Load alternative splicing annotation from AnnotationHub

Usage

```
loadAnnotation(annotation, cache = NULL)
```

Arguments

annotation	Character: annotation to load
cache	Character: directory path of cache (if NULL, default location is AnnotationHub::getAnnotationHubOpti

Value

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

See Also

Other functions for PSI quantification: filterPSI(), getSplicingEventTypes(), listSplicingAnnotations(), plotRowStats(), quantifySplicing()

Examples

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

## End(Not run)
```

---

loadGtexData *Download and load GTEx data*

Description

Download and load GTEx data

Usage

```
loadGtexData(
  folder = getDownloadsFolder(),
  data = getGtexDataTypes(),
  tissue = NULL,
  release = getGtexReleases()[[1]],
  progress = TRUE
)
```

Arguments

folder	Character: folder containing data
data	Character: data types to load (see getGtexDataTypes)
tissue	Character: tissues to load (if NULL, load all); tissue selection may speed up data loading
release	Numeric: GTEx data release to load
progress	Boolean: display progress?

Value

List with loaded data

See Also

Other functions associated with GTEx data retrieval: `getDownloadsFolder()`, `getGtexDataTypes()`, `getGtexTissues()`

Other functions to load data: `loadLocalFiles()`, `loadSRAproject()`, `loadTCGAdata()`

Examples

```
## Not run:
# Download and load all available GTEx data
data <- loadGtexData()

# Download and load only junction quantification and sample info from GTEx
getGtexDataTypes()
data <- loadGtexData(data=c("sampleInfo", "junctionQuant"))

# Download and load only data for specific tissues
getGtexTissues()
data <- loadGtexData(tissue=c("Stomach", "Small Intestine"))

# Download and load data from a specific GTEx data release
data <- loadGtexData(tissue=c("Stomach", "Small Intestine"), release=7)

## End(Not run)
```

---

loadLocalFiles *Load local files*

Description

Load local files

Usage

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

Arguments

folder	Character: path to folder or ZIP archive
ignore	Character: skip folders and filenames that match the expression
name	Character: name
verbose	Boolean: print steps?

Value

List of data frames from valid files

See Also

Other functions to load data: loadGtexData(), loadSRAproject(), loadTCGAdata()

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 via* R*hrefhttps://jhubiostatistics.shinyapps.io/recount/recount2*

Description

Download and load SRA projects via recount2

Usage

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

Arguments

project	Character: SRA project identifiers (check recount_abstract)
outdir	Character: directory to store the downloaded files

Value

List with loaded projects

See Also

Other functions associated with SRA data retrieval: `getDownloadsFolder()`

Other functions to load data: `loadGtexData()`, `loadLocalFiles()`, `loadTCGAdata()`

Examples

```
## Not run:
View(recount::recount_abstract)
sra <- loadSRAproject("SRP053101")
names(sra)
names(sra[[1]])

## End(Not run)
```

`loadTCGAdata` *Download and process TCGA data*

Description

TCGA data obtained via FireBrowse

Usage

```
loadTCGAdata(
  folder = getDownloadsFolder(),
  data = c("clinical", "junction_quantification", "RSEM_genes"),
  exclude = c(".aux.", ".mage-tab.", "MANIFEST.txt"),
  ...,
  download = TRUE
)
```

Arguments

`folder`	Character: directory to store the downloaded archives (by default, saves to `getDownloadsFolder()`)
`data`	Character: data to load (see `getTCGAdataTypes()`)
`exclude`	Character: files and folders to exclude from downloading and from loading into R (by default, exclude files containing `.aux.`, `.mage-tab.` and `MANIFEST.TXT`)
`...`	Arguments passed on to `queryFirebrowseData` `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 (optional) sort_by String: column used to sort the data (by default, sort by cohort)
download	Boolean: download missing files

Value

A list with the loaded data, unless required files are unavailable and download = FALSE (if so, it returns the URL of files to download)

See Also

Other functions associated with TCGA data retrieval: getDownloadsFolder(), getTCGAdataTypes(), isFirebrowseUp(), parseTCGAsampleTypes()

Other functions to load data: loadGtexData(), loadLocalFiles(), loadSRAproject()

Examples

```
getTCGAcohorts()
getTCGAdataTypes()
## Not run:
loadTCGAdata(cohort = "ACC", data_type = "Clinical")

## End(Not run)
```

---

normaliseGeneExpression

Filter and normalise gene expression

Description

Gene expression is filtered and normalised in the following steps:

- Filter gene expression;
- Normalise gene expression with calcNormFactors;
- If performVoom = FALSE, compute counts per million (CPM) using cpm and log2-transform values if log2transform = TRUE;
- If performVoom = TRUE, use voom to compute log2-CPM, quantile-normalise (if method = "quantile") and estimate mean-variance relationship to calculate observation-level weights.

Usage

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

normalizeGeneExpression(
  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 (if NULL, skip filtering)
method	Character: normalisation method, including TMM, RLE, upperquartile, none or quantile (see Details)
p	numeric value between 0 and 1 specifying which quantile of the counts should be used by 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 (using voom)?

Details

edgeR::calcNormFactors will be used to normalise gene expression if method is TMM, RLE, upperquartile or none. If performVoom = TRUE, voom will only normalise if method = "quantile".

Available normalisation methods:

- TMM is recommended for most RNA-seq data where more than half of the genes are believed not differentially expressed between any pair of samples;
- RLE calculates the median library from the geometric mean of all columns and the median ratio of each sample to the median library is taken as the scale factor;
- upperquartile calculates the scale factors from a given quantile of the counts for each library, after removing genes with zero counts in all libraries;
- quantile forces the entire empirical distribution of each column to be identical (only performed if performVoom = TRUE).

Value

Filtered and normalised gene expression

See Also

Other functions for gene expression pre-processing: `convertGeneIdentifiers()`, `filterGeneExpr()`, `plotGeneExprPerSample()`, `plotLibrarySize()`, `plotRowStats()`

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
)
```

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 are used 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, bounds are set to lower = 0 and upper = 1 if all data values are within that interval; otherwise, lower = min(data,na.rm = TRUE) and upper = max(data,na.rm = TRUE))

Value

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

See Also

Other functions to analyse survival: assignValuePerSubject(), getAttributesTime(), labelBasedOnCutoff(), plotSurvivalCurves(), plotSurvivalPvaluesByCutoff(), processSurvTerms(), survdiffTerms(), survfit.survTerms(), testSurvival()

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("subject", 1:3)
parseCategoricalGroups(df)
```

---

parseSplicingEvent *Parse alternative splicing event identifier*

Description

Parse alternative splicing event identifier

Usage

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

Arguments

event	Character: event identifier
char	Boolean: return character vector 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 = TRUE and char = TRUE

coords	Boolean: display extra coordinates regarding the alternative and constitutive regions of alternative splicing events? Only used if char = FALSE
data	Matrix or data frame: alternative splicing information

Value

Data.frame containing type of event, chromosome, strand, gene and position of alternative splicing events or character with that same information (depending on what is available)

Examples

```
events <- c(
  "A3SS_15_+_63353138_63353912_63353397_TPM1",
  "A3SS_11_-_61118463_61117115_61117894_CYB561A3",
  "A5SS_21_+_48055675_48056459_48056808_PRMT2",
  "A5SS_1_-_1274742_1274667_1274033_DVL1",
  "AFE_9_+_131902430_131901928_131904724_PPP2R4",
  "AFE_5_-_134686513_134688636_134681747_H2AFY",
  "ALE_12_+_56554104_56554410_56555171_MYL6",
  "ALE_8_-_38314874_38287466_38285953_FGFR1",
  "SE_9_+_6486925_6492303_6492401_6493826_UHRF2",
  "SE_19_-_5218431_5216778_5216731_5215606_PTPRS",
  "MXE_15_+_63335142_63335905_63336030_63336226_63336351_63349184_TPM1",
  "MXE_17_-_74090495_74087316_74087224_74086478_74086410_74085401_EXOC7")
parseSplicingEvent(events)
```

---

parseSuppaAnnotation *Parse events from alternative splicing annotation*

Description

Parse 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?
novelEvents	Boolean: parse events detected due to novel splice sites

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

See Also

Other functions to prepare alternative splicing annotations: prepareAnnotationFromEvents()

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)
```

---

parseTCGAsampleTypes *Parse sample information from TCGA sample identifiers*

Description

Parse sample information from TCGA sample identifiers

Usage

```
parseTCGAsampleTypes(
  samples,
  filename = system.file("extdata", "TCGAsampleType.RDS", package = "psichomics")
)

parseTCGAsampleInfo(samples, match = NULL)
```

Arguments

samples	Character: sample identifiers
filename	Character: path to RDS file containing corresponding types
match	Integer: match between samples and subjects (NULL by default; performs the match)

Value

Metadata associated with each TCGA sample

See Also

Other functions associated with TCGA data retrieval: `getDownloadsFolder()`, `getTCGAdataTypes()`, `isFirebrowseUp()`, `loadTCGAdata()`

Examples

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

Other functions to analyse independent components: plotICA()

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
...	Arguments passed on to stats::prcomp

Value

PCA result in a prcomp object

See Also

Other functions to analyse principal components: calculateLoadingsContribution(), plotPCAvariance(), plotPCA()

Examples

```
performPCA(USArrests)
```

---

plotDistribution *Plot sample distribution*

Description

The tooltip shows the median, variance, maximum, minimum and number of non-NA samples of each data series, as well as sample names if available.

Usage

```
plotDistribution(
  data,
  groups = NULL,
  rug = length(data) < 500,
  vLine = TRUE,
  ...,
  title = NULL,
  subtitle = NULL,
  type = c("density", "boxplot", "violin"),
```

```
  invertAxes = FALSE,
  psi = NULL,
  rugLabels = FALSE,
  rugLabelsRotation = 0,
  legend = TRUE,
  valueLabel = NULL
)
```

Arguments

data	Numeric, data frame or matrix: gene expression data or alternative splicing event quantification values (sample names are based on their names or colnames)
groups	List of sample names or vector containing the group name per data value (read Details); if NULL or a character vector of length 1, data values are considered from the same group
rug	Boolean: show rug plot?
vLine	Boolean: plot vertical lines (including descriptive statistics for each group)?
...	Arguments passed on to stats::density.default

bw the smoothing bandwidth to be used. The kernels are scaled such that this is the standard deviation of the smoothing kernel. (Note this differs from the reference books cited below, and from S-PLUS.)
bw can also be a character string giving a rule to choose the bandwidth. See bw.nrd.
The default, "nrd0", has remained the default for historical and compatibility reasons, rather than as a general recommendation, where e.g., "SJ" would rather fit, see also Venables and Ripley (2002).
The specified (or computed) value of bw is multiplied by adjust.

adjust the bandwidth used is actually adjust*bw. This makes it easy to specify values like 'half the default' bandwidth.

kernel a character string giving the smoothing kernel to be used. This must partially match one of "gaussian", "rectangular", "triangular", "epanechnikov", "biweight", "cosine" or "optcosine", with default "gaussian", and may be abbreviated to a unique prefix (single letter).
"cosine" is smoother than "optcosine", which is the usual 'cosine' kernel in the literature and almost MSE-efficient. However, "cosine" is the version used by S.

window a character string giving the smoothing kernel to be used. This must partially match one of "gaussian", "rectangular", "triangular", "epanechnikov", "biweight", "cosine" or "optcosine", with default "gaussian", and may be abbreviated to a unique prefix (single letter).
"cosine" is smoother than "optcosine", which is the usual 'cosine' kernel in the literature and almost MSE-efficient. However, "cosine" is the version used by S.

weights numeric vector of non-negative observation weights, hence of same length as x. The default NULL is equivalent to weights = rep(1/nx,nx) where nx is the length of (the finite entries of) x[].

width this exists for compatibility with S; if given, and bw is not, will set bw to width if this is a character string, or to a kernel-dependent multiple of width if this is numeric.

give.Rkern logical; if true, *no* density is estimated, and the 'canonical bandwidth' of the chosen kernel is returned instead.

n the number of equally spaced points at which the density is to be estimated. When n > 512, it is rounded up to a power of 2 during the calculations (as fft is used) and the final result is interpolated by approx. So it almost always makes sense to specify n as a power of two.

from the left and right-most points of the grid at which the density is to be estimated; the defaults are cut * bw outside of range(x).

to the left and right-most points of the grid at which the density is to be estimated; the defaults are cut * bw outside of range(x).

cut by default, the values of from and to are cut bandwidths beyond the extremes of the data. This allows the estimated density to drop to approximately zero at the extremes.

title Character: plot title

subtitle Character: plot subtitle

type Character: density, boxplot or violin plot

invertAxes Boolean: plot X axis as Y and vice-versa?

psi Boolean: are data composed of PSI values? If NULL, psi = TRUE if all data values are between 0 and 1

rugLabels Boolean: plot sample names in the rug?

rugLabelsRotation
Numeric: rotation (in degrees) of rug labels; this may present issues at different zoom levels and depending on the proximity of data values

legend Boolean: show legend?

valueLabel Character: label for the value (by default, either Inclusion levels or Gene expression)

Details

Argument groups can be either:

- a list of sample names, e.g. list("Group 1"=c("Sample A","Sample B"),"Group 2"=c("Sample C")))
- a character vector with the same length as data, e.g. c("Sample A","Sample C","Sample B").

Value

highchart object with density plot

See Also

Other functions to perform and plot differential analyses: diffAnalyses()

Examples

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

# Using colours
attr(groups, "Colour") <- c("Group A"="pink", "Group B"="orange")
plotDistribution(data, groups)
```

`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

See Also

Other functions for gene expression pre-processing: `convertGeneIdentifiers()`, `filterGeneExpr()`, `normaliseGeneExpression()`, `plotLibrarySize()`, `plotRowStats()`

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()`

Other functions for data grouping: `createGroupByAttribute()`, `getGeneList()`, `getSampleFromSubject()`, `getSubjectFromSample()`, `groupPerElem()`, `testGroupIndependence()`

Examples

```
elements <- paste("subjects", 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

See Also

Other functions to analyse independent components: `performICA()`

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)
```

---

plotLibrarySize *Plot library size*

Description

Plot library size

Usage

```
plotLibrarySize(
  data,
  log10 = TRUE,
  title = "Library size distribution across samples",
  subtitle = "Library size: total number of mapped reads",
  colour = "orange"
)
```

Arguments

data	Data frame or matrix: gene expression
log10	Boolean: log10-transform data?
title	Character: plot title
subtitle	Character: plot subtitle
colour	Character: data colour

Value

Library size distribution

See Also

Other functions for gene expression pre-processing: convertGeneIdentifiers(), filterGeneExpr(), normaliseGeneExpression(), plotGeneExprPerSample(), plotRowStats()

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)
plotLibrarySize(df)
```

---

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

See Also

Other functions to analyse principal components: calculateLoadingsContribution(), performPCA(), plotPCAvariance()

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)

# Only plot loadings
plotPCA(pca, pcX=2, pcY=3, loadings=TRUE, individuals=FALSE)
```

---

plotPCAvariance *Create the explained variance plot from a PCA*

Description

Create the explained variance plot from a PCA

Usage

```
plotPCAvariance(pca)
```

Arguments

pca	prcomp object

Value

Plot variance as an highchart object

See Also

Other functions to analyse principal components: calculateLoadingsContribution(), performPCA(), plotPCA()

Examples

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

---

plotProtein *Plot protein features*

Description

Plot protein features

Usage

```
plotProtein(molecule)
```

Arguments

molecule	Character: UniProt protein or Ensembl transcript identifier

Value

highcharter object

See Also

Other functions to retrieve external information: ensemblToUniprot(), plotTranscripts(), queryEnsemblByGene()

Examples

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

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

plotRowStats *Plot row-wise statistics*

Description

Scatter plot to compare between the row-wise mean, median, variance or range from a data frame or matrix. Also supports transformations of those variables, such as `log10(mean)`. If `y = NULL`, a density plot is rendered instead.

Usage

```
plotRowStats(
  data,
  x,
  y = NULL,
  subset = NULL,
  xmin = NULL,
  xmax = NULL,
  ymin = NULL,
  ymax = NULL,
  xlim = NULL,
  ylim = NULL,
  cache = NULL,
  verbose = FALSE,
  data2 = NULL,
  legend = FALSE,
  legendLabels = c("Original", "Highlighted")
)
```

Arguments

`data`	Data frame or matrix containing samples per column and, for instance, gene or alternative splicing event per row
`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)`); if `y = NULL`, the density of `x` will be plot instead
`subset`	Boolean or integer: `data` points to highlight
`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
`cache`	List of summary statistics for `data` previously calculated to avoid repeating calculations (output also returns cache in attribute named `cache` with appropriate data)
`verbose`	Boolean: print messages of the steps performed

data2	Same as data argument but points in data2 are highlighted (unless data2 = NULL)
legend	Boolean: show legend?
legendLabels	Character: legend labels

Value

Plot of data

See Also

Other functions for gene expression pre-processing: convertGeneIdentifiers(), filterGeneExpr(), normaliseGeneExpression(), plotGeneExprPerSample(), plotLibrarySize()

Other functions for PSI quantification: filterPSI(), getSplicingEventTypes(), listSplicingAnnotations(), loadAnnotation(), quantifySplicing()

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
```

---

plotSplicingEvent *Plot diagram of alternative splicing events*

Description

Plot diagram of alternative splicing events

Usage

```
plotSplicingEvent(
  ASevent,
  data = NULL,
  showText = TRUE,
  showPath = TRUE,
  showAlternative1 = TRUE,
  showAlternative2 = TRUE,
  constitutiveWidth = NULL,
  alternativeWidth = NULL,
  intronWidth = NULL,
  constitutiveFill = "lightgray",
  constitutiveStroke = "darkgray",
  alternative1Fill = "#ffb153",
  alternative1Stroke = "#faa000",
  alternative2Fill = "#caa06c",
  alternative2Stroke = "#9d7039",
  class = NULL,
  style = NULL
)
```

Arguments

ASevent	Character: alternative splicing event identifiers
data	Matrix or data frame: alternative splicing information
showText	Boolean: display coordinates and length (if available)
showPath	Boolean: display alternative splicing junctions
showAlternative1	Boolean: show alternative exon 1 and respective splicing junctions and text?
showAlternative2	Boolean: show alternative exon 2 and respective splicing junctions and text? (only related with mutually exclusive exons)
constitutiveWidth	Numeric: width of constitutive exon(s)
alternativeWidth	Numeric: width of alternative exon(s)
intronWidth	Numeric: width of intron's representation
constitutiveFill	Character: fill colour of constitutive exons
constitutiveStroke	Character: stroke colour of constitutive exons
alternative1Fill	Character: fill colour of alternative exon 1
alternative1Stroke	Character: stroke colour of alternative exon 1

alternative2Fill
: Character: fill colour of alternative exon 2

alternative2Stroke
: Character: stroke colour of alternative exon 2

class
: Character: class of SVG parent tag

style
: Character: style of SVG parent tag

Value

List of SVG (one for each alternative splicing event)

Examples

```
events <- c(
  "A3SS_15_+_63353138_63353912_63353397_TPM1",
  "A3SS_11_-_61118463_61117115_61117894_CYB561A3",
  "A5SS_21_+_48055675_48056459_48056808_PRMT2",
  "A5SS_1_-_1274742_1274667_1274033_DVL1",
  "AFE_9_+_131902430_131901928_131904724_PPP2R4",
  "AFE_5_-_134686513_134688636_134681747_H2AFY",
  "ALE_12_+_56554104_56554410_56555171_MYL6",
  "ALE_8_-_38314874_38287466_38285953_FGFR1",
  "SE_9_+_6486925_6492303_6492401_6493826_UHRF2",
  "SE_19_-_5218431_5216778_5216731_5215606_PTPRS",
  "MXE_15_+_63335142_63335905_63336030_63336226_63336351_63349184_TPM1",
  "MXE_17_-_74090495_74087316_74087224_74086478_74086410_74085401_EXOC7")
diagram <- plotSplicingEvent(events)

## Not run:
diagram[["A3SS_3_-_145796903_145794682_145795711_PLOD2"]]
diagram[[6]]
diagram

## End(Not run)
```

---

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?
interval	Boolean: show interval ranges?
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)?

Value

Plot of survival curves

See Also

Other functions to analyse survival: assignValuePerSubject(), getAttributesTime(), labelBasedOnCutoff(), optimalSurvivalCutoff(), plotSurvivalPvaluesByCutoff(), processSurvTerms(), survdiffTerms(), survfit.survTerms(), testSurvival()

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

See Also

Other functions to analyse survival: assignValuePerSubject(), getAttributesTime(), labelBasedOnCutoff(), optimalSurvivalCutoff(), plotSurvivalCurves(), processSurvTerms(), survdiffTerms(), survfit.survTerms(), testSurvival()

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")
names(clinical) <- c("patient.days_to_last_followup",
                     "patient.days_to_death",
                     "patient.stage_event.pathologic_stage",
```

```
                    "patient.gender")
clinical <- do.call(rbind, rep(list(clinical), 5))
rownames(clinical) <- paste("Subject", seq(nrow(clinical)))

# 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 <- c("Cancer 1"="Subject 3",
           "Cancer 2"="Subject 17",
           "Cancer 3"="Subject 21")

eventData <- assignValuePerSubject(psi[3, ], match)

event      <- "days_to_death"
timeStart  <- "days_to_death"
plotSurvivalPvaluesByCutoff(clinical, eventData, censoring="right",
                            event=event, timeStart=timeStart)
```

`plotTranscripts` *Plot transcripts*

Description

Plot transcripts

Usage

```
plotTranscripts(
  info,
  eventPosition = NULL,
  event = NULL,
  eventData = 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
`eventData`	Object containing event information to be parsed
`shiny`	Boolean: is the function running in a Shiny session?

Value

NULL (function is only used to modify the Shiny session's state or internal variables)

See Also

Other functions to retrieve external information: ensemblToUniprot(), plotProtein(), queryEnsemblByGene()

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)
```

---

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.

See Also

Other functions to prepare alternative splicing annotations: `parseSuppaAnnotation()`

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)
```

---

prepareSRAmetadata *Prepare user-provided files to be loaded into psichomics*

Description

Prepare user-provided 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 of (optionally named) input files (see Examples)
`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 (if `output != NULL`) and object

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 subject
formulaStr	Character: formula to use
coxph	Boolean: fit a Cox proportional hazards regression model?
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

The event time is only used to determine whether the event has occurred (1) or not (0) in case of missing values.

If survTime = NULL, survival times are obtained from the clinical dataset according to the names given in timeStart, timeStop, event and followup. This may become quite slow when used in a loop. If the aforementioned variables are constant, consider running getAttributesTime() outside the loop and using its output via the survTime argument of this function (see Examples).

Value

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

See Also

Other functions to analyse survival: assignValuePerSubject(), getAttributesTime(), labelBasedOnCutoff(), optimalSurvivalCutoff(), plotSurvivalCurves(), plotSurvivalPvaluesByCutoff(), survdiffTerms(), survfit.survTerms(), testSurvival()

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)

# If running multiple times, consider calculating survTime only once
survTime <- getAttributesTime(clinical, event, timeStart)
for (i in seq(5)) {
  survTerms <- processSurvTerms(clinical, censoring="right", event,
                                timeStart, formulaStr=formulaStr,
                                survTime=survTime)
}
```

`psichomics` *Start graphical interface of psichomics*

Description

Start graphical interface of psichomics

Usage

```
psichomics(
  ...,
  launch.browser = TRUE,
  reset = FALSE,
  shinyproxy = FALSE,
  testData = FALSE,
  unparsableEvents = 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 between 3000:8000, excluding ports that are blocked by Google Chrome for being considered unsafe: 3659, 4045, 5060, 5061, 6000, 6566, 6665:6669 and 6697. Up to twenty random ports will be tried. `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.
`launch.browser`	If true, the system's default web browser will be launched automatically after the app is started. Defaults to true in interactive sessions only. This value of this parameter can also be a function to call with the application's URL.
`reset`	Boolean: reset Shiny session? Requires package `devtools`
`shinyproxy`	Boolean: prepare visual interface to run in Shinyproxy?
`testData`	Boolean: load with test data
`unparsableEvents`	Boolean: when testing data, load alternative splicing quantification events that cannot be parsed?

Value

`NULL` (function is only used to modify the Shiny session's state or internal variables)

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: values whose number of total supporting read counts is below minReads are returned as NA
genes	Character: gene symbols for which to quantify splicing events (if NULL, events from all genes are quantified)

Value

Data frame with the quantification of the alternative splicing events

See Also

Other functions for PSI quantification: filterPSI(), getSplicingEventTypes(), listSplicingAnnotations(), loadAnnotation(), plotRowStats()

Examples

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

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 = NULL, assembly = NULL, data = NULL)
```

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
data	Matrix or data frame: alternative splicing information

Value

Information from Ensembl

See Also

Other functions to retrieve external information: ensemblToUniprot(), plotProtein(), plotTranscripts()

Examples

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

readFile *Load psichomics-specific file*

Description

Load psichomics-specific file

Usage

```
readFile(file)
```

Arguments

file	Character: path to the file

Value

Loaded file

Examples

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

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 roundoff issues.

Value

`survfit` object. See `survfit.object` for details. Methods defined for `survfit` objects are `print`, `plot`, `lines`, and `points`.

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.

See Also

Other functions to analyse survival: `assignValuePerSubject()`, `getAttributesTime()`, `labelBasedOnCutoff()`, `optimalSurvivalCutoff()`, `plotSurvivalCurves()`, `plotSurvivalPvaluesByCutoff()`, `processSurvTerms()`, `survfit.survTerms()`, `testSurvival()`

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 roundoff 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

survfit object. See survfit.object for details. Methods defined for survfit objects are print, plot, lines, and points.

See Also

Other functions to analyse survival: assignValuePerSubject(), getAttributesTime(), labelBasedOnCutoff(), optimalSurvivalCutoff(), plotSurvivalCurves(), plotSurvivalPvaluesByCutoff(), processSurvTerms(), survdiffTerms(), testSurvival()

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)
```

---

t.sticky *Preserve attributes of* sticky *objects when extracting or transposing object*

Description

Most attributes - with the exception of names, dim, dimnames, class and row.names - are preserved in simple transformations of objects from class sticky

Usage

```
## S3 method for class 'sticky'
t(x)

## S3 method for class 'sticky'
x[i, j, ...]
```

Arguments

x	Object
i, j, ...	Numeric or character: indices of elements to extract

Value

Transformed object with most attributes preserved

---

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

Description

Test multiple contingency tables comprised by two groups (one reference group and another containing remaining 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	Contingency table used for testing
pvalue	Fisher's exact test's p-value

See Also

parseCategoricalGroups() and plotGroupIndependence()

Other functions for data grouping: createGroupByAttribute(), getGeneList(), getSampleFromSubject(), getSubjectFromSample(), groupPerElem(), plotGroupIndependence()

Examples

```
elements <- paste("subjects", 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 subjects*

Description

Test the survival difference between groups of subjects

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 roundoff issues.

Value

p-value of the survival difference or `NA`

Note

Instead of raising errors, returns `NA`

See Also

Other functions to analyse survival: `assignValuePerSubject()`, `getAttributesTime()`, `labelBasedOnCutoff()`, `optimalSurvivalCutoff()`, `plotSurvivalCurves()`, `plotSurvivalPvaluesByCutoff()`, `processSurvTerms()`, `survdiffTerms()`, `survfit.survTerms()`

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` *Display results of correlation analyses*

Description

Plot, print and display as table the results of gene expression and alternative splicing

Usage

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

## 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
)

## 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()
genes	Character: genes
ASevents	Character: AS events
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

See Also

Other functions to correlate gene expression and alternative splicing: correlateGEandAS()

Other functions to correlate gene expression and alternative splicing: correlateGEandAS()

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)
```

Index

∗ **functions associated with GTEx data retrieval**
getDownloadsFolder, 18
getGtexDataTypes, 19
getGtexTissues, 20
loadGtexData, 29
∗ **functions associated with SRA data retrieval**
getDownloadsFolder, 18
loadSRAproject, 31
∗ **functions associated with TCGA data retrieval**
getDownloadsFolder, 18
getTCGAdataTypes, 25
isFirebrowseUp, 26
loadTCGAdata, 32
parseTCGAsampleTypes, 40
∗ **functions for PSI quantification**
filterPSI, 16
getSplicingEventTypes, 23
listSplicingAnnotations, 28
loadAnnotation, 28
plotRowStats, 53
quantifySplicing, 66
∗ **functions for data grouping**
createGroupByAttribute, 10
getGeneList, 19
getSampleFromSubject, 21
getSubjectFromSample, 24
groupPerElem, 25
plotGroupIndependence, 47
testGroupIndependence, 72
∗ **functions for gene expression pre-processing**
convertGeneIdentifiers, 8
filterGeneExpr, 14
normaliseGeneExpression, 33
plotGeneExprPerSample, 46
plotLibrarySize, 49
plotRowStats, 53
∗ **functions to analyse independent components**
performICA, 41
plotICA, 48
∗ **functions to analyse principal components**
calculateLoadingsContribution, 6
performPCA, 42
plotPCA, 50
plotPCAvariance, 51
∗ **functions to analyse survival**
assignValuePerSubject, 5
getAttributesTime, 17
labelBasedOnCutoff, 27
optimalSurvivalCutoff, 35
plotSurvivalCurves, 56
plotSurvivalPvaluesByCutoff, 57
processSurvTerms, 62
survdiffTerms, 68
survfit.survTerms, 70
testSurvival, 73
∗ **functions to correlate gene expression and alternative splicing**
[.GEandAScorrelation, 74
correlateGEandAS, 9
∗ **functions to load data**
loadGtexData, 29
loadLocalFiles, 30
loadSRAproject, 31
loadTCGAdata, 32
∗ **functions to load local files**
loadLocalFiles, 30
∗ **functions to perform and plot differential analyses**
diffAnalyses, 11
plotDistribution, 43
∗ **functions to prepare alternative splicing annotations**
parseSuppaAnnotation, 38

prepareAnnotationFromEvents, 60
∗ **functions to retrieve external information**
ensemblToUniprot, 13
plotProtein, 52
plotTranscripts, 59
queryEnsemblByGene, 67
.onAttach, 4
[.GEandAScorrelation, *9*, 74
[.sticky *(*t.sticky*)*, 71

approx, *45*
as.table.GEandAScorrelation *(*[.GEandAScorrelation*)*, 74
assignValuePerPatient *(*assignValuePerSubject*)*, 5
assignValuePerSubject, 5, *18*, *27*, *36*, *57, 58*, *64*, *69*, *71*, *74*

bw.nrd, *44*

calcNormFactors, *33*
calculateLoadingsContribution, 6, *43*, *51, 52*
colSums,EList-method, 7
convertGeneIdentifiers, 8, *14*, *35*, *46*, *50*, *54*
cor.test, *9*
correlateGEandAS, 9, *75, 76*
cpm, *33*
createGroupByAttribute, 10, *19*, *21*, *24*, *26*, *47*, *73*

diffAnalyses, 11, *45*
discardLowCoveragePSIvalues, 12

EList-class, *7*
ensemblToUniprot, 13, *52*, *60*, *67*

fft, *45*
filterByExpr, *14*
filterGeneExpr, *8*, 14, *35*, *46*, *50*, *54*
filterGroups, 15
filterPSI, 16, *24*, *28, 29*, *54*, *66*

getAttributesTime, *6*, 17, *27*, *36*, *57, 58*, *63, 64*, *69*, *71*, *74*
getDownloadsFolder, 18, *20*, *25, 26*, *30*, *32, 33*, *41*
getFirebrowseCohorts *(*getTCGAdataTypes*)*, 25
getFirebrowseDataTypes *(*getTCGAdataTypes*)*, 25
getFirebrowseDates *(*getTCGAdataTypes*)*, 25
getGeneList, *10*, 19, *21*, *24*, *26*, *47*, *73*
getGenesFromSplicingEvents *(*getSplicingEventFromGenes*)*, 22
getGtexDataTypes, *18*, 19, *20*, *30*
getGtexReleases *(*getGtexDataTypes*)*, 19
getGtexTissues, *18*, *20*, 20, *30*
getMatchingSamples *(*getSampleFromSubject*)*, 21
getPatientFromSample *(*getSubjectFromSample*)*, 24
getSampleFromPatient *(*getSampleFromSubject*)*, 21
getSampleFromSubject, *10*, *19*, 21, *24*, *26*, *47*, *73*
getSplicingEventData, 22
getSplicingEventFromGenes, 22
getSplicingEventTypes, *17*, 23, *28, 29*, *54*, *66*
getSubjectFromSample, *10*, *19*, *21*, 24, *26*, *47*, *73*
getTCGAcohorts *(*getTCGAdataTypes*)*, 25
getTCGAdataTypes, *18*, 25, *26*, *32, 33*, *41*
getTCGAdates *(*getTCGAdataTypes*)*, 25
getValuePerPatient *(*assignValuePerSubject*)*, 5
getValuePerSubject *(*assignValuePerSubject*)*, 5
groupPerElem, *10*, *19*, *21*, *24*, 25, *47*, *73*

isFirebrowseUp, *18*, *25*, 26, *33*, *41*

labelBasedOnCutoff, *6*, *18*, 27, *36*, *57, 58*, *64*, *69*, *71*, *74*
listSplicingAnnotations, *17*, *24*, 28, *29*, *54*, *66*
loadAnnotation, *17*, *24*, *28*, 28, *54*, *66*
loadFirebrowseData *(*loadTCGAdata*)*, 32
loadGtexData, *18*, *20*, 29, *31–33*
loadLocalFiles, *30*, 30, *32, 33*
loadSRAproject, *18*, *30, 31*, 31, *33*
loadTCGAdata, *18*, *25, 26*, *30–32*, 32, *41*

model.frame, *41*, *43*

normaliseGeneExpression, *8*, *14*, 33, *46*, *50*, *54*

normalizeGeneExpression (normaliseGeneExpression), 33

optimalSurvivalCutoff, *6*, *18*, *27*, 35, *57*, *58*, *64*, *69*, *71*, *74*

pairsD3::pairsD3, *48*
parseCategoricalGroups, 36, *47*, *73*
parseMatsAnnotation (parseSuppaAnnotation), 38
parseMisoAnnotation (parseSuppaAnnotation), 38
parseSampleGroups (parseTCGAsampleTypes), 40
parseSplicingEvent, 37
parseSuppaAnnotation, 38, *61*
parseTCGAsampleInfo (parseTCGAsampleTypes), 40
parseTcgaSampleInfo (parseTCGAsampleTypes), 40
parseTCGAsampleTypes, *18*, *25*, *26*, *33*, 40
parseVastToolsAnnotation (parseSuppaAnnotation), 38
performICA, 41, *48*, *49*
performPCA, *7*, 42, *51*, *52*
plot.GEandAScorrelation ([.GEandAScorrelation), 74
plotCorrelation ([.GEandAScorrelation), 74
plotDistribution, *12*, 43
plotGeneExprPerSample, *8*, *14*, *35*, 46, *50*, *54*
plotGroupIndependence, *10*, *19*, *21*, *24*, *26*, *37*, 47, *73*
plotICA, *42*, 48
plotLibrarySize, *8*, *14*, *35*, *46*, 49, *54*
plotPCA, *7*, *43*, 50, *52*
plotPCAvariance, *7*, *43*, *51*, 51
plotProtein, *13*, 52, *60*, *67*
plotRowStats, *8*, *14*, *17*, *24*, *28*, *29*, *35*, *46*, *50*, 53, *66*
plotSplicingEvent, 54
plotSurvivalCurves, *6*, *18*, *27*, *36*, 56, *58*, *64*, *69*, *71*, *74*
plotSurvivalPvaluesByCutoff, *6*, *18*, *27*, *36*, *57*, 57, *64*, *69*, *71*, *74*
plotTranscripts, *13*, *52*, 59, *67*
plotVariance (plotPCAvariance), 51
prepareAnnotationFromEvents, *39*, 60
prepareGeneQuant (prepareSRAmetadata), 61
prepareJunctionQuant (prepareSRAmetadata), 61
prepareSRAmetadata, 61
print.GEandAScorrelation ([.GEandAScorrelation), 74
processSurvTerms, *6*, *18*, *27*, *36*, *57*, *58*, 62, *69*, *71*, *74*
psichomics, 64

quantifySplicing, *17*, *24*, *28*, *29*, *54*, 66
queryEnsemblByEvent (queryEnsemblByGene), 67
queryEnsemblByGene, *13*, *52*, *60*, 67
queryFirebrowseData, *32*

readFile, 68
recount_abstract, *31*
renderBoxplot, *46*

scale, *42*, *43*
shiny::runApp, *65*
stats::density.default, *44*
stats::loess.smooth, *76*
survdiffTerms, *6*, *18*, *27*, *36*, *57*, *58*, *64*, 68, *71*, *74*
survfit.survTerms, *6*, *18*, *27*, *36*, *57*, *58*, *64*, *69*, 70, *74*
survival::survdiff, *68*, *70*, *73*

t.sticky, 71
testGroupIndependence, *10*, *19*, *21*, *24*, *26*, *37*, *47*, 72
testSurvival, *6*, *18*, *27*, *36*, *57*, *58*, *64*, *69*, *71*, 73

voom, *33*, *34*