

# Package ‘IOBR’

May 7, 2026

**Title** Immune Oncology Biological Research

**Version** 2.2.0

**Description** Provides six modules for tumor microenvironment (TME) analysis based on multi-omics data. These modules cover data preprocessing, TME estimation, TME infiltrating patterns, cellular interactions, genome and TME interaction, and visualization for TME relevant features, as well as modelling based on key features. It integrates multiple microenvironmental analysis algorithms and signature estimation methods, simplifying the analysis and downstream visualization of the TME. In addition to providing a quick and easy way to construct gene signatures from single-cell RNA-seq data, it also provides a way to construct a reference matrix for TME deconvolution from single-cell RNA-seq data. The analysis pipeline and feature visualization are user-friendly and provide a comprehensive description of the complex TME, offering insights into tumour-immune interactions (Zeng D, et al. (2024) <[doi:10.1016/j.crmeth.2024.100910](https://doi.org/10.1016/j.crmeth.2024.100910)>. Fang Y, et al. (2025) <[doi:10.1002/mdr2.70001](https://doi.org/10.1002/mdr2.70001)>).

**License** GPL-3

**URL** <https://doi.org/10.3389/fimmu.2021.687975> (paper),  
<https://iobr.github.io/book/> (docs),  
<https://iobr.github.io/IOBR/>

**BugReports** <https://github.com/IOBR/IOBR/issues>

**Depends** R (>= 3.6.0)

**Imports** cli, dplyr, ggplot2, glmnet, GSVA, methods, purrr, rlang, stringr, survival, survminer, tibble, tidy

**Suggests** BiocParallel, biomaRt, circlize, clusterProfiler, ComplexHeatmap, corrplot, DESeq2, doParallel, DOSE, e1071, easier, enrichplot, factoextra, FactoMineR, foreach, ggdensity, ggpp, ggpubr, ggsci, gridExtra, Hmisc, knitr, limma, limSolve, maftools, MASS, Matrix, msigdb, NbClust, org.Hs.eg.db, org.Mm.eg.db, patchwork, PMCMRplus, pracma, preprocessCore,

prettydoc, pROC, psych, RColorBrewer, reshape2, rmarkdown,  
 ROCR, sampling, scales, Seurat, SeuratObject, sva, testthat (>=  
 3.0.0), tidyHeatmap, timeROC, webr, WGCNA

**VignetteBuilder** knitr

**biocViews** GeneExpression, DifferentialExpression, ImmunoOncology,  
 Transcriptomics, Clustering, Survival, Visualization

**Config/testthat/edition** 3

**Encoding** UTF-8

**LazyData** FALSE

**RoxygenNote** 7.3.3

**NeedsCompilation** no

**Author** Dongqiang Zeng [aut],  
 Yiran Fang [aut],  
 Shixiang Wang [aut, cre] (ORCID:  
<https://orcid.org/0000-0001-9855-7357>),  
 Qingcong Luo [aut],  
 Hongqian Qian [aut]

**Maintainer** Shixiang Wang <w\_shixiang@163.com>

**Repository** CRAN

**Date/Publication** 2026-04-22 14:00:18 UTC

## Contents

add_iobr_mirror . . . . .	5
add_riskscore . . . . .	6
anno_eset . . . . .	8
assimilate_data . . . . .	9
batch_cor . . . . .	10
batch_kruskal . . . . .	11
batch_pcc . . . . .	12
batch_sig_surv_plot . . . . .	13
batch_surv . . . . .	15
batch_wilcoxon . . . . .	16
best_cutoff . . . . .	18
best_cutoff2 . . . . .	19
BinomialAUC . . . . .	20
BinomialModel . . . . .	21
CalculatePref . . . . .	22
CalculateTimeROC . . . . .	23
calculate_break_month . . . . .	24
calculate_sig_score . . . . .	24
calculate_sig_score_integration . . . . .	26
calculate_sig_score_pca . . . . .	27
calculate_sig_score_ssgsea . . . . .	28

calculate_sig_score_zscore . . . . .	30
cell_bar_plot . . . . .	31
check_cancer_types . . . . .	32
check_eset . . . . .	33
CIBERSORT . . . . .	34
clear_iobr_cache . . . . .	35
combine_pd_eset . . . . .	36
Construct_con . . . . .	37
ConvertRownameToLoci . . . . .	38
CoreAlg . . . . .	38
count2tpm . . . . .	39
creat_folder . . . . .	41
deconvolute_quantiseq.default . . . . .	41
deconvolute_timer.default . . . . .	43
deconvo_cibersort . . . . .	44
deconvo_epic . . . . .	45
deconvo_estimate . . . . .	46
deconvo_ips . . . . .	46
deconvo_mcpcounter . . . . .	47
deconvo_quantiseq . . . . .	48
deconvo_ref . . . . .	49
deconvo_timer . . . . .	50
deconvo_tme . . . . .	51
deconvo_xcell . . . . .	53
design_mytheme . . . . .	54
doPerm . . . . .	55
download_iobr_data . . . . .	56
DrawQQPlot . . . . .	57
Enet . . . . .	58
enrichment_barplot . . . . .	58
EPIC . . . . .	60
eset_distribution . . . . .	62
estimateScore . . . . .	63
exact_pvalue . . . . .	64
extract_sc_data . . . . .	65
feature_manipulation . . . . .	66
feature_select . . . . .	67
filterCommonGenes . . . . .	68
find_markers_in_bulk . . . . .	69
find_mutations . . . . .	70
find_outlier_samples . . . . .	72
find_variable_genes . . . . .	73
format_msigdb . . . . .	74
format_signatures . . . . .	75
generateRef . . . . .	76
generateRef_DEseq2 . . . . .	77
generateRef_limma . . . . .	78
generateRef_rnaseq . . . . .	79

generateRef_seurat . . . . .	80
GetFractions.Abbas . . . . .	81
getHRandCIfromCoxph . . . . .	82
GetOutlierGenes . . . . .	82
get_cols . . . . .	83
get_cor . . . . .	84
get_cor_matrix . . . . .	86
get_sig_sc . . . . .	88
high_var_fea . . . . .	89
imvigor210_pdata . . . . .	90
iobr_cor_plot . . . . .	91
iobr_deconvo_pipeline . . . . .	94
iobr_deg . . . . .	95
iobr_pca . . . . .	97
ipsmap . . . . .	98
IPS_calculation . . . . .	99
lasso_select . . . . .	100
limma.dif . . . . .	101
list_github_datasets . . . . .	102
list_iobr_mirrors . . . . .	102
load_data . . . . .	103
log2eset . . . . .	104
LR_cal . . . . .	104
make_mut_matrix . . . . .	105
mapbw . . . . .	107
mapcolors . . . . .	108
MCPcounter.estimate . . . . .	108
merge_duplicate . . . . .	109
merge_eset . . . . .	110
mouse2human_eset . . . . .	111
null_models . . . . .	112
outputGCT . . . . .	113
output_sig . . . . .	114
palettes . . . . .	114
parallel_doperm . . . . .	115
ParseInputExpression . . . . .	117
patterns_to_na . . . . .	118
pdata_stad . . . . .	118
percent_bar_plot . . . . .	119
pie_chart . . . . .	121
PlotAUC . . . . .	122
plotPurity . . . . .	123
PlotTimeROC . . . . .	124
ProcessingData . . . . .	125
PrognosticAUC . . . . .	126
PrognosticModel . . . . .	127
PrognosticResult . . . . .	128
random_strata_cells . . . . .	129

rbind_iobr . . . . .	131
RegressionResult . . . . .	132
RemoveBatchEffect . . . . .	133
remove_batcheffect . . . . .	134
remove_duplicate_genes . . . . .	135
remove_names . . . . .	137
reset_iobr_mirrors . . . . .	138
roc_time . . . . .	138
scale_matrix . . . . .	140
select_method . . . . .	140
signature_collection . . . . .	141
signature_score_calculation_methods . . . . .	142
sigScore . . . . .	143
sig_box . . . . .	144
sig_box_batch . . . . .	145
sig_forest . . . . .	147
sig_group . . . . .	149
sig_gsea . . . . .	150
sig_heatmap . . . . .	153
sig_pheatmap . . . . .	155
sig_roc . . . . .	157
sig_surv_plot . . . . .	158
SplitTrainTest . . . . .	160
stad_group . . . . .	161
subgroup_data . . . . .	162
subgroup_survival . . . . .	162
surv_group . . . . .	163
tcga_rna_preps . . . . .	165
tcga_stad_pdata . . . . .	166
test_for_infiltration . . . . .	167
timer_available_cancers . . . . .	168
timer_info . . . . .	169
tme_cluster . . . . .	169
tme_deconvolution_methods . . . . .	171
Top_probe . . . . .	172
transform_data . . . . .	172

**Index****174**


---

add_iobr_mirror	<i>Add Custom Download Mirror</i>
-----------------	-----------------------------------

---

**Description**

Adds a custom mirror URL to the default mirrors for the current session. The mirror URL should be a base URL that will be prepended to GitHub paths.

**Usage**

```
add_iobr_mirror(url, position = c("first", "last", "before_github"))
```

**Arguments**

`url` Character string. The mirror URL to add.

`position` Character. Where to add the mirror: "first", "last", or "before\_github". Default: "first".

**Value**

Invisibly returns the updated mirror list.

**Examples**

```
# Add a custom mirror to try first
add_iobr_mirror("https://my-mirror.com/https://github.com")

# Add mirror to try before default GitHub
add_iobr_mirror("https://fast-mirror.org", position = "before_github")

# Download with the new mirror
data <- download_iobr_data("BRef")
```

---

`add_riskscore`*Add Risk Score to Dataset*

---

**Description**

Computes a risk score for each observation based on Cox proportional hazards regression or binary logistic regression. The function fits the specified model and returns the dataset with an added risk score column.

**Usage**

```
add_riskscore(
  input,
  family = c("cox", "binary"),
  target = NULL,
  time = NULL,
  status = NULL,
  vars,
  new_var_name = "riskscore"
)
```

**Arguments**

input	Data frame containing the variables for analysis.
family	Character string specifying the model family: "cox" for Cox proportional hazards regression or "binary" for logistic regression. Default is "cox".
target	Character string specifying the target variable name. Required when 'family = "binary"'.
time	Character string specifying the time-to-event variable name. Required when 'family = "cox"'.
status	Character string specifying the event status variable name. Required when 'family = "cox"'.
vars	Character vector of variable names to include in the model.
new_var_name	Character string specifying the name for the new risk score column. Default is "riskscore".

**Value**

Data frame identical to 'input' with an additional column containing risk scores (linear predictors for Cox models or predicted probabilities for logistic models).

**Author(s)**

Dongqiang Zeng

**Examples**

```
set.seed(123)
input_data <- data.frame(
  time = rexp(100),
  status = rbinom(100, 1, 0.5),
  age = rnorm(100, 60, 10),
  score1 = rnorm(100),
  score2 = rnorm(100)
)
result <- add_riskscore(
  input_data,
  time = "time", status = "status",
  vars = c("age", "score1", "score2")
)
head(result$riskscore)
```

---

`anno_eset`*Annotate Gene Expression Matrix and Remove Duplicated Genes*

---

## Description

Annotates an expression matrix with gene symbols using provided annotation data, filters out missing or invalid symbols, handles duplicate gene entries, and removes uninformative rows. The function supports multiple aggregation methods for resolving duplicate gene symbols.

## Usage

```
anno_eset(  
  eset,  
  annotation,  
  symbol = "symbol",  
  probe = "probe_id",  
  method = "mean"  
)
```

## Arguments

<code>eset</code>	Expression matrix or ExpressionSet object containing gene expression data.
<code>annotation</code>	Data frame containing annotation information for probes. Built-in options include ‘anno_hug133plus2’, ‘anno_rnaseq’, and ‘anno_illumina’.
<code>symbol</code>	Character string specifying the column name in ‘annotation’ that represents gene symbols. Default is “symbol”.
<code>probe</code>	Character string specifying the column name in ‘annotation’ that represents probe identifiers. Default is “probe_id”.
<code>method</code>	Character string specifying the aggregation method for duplicate gene symbols. Options are “mean”, “sum”, or “sd”. Default is “mean”.

## Details

The function performs the following operations:

1. Filters probes with missing symbols or labeled as “NA\_NA”
2. Matches probes between expression set and annotation data
3. Merges annotation with expression data
4. Handles duplicate gene symbols using specified aggregation method
5. Removes rows with all zeros, all NAs, or missing values in the first column

## Value

Annotated and cleaned gene expression matrix with gene symbols as row names.

**Author(s)**

Dongqiang Zeng

**Examples**

```
# Annotate Affymetrix microarray data
eset_gse62254 <- load_data("eset_gse62254")
anno_hug133plus2 <- load_data("anno_hug133plus2")
eset <- anno_eset(eset = eset_gse62254, annotation = anno_hug133plus2)
head(eset)

# Annotate RNA-seq data with Ensembl IDs
eset_stad <- load_data("eset_stad")
anno_grch38 <- load_data("anno_grch38")
eset <- anno_eset(eset = eset_stad, annotation = anno_grch38, probe = "id")
head(eset)
```

---

`assimilate_data`*Harmonize Two Data Frames by Column Structure*

---

**Description**

Adds missing columns (filled with 'NA') to a secondary data frame so that its column set and order match a reference data frame. This is useful when combining data frames from different sources that should have the same structure but may be missing some columns.

**Usage**

```
assimilate_data(data_a, data_b)
```

**Arguments**

`data_a`            Data frame. Reference data frame whose column structure should be matched.  
`data_b`            Data frame. Data frame to be conformed to 'data\_a'.

**Value**

Data frame 'data\_b' with added missing columns (NA-filled) and reordered to match 'data\_a'.

**Examples**

```
# Create reference data frame
pdata_a <- data.frame(
  A = 1:5, B = 2:6, C = 3:7, D = 4:8, E = 5:9
)

# Create data frame with subset of columns
```

```
pdata_b <- data.frame(A = 1:3, C = 4:6, E = 7:9)

# Harmonize pdata_b to match pdata_a structure
pdata_b_harmonized <- assimilate_data(data_a = pdata_a, data_b = pdata_b)
print(names(pdata_b_harmonized)) # Now has A, B, C, D, E
```

---

batch\_cor

*Batch Correlation Analysis*


---

## Description

Performs correlation analysis between a target variable and multiple feature variables. Computes correlation coefficients, p-values, and adjusts for multiple testing using the Benjamini-Hochberg method.

## Usage

```
batch_cor(data, target, feature, method = c("spearman", "pearson", "kendall"))
```

## Arguments

data	Data frame containing the target and feature variables.
target	Character string specifying the name of the target variable.
feature	Character vector specifying the names of feature variables to correlate with the target.
method	Character string specifying the correlation method. Options are "spearman", "pearson", or "kendall". Default is "spearman".

## Value

Tibble containing the following columns for each feature:

**sig\_names** Feature name

**p.value** Raw p-value

**statistic** Correlation coefficient

**p.adj** Adjusted p-value (Benjamini-Hochberg)

**log10pvalue** Negative log10-transformed p-value

**stars** Significance stars: \*\*\*\* p<0.0001, \*\*\* p<0.001, \*\* p<0.01, \* p<0.05, + p<0.5

## Author(s)

Dongqiang Zeng

**Examples**

```
# Load TCGA-STAD signature data
sig_stad <- load_data("sig_stad")

# Perform batch correlation
results <- batch_cor(
  data = sig_stad,
  target = "CD_8_T_effector",
  feature = colnames(sig_stad)[69:ncol(sig_stad)]
)
head(results)
```

---

batch_kruskal	<i>Batch Kruskal-Wallis Test</i>
---------------	----------------------------------

---

**Description**

Performs Kruskal-Wallis rank sum tests on multiple continuous features across different groups. Computes p-values, adjusts for multiple testing, and ranks features by significance.

**Usage**

```
batch_kruskal(data, group, feature = NULL, feature_manipulation = FALSE)
```

**Arguments**

<code>data</code>	Data frame containing the dataset for analysis.
<code>group</code>	Character string specifying the name of the grouping variable.
<code>feature</code>	Character vector specifying the names of feature variables to test. If 'NULL', the user is prompted to select features (interactive mode only). Default is 'NULL'.
<code>feature_manipulation</code>	Logical indicating whether to apply feature manipulation to filter valid features. Default is 'FALSE'.

**Value**

Tibble containing:

<b>sig_names</b>	Feature name
<b>p.value</b>	Raw p-value from Kruskal-Wallis test
<b>statistic</b>	Test statistic (chi-squared)
<b>p.adj</b>	Adjusted p-value (Benjamini-Hochberg)
<b>log10pvalue</b>	Negative log10-transformed p-value
<b>stars</b>	Significance stars: **** p<0.0001, *** p<0.001, ** p<0.01, * p<0.05, + p<0.5
<b>group columns</b>	Mean-centered values for each group

**Author(s)**

Dongqiang Zeng

**Examples**

```
# Load TCGA-STAD signature data
sig_stad <- load_data("sig_stad")

# Test features by gender (if available in your dataset)
if ("Gender" %in% colnames(sig_stad)) {
  res <- batch_kruskal(
    data = sig_stad,
    group = "Gender",
    feature = colnames(sig_stad)[69:ncol(sig_stad)]
  )
  head(res)
}
```

batch\_pcc

*Batch Calculation of Partial Correlation Coefficients***Description**

Computes partial correlation coefficients between multiple features and a target variable while controlling for an interference (confounding) variable. Adjusts p-values for multiple testing using the Benjamini-Hochberg method.

**Usage**

```
batch_pcc(
  input,
  interferenceid,
  target,
  features,
  method = c("pearson", "spearman", "kendall")
)
```

**Arguments**

input	Data frame containing feature variables, target variable, and interference variable.
interferenceid	Character string specifying the column name of the interference (confounding) variable to control for.
target	Character string specifying the column name of the target variable.
features	Character vector specifying the column names of feature variables to correlate with the target.
method	Character string specifying the correlation method. Options are "pearson", "spearman", or "kendall". Default is "pearson".

**Value**

Tibble containing the following columns for each feature:

**sig\_names** Feature name

**p.value** Raw p-value

**statistic** Partial correlation coefficient

**p.adj** Adjusted p-value (Benjamini-Hochberg method)

**log10pvalue** Negative log10-transformed p-value

**stars** Significance stars: \*\*\*\* p.adj<0.0001, \*\*\* p.adj<0.001, \*\* p.adj<0.01, \* p.adj<0.05, + p.adj<0.5

**Author(s)**

Rongfang Shen

**Examples**

```
# Load TCGA-STAD signature data
sig_stad <- load_data("sig_stad")

# Calculate partial correlations controlling for tumor purity
res <- batch_pcc(
  input = sig_stad,
  interferenceid = "TumorPurity_estimate",
  target = "Pan_F_TBRs",
  method = "pearson",
  features = colnames(sig_stad)[70:ncol(sig_stad)]
)
head(res)
```

---

batch\_sig\_surv\_plot     *Batch Signature Survival Plot*

---

**Description**

Generates Kaplan-Meier survival plots for multiple projects or cohorts based on signature scores. Automatically determines optimal cutpoints for signature stratification and creates publication-ready survival curves.

**Usage**

```
batch_sig_surv_plot(
  input_pdata,
  signature,
  id = "ID",
  column_of_project = "ProjectID",
  project = NULL,
```

```

time = "time",
status = "status",
time_type = "day",
break_month = "auto",
palette = "jama",
cols = NULL,
mini_sig = "score",
save_path = NULL,
show_col = TRUE,
fig_type = "pdf"
)

```

### Arguments

input_pdata	Data frame containing survival data and signature scores.
signature	Character string specifying the column name of the target signature for survival analysis.
id	Character string specifying the column name containing unique identifiers. Default is "ID".
column_of_project	Character string specifying the column name containing project identifiers. Default is "ProjectID".
project	Character string or vector specifying project name(s) to analyze. If 'NULL', all projects are analyzed. Default is 'NULL'.
time	Character string specifying the column name containing time-to-event data. Default is "time".
status	Character string specifying the column name containing event status. Default is "status".
time_type	Character string specifying the time unit. Options are "day" or "month". Default is "day".
break_month	Numeric value or "auto" specifying the interval for time axis breaks in months. Default is "auto".
palette	Character string specifying the color palette. Default is "jama".
cols	Character vector of custom colors. If 'NULL', palette is used. Default is 'NULL'.
mini_sig	Character string for the signature label in the legend. Default is "score".
save_path	Character string or 'NULL'. Directory path for saving plots. If 'NULL', plots are not saved. Default is 'NULL'.
show_col	Logical indicating whether to display color information. Default is 'TRUE'.
fig_type	Character string specifying the output file format ("pdf", "png", etc.). Default is "pdf".

### Value

Data frame containing combined survival analysis results from all projects.

**Author(s)**

Dongqiang Zeng

**Examples**

```
sig_stad <- load_data("sig_stad")
result <- batch_sig_surv_plot(
  input_pdata = sig_stad,
  signature = "T.cells.CD8",
  id = "ID",
  column_of_project = "ProjectID",
  project = NULL,
  time = "OS_time",
  status = "OS_status",
  time_type = "month",
  break_month = "auto",
  palette = "jama",
  cols = NULL,
  mini_sig = "score",
  show_col = TRUE,
  fig_type = "pdf"
)
```

---

`batch_surv`*Batch Survival Analysis*

---

**Description**

Performs Cox proportional hazards regression analysis on multiple variables. Optionally determines optimal cutoffs to dichotomize continuous predictors before modeling. Returns hazard ratios, confidence intervals, and p-values for each variable.

**Usage**

```
batch_surv(
  pdata,
  variable,
  time = "time",
  status = "status",
  best_cutoff = FALSE
)
```

**Arguments**

<code>pdata</code>	Data frame containing survival time, event status, and predictor variables.
<code>variable</code>	Character vector specifying the names of predictor variables to analyze.

time	Character string specifying the column name containing follow-up time. Default is "time".
status	Character string specifying the column name containing event status (1 = event occurred, 0 = censored). Default is "status".
best_cutoff	Logical indicating whether to compute optimal cutoffs for continuous variables and analyze dichotomized versions. Default is 'FALSE'.

### Value

Data frame containing hazard ratios (HR), 95 and p-values for each variable, sorted by p-value.

### Author(s)

Dongqiang Zeng

### Examples

```
sig_stad <- load_data("sig_stad")
batch_surv(
  pdata = sig_stad,
  variable = colnames(sig_stad)[69:ncol(sig_stad)],
  time = "OS_time",
  status = "OS_status"
)
```

---

batch\_wilcoxon

*Batch Wilcoxon Rank-Sum Test Between Two Groups*

---

### Description

Performs Wilcoxon rank-sum tests (Mann-Whitney U tests) to compare the distribution of specified features between two groups. Computes p-values, adjusts for multiple testing, and ranks features by significance.

### Usage

```
batch_wilcoxon(
  data,
  target = "group",
  feature = NULL,
  feature_manipulation = FALSE
)
```

**Arguments**

<code>data</code>	Data frame containing the dataset for analysis.
<code>target</code>	Character string specifying the column name of the grouping variable. Default is <code>"group"</code> .
<code>feature</code>	Character vector specifying feature names to analyze. If <code>'NULL'</code> , prompts for selection (interactive mode only). Default is <code>'NULL'</code> .
<code>feature_manipulation</code>	Logical indicating whether to apply feature manipulation filtering. Default is <code>'FALSE'</code> .

**Value**

Tibble with columns:

**sig\_name** Feature name

**p.value** Raw p-value

**statistic** Difference in means between groups

**p.adj** Adjusted p-value (Benjamini-Hochberg)

**log10pvalue** Negative log10-transformed p-value

**stars** Significance stars: \*\*\*\* p<0.0001, \*\*\* p<0.001, \*\* p<0.01, \* p<0.05, + p<0.5

**group1, group2** Mean values for each group

**Author(s)**

Dongqiang Zeng

**Examples**

```
# Load TCGA-STAD signature data
sig_stad <- load_data("sig_stad")

# Compare features by gender
res <- batch_wilcoxon(
  data = sig_stad,
  target = "Gender",
  feature = colnames(sig_stad)[69:ncol(sig_stad)]
)
head(res)
```

---

`best_cutoff`*Extract Best Cutoff and Add Binary Variable to Data Frame*

---

### Description

Determines the optimal cutoff point for a continuous variable in survival analysis using the maximally selected rank statistics method. Creates a binary variable based on the identified cutoff and adds it to the input data frame.

### Usage

```
best_cutoff(  
  pdata,  
  variable,  
  time = "time",  
  status = "status",  
  print_result = TRUE  
)
```

### Arguments

<code>pdata</code>	Data frame containing survival information and the continuous variable.
<code>variable</code>	Character string specifying the name of the continuous variable for which the optimal cutoff should be determined.
<code>time</code>	Character string specifying the column name containing time-to-event data. Default is "time".
<code>status</code>	Character string specifying the column name containing event status (censoring information). Default is "status".
<code>print_result</code>	Logical indicating whether to print detailed results including cutoff value and Cox model summaries. Default is 'TRUE'.

### Value

Data frame identical to 'pdata' with an additional binary column named '<variable>\_binary' containing "High" and "Low" categories based on the optimal cutoff.

### Author(s)

Dongqiang Zeng

### Examples

```
set.seed(123)  
pdata <- data.frame(  
  time = rexp(100),  
  status = rbinom(100, 1, 0.5),  
  score = rnorm(100, mean = 50, sd = 10)
```

```

)
result <- best_cutoff(pdata, variable = "score", print_result = FALSE)
table(result$score_binary)

```

---

best\_cutoff2

*Extract Best Cutoff and Add Binary Variable to Data Frame*


---

## Description

Finds the best cutoff point for a continuous variable in survival analysis. Takes input data containing a continuous variable and survival information (time and status). Returns the modified input data with a new binary variable created based on the best cutoff point.

## Usage

```

best_cutoff2(
  pdata,
  variable,
  time = "time",
  status = "status",
  print_result = TRUE
)

```

## Arguments

<code>pdata</code>	Data frame containing survival information and features.
<code>variable</code>	Character string specifying the continuous variable name.
<code>time</code>	Character string specifying the time-to-event column name. Default is "time".
<code>status</code>	Character string specifying the event status column name. Default is "status".
<code>print_result</code>	Logical indicating whether to print results. Default is 'TRUE'.

## Value

List containing:

**pdata** Data frame with binary variable added

**best\_cutoff** Numeric cutoff value

**cox\_continuous\_object** Cox model summary for continuous variable

**summary\_binary\_variable** Summary of binary variable

**cox\_binary\_object** Cox model summary for binary variable

## Author(s)

Dongqiang Zeng

**Examples**

```

set.seed(123)
pdata <- data.frame(
  time = rexp(100),
  status = rbinom(100, 1, 0.5),
  score = rnorm(100, mean = 50, sd = 10)
)
result <- best_cutoff2(pdata, variable = "score", print_result = FALSE)
result$best_cutoff

```

---

BinomialAUC

*Calculate AUC for Binomial Model*


---

**Description**

Computes Area Under the ROC Curve (AUC) for model predictions using the ROCR package. Handles binary classification models from glmnet.

**Usage**

```
BinomialAUC(model, newx, s, acture.y)
```

**Arguments**

model	Fitted glmnet model object.
newx	New data matrix for prediction.
s	Lambda value for prediction (e.g., "lambda.min" or numeric).
acture.y	Actual binary outcomes (numeric 0/1 or factor).

**Value**

Numeric AUC value between 0 and 1.

**Examples**

```

if (requireNamespace("glmnet", quietly = TRUE) &&
    requireNamespace("ROCR", quietly = TRUE)) {
  set.seed(123)
  train_data <- matrix(rnorm(100 * 5), ncol = 5)
  train_outcome <- rbinom(100, 1, 0.5)
  test_data <- matrix(rnorm(50 * 5), ncol = 5)
  test_outcome <- rbinom(50, 1, 0.5)
  fitted_model <- glmnet::cv.glmnet(train_data, train_outcome, family = "binomial", nfolds = 5)
  auc_value <- BinomialAUC(fitted_model, test_data, fitted_model$lambda.min, test_outcome)
  print(auc_value)
}

```

---

 BinomialModel

*Binomial Model Construction*


---

### Description

Constructs and evaluates binomial logistic regression models using Lasso and Ridge regularization. Processes input data, scales features if specified, splits data into training/testing sets, and fits both Lasso and Ridge models. Optionally generates AUC plots for model evaluation.

### Usage

```
BinomialModel(
  x,
  y,
  seed = 123456,
  scale = TRUE,
  train_ratio = 0.7,
  nfold = 10,
  plot = TRUE,
  palette = "jama",
  cols = NULL
)
```

### Arguments

x	A data frame containing sample ID and features. First column must be sample ID.
y	A data frame where first column is sample ID and second column is outcome (numeric or factor).
seed	Integer for random seed. Default is '123456'.
scale	Logical indicating whether to scale features. Default is 'TRUE'.
train_ratio	Numeric between 0 and 1 for training proportion. Default is '0.7'.
nfold	Integer for cross-validation folds. Default is '10'.
plot	Logical indicating whether to generate AUC plots. Default is 'TRUE'.
palette	Character string for color palette. Default is "jama".
cols	Optional color vector for ROC curves. Default is 'NULL'.

### Value

List containing:

**lasso\_result** Lasso model results

**ridge\_result** Ridge model results

**train.x** Training data with IDs

**Author(s)**

Dongqiang Zeng

**Examples**

```
set.seed(123)
x <- data.frame(
  ID = paste0("Sample", 1:50),
  Feature1 = rnorm(50),
  Feature2 = rnorm(50),
  Feature3 = rnorm(50)
)
y <- data.frame(
  ID = x$ID,
  Outcome = factor(rbinom(50, 1, 0.5))
)
result <- BinomialModel(x = x, y = y, plot = FALSE, nfold = 5)
str(result, max.level = 1)
```

---

**CalculatePref***Calculate Performance Metrics*

---

**Description**

Computes True Positive Rate (TPR) and False Positive Rate (FPR) for ROC analysis using the ROCR package. Used internally for ROC curve generation.

**Usage**

```
CalculatePref(model, newx, s, acture.y)
```

**Arguments**

model	Fitted glmnet model.
newx	New data matrix for prediction.
s	Lambda value for prediction.
acture.y	Actual binary outcomes.

**Value**

ROCR performance object containing TPR and FPR values.

**Examples**

```
if (requireNamespace("glmnet", quietly = TRUE) &&
    requireNamespace("ROCR", quietly = TRUE)) {
  fitted_model <- glmnet::cv.glmnet(matrix(rnorm(100), ncol = 2), rbinom(50, 1, 0.5), nfolds = 3)
  perf <- CalculatePref(fitted_model, matrix(rnorm(20), ncol = 2), "lambda.min", rbinom(10, 1, 0.5))
}
```

---

CalculateTimeROC	<i>Calculate Time-Dependent ROC Curve</i>
------------------	---

---

**Description**

Computes time-dependent ROC curve for survival models using the ‘timeROC’ package. Evaluates predictive accuracy at a specified time quantile.

**Usage**

```
CalculateTimeROC(model, newx, s, acture.y, modelname, time_prob = 0.9)
```

**Arguments**

model	A fitted survival model object.
newx	A matrix or data frame of new data for prediction.
s	Lambda value for prediction.
acture.y	Data frame with ‘time’ and ‘status’ columns.
modelname	Character string for model identification.
time_prob	Numeric quantile for ROC calculation. Default is ‘0.9’.

**Value**

An object of class ‘timeROC’ containing ROC curve information.

**Author(s)**

Dongqiang Zeng

**Examples**

```
if (requireNamespace("glmnet", quietly = TRUE) &&
    requireNamespace("survival", quietly = TRUE) &&
    requireNamespace("timeROC", quietly = TRUE)) {
  library(survival)
  dat <- na.omit(lung[, c("time", "status", "age", "sex", "ph.ecog")])
  dat$status <- dat$status - 1
  x <- as.matrix(dat[, c("age", "sex", "ph.ecog")])
  y <- Surv(dat$time, dat$status)
  fit <- glmnet::glmnet(x, y, family = "cox")
  actual_outcome <- data.frame(time = dat$time, status = dat$status)
  roc_info <- CalculateTimeROC(
    model = fit, newx = x, s = 0.01, acture.y = actual_outcome,
    modelname = "glmnet Cox Model", time_prob = 0.5
  )
  print(roc_info$AUC)
}
```

---

calculate\_break\_month *Break Time Into Blocks*

---

**Description**

Divides time duration into specified blocks for analysis.

**Usage**

```
calculate_break_month(input, block = 6, time_type = c("month", "day"))
```

**Arguments**

input	Numeric vector of time durations.
block	Number of blocks. Default is 6.
time_type	Units: "month" or "day". Default is "month".

**Value**

Numeric vector of breakpoints, rounded to nearest multiple of 5.

**Author(s)**

Dongqiang Zeng

**Examples**

```
time_data <- c(24, 36, 12, 48)
blocks <- calculate_break_month(input = time_data)
```

---

calculate\_sig\_score *Calculate Signature Score*

---

**Description**

Main interface for calculating signature scores from gene expression data. Supports PCA, z-score, ssGSEA, and integration methods.

**Usage**

```

calculate_sig_score(
  pdata = NULL,
  eset,
  signature = NULL,
  method = c("pca", "ssgsea", "zscore", "integration"),
  mini_gene_count = 3,
  column_of_sample = "ID",
  print_gene_proportion = FALSE,
  print_filtered_signatures = FALSE,
  adjust_eset = FALSE,
  parallel.size = 1L,
  ...
)

```

**Arguments**

<code>pdata</code>	Phenotype data. If 'NULL', created from 'eset' column names.
<code>eset</code>	Expression matrix (CPM, TPM, RPKM, FPKM, etc.).
<code>signature</code>	List of gene signatures. Can also be a character string naming a built-in signature collection (e.g., "signature_collection", "signature_tme", "go_bp", "kegg", "hallmark").
<code>method</code>	Scoring method: "pca", "ssgsea", "zscore", or "integration". Default is "pca".
<code>mini_gene_count</code>	Minimum genes required per signature. Default is 3 (or 5 for ssGSEA).
<code>column_of_sample</code>	Column with sample IDs in 'pdata'. Default is "ID".
<code>print_gene_proportion</code>	Logical: print gene coverage. Default is 'FALSE'.
<code>print_filtered_signatures</code>	Logical: print filtered signatures. Default is 'FALSE'.
<code>adjust_eset</code>	Logical: clean problematic features. Default is 'FALSE'.
<code>parallel.size</code>	Parallel workers for ssGSEA. Default is 1.
<code>...</code>	Additional arguments passed to specific methods.

**Value**

Tibble with phenotype data and signature scores.

**Author(s)**

Dongqiang Zeng

## References

1. Hänzelmann S, Castelo R, Guinney J. GSVA: gene set variation analysis. *BMC Bioinformatics*. 2013;14:7.
2. Mariathasan S, et al. TGF $\beta$  attenuates tumour response to PD-L1 blockade. *Nature*. 2018;554:544-548.

## Examples

```
set.seed(123)
eset <- matrix(rnorm(1000), nrow = 100, ncol = 10)
rownames(eset) <- paste0("Gene", 1:100)
colnames(eset) <- paste0("Sample", 1:10)
signature <- list(
  Signature1 = paste0("Gene", 1:10),
  Signature2 = paste0("Gene", 11:20)
)

result <- calculate_sig_score(eset = eset, signature = signature, method = "pca")
```

---

calculate\_sig\_score\_integration

*Calculate Signature Score Using Integration Method*

---

## Description

Computes signature scores using PCA, z-score, and ssGSEA methods combined.

## Usage

```
calculate_sig_score_integration(
  pdata = NULL,
  eset,
  signature,
  mini_gene_count = 2,
  column_of_sample = "ID",
  adjust_eset = FALSE,
  parallel.size = 1L
)
```

## Arguments

pdata	Data frame with phenotype data. If 'NULL', created from 'eset' column names.
eset	Expression matrix (genes as rows, samples as columns).
signature	List of gene signatures.
mini_gene_count	Minimum genes required per signature. Default is 3.

column\_of\_sample      Column in 'pdata' with sample IDs. Default is "ID".  
adjust\_eset      Logical: remove problematic features. Default is 'FALSE'.  
parallel.size      Number of parallel workers. Default is 1.

**Value**

Tibble with signature scores from all three methods.

**Author(s)**

Dongqiang Zeng

**Examples**

```
set.seed(123)
eset <- matrix(rnorm(1000), nrow = 100, ncol = 10)
rownames(eset) <- paste0("Gene", 1:100)
colnames(eset) <- paste0("Sample", 1:10)
signature <- list(
  Signature1 = paste0("Gene", 1:15),
  Signature2 = paste0("Gene", 16:30)
)

result <- calculate_sig_score_integration(eset = eset, signature = signature)
```

---

calculate\_sig\_score\_pca

*Calculate Signature Score Using PCA Method*

---

**Description**

Computes signature scores using Principal Component Analysis. The first principal component is used as the signature score.

**Usage**

```
calculate_sig_score_pca(
  pdata = NULL,
  eset,
  signature,
  mini_gene_count = 3,
  column_of_sample = "ID",
  adjust_eset = FALSE
)
```

**Arguments**

pdata	Data frame with phenotype data. If 'NULL', created from 'eset' column names.
eset	Expression matrix (genes as rows, samples as columns).
signature	List of gene signatures.
mini_gene_count	Minimum genes required per signature. Default is 3.
column_of_sample	Column in 'pdata' with sample IDs. Default is "ID".
adjust_eset	Logical: remove problematic features. Default is 'FALSE'.

**Value**

Tibble with signature scores.

**Author(s)**

Dongqiang Zeng

**Examples**

```
set.seed(123)
eset <- matrix(rnorm(1000), nrow = 100, ncol = 10)
rownames(eset) <- paste0("Gene", 1:100)
colnames(eset) <- paste0("Sample", 1:10)
signature <- list(
  Signature1 = paste0("Gene", 1:10),
  Signature2 = paste0("Gene", 11:20)
)

result <- calculate_sig_score_pca(eset = eset, signature = signature)
```

---

calculate\_sig\_score\_ssgsea

*Calculate Signature Score Using ssGSEA Method*

---

**Description**

Computes signature scores using single-sample Gene Set Enrichment Analysis.

**Usage**

```
calculate_sig_score_ssgsea(  
  pdata = NULL,  
  eset,  
  signature,  
  mini_gene_count = 5,  
  column_of_sample = "ID",  
  adjust_eset = FALSE,  
  parallel.size = 1L  
)
```

**Arguments**

<code>pdata</code>	Data frame with phenotype data. If 'NULL', created from 'eset' column names.
<code>eset</code>	Expression matrix (genes as rows, samples as columns).
<code>signature</code>	List of gene signatures.
<code>mini_gene_count</code>	Minimum genes required per signature. Default is 3.
<code>column_of_sample</code>	Column in 'pdata' with sample IDs. Default is "ID".
<code>adjust_eset</code>	Logical: remove problematic features. Default is 'FALSE'.
<code>parallel.size</code>	Number of parallel workers. Default is 1.

**Value**

Tibble with signature scores.

**Author(s)**

Dongqiang Zeng

**Examples**

```
set.seed(123)  
eset <- matrix(rnorm(1000), nrow = 100, ncol = 10)  
rownames(eset) <- paste0("Gene", 1:100)  
colnames(eset) <- paste0("Sample", 1:10)  
signature <- list(  
  Signature1 = paste0("Gene", 1:15),  
  Signature2 = paste0("Gene", 16:30)  
)  
  
result <- calculate_sig_score_ssgsea(eset = eset, signature = signature)
```

---

`calculate_sig_score_zscore`*Calculate Signature Score Using Z-Score Method*

---

**Description**

Computes signature scores using z-score transformation.

**Usage**

```
calculate_sig_score_zscore(  
  pdata = NULL,  
  eset,  
  signature,  
  mini_gene_count = 3,  
  column_of_sample = "ID",  
  adjust_eset = FALSE  
)
```

**Arguments**

<code>pdata</code>	Data frame with phenotype data. If 'NULL', created from 'eset' column names.
<code>eset</code>	Expression matrix (genes as rows, samples as columns).
<code>signature</code>	List of gene signatures.
<code>mini_gene_count</code>	Minimum genes required per signature. Default is 3.
<code>column_of_sample</code>	Column in 'pdata' with sample IDs. Default is "ID".
<code>adjust_eset</code>	Logical: remove problematic features. Default is 'FALSE'.

**Value**

Tibble with signature scores.

**Author(s)**

Dongqiang Zeng

**Examples**

```
set.seed(123)  
eset <- matrix(rnorm(1000), nrow = 100, ncol = 10)  
rownames(eset) <- paste0("Gene", 1:100)  
colnames(eset) <- paste0("Sample", 1:10)  
signature <- list(  
  Signature1 = paste0("Gene", 1:10),  
  Signature2 = paste0("Gene", 11:20)
```

```
)
result <- calculate_sig_score_zscore(eset = eset, signature = signature)
```

---

cell\_bar\_plot

*Visualize Cell Fractions as Stacked Bar Chart*


---

### Description

Creates stacked bar charts to visualize tumor microenvironment (TME) cell fractions. Supports batch visualization of deconvolution results from methods such as CIBERSORT, EPIC, and quanTIseq.

### Usage

```
cell_bar_plot(
  input,
  id = "ID",
  title = "Cell Fraction",
  features = NULL,
  pattern = NULL,
  legend.position = "bottom",
  coord_flip = TRUE,
  palette = 3,
  show_col = FALSE,
  cols = NULL
)
```

### Arguments

input	Data frame containing deconvolution results.
id	Character string specifying the column name containing sample identifiers. Default is "ID".
title	Character string specifying the plot title. Default is "Cell Fraction".
features	Character vector specifying column names representing cell types to plot. If NULL, columns are selected based on 'pattern'. Default is NULL.
pattern	Character string or regular expression to match column names for automatic feature selection. Used when 'features' is NULL. Default is NULL.
legend.position	Character string specifying legend position ("bottom", "top", "left", "right"). Default is "bottom".
coord_flip	Logical indicating whether to flip plot coordinates using 'coord_flip()'. Default is TRUE.
palette	Integer specifying the color palette to use. Default is 3.
show_col	Logical indicating whether to display color information. Default is FALSE.
cols	Character vector of custom colors. If NULL, palette is used. Default is NULL.

**Value**

A ggplot2 object representing the stacked bar chart.

**Author(s)**

Dongqiang Zeng

**Examples**

```
set.seed(123)
input_data <- data.frame(
  ID = paste0("Sample", 1:10),
  Cell_A = runif(10, 0, 0.4),
  Cell_B = runif(10, 0, 0.3),
  Cell_C = runif(10, 0, 0.3)
)
cell_bar_plot(input = input_data, id = "ID", features = c("Cell_A", "Cell_B", "Cell_C"))
```

---

check\_cancer\_types      *Process Batch Table and Validate Cancer Types*

---

**Description**

Processes input data containing cancer types and validates each category against a predefined list of supported cancer types ('timer\_available\_cancers').

**Usage**

```
check_cancer_types(args)
```

**Arguments**

**args**                    A list containing input parameters:  
**batch**                    Character. Path to a CSV file (optional).  
**expression**                Character vector of expression identifiers (used if 'batch' is NULL).  
**category**                  Character vector of cancer types (used if 'batch' is NULL).

**Value**

A character matrix with two columns:

**Column 1** Expression identifiers

**Column 2** Cancer categories

## Examples

```
args <- list(
  expression = c("exp1", "exp2"),
  category = c("luad", "brca"),
  batch = NULL
)
result <- check_cancer_types(args)
```

---

check\_eset

*Check Integrity and Outliers of Expression Set*

---

## Description

Performs quality checks on an expression matrix to identify missing values, infinite values, and features with zero variance. Issues warnings when potential problems are detected that may affect downstream analyses.

## Usage

```
check_eset(eset, print_result = FALSE, estimate_sd = FALSE)
```

## Arguments

eset	Expression matrix or data frame with genes/features in rows and samples in columns.
print_result	Logical indicating whether to print detailed check results to the console. Default is 'FALSE'.
estimate_sd	Logical indicating whether to check for features with zero standard deviation. Default is 'FALSE'.

## Value

Invisibly returns 'NULL'. Function is called for its side effects (printing messages and issuing warnings).

## Author(s)

Dongqiang Zeng

## Examples

```
set.seed(123)
eset <- matrix(rnorm(1000), nrow = 100, ncol = 10)
rownames(eset) <- paste0("Gene", 1:100)
colnames(eset) <- paste0("Sample", 1:10)
check_eset(eset, print_result = TRUE)
```

**Description**

An analytical tool to estimate cell type abundances in mixed cell populations using gene expression data.

**Usage**

```
CIBERSORT(
  sig_matrix = NULL,
  mixture_file,
  perm,
  QN = TRUE,
  absolute = FALSE,
  abs_method = "sig.score",
  parallel = FALSE,
  num_cores = 2,
  seed = NULL
)
```

**Arguments**

<code>sig_matrix</code>	Cell type GEP barcode matrix: row 1 = sample labels; column 1 = gene symbols; no missing values; default = LM22.txt download from CIBERSORT ( <a href="https://cibersortx.stanford.edu/runci">https://cibersortx.stanford.edu/runci</a> )
<code>mixture_file</code>	GEP matrix: row 1 = sample labels; column 1 = gene symbols; no missing values
<code>perm</code>	Set permutations for statistical analysis ( $\geq 100$ permutations recommended).
<code>QN</code>	Quantile normalization of input mixture (default = TRUE)
<code>absolute</code>	Run CIBERSORT in absolute mode (default = FALSE) - note that cell subsets will be scaled by their absolute levels and will not be represented as fractions (to derive the default output, normalize absolute levels such that they sum to 1 for each mixture sample) - the sum of all cell subsets in each mixture sample will be added to the output ('Absolute score'). If LM22 is used, this score will capture total immune content.
<code>abs_method</code>	If absolute is set to TRUE, choose method: 'no.sumto1' or 'sig.score' - sig.score = for each mixture sample, define S as the median expression level of all genes in the signature matrix divided by the median expression level of all genes in the mixture. Multiple cell subset fractions by S. - no.sumto1 = remove sum to 1 constraint
<code>parallel</code>	Logical. Enable parallel execution? (default = FALSE)
<code>num_cores</code>	Integer. Number of cores to use when parallel = TRUE (default = 2)
<code>seed</code>	Integer. Random seed for reproducible permutation testing. If NULL (default), uses current random state. Set to a specific value (e.g., 123) for reproducible results across runs. Applies to both parallel and serial permutation.

**Value**

A matrix object containing the estimated cibersort-cell fractions, p-values, correlation coefficients, and RMSE values.

**Author(s)**

Aaron M. Newman, Stanford University (amnewman@stanford.edu)

**Examples**

```
lm22 <- load_data("lm22")
common_genes <- rownames(lm22)[1:500]
sim_mixture <- as.data.frame(matrix(
  rnorm(length(common_genes) * 10, mean = 5, sd = 2),
  nrow = length(common_genes), ncol = 10
))
rownames(sim_mixture) <- common_genes
colnames(sim_mixture) <- paste0("Sample", 1:10)
result <- CIBERSORT(
  sig_matrix = lm22,
  mixture_file = sim_mixture,
  perm = 10, QN = FALSE, absolute = FALSE,
  parallel = FALSE
)
head(result)
```

---

clear_iobr_cache	<i>Clear IOBR Data Cache</i>
------------------	------------------------------

---

**Description**

Removes all cached data files downloaded from GitHub.

**Usage**

```
clear_iobr_cache()
```

**Value**

Invisible NULL. Called for side effects of clearing the cache.

---

combine_pd_eset	<i>Combine Phenotype Data and Expression Set</i>
-----------------	--

---

**Description**

Merges phenotype data with an expression matrix by matching sample IDs. Optionally filters features, applies feature manipulation, and scales expression data before combining.

**Usage**

```
combine_pd_eset(
  eset,
  pdata,
  id_pdata = "ID",
  feas = NULL,
  feature_manipulation = TRUE,
  scale = TRUE,
  choose_who_when_duplicate = c("eset", "pdata")
)
```

**Arguments**

eset	Expression matrix with genes/features in rows and samples in columns.
pdata	Data frame containing phenotype/clinical data.
id_pdata	Character string specifying the column name in ‘pdata’ containing sample identifiers. Default is “ID”.
feas	Character vector specifying features to include from ‘eset’. If ‘NULL’, all features are used. Default is ‘NULL’.
feature_manipulation	Logical indicating whether to apply feature manipulation to filter valid features. Default is ‘TRUE’.
scale	Logical indicating whether to scale (standardize) expression data. Default is ‘TRUE’.
choose_who_when_duplicate	Character string specifying which data to prefer when duplicate columns exist. Options are “eset” or “pdata”. Default is “eset”.

**Value**

Data frame combining phenotype data and (transposed) expression data, with samples in rows and features/phenotypes in columns.

**Author(s)**

Dongqiang Zeng

## Examples

```
set.seed(123)
eset <- matrix(rnorm(1000), nrow = 100, ncol = 10)
rownames(eset) <- paste0("Gene", 1:100)
colnames(eset) <- paste0("Sample", 1:10)
pdata <- data.frame(
  ID = colnames(eset),
  group = rep(c("A", "B"), each = 5),
  age = rnorm(10, 50, 10)
)
result <- combine_pd_eset(eset = eset, pdata = pdata, scale = FALSE)
dim(result)
```

---

Construct\_con

*Construct Contrast Matrix*

---

## Description

Creates a contrast matrix for differential analysis, where each phenotype level is contrasted against all other levels combined.

## Usage

```
Construct_con(pheno)
```

## Arguments

pheno                    Factor with different levels representing groups to contrast.

## Value

Square matrix with dimensions equal to the number of levels in 'pheno'. Each row represents a contrast where the corresponding level is compared against the average of others.

## Examples

```
pheno <- factor(c("A", "B", "C", "D"))
contrast_matrix <- Construct_con(pheno)
print(contrast_matrix)
```

---

ConvertRownameToLoci    *Convert Rowname To Loci*

---

### Description

Processes a gene expression data matrix by modifying its row names. Extracts the gene identifier from row names formatted as 'GENE|ID', simplifying them to 'GENE'.

### Usage

```
ConvertRownameToLoci(cancerGeneExpression)
```

### Arguments

cancerGeneExpression  
Matrix or data frame. Gene expression data with row names in the format 'GENE|ID'.

### Value

Matrix with modified gene expression data with updated row names. Rows without a valid identifier are removed.

### Examples

```
example_data <- matrix(runif(20), ncol = 5)
rownames(example_data) <- c("LOC101", "LOC102", "LOC103", "LOC104")
processed_data <- ConvertRownameToLoci(example_data)
print(processed_data)
```

---

CoreAlg

*Core Algorithm for CIBERSORT Deconvolution*

---

### Description

Performs nu-regression using support vector machines (SVM) to estimate cell type proportions. This is the core computational engine of CIBERSORT, using nu-SVR with linear kernel to decompose mixed gene expression signals.

### Usage

```
CoreAlg(X, y, absolute, abs_method)
```

**Arguments**

<code>X</code>	Matrix or data frame containing signature matrix (predictor variables). Rows are genes, columns are cell types.
<code>y</code>	Numeric vector containing the mixture sample expression (response variable).
<code>absolute</code>	Logical indicating whether to use absolute space for weights. Default is FALSE (relative proportions).
<code>abs_method</code>	String specifying the method for absolute space weights: "sig.score" or "no.sumto1".

**Value**

List containing:

**w** Estimated cell type weights/proportions

**mix\_rmse** Root mean squared error of the deconvolution

**mix\_r** Correlation coefficient between observed and predicted mixture

**Examples**

```
X <- matrix(rnorm(100), nrow = 10)
y <- rnorm(10)
result <- CoreAlg(X, y, absolute = FALSE, abs_method = "sig.score")
```

---

count2tpm

*Convert Read Counts to Transcripts Per Million (TPM)*

---

**Description**

Transforms gene expression count data into Transcripts Per Million (TPM) values, normalizing for gene length and library size. Supports multiple gene ID types and can retrieve gene length information from BioMart or use local datasets.

**Usage**

```
count2tpm(
  countMat,
  idType = "Ensembl",
  org = c("hsa", "mmus"),
  source = c("local", "biomart"),
  effLength = NULL,
  id = "id",
  gene_symbol = "symbol",
  length = "eff_length",
  check_data = FALSE
)
```

**Arguments**

countMat	Numeric matrix of raw read counts with genes in rows and samples in columns.
idType	Character string specifying the gene identifier type. Options are "Ensembl", "Entrez", or "Symbol". Default is "Ensembl".
org	Character string specifying the organism. Options include "hsa" (human) or "mmus" (mouse). Default is "hsa".
source	Character string specifying the source for gene length information. Options are "biomart" (retrieve from Ensembl BioMart) or "local" (use local dataset). Default is "local".
effLength	Data frame containing effective gene length information. If 'NULL', lengths are retrieved based on 'source'. Default is 'NULL'.
id	Character string specifying the column name in 'effLength' containing gene identifiers. Default is "id".
gene_symbol	Character string specifying the column name in 'effLength' containing gene symbols. Default is "symbol".
length	Character string specifying the column name in 'effLength' containing gene lengths. Default is "eff_length".
check_data	Logical indicating whether to check for missing values in the count matrix. Default is 'FALSE'.

**Value**

Data frame of TPM-normalized expression values with genes in rows and samples in columns. Gene identifiers are converted to gene symbols in the output, regardless of the input 'idType'.

**Author(s)**

Wubing Zhang, Dongqiang Zeng, Yiran Fang

**Examples**

```
# Load TCGA count data
eset_stad <- load_data("eset_stad")

# Transform to TPM using local gene annotation
eset <- count2tpm(countMat = eset_stad, source = "local", idType = "ensembl")
head(eset)
```

---

creat_folder	<i>Create Nested Output Folders</i>
--------------	-------------------------------------

---

### Description

Creates one to three nested folders (if not existing) under the current working directory and returns their names and absolute paths.

### Usage

```
creat_folder(f1, f2 = NULL, f3 = NULL, return = NULL)
```

### Arguments

f1	Character. First-level folder name.
f2	Character or 'NULL'. Second-level folder name. Default is 'NULL'.
f3	Character or 'NULL'. Third-level folder name. Default is 'NULL'.
return	Deprecated (not used). Kept for backward compatibility.

### Value

List with elements:

**folder\_name** Relative path to the created folder

**abspath** Absolute path ending with '/'

### Examples

```
creat_folder(file.path(tempdir(), "1-result"))
creat_folder(file.path(tempdir(), "1-result"), "figures", "correlation")
```

---

deconvolute\_quantiseq.default

*Use quanTIseq to Deconvolute a Gene Expression Matrix*

---

### Description

Deconvolutes gene expression data to estimate immune cell fractions using the quanTIseq method. Source code from <https://github.com/FFinotello/quanTIseq>.

**Usage**

```
deconvolute_quantiseq.default(
  mix.mat,
  arrays = FALSE,
  signame = "TIL10",
  tumor = FALSE,
  mRNAscale = TRUE,
  method = c("lsei", "hampel", "huber", "bisquare"),
  rmgenes = "unassigned"
)
```

**Arguments**

mix.mat	Data frame or matrix. Gene expression matrix with gene symbols on the first column and sample IDs on the first row. Expression data must be on non-log scale (TPM for RNA-seq or expression values for microarrays).
arrays	Logical. Whether expression data are from microarrays. Default is FALSE. If TRUE, the rmgenes parameter is set to "none".
signame	Character. Name of the signature matrix. Currently only "TIL10" is available. Default is "TIL10".
tumor	Logical. Whether expression data are from tumor samples. If TRUE, signature genes with high expression in tumor samples are removed. Default is FALSE.
mRNAscale	Logical. Whether cell fractions must be scaled to account for cell-type-specific mRNA content. Default is TRUE.
method	Character. Deconvolution method: "hampel", "huber", "bisquare" for robust regression, or "lsei" for constrained least squares. Default is "lsei".
rmgenes	Character. Genes to remove: "unassigned" (default), "default", "none", or "path".

**Value**

Data frame with cell fractions for each sample.

**Author(s)**

Finotello F, et al. (adapted for IOBR)

**References**

F. Finotello, C. Mayer, C. Plattner, G. Laschober, D. Rieder, H. Hackl, A. Krogsdam, W. Posch, D. Wilflingseder, S. Sopper, M. Jsselsteijn, D. Johnsons, Y. Xu, Y. Wang, M. E. Sanders, M. V. Estrada, P. Ericsson-Gonzalez, J. Balko, N. F. de Miranda, Z. Trajanoski. "quanTlseq: quantifying immune contexture of human tumors". bioRxiv 223180. <https://doi.org/10.1101/223180>.

## Examples

```
lm22 <- load_data("lm22")
common_genes <- rownames(lm22)[1:500]
tpm_matrix <- as.data.frame(matrix(
  rnorm(length(common_genes) * 5, mean = 5, sd = 2),
  nrow = length(common_genes), ncol = 5
))
rownames(tpm_matrix) <- common_genes
colnames(tpm_matrix) <- paste0("Sample", 1:5)
results <- deconvolute_quantiseq.default(mix.mat = tpm_matrix)
```

---

deconvolute\_timer.default

*Deconvolute Tumor Microenvironment Using TIMER*

---

## Description

Performs deconvolution of the tumor microenvironment using the TIMER algorithm. Processes multiple cancer datasets, removes batch effects, and estimates immune cell type abundances.

## Usage

```
deconvolute_timer.default(args)
```

## Arguments

**args** List or environment containing parameters:  
**outdir** Character. Output directory path.  
**batch** Character. File containing paths to expression data and cancer types.

## Value

Matrix of abundance scores for different immune cell types across multiple cancer samples.

## Examples

```
## Not run:
# This example requires actual expression data files
# Create a batch file with paths to expression data and cancer types
batch_file <- "batch.csv"
# batch.csv format: each row contains expression_file_path,cancer_type
# Example content:
# /path/to/exp1.txt,luad
# /path/to/exp2.txt,brca
outdir <- tempdir()
args <- list(outdir = outdir, batch = batch_file)
results <- deconvolute_timer.default(args)

## End(Not run)
```

---

deconvo\_cibersort      *Deconvolve Using CIBERSORT*

---

### Description

CIBERSORT is freely available to academic users. License and binary can be obtained from <https://cibersortx.stanford.edu>.

### Usage

```
deconvo_cibersort(  
  eset,  
  project = NULL,  
  arrays = FALSE,  
  perm = 1000,  
  absolute = FALSE,  
  abs_method = "sig.score",  
  parallel = FALSE,  
  num_cores = 2,  
  seed = NULL  
)
```

### Arguments

eset	Expression matrix with gene symbols as row names.
project	Optional project name. Default is 'NULL'.
arrays	Logical: optimized for microarray data. Default is 'FALSE'.
perm	Permutations for statistical analysis. Default is 1000.
absolute	Logical: run in absolute mode. Default is 'FALSE'.
abs_method	Method for absolute mode: "sig.score" or "no.sumto1". Default is "sig.score".
parallel	Enable parallel execution. Default is 'FALSE'.
num_cores	Number of cores for parallel mode. Default is 2.
seed	Random seed for reproducibility. Default is 'NULL'.

### Value

Data frame with CIBERSORT cell fractions. Columns suffixed with '\_CIBERSORT'.

### Author(s)

Dongqiang Zeng

**Examples**

```
eset_tme_stad <- load_data("eset_tme_stad")
lm22 <- load_data("lm22")

cibersort_result <- deconvo_cibersort(
  eset = eset_tme_stad,
  project = "TCGA-STAD",
  perm = 100
)
```

---

deconvo\_epic

*Deconvolve Immune Microenvironment Using EPIC*

---

**Description**

Estimates immune cell fractions using EPIC algorithm.

**Usage**

```
deconvo_epic(eset, project = NULL, tumor = TRUE)
```

**Arguments**

eset	Gene expression matrix with genes as row names.
project	Optional project name. Default is 'NULL'.
tumor	Logical indicating tumor ('TRUE') or normal ('FALSE') samples.

**Value**

Data frame with EPIC cell fraction estimates. Columns suffixed with '\_EPIC'.

**Author(s)**

Dongqiang Zeng

**Examples**

```
eset_stad <- load_data("eset_stad")
anno_grch38 <- load_data("anno_grch38")
eset <- anno_eset(eset = eset_stad, annotation = anno_grch38, probe = "id")
eset <- eset[1:500, 1:5]
epic_result <- deconvo_epic(eset = eset, project = "Example", tumor = TRUE)
head(epic_result)
```

---

deconvo_estimate	<i>Calculate ESTIMATE Scores</i>
------------------	----------------------------------

---

**Description**

Calculates stromal, immune, and ESTIMATE scores from gene expression.

**Usage**

```
deconvo_estimate(eset, project = NULL, platform = "affymetrix")
```

**Arguments**

eset	Gene expression matrix with gene symbols.
project	Optional project name. Default is 'NULL'.
platform	Platform type: "affymetrix" or "illumina". Default is "affymetrix".

**Value**

Data frame with ESTIMATE scores. Columns suffixed with '\_estimate'.

**Author(s)**

Dongqiang Zeng

**Examples**

```
eset_stad <- load_data("eset_stad")
anno_grch38 <- load_data("anno_grch38")
eset <- anno_eset(eset = eset_stad, annotation = anno_grch38, probe = "id")

estimate_result <- deconvo_estimate(eset, project = "TCGA-STAD")
```

---

deconvo_ips	<i>Calculate Immunophenoscore (IPS)</i>
-------------	---

---

**Description**

Calculates immune phenotype scores from gene expression data.

**Usage**

```
deconvo_ips(eset, project = NULL, plot = FALSE)
```

**Arguments**

eset            Gene expression matrix.  
project        Optional project name. Default is 'NULL'.  
plot           Logical: generate visualization. Default is 'FALSE'.

**Value**

Data frame with IPS scores. Columns suffixed with '\_IPS'.

**Author(s)**

Dongqiang Zeng

**Examples**

```
eset_stad <- load_data("eset_stad")  
anno_grch38 <- load_data("anno_grch38")  
eset <- anno_eset(eset = eset_stad, annotation = anno_grch38, probe = "id")  
  
ips_result <- deconvo_ips(eset = eset, project = "TCGA-STAD")
```

---

deconvo\_mcpcounter      *Deconvolve Immune Microenvironment Using MCP-Counter*

---

**Description**

Estimates immune cell abundances using MCP-counter.

**Usage**

```
deconvo_mcpcounter(eset, project = NULL)
```

**Arguments**

eset            Gene expression matrix with HGNC symbols as row names.  
project        Optional project name. Default is 'NULL'.

**Value**

Data frame with MCP-counter scores. Columns suffixed with '\_MCPcounter'.

**Author(s)**

Dongqiang Zeng

## Examples

```
eset_stad <- load_data("eset_stad")
anno_grch38 <- load_data("anno_grch38")
eset <- anno_eset(eset = eset_stad, annotation = anno_grch38, probe = "id")
eset <- eset[1:500, 1:3]

mcp_result <- deconvo_mcpcounter(eset = eset, project = "TCGA-STAD")
head(mcp_result)
```

---

deconvo\_quantiseq      *Deconvolve Using quanTIseq*

---

## Description

quanTIseq deconvolution for RNA-seq immune cell fractions.

## Usage

```
deconvo_quantiseq(eset, project = NULL, tumor, arrays, scale_mrna)
```

## Arguments

eset	Gene expression matrix.
project	Optional project name. Default is 'NULL'.
tumor	Logical: tumor samples. Must be specified.
arrays	Logical: microarray data. Must be specified.
scale_mrna	Logical: correct for mRNA content. Must be specified.

## Value

Data frame with quanTIseq cell fractions. Columns suffixed with '\_quantiseq'.

## Author(s)

Dongqiang Zeng

## Examples

```
eset_stad <- load_data("eset_stad")
anno_grch38 <- load_data("anno_grch38")
eset <- anno_eset(eset = eset_stad, annotation = anno_grch38, probe = "id")
eset <- eset[1:500, 1:3]

res <- deconvo_quantiseq(
  eset = eset, project = "stad", tumor = TRUE,
  arrays = FALSE, scale_mrna = FALSE
```

```
)
head(res)
```

---

deconvo\_ref

*Deconvolve Using Custom Reference*


---

## Description

Cell fraction estimation using SVR or lsei methods with custom reference.

## Usage

```
deconvo_ref(
  eset,
  project = NULL,
  arrays = TRUE,
  method = c("svr", "lsei"),
  perm = 100,
  reference,
  scale_reference = TRUE,
  absolute.mode = FALSE,
  abs.method = "sig.score"
)
```

## Arguments

eset	Gene expression matrix.
project	Optional project name. Default is 'NULL'.
arrays	Logical: use quantile normalization. Default is 'TRUE'.
method	Method: "svr" or "lsei". Default is "svr".
perm	Permutations for SVR. Default is 100.
reference	Custom reference matrix (e.g., lm22, lm6).
scale_reference	Logical: scale reference. Default is 'TRUE'.
absolute.mode	Logical: absolute mode for SVR. Default is 'FALSE'.
abs.method	Method for absolute mode. Default is "sig.score".

## Value

Data frame with cell fractions. Columns suffixed with '\_CIBERSORT'.

## Author(s)

Dongqiang Zeng, Rongfang Shen

**Examples**

```
lm22 <- load_data("lm22")
common_genes <- rownames(lm22)[1:500]
sim_eset <- as.data.frame(matrix(
  rnorm(length(common_genes) * 5, mean = 5, sd = 2),
  nrow = length(common_genes), ncol = 5
))
rownames(sim_eset) <- common_genes
colnames(sim_eset) <- paste0("Sample", 1:5)

deconvo_ref(eset = sim_eset, reference = lm22, method = "lse")
```

---

deconvo\_timer

*Deconvolve Using TIMER*


---

**Description**

TIMER deconvolution for cancer-specific immune estimation.

**Usage**

```
deconvo_timer(eset, project = NULL, indications = NULL)
```

**Arguments**

eset	Gene expression matrix.
project	Optional project name. Default is 'NULL'.
indications	Cancer type for each sample (e.g., "brca", "stad"). Must match number of columns in 'eset'.

**Value**

Data frame with TIMER cell fractions. Columns suffixed with '\_TIMER'.

**Author(s)**

Dongqiang Zeng

**Examples**

```
eset_stad <- load_data("eset_stad")
anno_grch38 <- load_data("anno_grch38")
eset <- anno_eset(eset = eset_stad, annotation = anno_grch38, probe = "id")

res <- deconvo_timer(
  eset = eset, project = "stad",
  indications = rep("stad", ncol(eset))
```

```
)
head(res)
```

---

deconvo\_tme

*Main TME Deconvolution Function*


---

## Description

Unified interface for multiple TME deconvolution methods.

## Usage

```
deconvo_tme(
  eset,
  project = NULL,
  method = tme_deconvolution_methods,
  arrays = FALSE,
  tumor = TRUE,
  perm = 1000,
  reference,
  scale_reference = TRUE,
  plot = FALSE,
  scale_mrna = TRUE,
  group_list = NULL,
  platform = "affymetrix",
  absolute.mode = FALSE,
  abs.method = "sig.score",
  ...
)
```

## Arguments

eset	Gene expression matrix with HGNC symbols as row names.
project	Optional project name. Default is 'NULL'.
method	Deconvolution method. See [tme_deconvolution_methods].
arrays	Logical: microarray-optimized mode. Default is 'FALSE'.
tumor	Logical: tumor-optimized mode (EPIC). Default is 'TRUE'.
perm	Permutations (CIBERSORT/SVR). Default is 1000.
reference	Custom reference matrix (SVR/lsei).
scale_reference	Logical: scale reference (SVR/lsei).
plot	Logical: generate plots (IPS). Default is 'FALSE'.
scale_mrna	Logical: mRNA correction (quanTIseq/EPIC).

group_list	Cancer types for TIMER (vector).
platform	Platform for ESTIMATE. Default is "affymetrix".
absolute.mode	Logical: absolute mode (CIBERSORT/SVR). Default is 'FALSE'.
abs.method	Absolute mode method. Default is "sig.score".
...	Additional arguments passed to method.

**Value**

Tibble with cell fractions and 'ID' column.

**Author(s)**

Dongqiang Zeng, Rongfang Shen

**References**

1. Newman et al. (2015). Robust enumeration of cell subsets from tissue expression profiles. *Nature Methods*.
2. Vegesna et al. (2013). Inferring tumour purity and stromal/immune cell admixture. *Nature Communications*.
3. Finotello et al. (2019). Molecular and pharmacological modulators of the tumor immune contexture. *Genome Medicine*.
4. Li et al. (2016). Comprehensive analyses of tumor immunity. *Genome Biology*.
5. Charoentong et al. (2017). Pan-cancer Immunogenomic Analyses. *Cell Reports*.
6. Becht et al. (2016). Estimating population abundance of tissue-infiltrating immune cells. *Genome Biology*.
7. Aran et al. (2017). xCell: digitally portraying tissue cellular heterogeneity. *Genome Biology*.
8. Racle et al. (2017). Simultaneous enumeration of cancer and immune cell types. *ELife*.

**Examples**

```
lm22 <- load_data("lm22")
common_genes <- rownames(lm22)[1:500]
sim_eset <- as.data.frame(matrix(
  rnorm(length(common_genes) * 5, mean = 5, sd = 2),
  nrow = length(common_genes), ncol = 5
))
rownames(sim_eset) <- common_genes
colnames(sim_eset) <- paste0("Sample", 1:5)

res <- deconvo_tme(eset = sim_eset, method = "cibersort", perm = 10)
```

---

deconvo_xcell	<i>Deconvolve Immune Microenvironment Using xCell</i>
---------------	---

---

## Description

Estimates immune cell fractions using the xCell algorithm. xCell provides cell type enrichment scores for 64 immune and stromal cell types from gene expression data.

## Usage

```
deconvo_xcell(eset, project = NULL, arrays = FALSE)
```

## Arguments

eset	Gene expression matrix with HGNC gene symbols as row names and samples as columns.
project	Optional project name to add as 'ProjectID' column. Default is 'NULL'.
arrays	Logical indicating microarray data ('TRUE') or RNA-seq ('FALSE'). Default is 'FALSE'.

## Value

Data frame with xCell enrichment scores. Cell type columns are suffixed with '\_xCell'.

## Author(s)

Dongqiang Zeng

## Examples

```
eset_stad <- load_data("eset_stad")
anno_grch38 <- load_data("anno_grch38")
eset <- anno_eset(eset = eset_stad, annotation = anno_grch38, probe = "id")
xcell_result <- deconvo_xcell(eset = eset[, 1:3], project = "TCGA-STAD")
head(xcell_result)[, 1:5]
```

---

`design_mytheme`*Design Custom Theme for ggplot2 Plots*

---

## Description

Creates a customized ggplot2 theme based on user-specified parameters for plot elements such as title size, axis sizes, legend settings, and theme style. Supports various base themes and allows fine-tuning of visual aspects.

## Usage

```
design_mytheme(  
  theme = c("light", "bw", "classic", "classic2"),  
  plot_title_size = 2,  
  axis_title_size = 2,  
  axis_text_size = 12,  
  axis_angle = 60,  
  hjust = 1,  
  legend.position = "bottom",  
  legend.direction = "horizontal",  
  legend.size = 0.25,  
  legend.key.height = 0.5,  
  legend.key.width = 0.5,  
  legend.size.text = 10,  
  legend.box = "horizontal"  
)
```

## Arguments

<code>theme</code>	Base theme: "light", "bw", "classic", "classic2". Default is "light".
<code>plot_title_size</code>	Relative size of plot title. Default is 2.
<code>axis_title_size</code>	Relative size of axis titles. Default is 2.
<code>axis_text_size</code>	Size of axis tick labels. Default is 12.
<code>axis_angle</code>	Angle of x-axis tick labels. Default is 60.
<code>hjust</code>	Horizontal justification for x-axis text. Default is 1.
<code>legend.position</code>	Legend position: "none", "left", "right", "bottom", "top". Default is "bottom".
<code>legend.direction</code>	Direction of legend items: "horizontal" or "vertical". Default is "horizontal".
<code>legend.size</code>	Size of legend key. Default is 0.25.
<code>legend.key.height</code>	Height of legend key in cm. Default is 0.5.

legend.key.width	Width of legend key in cm. Default is 0.5.
legend.size.text	Size of legend text labels. Default is 10.
legend.box	Orientation of legend box: "horizontal" or "vertical". Default is "horizontal".

**Value**

A ggplot2 theme object.

**Author(s)**

Dongqiang Zeng

**Examples**

```
library(ggplot2)
p <- ggplot(mtcars, aes(wt, mpg)) +
  geom_point()
mytheme <- design_mytheme(theme = "bw", plot_title_size = 1.5, axis_text_size = 14)
p + mytheme + ggtitle("Example Plot")
```

doPerm

*Permutation Test for CIBERSORT***Description**

Performs permutation-based sampling to generate an empirical null distribution of correlation coefficients for p-value calculation in CIBERSORT analysis. Randomly samples from the mixture data to create null distributions.

**Usage**

```
doPerm(perm, X, Y, absolute, abs_method, seed = NULL)
```

**Arguments**

perm	Integer. Number of permutations to perform ( $\geq 100$ recommended for reliable p-value estimation).
X	Matrix or data frame containing signature matrix (predictor variables).
Y	Numeric vector containing the mixture sample expression.
absolute	Logical indicating whether to use absolute space for weights.
abs_method	String specifying the method for absolute space weights: "sig.score" or "no.sumto1".
seed	Integer. Random seed for reproducibility. If NULL (default), uses current random state.

**Value**

List containing:

**dist** Numeric vector of correlation coefficients from permutations

**Examples**

```
X <- matrix(rnorm(100), nrow = 10)
Y <- rnorm(10)
result <- doPerm(100, X, Y, absolute = FALSE, abs_method = "sig.score")
```

---

download\_iobr\_data      *Download IOBR Data from GitHub with Mirror Support*

---

**Description**

Downloads large datasets from GitHub releases to avoid CRAN size limits. Supports multiple download mirrors for users in different regions. Data is cached locally after first download.

**Usage**

```
download_iobr_data(
  name,
  force = FALSE,
  verbose = TRUE,
  mirrors = get_default_mirrors()
)
```

**Arguments**

name	Character string. Name of the dataset to download.
force	Logical. Whether to force re-download even if cached. Default: FALSE.
verbose	Logical. Whether to print progress messages. Default: TRUE.
mirrors	Character vector. URLs of mirrors to try. Default uses get_default_mirrors().

**Value**

The requested dataset.

**Examples**

```
# Download TCGA STAD signature data
tcga_sig <- download_iobr_data("tcga_stad_sig")

# Download with custom mirrors
eset <- download_iobr_data("eset_stad",
  mirrors = c(
```

```
    "https://ghproxy.vip/https://github.com",  
    "https://gh-proxy.org/https://github.com"  
  )  
)
```

---

DrawQQPlot

*Draw QQ Plot Comparing Cancer and Immune Expression*

---

## Description

Creates a quantile-quantile (QQ) plot to compare gene expression distributions between cancer and immune samples. Points along the diagonal indicate similar distributions.

## Usage

```
DrawQQPlot(cancer.exp, immune.exp, name = "")
```

## Arguments

cancer.exp	Vector. Gene expression data for cancer samples.
immune.exp	Vector. Gene expression data for immune samples.
name	Character. Optional subtitle with additional information.

## Value

Generates a QQ plot.

## Examples

```
cancer_exp <- rnorm(100, mean = 5, sd = 1.5)  
immune_exp <- rnorm(100, mean = 5, sd = 1.5)  
DrawQQPlot(  
  cancer.exp = cancer_exp,  
  immune.exp = immune_exp,  
  name = "Comparison of Gene Expression"  
)
```

Enet

*Elastic Net Model Fitting*

---

**Description**

Fits elastic net model with cross-validation to find optimal alpha and lambda. Searches across a grid of alpha values (0 to 1) and lambda values to minimize cross-validation error.

**Usage**

```
Enet(train.x, train.y, lambdamax, nfold = 10)
```

**Arguments**

<code>train.x</code>	Training predictors matrix.
<code>train.y</code>	Training binary outcomes (0/1 or factor).
<code>lambdamax</code>	Maximum lambda value for the grid search.
<code>nfold</code>	Number of CV folds. Default is '10'.

**Value**

List containing:

**chose\_alpha** Optimal alpha value (0-1)

**chose\_lambda** Optimal lambda value

**Examples**

```
if (requireNamespace("glmnet", quietly = TRUE)) {  
  set.seed(123)  
  train_data <- matrix(rnorm(50 * 5), ncol = 5)  
  train_outcome <- rbinom(50, 1, 0.5)  
  result <- Enet(train.x = train_data, train.y = train_outcome, lambdamax = 1, nfold = 5)  
}
```

---

enrichment\_barplot*Enrichment Bar Plot with Two Directions*

---

**Description**

Creates a bar plot visualizing enrichment results for up-regulated and down-regulated terms, using  $-\log_{10}(\text{p-values})$  to indicate significance.

**Usage**

```
enrichment_barplot(  
  up_terms,  
  down_terms,  
  terms = "Description",  
  pvalue = "pvalue",  
  group = "group",  
  palette = "jama",  
  cols = NULL,  
  title = "Gene Ontology Enrichment",  
  width_wrap = 30,  
  font_terms = 15  
)
```

**Arguments**

up_terms	Data frame for up-regulated terms.
down_terms	Data frame for down-regulated terms.
terms	Column name for term descriptions. Default is "Description".
pvalue	Column name for p-values. Default is "pvalue".
group	Column name for group indicator. Default is "group".
palette	Color palette. Default is "jama".
cols	Character vector. Custom colors for bars. If NULL, uses palette. Default is NULL.
title	Plot title. Default is "Gene Ontology Enrichment".
width_wrap	Maximum width for wrapping pathway names. Default is 30.
font_terms	Font size for axis labels. Default is 15.

**Value**

A ggplot object of the enrichment bar plot.

**Author(s)**

Dongqiang Zeng

**Examples**

```
up_terms <- data.frame(  
  Description = c("Pathway1", "Pathway2"),  
  pvalue = c(0.001, 0.01)  
)  
down_terms <- data.frame(  
  Description = c("Pathway4", "Pathway5"),  
  pvalue = c(0.005, 0.02)  
)  
p <- enrichment_barplot(  
  up_terms,  
  down_terms,  
  terms = "Description",  
  pvalue = "pvalue",  
  group = "group",  
  palette = "jama",  
  cols = NULL,  
  title = "Gene Ontology Enrichment",  
  width_wrap = 30,  
  font_terms = 15  
)
```

```

up_terms = up_terms,
down_terms = down_terms,
title = "Custom Enrichment Plot"
)
p

```

---

EPIC

*Estimate the proportion of immune and cancer cells.*


---

## Description

EPIC takes as input bulk gene expression data (RNA-seq) and returns the proportion of mRNA and cells composing the various samples.

## Usage

```

EPIC(
  bulk,
  reference = NULL,
  mRNA_cell = NULL,
  mRNA_cell_sub = NULL,
  sigGenes = NULL,
  scaleExprs = TRUE,
  withOtherCells = TRUE,
  constrainedSum = TRUE,
  rangeBasedOptim = FALSE
)

```

## Arguments

bulk	A matrix (nGenes x nSamples) of the genes expression from each bulk sample (the counts should be given in TPM, RPKM or FPKM when using the prebuilt reference profiles). This matrix needs to have rownames telling the gene names (corresponds to the gene symbol in the prebuilt reference profiles (e.g. CD8A, MS4A1) - no conversion of IDs is performed at the moment). It is advised to keep all genes in the bulk instead of a subset of signature genes (except if scaleExprs = FALSE in which case it doesn't make any difference).
reference	(optional): A string or a list defining the reference cells. It can take multiple formats: - 'NULL': to use the default reference profiles and genes signature TRef - A character: either "BRef" or "TRef" to use the reference cells and genes signature of the corresponding datasets (see BRef and TRef) - A list containing: - '\$refProfiles': a matrix (nGenes x nCellTypes) of the reference cells genes expression (don't include a column of the 'other cells' (representing usually the cancer cells for which such a profile is usually not conserved between samples); the rownames needs to be defined as well as the colnames giving the names of each gene and reference cell types respectively. It is advised to keep all genes in this refProfiles matrix instead of a subset of signature genes - '\$sigGenes':

a character vector of the gene names to use as signature - sigGenes can also be given as a direct input to EPIC function - '\$refProfiles.var' (optional): a matrix (nGenes x nCellTypes) of the variability of each gene expression for each cell type, which is used to define weights on each gene for the optimization (if this is absent, we assume an identical variability for all genes in all cells) - it needs to have the same dimnames than refProfiles

- mRNA\_cell (optional): A named numeric vector: tells (in arbitrary units) the amount of mRNA for each of the reference cells and of the other uncharacterized (cancer) cell. Two names are of special meaning: "otherCells" - used for the mRNA/cell value of the "other cells" from the sample (i.e. the cell type that don't have any reference gene expression profile) ; and *default* - used for the mRNA/cell of the cells from the reference profiles for which no specific value is given in mRNA\_cell (i.e. if mRNA\_cell=c(Bcells=2, NKcells=2.1, otherCells=3.5, default=1), then if the refProfiles described Bcells, NKcells and Tcells, we would use a value of 3.5 for the "otherCells" that didn't have any reference profile and a default value of 1 for the Tcells when computing the cell fractions). To note: if data is in tpm, this mRNA per cell would ideally correspond to some number of transcripts per cell.
- mRNA\_cell\_sub (optional): This can be given instead of mRNA\_cell (or in addition to it). It is also a named numeric vector, used to replace only the mRNA/cell values from some cell types (or to add values for new cell types). The values given in mRNA\_cell\_sub will overwrite the default values as well as those that might have been given by mRNA\_cell.
- sigGenes (optional): a character vector of the gene names to use as signature for the deconvolution. In principle this is given with the reference as the "reference\$sigGenes" but if we give a value for this input variable, it is these signature genes that will be used instead of the ones given with the reference profile.
- scaleExprs (optional, default is TRUE): boolean telling if the bulk samples and reference gene expression profiles should be rescaled based on the list of genes in common between the them (such a rescaling is recommended).
- withOtherCells (optional, default is TRUE): if EPIC should allow for an additional cell type for which no gene expression reference profile is available or if the bulk is assumed to be composed only of the cells with reference profiles.
- constrainedSum (optional, default is TRUE): tells if the sum of all cell types should be constrained to be < 1. When withOtherCells=FALSE, there is additionally a constrain the the sum of all cell types with reference profiles must be > 0.99.
- rangeBasedOptim (optional): when this is FALSE (the default), the least square optimization is performed as described in *Racle et al., 2017, eLife*, which is recommended. When this variable is TRUE, EPIC uses the variability of each gene from the reference profiles in another way: instead of defining weights (based on the variability) for the fit of each gene, we define a range of values accessible for each gene (based on the gene expression value in the reference profile +/- the variability values). The error that the optimization tries to minimize is by how much the predicted gene expression is outside of this allowed range of values.

## Details

This function uses a constrained least square minimization to estimate the proportion of each cell type with a reference profile and another uncharacterized cell type in bulk gene expression samples.

The names of the genes in the bulk samples, the reference samples and in the gene signature list need to be the same format (gene symbols are used in the predefined reference profiles). The full list of gene names don't need to be exactly the same between the reference and bulk samples: *EPIC* will use the intersection of the genes. In case of duplicate gene names, *EPIC* will use the median value per duplicate - if you want to consider these cases differently, you can remove the duplicates before calling *EPIC*.

## Value

A list of 3 matrices: - 'mRNAProportions': (nSamples x (nCellTypes+1)) the proportion of mRNA coming from all cell types with a ref profile + the uncharacterized other cell - 'cellFractions': (nSamples x (nCellTypes+1)) this gives the proportion of cells from each cell type after accounting for the mRNA / cell value - 'fit.gof': (nSamples x 12) a matrix telling the quality for the fit of the signature genes in each sample. It tells if the minimization converged, and other info about this fit comparing the measured gene expression in the sigGenes vs predicted gene expression in the sigGenes

## Examples

```
melanoma_data <- load_data("melanoma_data")
TRef <- load_data("TRef")
res1 <- EPIC(melanoma_data$counts)
res1$cellFractions
res2 <- EPIC(melanoma_data$counts, TRef)
res3 <- EPIC(bulk = melanoma_data$counts, reference = TRef)
res4 <- EPIC(melanoma_data$counts, reference = "TRef")
res5 <- EPIC(melanoma_data$counts, mRNA_cell_sub = c(
  Bcells = 1,
  otherCells = 5
))
# Various possible ways of calling EPIC function. res 1 to 4 should
# give exactly the same outputs, and the elements res1$cellFractions
# should be equal to the example predictions found in
# melanoma_data$cellFractions.pred for these first 4 results.
# The values of cellFraction for res5 will be different due to the use of
# other mRNA per cell values for the B and other cells.
```

## Description

Generates boxplots and density plots to analyze the distribution of expression values in an expression set. Useful for quality control and assessing data normalization.

**Usage**

```
eset_distribution(eset, quantile = 3, log = TRUE, project = NULL)
```

**Arguments**

eset	Expression matrix or data frame with genes in rows and samples in columns.
quantile	Integer specifying the divisor for sampling columns. Default is 3 (samples 1/3 of columns).
log	Logical indicating whether to perform log2 transformation. Default is 'TRUE'.
project	Optional output directory path for saving files. If 'NULL', no files are saved. Default is 'NULL'.

**Value**

Invisibly returns 'NULL'. If 'project' is provided, saves PNG files to disk.

**Examples**

```
eset_stad <- load_data("eset_stad")
anno_rnaseq <- load_data("anno_rnaseq")
eset <- anno_eset(eset = eset_stad, annotation = anno_rnaseq)
eset_distribution(eset)
eset_distribution(eset, project = file.path(tempdir(), "ESET"))
```

---

estimateScore

*estimateScore*

---

**Description**

This function reads a gene expression dataset in GCT format, calculates enrichment scores for specific gene sets, and writes the computed scores to an output file. It supports multiple platform types and performs platform-specific calculations if necessary.

**Usage**

```
estimateScore(
  input.ds,
  output.ds,
  platform = c("affymetrix", "agilent", "illumina")
)
```

**Arguments**

input.ds	A character string specifying the path to the input dataset file in GCT format. The file should have gene expression data with appropriate headers.
output.ds	A character string specifying the path to the output dataset file, where the calculated scores will be written.
platform	A character vector indicating the platform type. Must be one of "affymetrix", "agilent", or "illumina". Platform-specific calculations are performed based on this parameter.

**Value**

This function does not return a value but writes the computed scores to the specified output file in GCT format.

**Examples**

```

eset_stad <- load_data("eset_stad")
anno_grch38 <- load_data("anno_grch38")
eset <- anno_eset(eset = eset_stad, annotation = anno_grch38, probe = "id")
eset <- as.data.frame(eset)
eset <- tibble::rownames_to_column(eset, var = "symbol")
input_file <- tempfile(pattern = "estimate_", fileext = ".gct")
output_file <- tempfile(pattern = "estimate_score_", fileext = ".gct")
writeLines(c("#1.2", paste(nrow(eset), ncol(eset) - 1, sep = "\t")), input_file)
utils::write.table(
  eset,
  input_file,
  sep = "\t", row.names = FALSE, col.names = TRUE, append = TRUE, quote = FALSE
)
estimateScore(input.ds = input_file, output.ds = output_file, platform = "affymetrix")

```

---

exact\_pvalue

*Calculate Exact P-Value for Correlation*

---

**Description**

Computes the exact p-value for the correlation between two numeric variables using a specified correlation method.

**Usage**

```
exact_pvalue(x, y, method)
```

**Arguments**

x	Numeric vector representing the first variable.
y	Numeric vector representing the second variable.
method	Character string specifying the correlation method: "spearman", "pearson", or "kendall".

**Value**

Numeric value representing the exact p-value.

**Author(s)**

Dongqiang Zeng

**Examples**

```
sig_stad <- load_data("sig_stad")
p_val <- exact_pvalue(
  x = sig_stad$CD8.T.cells,
  y = sig_stad$CD_8_T_effector,
  method = "spearman"
)
print(p_val)
```

---

extract\_sc\_data

*Extract Data Frame from Seurat Object*

---

**Description**

Extracts and combines a data frame with cells as rows and features as columns from Seurat assay data. Supports multiple assays and optional metadata integration.

**Usage**

```
extract_sc_data(
  sce,
  vars = NULL,
  assay,
  slot = "scale.data",
  combine_meta_data = TRUE
)
```

**Arguments**

sce	Seurat object.
vars	Character vector of feature names to extract. If 'NULL', all features are extracted.
assay	Character vector specifying assay(s) to pull data from.
slot	Character string specifying the assay data slot. Default is "scale.data".
combine_meta_data	Logical indicating whether to combine metadata with the extracted data frame. Default is 'TRUE'.

**Value**

Data frame with cells as rows and features as columns.

**Author(s)**

Dongqiang Zeng

**Examples**

```
if (requireNamespace("Seurat", quietly = TRUE)) {  
  pbmc <- SeuratObject::pbmc_small  
  vars <- c("PPBP", "IGLL5", "VDAC3", "CD1C", "AKR1C3")  
  eset <- extract_sc_data(sce = pbmc, vars = vars, assay = "RNA")  
}
```

---

feature\_manipulation *Feature Quality Control and Filtering*

---

**Description**

Filters features (variables) in a matrix or data frame by removing those with missing values, non-numeric types, infinite values, or zero variance. This is useful for preparing data for downstream statistical analyses.

**Usage**

```
feature_manipulation(  
  data,  
  feature = NULL,  
  is_matrix = FALSE,  
  print_result = FALSE  
)
```

**Arguments**

data	A matrix or data frame containing features to filter.
feature	Character vector of feature names to check. If 'is_matrix = TRUE', features are extracted from row names of the matrix.
is_matrix	Logical indicating whether 'data' is a gene expression matrix (features as rows, samples as columns). If 'TRUE', the matrix is transposed for processing. Default is 'FALSE'.
print_result	Logical indicating whether to print filtering statistics. Default is 'FALSE'.

**Value**

Character vector of feature names that pass all quality checks.

**Author(s)**

Dongqiang Zeng

**Examples**

```
eset_stad <- load_data("eset_stad")
feas <- feature_manipulation(
  data = eset_stad,
  feature = rownames(eset_stad),
  is_matrix = TRUE,
  print_result = TRUE
)
```

---

`feature_select`*Feature Selection via Correlation or Differential Expression*

---

**Description**

Selects informative features using either correlation with a quantitative response or differential expression (limma) for binary/continuous responses.

**Usage**

```
feature_select(
  x,
  y,
  method = c("cor", "dif"),
  family = c("spearman", "pearson"),
  cutoff = NULL,
  padjcut = NULL
)
```

**Arguments**

<code>x</code>	Numeric matrix. Features (rows) by samples (columns).
<code>y</code>	Numeric or factor. Response vector (quantitative or binary).
<code>method</code>	Character. "cor" (correlation) or "dif" (differential expression). Default <code>c("cor","dif")</code> .
<code>family</code>	Character. Correlation method if <code>method = "cor"</code> : "spearman" or "pearson".
<code>cutoff</code>	Numeric. Absolute correlation (for cor) or <code>llog2FC</code> (for dif) threshold.
<code>padjcut</code>	Numeric. Adjusted p-value cutoff.

**Value**

Character vector of selected feature names.

**Examples**

```

imvigor210_eset <- load_data("imvigor210_eset")
mad <- apply(imvigor210_eset, 1, mad)
imvigor210_eset <- imvigor210_eset[mad > 0.5, ]
pd1 <- as.numeric(imvigor210_eset["PDCD1", ])
group <- ifelse(pd1 > mean(pd1), "high", "low")
pd1_cor <- feature_select(
  x = imvigor210_eset, y = pd1, method = "cor",
  family = "pearson", padjcut = 0.05, cutoff = 0.5
)
pd1_dif <- feature_select(
  x = imvigor210_eset, y = pd1, method = "dif",
  padjcut = 0.05, cutoff = 2
)
pd1_dif_2 <- feature_select(
  x = imvigor210_eset, y = group,
  method = "dif", padjcut = 0.05, cutoff = 2
)

```

---

filterCommonGenes      *filterCommonGenes*

---

**Description**

This function filters and merges a dataset with a set of common genes.

**Usage**

```
filterCommonGenes(input.f, output.f, id = c("GeneSymbol", "EntrezID"))
```

**Arguments**

input.f	A character string specifying the path to the input file or a connection object. The file should be a tab-separated table with row names.
output.f	A character string specifying the path to the output file.
id	A character string indicating the type of gene identifier to use. Can be either "GeneSymbol" or "EntrezID".

**Value**

No return value. The function writes the merged dataset to the specified output file.

**Examples**

```

# Create a sample input dataframe
input_data <- data.frame(
  GeneSymbol = c("BRCA1", "TP53", "EGFR", "NOTCH1"),
  Value = c(10, 15, 8, 12),

```

```
  stringsAsFactors = FALSE
)

# Write the input data to temporary file
input_file <- tempfile(fileext = ".txt")
output_file <- tempfile(fileext = ".txt")
write.table(input_data,
  file = input_file, sep = "\t", row.names = TRUE,
  quote = FALSE
)

# Call the filterCommonGenes function
filterCommonGenes(input_file, output_file, id = "GeneSymbol")
```

---

find\_markers\_in\_bulk *Identify Marker Features in Bulk Expression Data*

---

## Description

Identifies informative marker features across groups from bulk gene expression or signature score matrices using Seurat workflows. Performs feature selection, scaling, PCA, clustering, and marker discovery.

## Usage

```
find_markers_in_bulk(
  pdata,
  eset,
  group,
  id_pdata = "ID",
  nfeatures = 2000,
  top_n = 20,
  thresh.use = 0.25,
  only.pos = TRUE,
  min.pct = 0.25,
  npcs = 30
)
```

## Arguments

pdata	Data frame. Sample metadata.
eset	Matrix. Gene expression or signature score matrix.
group	Character. Column name in pdata specifying grouping variable.
id_pdata	Character. Column name for sample IDs. Default is "ID".
nfeatures	Integer. Number of top variable features to select. Default is 2000.
top_n	Integer. Number of top markers to retain per cluster. Default is 20.

thresh.use	Numeric. Threshold for marker selection. Default is 0.25.
only.pos	Logical. Whether to retain only positive markers. Default is TRUE.
min.pct	Numeric. Minimum expression percentage threshold. Default is 0.25.
npcs	Integer. Number of principal components to use. Default is 30.

### Value

List with components: ‘sce’ (Seurat object), ‘markers’ (all markers), ‘top\_markers’ (top markers per group).

### Examples

```

eset_tme_stad <- load_data("eset_tme_stad")
colnames(eset_tme_stad) <- substring(colnames(eset_tme_stad), 1, 12)
pdata_sig_tme <- load_data("pdata_sig_tme")

res <- find_markers_in_bulk(
  pdata = pdata_sig_tme, eset = eset_tme_stad,
  group = "TMEcluster"
)
# Extract top markers per cluster using base R
top_markers <- res$top_markers

```

---

find\_mutations

*Analyze Mutations Related to Signature Scores*

---

### Description

This function identifies mutations associated with a specific signature score, performs statistical tests for significance, and generates oncoprints and box plots to visualize relationships.

### Usage

```

find_mutations(
  mutation_matrix,
  signature_matrix,
  id_signature_matrix = "ID",
  signature,
  min_mut_freq = 0.05,
  plot = TRUE,
  method = "multi",
  point_alpha = 0.1,
  save_path = NULL,
  palette = "jco",
  cols = NULL,
  show_plot = TRUE,

```

```

    show_col = FALSE,
    width = 8,
    height = 4,
    oncoprint_group_by = "mean",
    oncoprint_col = "#224444",
    gene_counts = 10,
    jitter = FALSE,
    genes = NULL,
    point_size = 4.5
  )

```

### Arguments

<code>mutation_matrix</code>	A matrix of mutation data with samples in rows and genes in columns.
<code>signature_matrix</code>	A data frame with sample identifiers and signature scores.
<code>id_signature_matrix</code>	Column name in 'signature_matrix' for sample identifiers.
<code>signature</code>	Name of the target signature for analysis.
<code>min_mut_freq</code>	Minimum mutation frequency required for gene inclusion. Default is 0.05.
<code>plot</code>	Logical indicating whether to generate and save plots. Default is TRUE.
<code>method</code>	Statistical test method: "multi" for both Cuzick and Wilcoxon, or "Wilcoxon" only. Default is "multi".
<code>point_alpha</code>	Transparency of points in box plot. Default is 0.1.
<code>save_path</code>	Directory to save plots and results. If NULL, no files are saved.
<code>palette</code>	Color palette for box plots(used when cols is NULL). Default is "jco".
<code>cols</code>	Character vector. Custom colors for box plots. If NULL, uses palette. Default is NULL.
<code>show_plot</code>	Logical indicating whether to display plots. Default is TRUE.
<code>show_col</code>	Logical indicating whether to show color codes. Default is FALSE.
<code>width</code>	Width of oncoprint plot. Default is 8.
<code>height</code>	Height of oncoprint plot. Default is 4.
<code>oncoprint_group_by</code>	Grouping method for oncoprint: "mean" or "quantile". Default is "mean".
<code>oncoprint_col</code>	Color for mutations in oncoprint. Default is "#224444".
<code>gene_counts</code>	Number of genes to display in oncoprint. Default is 10.
<code>jitter</code>	Logical indicating whether to add jitter to box plot points. Default is FALSE.
<code>genes</code>	Optional vector of gene names; if NULL, selects based on frequency.
<code>point_size</code>	Size of points in box plot. Default is 4.5.

### Value

A list containing statistical test results, oncoprint plots, and box plots.

**Author(s)**

Dongqiang Zeng

**Examples**

```
## Not run:
# This example requires a MAF file from TCGA or maftools
# See maftools or TCGAbiolinks documentation for obtaining MAF files
mut_list <- make_mut_matrix(
  maf = "path_to_maf_file", isTCGA = TRUE,
  category = "multi"
)
mut <- mut_list$snp
results <- find_mutations(
  mutation_matrix = mut, signature_matrix = signature_data,
  id_signature_matrix = "ID", signature = "CD_8_T_effector",
  min_mut_freq = 0.01, plot = TRUE, method = "multi"
)

## End(Not run)
```

---

find\_outlier\_samples *Identify Outlier Samples in Gene Expression Data*

---

**Description**

Analyzes gene expression data to identify potential outlier samples using connectivity analysis via the WGCNA package. Calculates normalized adjacency and connectivity z-scores for each sample, generates connectivity plots, and optionally performs hierarchical clustering.

**Usage**

```
find_outlier_samples(
  eset,
  yinter = -3,
  project = NULL,
  plot_hculst = FALSE,
  show_plot = TRUE,
  index = NULL,
  save = FALSE
)
```

**Arguments**

eset	Numeric matrix. Gene expression data with genes as rows and samples as columns.
yinter	Numeric. Z-score threshold for identifying outliers. Default is -3.

project	Character or 'NULL'. Output directory path for saving plots. Required if 'save = TRUE'. Default is 'NULL'.
plot_hculst	Logical. Whether to plot hierarchical clustering. Default is 'FALSE'.
show_plot	Logical. Whether to display the connectivity plot. Default is 'TRUE'.
index	Integer or 'NULL'. Index for output file naming. Default is 'NULL'.
save	Logical. Whether to save plots to files. Default is 'FALSE'.

### Value

Character vector of sample names identified as potential outliers.

### Author(s)

Dongqiang Zeng

### Examples

```
eset_tme_stad <- load_data("eset_tme_stad")
outs <- find_outlier_samples(eset = eset_tme_stad)
print(outs)
```

---

find\_variable\_genes    *Identify Variable Genes in Expression Data*

---

### Description

Identifies variable genes from a gene expression dataset using specified selection criteria. Supports multiple methods, including expression thresholding and variability estimation via median absolute deviation (MAD).

### Usage

```
find_variable_genes(
  eset,
  data_type = c("count", "normalized"),
  methods = c("low", "mad"),
  prop = 0.7,
  quantile = c(0.75, 0.5, 0.25),
  min.mad = 0.1,
  feas = NULL
)
```

**Arguments**

eset	Numeric matrix. Gene expression data (genes as rows, samples as columns).
data_type	Character. Type of data: "count" or "normalized". Default is "count".
methods	Character vector. Methods for gene selection: "low", "mad". Default is 'c("low", "mad")'.
prop	Numeric. Proportion of samples in which a gene must be expressed. Default is 0.7.
quantile	Numeric. Quantile threshold for minimum MAD (0.25, 0.5, 0.75). Default is 0.75.
min.mad	Numeric. Minimum allowable MAD value. Default is 0.1.
feas	Character vector or 'NULL'. Additional features to include. Default is 'NULL'.

**Value**

Matrix subset of 'eset' containing variable genes.

**Author(s)**

Dongqiang Zeng

**Examples**

```
eset_tme_stad <- load_data("eset_tme_stad")
eset <- find_variable_genes(
  eset = eset_tme_stad,
  data_type = "normalized",
  methods = "mad",
  quantile = 0.25
)
```

---

format\_msigdb

*Format Input Signatures from MSigDB*


---

**Description**

Reads a GMT file from MSigDB and converts it into a named list of gene sets suitable for IOBR functions.

**Usage**

```
format_msigdb(gmt, ont = "term", gene = "gene")
```

**Arguments**

gmt	Character string. Path to a GMT file.
ont	Character string. Name of the signature/set column in the parsed GMT table. Default is "term".
gene	Character string. Name of the gene column in the parsed GMT table. Default is "gene".

**Value**

Named list of character vectors, where each element contains the genes belonging to one signature.

**Examples**

```
tf <- tempfile(fileext = ".gmt")
writeLines(
  c(
    "HALLMARK_TNFA_SIGNALING_VIA_NFKB\tNA\tTNF\tNFKB1\tNFKB2",
    "HALLMARK_P53_PATHWAY\tNA\tTP53\tMDM2\tCDKN1A"
  ),
  con = tf
)

sig_list <- format_msigdb(tf, ont = "term", gene = "gene")
names(sig_list)
sig_list[[1]]
```

---

format\_signatures      *Transform Signature Data into List Format*

---

**Description**

Converts signature data from a data frame (with signatures as columns and genes as rows) into a list format suitable for IOBR functions. Handles NA values appropriately.

**Usage**

```
format_signatures(sig_data, save_signature = FALSE, output_path = NULL)
```

**Arguments**

sig_data	Data frame with signature names as columns and genes in rows. Use 'NA' for missing values.
save_signature	Logical. Whether to save the signature list as RData. Default is 'FALSE'.
output_path	Character. Full path (without extension) for output file. Required if 'save_signature = TRUE'. Default is 'NULL'.

**Value**

List of signatures.

**Author(s)**

Dongqiang Zeng

**Examples**

```
sig_data <- data.frame(
  Signature1 = c("Gene1", "Gene2", "Gene3", NA),
  Signature2 = c("Gene4", "Gene5", NA, NA)
)
format_signatures(sig_data)
```

---

generateRef

*Generate Reference Signature Matrix*

---

**Description**

Generates a reference signature matrix for cell types based on differential expression analysis. Supports both limma for normalized data and DESeq2 for raw count data.

**Usage**

```
generateRef(dds, pheno, FDR = 0.05, dat, method = "limma")
```

**Arguments**

dds	Matrix. Raw count data from RNA-seq. Required if ‘method = "DESeq2"’.
pheno	Character vector. Cell type class of the samples.
FDR	Numeric. Genes with BH adjusted p-value < FDR are considered significant. Default is 0.05.
dat	Matrix or data frame. Normalized transcript quantification data (e.g., FPKM, TPM).
method	Character. Method for differential expression: “limma” or “DESeq2”. Default is “limma”.

**Value**

List containing: - ‘reference\_matrix’: Data frame of median expression for significant genes across cell types. - ‘G’: Optimal number of probes minimizing condition number. - ‘condition\_number’: Minimum condition number. - ‘whole\_matrix’: Full median expression matrix.

## Examples

```
expressionData <- matrix(runif(1000 * 4, min = 0, max = 10), ncol = 4)
rownames(expressionData) <- paste("Gene", 1:1000, sep = "_")
colnames(expressionData) <- paste("Sample", 1:4, sep = "_")

phenotype <- c("celltype1", "celltype2", "celltype1", "celltype2")

rawCountData <- matrix(sample(1:100, 1000 * 4, replace = TRUE), ncol = 4)
rownames(rawCountData) <- paste("Gene", 1:1000, sep = "_")
colnames(rawCountData) <- paste("Sample", 1:4, sep = "_")

result <- generateRef(
  dds = rawCountData, pheno = phenotype,
  FDR = 0.05, dat = expressionData, method = "DESeq2"
)
```

---

generateRef\_DEseq2      *Generate Reference Signature Matrix Using DESeq2*

---

## Description

Uses DESeq2 to perform differential expression analysis across cell types, identifies significantly expressed genes, and creates a reference signature matrix from median expression levels.

## Usage

```
generateRef_DEseq2(dds, pheno, FDR = 0.05, dat)
```

## Arguments

dds	Matrix. Raw count data from RNA-seq.
pheno	Character vector. Cell type classes for samples.
FDR	Numeric. Threshold for adjusted p-values. Default is 0.05.
dat	Matrix. Normalized expression data (e.g., FPKM, TPM) for calculating median expression.

## Value

List containing: - 'reference\_matrix': Data frame of median expression for significant genes across cell types. - 'G': Optimal number of probes minimizing condition number. - 'condition\_number': Minimum condition number. - 'whole\_matrix': Full median expression matrix.

**Examples**

```

set.seed(123)
dds <- matrix(sample(0:1000, 2000, replace = TRUE), nrow = 100, ncol = 20)
colnames(dds) <- paste("Sample", 1:20, sep = "_")
rownames(dds) <- paste("Gene", 1:100, sep = "_")
pheno <- rep(c("Type1", "Type2"), each = 10)
dat <- matrix(runif(2000), nrow = 100, ncol = 20)
rownames(dat) <- rownames(dds)
colnames(dat) <- colnames(dds)

result <- generateRef_DEseq2(dds = dds, pheno = pheno, FDR = 0.05, dat = dat)
print(result$reference_matrix)

```

---

generateRef\_limma

*Generate Reference Signature Matrix Using Limma*


---

**Description**

Performs differential expression analysis using the limma package to identify significantly expressed genes across different cell types. Computes median expression levels of these significant genes to create a reference signature matrix.

**Usage**

```
generateRef_limma(dat, pheno, FDR = 0.05)
```

**Arguments**

dat	Matrix or data frame. Gene probes in rows and samples in columns.
pheno	Character vector. Cell type class of the samples.
FDR	Numeric. Genes with BH adjusted p-value < FDR are considered significant. Default is 0.05.

**Value**

List containing: - 'reference\_matrix': Data frame of median expression values for significantly expressed genes. - 'G': Number of probes used that resulted in the lowest condition number. - 'condition\_number': Minimum condition number obtained. - 'whole\_matrix': Matrix of median values across all samples.

**Examples**

```

dat <- matrix(rnorm(2000), nrow = 100)
rownames(dat) <- paste("Gene", 1:100, sep = "_")
colnames(dat) <- paste("Sample", 1:20, sep = "_")
pheno <- sample(c("Type1", "Type2", "Type3"), 20, replace = TRUE)
results <- generateRef_limma(dat, pheno)
print(results)

```

---

generateRef\_rnaseq      *Generate Reference Gene Matrix from RNA-seq DEGs*

---

### Description

Uses DESeq2 to identify differentially expressed genes and create a reference matrix from median expression levels across cell types.

### Usage

```
generateRef_rnaseq(dds, pheno, mode = "oneVSothers", FDR = 0.05, dat)
```

### Arguments

dds	Matrix. Raw count data from RNA-seq.
pheno	Character vector. Cell type classes.
mode	Character. DEG identification mode: "oneVSothers" or "pairs". Default is "oneVSothers".
FDR	Numeric. Threshold for adjusted p-values. Default is 0.05.
dat	Matrix. Normalized expression data (e.g., FPKM, TPM).

### Value

List containing: - 'reference\_matrix': Data frame of median expression for significant genes across cell types. - 'G': Optimal number of probes minimizing condition number. - 'condition\_number': Minimum condition number. - 'whole\_matrix': Full median expression matrix.

### Author(s)

Rongfang Shen

### Examples

```
dds <- matrix(rpois(200 * 10, lambda = 10), ncol = 10)
rownames(dds) <- paste("Gene", 1:200, sep = "_")
colnames(dds) <- paste("Sample", 1:10, sep = "_")
pheno <- sample(c("Type1", "Type2", "Type3"), 10, replace = TRUE)
dat <- matrix(rnorm(200 * 10), ncol = 10)
rownames(dat) <- rownames(dds)
colnames(dat) <- colnames(dds)

results <- generateRef_rnaseq(dds = dds, pheno = pheno, FDR = 0.05, dat = dat)
```

---

generateRef\_seurat      *Generate Reference Matrix from Seurat Object*

---

### Description

Generates reference gene expression data from a Seurat object by identifying marker genes for each cell type and aggregating expression data.

### Usage

```
generateRef_seurat(
  sce,
  celltype = NULL,
  proportion = NULL,
  assay_deg = "RNA",
  slot_deg = "data",
  adjust_assay = FALSE,
  assay_out = "RNA",
  slot_out = "data",
  verbose = FALSE,
  only.pos = TRUE,
  n_ref_genes = 50,
  logfc.threshold = 0.15,
  test.use = "wilcox"
)
```

### Arguments

sce	Seurat object containing single-cell RNA-seq data.
celltype	Character. Cell type column name in metadata. Default is 'NULL' (uses default identity).
proportion	Numeric. Proportion of cells to randomly select for analysis. Default is 'NULL' (use all cells).
assay_deg	Character. Assay for finding markers. Default is "RNA".
slot_deg	Character. Slot for finding markers. Default is "data".
adjust_assay	Logical. Whether to adjust assay for SCT. Default is 'FALSE'.
assay_out	Character. Assay for output. Default is "RNA".
slot_out	Character. Slot for output. Default is "data".
verbose	Logical. Print verbose messages. Default is 'FALSE'.
only.pos	Logical. Return only positive markers. Default is 'TRUE'.
n_ref_genes	Integer. Number of reference genes per cell type. Default is 50.
logfc.threshold	Numeric. Log fold change threshold. Default is 0.15.
test.use	Character. Statistical test for marker identification. Default is "wilcox".

**Value**

Matrix containing aggregated expression data for reference genes.

**Author(s)**

Dongqiang Zeng

**Examples**

```
## Not run:
if (requireNamespace("Seurat", quietly = TRUE)) {
  # Requires a Seurat object with sufficient cells and markers
  sm <- generateRef_seurat(sce = seurat_obj, celltype = "cell_type", slot_out = "data")
}

## End(Not run)
```

---

GetFractions.Abbas      *Constrained Regression Method (Abbas et al., 2009)*

---

**Description**

Implements a constrained regression approach described by Abbas et al. (2009). Estimates proportions of immune cell types within mixed cancer tissue samples based on gene expression data. Iteratively adjusts regression coefficients to ensure non-negative values.

**Usage**

```
GetFractions.Abbas(XX, YY, w = NA)
```

**Arguments**

XX	Matrix. Immune expression data with genes as rows and cell types as columns.
YY	Vector. Cancer expression data with gene expression levels.
w	Vector or NA. Weights for regression. Default is NA (no weights).

**Value**

Vector with non-negative coefficients representing proportions of each cell type.

**Examples**

```
XX <- matrix(runif(100), nrow = 10, ncol = 10)
colnames(XX) <- paste("CellType", 1:10, sep = "")
YY <- runif(10)
results <- GetFractions.Abbas(XX, YY)
print(results)
```

---

getHRandCIfromCoxph     *Extract Hazard Ratio and Confidence Intervals from Cox Model*

---

**Description**

Extracts hazard ratio (HR) and 95 Cox proportional hazards model.

**Usage**

```
getHRandCIfromCoxph(coxphData)
```

**Arguments**

coxphData     Fitted Cox model object from 'survival::coxph()'.

**Value**

Data frame with p-values, HR, and confidence intervals.

**Author(s)**

Dorothee Nickles

Dongqiang Zeng

**Examples**

```
library(survival)
set.seed(123)
df <- data.frame(
  TTE = rexp(200, rate = 0.1),
  Cens = rbinom(200, size = 1, prob = 0.7),
  group = sample(c("Treatment", "Control"), 200, replace = TRUE)
)
coxphData <- survival::coxph(survival::Surv(TTE, Cens) ~ group, data = df)
results <- getHRandCIfromCoxph(coxphData)
print(results)
```

---

GetOutlierGenes     *Get Outlier Genes*

---

**Description**

Identifies outlier genes from multiple cancer datasets. Treats the top 5 expressed genes in each sample as outliers and returns unique outlier genes.

**Usage**

```
GetOutlierGenes(cancers)
```

**Arguments**

cancers            Data frame. One column containing paths to gene expression files.

**Value**

Vector of unique gene names identified as outliers.

**Examples**

```
tf1 <- tempfile(fileext = ".tsv")
tf2 <- tempfile(fileext = ".tsv")

expr1 <- data.frame(
  gene = c("GeneA", "GeneB", "GeneC", "GeneD", "GeneE", "GeneF"),
  Sample1 = c(10, 50, 30, 80, 60, 20),
  Sample2 = c(15, 40, 25, 90, 55, 10)
)

expr2 <- data.frame(
  gene = c("GeneA", "GeneB", "GeneC", "GeneD", "GeneE", "GeneF"),
  Sample3 = c(100, 20, 10, 60, 30, 80),
  Sample4 = c(95, 25, 15, 70, 35, 85)
)

write.table(expr1, tf1, sep = "\t", row.names = FALSE, quote = FALSE)
write.table(expr2, tf2, sep = "\t", row.names = FALSE, quote = FALSE)

cancers <- data.frame(ExpressionFiles = c(tf1, tf2))
outlier_genes <- GetOutlierGenes(cancers)
```

---

get\_cols

*Set and View Color Palettes*

---

**Description**

Retrieves color palettes from the IOBR package with options for randomization and visualization. Users can specify predefined palettes or provide custom colors.

**Usage**

```
get_cols(cols = "normal", palette = 1, show_col = TRUE, seed = 123)
```

**Arguments**

cols	Character vector of colors, or one of: - "normal": Use standard palette - "random": Randomly shuffle the palette Default is "normal".
palette	Numeric or character specifying the palette. Options are 1, 2, 3, 4, or palette name. Default is 1.
show_col	Logical indicating whether to display the color palette. Default is 'TRUE'.
seed	Integer seed for random number generator when 'cols = "random"'. Default is 123.

**Value**

Character vector of colors.

**Examples**

```
# Get default palette
mycols <- get_cols()

# Get random palette
mycols <- get_cols(cols = "random", seed = 456)

# Use custom colors
mycols <- get_cols(cols = c("red", "blue", "green"))
```

---

get\_cor

*Calculate and Visualize Correlation Between Two Variables*

---

**Description**

Calculates and visualizes the correlation between two variables with options for scaling, handling missing values, and incorporating grouping data.

**Usage**

```
get_cor(
  eset,
  pdata = NULL,
  var1,
  var2,
  is.matrix = FALSE,
  id_eset = "ID",
  id_pdata = "ID",
  scale = TRUE,
  subtype = NULL,
  na.subtype.rm = FALSE,
  color_subtype = NULL,
  palette = "jama",
```

```

index = NULL,
method = c("spearman", "pearson", "kendall"),
show_cor_result = TRUE,
col_line = NULL,
id = NULL,
show_label = FALSE,
point_size = 4,
title = NULL,
alpha = 0.5,
title_size = 1.5,
text_size = 10,
axis_angle = 0,
hjust = 0,
show_plot = TRUE,
save_plot = FALSE,
path = NULL,
fig.format = "png",
fig.width = 7,
fig.height = 7.3,
add.hdr.line = FALSE
)

```

### Arguments

eset	Dataset containing the variables (data frame or matrix).
pdata	Optional phenotype data frame. Default is 'NULL'.
var1	Name of the first variable.
var2	Name of the second variable.
is.matrix	Logical indicating if 'eset' is a matrix with features as rows. Default is 'FALSE'.
id_eset	ID column in 'eset'. Default is "ID".
id_pdata	ID column in 'pdata'. Default is "ID".
scale	Logical indicating whether to scale data. Default is 'TRUE'.
subtype	Optional grouping variable for coloring points. Default is 'NULL'.
na.subtype.rm	Logical indicating whether to remove NA in subtype. Default is 'FALSE'.
color_subtype	Colors for subtypes. Default is 'NULL'.
palette	Color palette name. Default is "jama".
index	Plot index for filename. Default is 'NULL' (uses 1).
method	Correlation method: "spearman", "pearson", or "kendall". Default is "spearman".
show_cor_result	Logical indicating whether to print correlation result. Default is 'TRUE'.
col_line	Color of regression line. Default is 'NULL' (auto-determine).
id	Column for point labels. Default is 'NULL'.
show_label	Logical indicating whether to show labels. Default is 'FALSE'.

point_size	Size of points. Default is 4.
title	Plot title. Default is 'NULL'.
alpha	Transparency of points. Default is 0.5.
title_size	Title font size. Default is 1.5.
text_size	Text font size. Default is 10.
axis_angle	Axis label angle. Default is 0.
hjust	Horizontal justification. Default is 0.
show_plot	Logical indicating whether to display plot. Default is 'TRUE'.
save_plot	Logical indicating whether to save plot. Default is 'FALSE'.
path	Save path. Default is 'NULL'.
fig.format	Figure format: "png" or "pdf". Default is "png".
fig.width	Figure width in inches. Default is 7.
fig.height	Figure height in inches. Default is 7.3.
add.hdr.line	Logical for adding HDR (high density region) lines. Default is 'FALSE'.

**Value**

A ggplot object of the correlation plot.

**Author(s)**

Dongqiang Zeng

**Examples**

```

eset_tme_stad <- load_data("eset_tme_stad")
get_cor(
  eset = eset_tme_stad,
  is.matrix = TRUE,
  var1 = "GZMB",
  var2 = "CD274"
)

```

---

get_cor_matrix	<i>Calculate and Visualize Correlation Matrix Between Two Variable Sets</i>
----------------	---

---

**Description**

Calculates and visualizes the correlation matrix between two sets of variables. Supports Pearson, Spearman, and Kendall correlation methods. The function generates a customizable heatmap with significance stars.

**Usage**

```

get_cor_matrix(
  data,
  feas1,
  feas2,
  method = c("pearson", "spearman", "kendall"),
  path = NULL,
  index = 1,
  fig.type = "pdf",
  width = NULL,
  height = NULL,
  project = NULL,
  is.matrix = FALSE,
  scale = TRUE,
  font.size = 15,
  fill_by_cor = FALSE,
  round.num = 1,
  font.size.star = 8,
  cols = NULL
)

```

**Arguments**

<code>data</code>	Input data frame or matrix. Variables should be in columns.
<code>feas1</code>	Character vector of variable names for the first set.
<code>feas2</code>	Character vector of variable names for the second set.
<code>method</code>	Correlation method: "pearson", "spearman", or "kendall". Default is "pearson".
<code>path</code>	Directory to save the plot. If 'NULL', plot is not saved. Default is 'NULL'.
<code>index</code>	Numeric prefix for output filename. Default is 1.
<code>fig.type</code>	File format: "pdf", "png", etc. Default is "pdf".
<code>width</code>	Plot width in inches. Auto-calculated if 'NULL'.
<code>height</code>	Plot height in inches. Auto-calculated if 'NULL'.
<code>project</code>	Project name for plot title. Default is 'NULL'.
<code>is.matrix</code>	Logical: if 'TRUE', data is transposed. Default is 'FALSE'.
<code>scale</code>	Logical: scale variables before correlation. Default is 'TRUE'.
<code>font.size</code>	Font size for axis labels. Default is 15.
<code>fill_by_cor</code>	Logical: show correlation values instead of stars. Default is 'FALSE'.
<code>round.num</code>	Decimal places for correlation values. Default is 1.
<code>font.size.star</code>	Font size for significance stars. Default is 8.
<code>cols</code>	Custom colors for gradient (low, mid, high). If 'NULL', uses blue-white-red. Default is 'NULL'.

**Value**

ggplot object displaying the correlation matrix heatmap.

**Author(s)**

Dongqiang Zeng

**Examples**

```
set.seed(123)
data <- as.data.frame(matrix(rnorm(1000), nrow = 100, ncol = 10))
colnames(data) <- paste0("Gene_", 1:10)

feas1 <- c("Gene_1", "Gene_2", "Gene_3")
feas2 <- c("Gene_4", "Gene_5", "Gene_6")

cor_plot <- get_cor_matrix(
  data = data,
  feas1 = feas1,
  feas2 = feas2,
  method = "spearman",
  project = "Example Correlation"
)
```

---

get\_sig\_sc

*Extract Top Marker Genes from Single-Cell Differential Results*

---

**Description**

Selects the top N marker genes per cluster from a ranked differential expression result table.

**Usage**

```
get_sig_sc(
  deg,
  cluster = "cluster",
  gene = "gene",
  avg_log2FC = "avg_log2FC",
  n = 100
)
```

**Arguments**

deg	Data frame or matrix. Ranked marker statistics.
cluster	Character. Column name containing cluster identifiers. Default is "cluster".
gene	Character. Column name containing gene identifiers. Default is "gene".
avg_log2FC	Character. Column name for average log2 fold change. Default is "avg_log2FC".
n	Integer. Number of top markers per cluster. Default is 100.

**Value**

List of character vectors; each element contains the top  $N$  genes for a cluster.

**Examples**

```
deg <- load_data("deg")
get_sig_sc(deg, cluster = "cluster", gene = "gene", avg_log2FC = "avg_log2FC", n = 100)
```

---

 high\_var\_fea

*Identify High-Variance Features from Statistical Results*


---

**Description**

Selects top variable (up- and down-regulated) features based on adjusted p-value and log fold-change thresholds.

**Usage**

```
high_var_fea(
  result,
  target,
  name_padj = "padj",
  padj_cutoff = 1,
  name_logfc,
  logfc_cutoff = 0,
  n = 10,
  data_type = NULL
)
```

**Arguments**

result	Data frame or tibble. Statistical results containing feature, adjusted p-value, and logFC columns.
target	Character. Column name of feature identifiers.
name_padj	Character. Adjusted p-value column name. Default is "padj".
padj_cutoff	Numeric. Adjusted p-value threshold. Default is 1.
name_logfc	Character. log2 fold-change column name.
logfc_cutoff	Numeric. Absolute log2 fold-change threshold. Default is 0.
n	Integer. Number of top up and top down features to select. Default is 10.
data_type	Character or 'NULL'. If "survival", adjusts logFC interpretation. Default is 'NULL'.

**Value**

Character vector of selected feature names (combined up and down sets).

**Author(s)**

Dongqiang Zeng

**Examples**

```

result_data <- data.frame(
  gene = c("Gene1", "Gene2", "Gene3", "Gene4", "Gene5"),
  padj = c(0.01, 0.02, 0.05, 0.001, 0.03),
  logfc = c(-2, 1.5, -3, 2.5, 0.5)
)
high_var_fea(
  result = result_data,
  target = "gene",
  name_padj = "padj",
  name_logfc = "logfc",
  n = 2,
  padj_cutoff = 0.05,
  logfc_cutoff = 1.5
)

```

imvigor210\_pdata

*IMvigor210 Bladder Cancer Immunotherapy Cohort Data***Description**

Clinical and biomarker data from the IMvigor210 clinical trial cohort. Includes treatment response, survival outcomes, and immune biomarker measurements for bladder cancer patients treated with atezolizumab.

**Usage**

```
data(imvigor210_pdata)
```

**Format**

A data frame with patients as rows and variables as columns:

**ID** Patient sample identifier

**BOR** Best overall response (CR, PR, SD, PD, NA)

**BOR\_binary** Binary response classification (R=responder, NR=non-responder)

**OS\_days** Overall survival time in days

**OS\_status** Overall survival status (0=alive, 1=dead)

**Mutation\_Load** Tumor mutation burden

**Neo\_antigen\_Load** Neoantigen load

**CD\_8\_T\_effector** CD8+ T effector signature score

**Immune\_Checkpoint** Immune checkpoint signature score

**Pan\_F\_TBRs** Pan-fibroblast TGF- $\beta$  response signature

**Mismatch\_Repair** Mismatch repair status or signature

**TumorPurity** Estimated tumor purity

### Source

IMvigor210 clinical trial (NCT02108652)

### References

Mariathasan S et al. TGF $\beta$  attenuates tumour response to PD-L1 blockade by contributing to exclusion of T cells. *Nature* 554, 544-548 (2018). doi:10.1038/nature25501

### Examples

```
data(imvigor210_pdata)
head(imvigor210_pdata)
```

---

iobr\_cor\_plot

*Integrative Correlation Analysis Between Phenotype and Features*

---

### Description

Performs comprehensive correlation analysis between phenotype data and feature data, supporting both continuous and categorical phenotypes. Filters features based on statistical significance and generates publication-ready visualizations including box plots, heatmaps, and correlation plots.

### Usage

```
iobr_cor_plot(
  pdata_group,
  id1 = "ID",
  feature_data,
  id2 = "ID",
  target = NULL,
  group = "group3",
  is_target_continuous = TRUE,
  padj_cutoff = 1,
  index = 1,
  category = "signature",
  signature_group = NULL,
  ProjectID = "TCGA",
  palette_box = "nrc",
  cols_box = NULL,
  palette_corplot = "pheatmap",
  palette_heatmap = 2,
  feature_limit = 26,
```

```

character_limit = 60,
show_heatmap_col_name = FALSE,
show_col = FALSE,
show_plot = FALSE,
path = NULL,
discrete_x = 20,
discrete_width = 20,
show_palettes = FALSE,
fig.type = "pdf"
)

```

### Arguments

<code>pdata_group</code>	Data frame containing phenotype data with an identifier column.
<code>id1</code>	Character string specifying the column name in <code>'pdata_group'</code> serving as the sample identifier. Default is <code>"ID"</code> .
<code>feature_data</code>	Data frame containing feature data with corresponding identifiers.
<code>id2</code>	Character string specifying the column name in <code>'feature_data'</code> serving as the sample identifier. Default is <code>"ID"</code> .
<code>target</code>	Character string specifying the target variable column name for continuous analysis. Default is <code>'NULL'</code> .
<code>group</code>	Character string specifying the grouping variable name for categorical analysis. Default is <code>"group3"</code> .
<code>is_target_continuous</code>	Logical indicating whether the target variable is continuous, which affects grouping strategy. Default is <code>'TRUE'</code> .
<code>padj_cutoff</code>	Numeric value specifying the adjusted p-value cutoff for filtering features. Default is <code>'1'</code> .
<code>index</code>	Numeric index used for ordering output file names. Default is <code>'1'</code> .
<code>category</code>	Character string specifying the data category: <code>"signature"</code> or <code>"gene"</code> .
<code>signature_group</code>	List specifying the grouping variable for signatures. Options include <code>"sig_group"</code> for signature grouping or <code>"signature_collection"/"signature_tme"</code> for gene grouping.
<code>ProjectID</code>	Character string specifying the project identifier for file naming.
<code>palette_box</code>	Character string or integer specifying the color palette for box plots. Default is <code>"nrc"</code> .
<code>cols_box</code>	Character vector of specific colors for box plots. Default is <code>'NULL'</code> .
<code>palette_corplot</code>	Character string or integer specifying the color palette for correlation plots. Default is <code>"pheatmap"</code> .
<code>palette_heatmap</code>	Integer specifying the color palette index for heatmaps. Default is <code>'2'</code> .
<code>feature_limit</code>	Integer specifying the maximum number of features to display. Default is <code>'26'</code> .

character_limit	Integer specifying the maximum number of characters for variable labels. Default is '60'.
show_heatmap_col_name	Logical indicating whether to display column names on heatmaps. Default is 'FALSE'.
show_col	Logical indicating whether to display color codes for palettes. Default is 'FALSE'.
show_plot	Logical indicating whether to display plots. Default is 'FALSE'.
path	Character string specifying the directory path for saving output files. Default is 'NULL'.
discrete_x	Numeric threshold for character length beyond which labels are discretized. Default is '20'.
discrete_width	Numeric value specifying the width for label wrapping in plots. Default is '20'.
show_palettes	Logical indicating whether to display color palettes. Default is 'FALSE'.
fig.type	Character string specifying the format for saving figures ("pdf", "png", etc.). Default is "pdf".

**Value**

Depending on configuration, returns ggplot2 objects (box plots, heatmaps, correlation plots) and/or a data frame containing statistical analysis results.

**Author(s)**

Dongqiang Zeng

**Examples**

```
set.seed(123)

pdata_group <- data.frame(
  ID = 1:100,
  phenotype_score = rnorm(100)
)

feature_data <- data.frame(
  ID = 1:100,
  Feature1 = rnorm(100),
  Feature2 = rnorm(100),
  Feature3 = rnorm(100)
)

sig_group_example <- list(
  signature = c("Feature1", "Feature2", "Feature3")
)

results <- iobr_cor_plot(
  pdata_group = pdata_group,
  feature_data = feature_data,
```

```

    id1 = "ID",
    id2 = "ID",
    target = "phenotype_score",
    is_target_continuous = TRUE,
    category = "signature",
    signature_group = sig_group_example,
    show_plot = FALSE,
    path = tempdir()
  )

print(results)

```

---

iobr\_deconvo\_pipeline *Tumor Microenvironment (TME) Deconvolution Pipeline*

---

### Description

Executes an integrated TME analysis on a gene expression matrix: performs immune/stromal cell deconvolution using multiple algorithms, computes signature scores, and aggregates results. Designed for exploratory immunogenomic profiling.

### Usage

```

iobr_deconvo_pipeline(
  eset,
  project,
  array,
  tumor_type,
  path = NULL,
  permutation = 1000
)

```

### Arguments

eset	Numeric matrix. Gene expression (TPM/log scale) with genes in rows.
project	Character. Project name (used in output naming).
array	Logical. Whether data originated from an array platform. Affects deconvolution choices.
tumor_type	Character. Tumor type code (e.g., "stad") used by certain methods.
path	Character. Output directory. Default is NULL (uses tempdir()).
permutation	Integer. Number of permutations for CIBERSORT (and similar). Default is 1000.

### Value

Data frame integrating cell fractions and signature scores (also writes intermediate outputs to disk).

**Author(s)**

Dongqiang Zeng

**Examples**

```
eset_stad <- load_data("eset_stad")
anno_grch38 <- load_data("anno_grch38")
eset <- anno_eset(eset = eset_stad, annotation = anno_grch38, probe = "id")

res <- iobr_deconvo_pipeline(
  eset = eset, project = "STAD",
  array = FALSE, tumor_type = "stad",
  path = tempdir(), permutation = 10
)
```

---

*iobr\_deg**Differential Expression Analysis*

---

**Description**

Performs differential expression analysis on gene expression data using either DESeq2 or limma. Includes pre-processing steps like filtering low count data, and calculates fold changes and adjusted p-values. Optionally generates volcano plots and heatmaps.

**Usage**

```
iobr_deg(
  eset,
  annotation = NULL,
  id_anno = NULL,
  pdata,
  group_id = "group",
  pdata_id = "ID",
  array = FALSE,
  method = c("DESeq2", "limma"),
  contrast = c("High", "Low"),
  path = NULL,
  padj_cutoff = 0.01,
  logfc_cutoff = 0.5,
  volcano_plot = FALSE,
  col_volcano = 1,
  heatmap = TRUE,
  col_heatmap = 1,
  parallel = FALSE
)
```

**Arguments**

eset	A matrix of gene expression data where rows represent genes and columns represent samples.
annotation	Optional data frame for mapping gene IDs to gene names. Default is 'NULL'.
id_anno	Character string specifying the identifier column in annotation. Default is 'NULL'.
pdata	A data frame containing sample information and grouping labels.
group_id	Character string specifying the column name in 'pdata' containing grouping labels. Default is "group".
pdata_id	Character string specifying the column name in 'pdata' for sample IDs. Default is "ID".
array	Logical indicating whether to perform quantile normalization. Default is 'FALSE'.
method	Character string specifying the method: "DESeq2" or "limma". Default is "DESeq2".
contrast	Character vector of length 2 specifying contrast groups. Default is 'c("High", "Low")'.
path	Character string for output directory. Default is 'NULL'.
padj_cutoff	Numeric cutoff for adjusted p-values. Default is '0.01'.
logfc_cutoff	Numeric log2 fold change cutoff. Default is '0.5'.
volcano_plot	Logical indicating whether to generate a volcano plot. Default is 'FALSE'.
col_volcano	Integer specifying color index for volcano plot. Default is '1'.
heatmap	Logical indicating whether to generate a heatmap. Default is 'TRUE'.
col_heatmap	Integer specifying color index for heatmap. Default is '1'.
parallel	Logical indicating whether to run in parallel. Default is 'FALSE'.

**Value**

Data frame containing differentially expressed genes with statistics including log2 fold changes and adjusted p-values.

**Author(s)**

Dongqiang Zeng

**Examples**

```

eset_stad <- load_data("eset_stad")
stad_group <- load_data("stad_group")
deg <- iobr_deg(
  eset = eset_stad, pdata = stad_group,
  group_id = "subtype", pdata_id = "ID", array = FALSE,
  method = "DESeq2", contrast = c("EBV", "GS"),
  path = file.path(tempdir(), "STAD")
)
head(deg)

```

iobr\_pca

*Principal Component Analysis (PCA) Visualization***Description**

This function performs Principal Component Analysis (PCA) on gene expression data, reduces dimensionality while preserving variance, and generates a scatter plot visualization.

**Usage**

```
iobr_pca(
  data,
  is.matrix = TRUE,
  scale = TRUE,
  is.log = FALSE,
  pdata,
  id_pdata = "ID",
  group = NULL,
  geom.ind = "point",
  cols = "normal",
  palette = "jama",
  repel = FALSE,
  ncp = 5,
  axes = c(1, 2),
  addEllipses = TRUE
)
```

**Arguments**

<code>data</code>	Input data for PCA: matrix or data frame.
<code>is.matrix</code>	Logical indicating if input is a matrix. Default is TRUE.
<code>scale</code>	Logical indicating whether to scale the data. Default is TRUE.
<code>is.log</code>	Logical indicating whether to log-transform the data. Default is FALSE.
<code>pdata</code>	Data frame with sample IDs and grouping information.
<code>id_pdata</code>	Column name in 'pdata' for sample IDs. Default is "ID".
<code>group</code>	Column name in 'pdata' for grouping variable. Default is NULL.
<code>geom.ind</code>	Type of geometric representation for points. Default is "point".
<code>cols</code>	Color scheme for groups. Default is "normal".
<code>palette</code>	Color palette for groups. Default is "jama".
<code>repel</code>	Logical indicating whether to repel overlapping points. Default is FALSE.
<code>ncp</code>	Number of principal components to retain. Default is 5.
<code>axes</code>	Principal components to plot (e.g., c(1, 2)). Default is c(1, 2).
<code>addEllipses</code>	Logical indicating whether to add concentration ellipses. Default is TRUE.

**Value**

A ggplot object of the PCA plot.

**Author(s)**

Dongqiang Zeng

**Examples**

```
if (requireNamespace("FactoMineR", quietly = TRUE) &&
    requireNamespace("factoextra", quietly = TRUE)) {
  set.seed(123)
  eset <- matrix(rnorm(1000), nrow = 100, ncol = 10)
  rownames(eset) <- paste0("Gene", 1:100)
  colnames(eset) <- paste0("Sample", 1:10)
  pdata <- data.frame(
    ID = colnames(eset),
    group = rep(c("A", "B"), each = 5)
  )
  iobr_pca(eset, pdata = pdata, id_pdata = "ID", group = "group", addEllipses = FALSE)
}
```

---

ipsmap

*Map Score to Immunophenoscore*

---

**Description**

Maps input score to Immunophenoscore (IPS) on a 0-10 scale. Scores  $\leq 0$  map to 0, scores  $\geq 3$  map to 10, and intermediate scores are linearly scaled.

**Usage**

```
ipsmap(x)
```

**Arguments**

x                      Numeric value representing the aggregate z-score.

**Value**

Integer value between 0 and 10 representing the Immunophenoscore.

**Examples**

```
ips <- ipsmap(2.5)
ips <- ipsmap(-1)
ips <- ipsmap(5)
```

---

IPS_calculation	<i>Calculate Immunophenoscore (IPS)</i>
-----------------	---

---

### Description

Calculates Immunophenoscore (IPS) from gene expression data. IPS is a composite score measuring immunophenotype based on four major categories: MHC molecules, immunomodulators, effector cells, and suppressor cells.

### Usage

```
IPS_calculation(project = NULL, eset, plot = FALSE)
```

### Arguments

project	Character string for project identifier. Default is NULL.
eset	Gene expression matrix with official human gene symbols (HGNC) as row-names. Expression values should be $\log_2(\text{TPM}+1)$ or will be transformed if max value > 100.
plot	Logical. Whether to generate immunophenogram plots. Default is FALSE.

### Value

Data frame containing:

**MHC** MHC molecules score  
**EC** Effector cells score  
**SC** Suppressor cells score  
**CP** Checkpoints/Immunomodulators score  
**AZ** Aggregate score (sum of MHC, CP, EC, SC)  
**IPS** Immunophenoscore (0-10 scale)

### Examples

```
# IPS requires gene symbols as rownames
# Create a simple example with gene symbols
example_genes <- c(
  "HLA-A", "HLA-B", "HLA-C", "CD274", "PDCD1", "CTLA4",
  "CD8A", "CD8B", "GZMB", "PRF1", "FOXP3", "IL10"
)
sim_eset <- as.data.frame(matrix(
  rnorm(length(example_genes) * 10, mean = 5, sd = 2),
  nrow = length(example_genes), ncol = 10
))
rownames(sim_eset) <- example_genes
colnames(sim_eset) <- paste0("Sample", 1:10)
ips_result <- IPS_calculation(eset = sim_eset, project = "Example", plot = FALSE)
head(ips_result)
```

---

lasso_select	<i>Feature Selection for Predictive or Prognostic Models Using LASSO Regression</i>
--------------	---

---

### Description

Applies LASSO (Least Absolute Shrinkage and Selection Operator) regression to construct predictive or prognostic models. Supports both binary and survival response variables, utilizing cross-validation for optimal model selection.

### Usage

```
lasso_select(  
  x,  
  y,  
  type = c("binary", "survival"),  
  nfold = 10,  
  lambda = c("lambda.min", "lambda.1se")  
)
```

### Arguments

x	A numeric matrix. Features (e.g., gene symbols or CGI) as row names and samples as column names.
y	A response variable vector. Can be binary (0/1) or survival data (e.g., survival time and event status).
type	Character. Model type: "binary" for binary response or "survival" for survival analysis. Default is "binary".
nfold	Integer. Number of folds for cross-validation. Default is 10.
lambda	Character. Regularization parameter selection: "lambda.min" (minimum mean cross-validated error) or "lambda.1se" (one standard error from minimum). Default is "lambda.min".

### Value

Character vector of selected feature names with non-zero coefficients in the optimal LASSO model.

### Author(s)

Dongqiang Zeng

### Examples

```
set.seed(123)  
gene_expression <- matrix(rnorm(100 * 20), nrow = 100, ncol = 20)  
rownames(gene_expression) <- paste0("Gene", 1:100)  
colnames(gene_expression) <- paste0("Sample", 1:20)
```

```
# Binary response example
binary_outcome <- sample(c(0, 1), 20, replace = TRUE)
lasso_select(
  x = gene_expression,
  y = binary_outcome,
  type = "binary",
  nfold = 5
)
```

---

limma.dif

*Differential Expression Analysis Using Limma*

---

### Description

Performs differential expression analysis using the limma package on a given gene expression dataset. Constructs a design matrix from phenotype data, fits a linear model, applies contrasts, and computes statistics for differential expression.

### Usage

```
limma.dif(exprdata, pdata, contrastfml)
```

### Arguments

exprdata	A matrix with rownames as features like gene symbols or cgi, and colnames as samples.
pdata	A two-column dataframe where the first column matches the colnames of exprdata and the second column contains the grouping variable.
contrastfml	A character vector for contrasts to be tested (see ?makeContrasts for more details).

### Value

Returns a dataframe from `limma::topTable`, which includes genes as rows and columns like `genelist`, `logFC`, `AveExpr`, etc.

### Examples

```
# Toy example with 100 genes and 6 samples
set.seed(123)
exprdata <- matrix(
  rnorm(100 * 6),
  nrow = 100,
  ncol = 6,
  dimnames = list(
    paste0("gene", 1:100),
    paste0("sample", 1:6)
  )
)
```

```
)  
)  
  
# Phenotype data: 3 vs 3  
pdata <- data.frame(  
  sample = colnames(exprdata),  
  group = rep(c("group1", "group2"), each = 3),  
  stringsAsFactors = FALSE  
)  
  
# Differential expression: group1 vs group2  
res <- limma.dif(  
  exprdata = exprdata,  
  pdata = pdata,  
  contrastfml = "group1 - group2"  
)  
head(res)
```

---

list\_github\_datasets    *List Available GitHub Datasets*

---

**Description**

List Available GitHub Datasets

**Usage**

```
list_github_datasets()
```

**Value**

Character vector of available dataset names.

**Examples**

```
list_github_datasets()
```

---

list\_iobr\_mirrors    *List Current Download Mirrors*

---

**Description**

Returns the current list of download mirrors.

**Usage**

```
list_iobr_mirrors()
```

**Value**

Character vector of mirror URLs.

**Examples**

```
list_ioibr_mirrors()
```

---

load_data	<i>Load IOBR Datasets</i>
-----------	---------------------------

---

**Description**

Loads internal datasets from the IOBR package. Supports both sysdata (internal) and exported data files included in the package.

**Usage**

```
load_data(name)
```

**Arguments**

name	Character string. Name of the dataset to load. Must be a single value. Available datasets include: - Expression data: "eset_stad", "imvigor210_eset", "melanoma_data" - Signatures: "signature_tme", "signature_metabolism", "signature_collection" - Gene sets: "hallmark", "kegg", "go_bp", "go_cc", "go_mf" - Cell markers: "cellmarkers", "mcp_genes" - Phenotype data: "pdata_stad", "pdata_sig_tme", "pdata_acrg" - Reference data: "xCell.data", "quantiseq_data", "TRef", "BRef" - Color palettes: "palette1", "palette2", "palette3", "palette4"
------	---

**Value**

Dataset object, typically a 'list', 'data.frame', or 'matrix'. The exact type depends on the requested dataset.

**Examples**

```
# Load signature collection
sig_tme <- load_data("signature_tme")

# Load expression data
eset <- load_data("eset_stad")

# Load color palette
colors <- load_data("palette1")

# Error handling with suggestions for similar names
try(load_data("sign_tme")) # Will suggest "signature_tme"
```

---

log2eset	<i>Log2 Transformation of Gene Expression Matrix</i>
----------	--

---

**Description**

Determines whether a gene expression matrix requires log2 transformation based on the distribution of values, and applies it if necessary. This is useful for automatically detecting raw counts or linear-scale data that should be log-transformed for downstream analysis.

**Usage**

```
log2eset(eset)
```

**Arguments**

eset            Numeric matrix. Gene expression data with genes as rows and samples in columns.

**Value**

Numeric matrix. Log2-transformed gene expression data (if transformation was needed), or the original data otherwise.

**Examples**

```
set.seed(123)
eset <- matrix(rnorm(1000, mean = 10, sd = 2), nrow = 100, ncol = 10)
rownames(eset) <- paste0("Gene", 1:100)
colnames(eset) <- paste0("Sample", 1:10)
eset_transformed <- log2eset(eset)
```

---

LR_cal	<i>Calculate Ligand-Receptor Interaction Scores</i>
--------	---

---

**Description**

Quantifies ligand-receptor interactions in the tumor microenvironment from bulk gene expression data using the easier package. This function processes raw counts or TPM data and computes interaction scores for each sample.

**Usage**

```
LR_cal(  
  eset,  
  data_type = c("count", "tpm"),  
  id_type = "ensembl",  
  cancer_type = "pancan"  
)
```

**Arguments**

eset	Gene expression matrix with genes as rows and samples as columns.
data_type	Type of input data. Options are "count" or "tpm". If "count", data will be converted to TPM before analysis.
id_type	Type of gene identifier. Default is "ensembl".
cancer_type	Character string specifying the cancer type for easier. Default is "pancan" for pan-cancer analysis.

**Value**

Data frame containing ligand-receptor interaction scores with sample IDs as row names.

**References**

Lapunte-Santana, van Genderen, M., Hilbers, P., Finotello, F., & Eduati, F. (2021). Interpretable systems biomarkers predict response to immune-checkpoint inhibitors. *Patterns (New York, N.Y.)*, 2(8), 100293. <https://doi.org/10.1016/j.patter.2021.100293>

**Examples**

```
# LR_cal requires HGNC gene symbols as rownames
# Create a simple example with gene symbols
example_genes <- c(
  "TGFB1", "EGFR", "VEGFA", "PDGFB", "FGF2", "CXCL12",
  "CXCR4", "IL6", "IL6R", "TNF", "TNFRSF1A", "IFNG"
)
sim_eset <- as.data.frame(matrix(
  rnorm(length(example_genes) * 10, mean = 5, sd = 2),
  nrow = length(example_genes), ncol = 10
))
rownames(sim_eset) <- example_genes
colnames(sim_eset) <- paste0("Sample", 1:10)

if (requireNamespace("easier", quietly = TRUE)) {
  lr <- LR_cal(eset = sim_eset, data_type = "tpm")
  head(lr)
}
```

---

make\_mut\_matrix

*Construct Mutation Matrices from MAF Data*


---

**Description**

Builds mutation presence/absence matrices from MAF input (file path or MAF object). Supports multiple categories: all mutations, SNPs, indels, and frameshift mutations. When category = "multi", returns a list of matrices for each category. Compatible with TCGA-formatted data.

**Usage**

```
make_mut_matrix(
  maf = NULL,
  mut_data = NULL,
  isTCGA = TRUE,
  category = c("multi", "all", "snp", "indel", "frameshift"),
  Tumor_Sample_Barcode = "Tumor_Sample_Barcode",
  Hugo_Symbol = "Hugo_Symbol",
  Variant_Classification = "Variant_Classification",
  Variant_Type = "Variant_Type"
)
```

**Arguments**

maf	Character or MAF object. Path to MAF file or an already loaded MAF object.
mut_data	Data frame or NULL. Preloaded MAF-like data (used if 'maf' is NULL).
isTCGA	Logical. Whether the MAF follows TCGA conventions. Default is TRUE.
category	Character. Mutation category: "all", "snp", "indel", "frameshift", or "multi". Default is "multi".
Tumor_Sample_Barcode	Character. Column name for tumor sample IDs. Default is "Tumor_Sample_Barcode".
Hugo_Symbol	Character. Column name for gene symbols. Default is "Hugo_Symbol".
Variant_Classification	Character. Column name for variant classification (e.g., Frame_Shift_Del). Default is "Variant_Classification".
Variant_Type	Character. Column name for variant type (e.g., SNP, INS, DEL). Default is "Variant_Type".

**Value**

List of mutation matrices (if category = "multi") or a single matrix for the specified category.

**Note**

Some users may encounter errors from upstream data import (e.g. "Can't combine ..\$Tumor\_Seq\_Allele2" when using TCGAAbiolinks or TCGAmutations). This is due to inconsistent column types in the source MAF tables, not an issue of this function. Please ensure your MAF or merged data frame uses consistent column types (e.g. convert allele columns to character before input).

**Author(s)**

Dongqian Zeng

Shixiang Huang

## Examples

```
## Not run:  
# See maftools or TCGAbiolinks documentation for obtaining MAF input  
# Example: Download MAF file from TCGA portal  
mut_list <- make_mut_matrix(maf = "path_to_maf_file.maf", isTCGA = TRUE, category = "multi")  
  
## End(Not run)
```

---

mapbw

*Map Score to Black and White Color*

---

## Description

Maps a numeric input value to a color from a black-white gradient palette. Values are mapped to a 1001-color palette where -2 maps to black and +2 maps to white.

## Usage

```
mapbw(x, my_palette2 = NULL)
```

## Arguments

x	Numeric value to be mapped to a color (typically between -2 and 2).
my_palette2	Color palette vector (should have 1001 colors). Default uses black-white gradient.

## Value

A color from the black-white palette as a hex code.

## Examples

```
my_palette2 <- grDevices::colorRampPalette(c("black", "white"))(1001)  
color <- mapbw(1.5, my_palette2)  
color <- mapbw(-1, my_palette2)
```

---

mapcolors	<i>Map Score to Color</i>
-----------	---------------------------

---

**Description**

Maps a numeric input value to a color from a blue-white-red gradient palette. Values are mapped to a 1001-color palette where -3 maps to blue, 0 maps to white, and +3 maps to red.

**Usage**

```
mapcolors(x, my_palette = NULL)
```

**Arguments**

x	Numeric value to be mapped to a color (typically between -3 and 3).
my_palette	Color palette vector (should have 1001 colors). Default uses blue-white-red gradient.

**Value**

A color from the palette as a hex code.

**Examples**

```
my_palette <- grDevices::colorRampPalette(c("blue", "white", "red"))(1001)
color <- mapcolors(2, my_palette)
color <- mapcolors(-2, my_palette)
```

---

MCPcounter.estimate	<i>MCP-counter Cell Population Abundance Estimation</i>
---------------------	---

---

**Description**

Estimates the abundance of different immune and stromal cell populations using the MCP-counter method. Works with various gene identifiers including Affymetrix probesets, HUGO gene symbols, Entrez IDs, and Ensembl IDs.

**Usage**

```
MCPcounter.estimate(
  expression,
  featuresType = c("affy133P2_probesets", "HUGO_symbols", "ENTREZ_ID", "ENSEMBL_ID"),
  probesets = read.table(system.file("extdata/probesets.txt", package = "IOBR"), sep =
    "\t", stringsAsFactors = FALSE, colClasses = "character"),
  genes = read.table(system.file("extdata/genes.txt", package = "IOBR"), sep = "\t",
    stringsAsFactors = FALSE, header = TRUE, colClasses = "character", check.names =
    FALSE)
)
```

**Arguments**

expression	Matrix or data.frame with features in rows and samples in columns.
featuresType	Type of identifiers for expression features. Options: "affy133P2_probesets", "HUGO_symbols", "ENTREZ_ID", "ENSEMBL_ID". Default is "affy133P2_probesets".
probesets	Probesets data table. Default loads from GitHub.
genes	Genes data table. Default loads from GitHub.

**Value**

Matrix with cell populations in rows and samples in columns.

**Author(s)**

Etienne Becht

**Examples**

```
expr <- matrix(runif(1000), nrow = 100, ncol = 10)
rownames(expr) <- paste0("Gene", 1:100)
estimates <- MCPcounter.estimate(expr, featuresType = "HUGO_symbols")
```

---

merge\_duplicate

*Merge Data Frames with Duplicated Column Names*

---

**Description**

Merges two data frames, resolving duplicated column names according to user preference. Allows selection of which data frame's duplicated columns to retain, ensuring data integrity during merging.

**Usage**

```
merge_duplicate(
  x,
  y,
  by.x,
  by.y,
  all.x = FALSE,
  all.y = FALSE,
  all = NULL,
  choose = c("x", "y")
)
```

**Arguments**

x	Data frame. First data frame to merge.
y	Data frame. Second data frame to merge.
by.x	Character. Column name(s) in 'x' used for merging.
by.y	Character. Column name(s) in 'y' used for merging.
all.x	Logical. Include all rows from 'x' in output. Default is 'FALSE'.
all.y	Logical. Include all rows from 'y' in output. Default is 'FALSE'.
all	Logical or 'NULL'. If not 'NULL', include all rows from both 'x' and 'y', overriding 'all.x' and 'all.y'.
choose	Character. Which data frame's duplicated non-joining columns to retain: "x" or "y". Default is "x".

**Value**

Data frame resulting from merging 'x' and 'y' according to specified parameters.

**Examples**

```
df1 <- data.frame(ID = 1:3, Name = c("A", "B", "C"), Value = 1:3)
df2 <- data.frame(ID = 1:3, Name = c("X", "Y", "Z"), Score = 4:6)

# Merge keeping duplicated columns from x
merged_df <- merge_duplicate(df1, df2,
  by.x = "ID", by.y = "ID",
  all.x = TRUE, choose = "x"
)
print(merged_df)

# Merge keeping duplicated columns from y
merged_df2 <- merge_duplicate(df1, df2,
  by.x = "ID", by.y = "ID",
  all = TRUE, choose = "y"
)
```

---

merge\_eset

*Merge Expression Sets by Row Names*


---

**Description**

Merges two or three expression sets (matrices or data frames) by row names (gene symbols), removing duplicates. The function ensures common genes across all input expression sets are retained.

**Usage**

```
merge_eset(eset1, eset2, eset3 = NULL)
```

**Arguments**

eset1            First expression set (matrix or data frame with row names).  
eset2            Second expression set (matrix or data frame with row names).  
eset3            Optional third expression set. Default is 'NULL'.

**Value**

Merged expression set (data frame) with duplicates removed. Row names correspond to gene symbols.

**Author(s)**

Dongqiang Zeng

**Examples**

```
# Load example data
eset_stad <- load_data("eset_stad")

# Create mock expression sets with common genes
common_genes <- c("TP53", "BRCA1", "EGFR", "MYC")
eset1 <- matrix(rnorm(12),
  nrow = 4,
  dimnames = list(common_genes, paste0("S", 1:3))
)
eset2 <- matrix(rnorm(16),
  nrow = 4,
  dimnames = list(common_genes, paste0("S", 4:7))
)

# Merge two expression sets
merged_eset <- merge_eset(eset1, eset2)
print(dim(merged_eset))
```

---

mouse2human\_eset

*Convert Mouse Gene Symbols to Human Gene Symbols*

---

**Description**

Converts mouse gene symbols to human gene symbols in an expression dataset. Supports using either an online resource (Ensembl) or a local dataset for conversion.

**Usage**

```
mouse2human_eset(
  eset,
  source = c("local", "ensembl"),
  is_matrix = TRUE,
```

```

    column_of_symbol = NULL,
    verbose = FALSE
  )

```

### Arguments

eset	Matrix or data frame. Expression matrix with genes in rows.
source	Character. Data source for conversion: "ensembl" (online) or "local". Default is "ensembl". If Ensembl fails, use "local" which uses the internal 'mus_human_gene_symbol' dataset.
is_matrix	Logical. Whether 'eset' is a matrix with gene symbols as row names. Default is 'TRUE'. If 'FALSE', 'column_of_symbol' must be specified.
column_of_symbol	Character or 'NULL'. Column name containing gene symbols if 'eset' is not a matrix. Default is 'NULL'.
verbose	Logical. If 'TRUE', prints available Ensembl datasets. Default is 'FALSE'.

### Value

Expression set with human gene symbols.

### Author(s)

Dongqiang Zeng

### Examples

```

# Create example mouse expression data
anno_gc_vm32 <- load_data("anno_gc_vm32")
num_rows <- 200
num_cols <- 10
sample_names <- paste0("Sample", 1:num_cols)
data <- matrix(runif(num_rows * num_cols), nrow = num_rows, ncol = num_cols)
rownames(data) <- anno_gc_vm32$symbol[1:200]
colnames(data) <- sample_names

# Convert using local database
human_data <- mouse2human_eset(data, source = "local", is_matrix = TRUE)

```

---

null\_models

*NULL Model Coefficients for MCPcounter*

---

### Description

NULL Model Coefficients for MCPcounter

**Usage**

```
data(null_models)
```

**Format**

A 'data.frame' with cell types in rows and coefficients in columns.

**Examples**

```
data(null_models)
head(null_models)
```

---

outputGCT

*outputGCT*

---

**Description**

This function converts a gene expression dataset to a GCT format file.

**Usage**

```
outputGCT(input.f, output.f)
```

**Arguments**

`input.f` A data frame or a character string specifying the path to the input file. If a character string, the file should be a tab-separated table with row names.

`output.f` A character string specifying the path to the output file.

**Value**

No return value. The function writes the dataset to the specified output file in GCT format.

**Examples**

```
# Create a sample input data frame
sample_data <- data.frame(
  Gene = c("BRCA1", "TP53", "EGFR"),
  Sample1 = c(10, 15, 8),
  Sample2 = c(12, 18, 7),
  stringsAsFactors = FALSE
)
rownames(sample_data) <- sample_data$Gene
sample_data <- sample_data[, -1]

# Convert the input data frame to GCT format and save to temporary file
output_file <- tempfile(fileext = ".gct")
outputGCT(sample_data, output_file)
```

---

output_sig	<i>Save Signature Data to File</i>
------------	------------------------------------

---

### Description

Saves signature data to a specified file format, supporting CSV or RData. Handles single signatures or lists of signatures, converting them to a data frame for storage.

### Usage

```
output_sig(signatures, format = c("csv", "rdata"), file.name)
```

### Arguments

signatures	Signature data: a list or single string of signatures.
format	Character. Output format: "csv" or "rdata". Default is "csv".
file.name	Character. Name of the output file without extension.

### Value

Data frame containing the processed signature data, also saved to the specified file.

### Author(s)

Dongqiang Zeng

### Examples

```
signature_collection <- load_data("signature_collection")
tmpfile <- tempfile(fileext = ".csv")
output_sig(
  signatures = signature_collection, format = "csv",
  file.name = tools::file_path_sans_ext(tmpfile)
)
```

---

palettes	<i>Select Color Palettes for Visualization</i>
----------	--

---

### Description

Provides curated qualitative, sequential, and diverging palettes for multiple plot types. Supports intensity adjustment and preview.

**Usage**

```
palettes(  
  category = "box",  
  palette = "nrc",  
  alpha = 1,  
  counts = 50,  
  show_col = TRUE,  
  show_message = FALSE  
)
```

**Arguments**

category	Character. Plot/palette category: one of 'box', 'continue2', 'continue', 'random', 'heatmap', 'heatmap3', 'tidyheatmap'.
palette	Character or numeric. Palette name or index (varies by category).
alpha	Numeric. Alpha (transparency) scaling factor. Default is 1.
counts	Integer. Number of colors (for continuous palettes). Default is 50.
show_col	Logical. If TRUE, prints the palette. Default is TRUE.
show_message	Logical. If TRUE, prints available options. Default is FALSE.

**Value**

Character vector of hex color codes.

**Author(s)**

Dongqiang Zeng

**Examples**

```
colors <- palettes(category = "box", palette = "nrc", show_col = FALSE)  
heatmap_colors <- palettes(  
  category = "heatmap", palette = 1, counts = 10, show_col = FALSE  
)
```

---

parallel\_doperm

*Parallel Permutation Test for CIBERSORT*

---

**Description**

Parallel version of doPerm. Performs permutation-based sampling and runs the CoreAlg function iteratively using multiple CPU cores to accelerate computation. This function generates an empirical null distribution of correlation coefficients for p-value calculation in CIBERSORT analysis.

**Usage**

```
parallel_doperm(
  perm1,
  X1,
  Y1,
  absolute1,
  abs_method1,
  num_cores1 = 2,
  seed = NULL
)
```

**Arguments**

perm1	Integer. Number of permutations to perform ( $\geq 100$ recommended).
X1	Matrix or data frame. Signature matrix (cell type GEP barcode).
Y1	Matrix. Mixture file containing gene expression profiles.
absolute1	Logical. Whether to run in absolute mode (default: FALSE).
abs_method1	String. Method for absolute mode: 'sig.score' or 'no.sumto1'.
num_cores1	Integer. Number of CPU cores for parallel computation (default: 2).
seed	Integer. Random seed for reproducibility. If NULL (default), uses current random state. Set to a specific value (e.g., 123) for reproducible results across runs.

**Details**

This function utilizes the `foreach` and `doParallel` packages to distribute permutation iterations across multiple cores. It automatically handles cluster setup/teardown via `on.exit()` to prevent resource leaks.

Note: Windows users may experience slower performance due to socket-based parallelization (PSOCK) versus forking on Unix systems.

**Value**

List containing:

**dist** Numeric vector of correlation coefficients from permutations, representing the empirical null distribution.

**See Also**

[doPerm](#) for the sequential version, [CoreAlg](#), [CIBERSORT](#)

**Examples**

```
X <- matrix(rnorm(1000), nrow = 100)
Y <- matrix(rnorm(500), nrow = 100)
rownames(X) <- rownames(Y) <- paste0("Gene", 1:100)

result <- parallel_doperm(
```

```
perm1 = 100, X1 = X, Y1 = Y,  
absolute1 = FALSE, abs_method1 = "sig.score", num_cores1 = 2  
)  
str(result$dist)
```

---

ParseInputExpression *Parse Input Gene Expression Data*

---

### Description

Reads gene expression data from a tab-delimited text file, using the first column as row names. Converts data into a numeric matrix for analysis.

### Usage

```
ParseInputExpression(path)
```

### Arguments

path                    Character. Path to a tab-delimited gene expression file. First column should contain gene identifiers.

### Value

Numeric matrix of gene expression values with genes as rows and samples as columns.

### Examples

```
tf <- tempfile(fileext = ".tsv")  
expr <- data.frame(  
  gene = c("GeneA", "GeneB", "GeneC"),  
  Sample1 = c(10, 20, 30),  
  Sample2 = c(15, 25, 35)  
)  
write.table(expr, tf, sep = "\t", row.names = FALSE, quote = FALSE)  
gene_expression_data <- ParseInputExpression(tf)  
print(gene_expression_data)
```

---

patterns_to_na	<i>Default Pattern List for Name Cleaning</i>
----------------	---

---

**Description**

A character vector of common substrings to remove from feature names. Used in [remove\_names()] and other helper functions.

**Usage**

```
patterns_to_na
```

**Format**

A character vector of length 12.

**Value**

Character vector of patterns to remove.

**Examples**

```
# View default patterns
patterns_to_na
```

---

pdata_stad	<i>Toy STAD Phenotype Data</i>
------------	--------------------------------

---

**Description**

A data frame containing clinical and pathological annotations for the TCGA stomach adenocarcinoma (STAD) cohort. Each row corresponds to one tumour sample and can be matched to the columns of eset\_stad via the ID column. This dataset is typically used together with eset\_stad in examples of survival analysis, subgroup comparison and immune deconvolution in the IOBR package.

**Usage**

```
data(pdata_stad)
```

**Format**

A data frame with one row per TCGA-STAD sample and 8 variables:

**ID** Character. TCGA sample barcode, matching the column names of eset\_stad.

**stage** Factor. Pathological stage (e.g. "Stage\_I", "Stage\_II", "Stage\_III", "Stage\_IV").

**status** Factor. Vital status at last follow-up ("Alive" or "Dead").

**Lauren** Factor. Lauren classification of gastric cancer ("Intestinal", "Diffuse", "Mixed" or NA).

**subtype** Factor. Molecular subtype (e.g. "CIN", "EBV", "GS", "MSI").

**EBV** Factor. EBV status of the tumour ("Positive" or "Negative").

**time** Numeric. Overall survival or follow-up time, typically measured in months.

**OS\_status** Integer/binary. Overall survival status indicator. (1 = death, 0 = censored)

**Examples**

```
data(pdata_stad)
head(pdata_stad)
```

---

percent\_bar\_plot      *Create a Percent Bar Plot*

---

**Description**

Generates a bar plot visualizing the percentage distribution of a variable grouped by another variable.

**Usage**

```
percent_bar_plot(
  input,
  x,
  y,
  subset.x = NULL,
  color = NULL,
  palette = NULL,
  title = NULL,
  axis_angle = 0,
  coord_flip = FALSE,
  add_freq = TRUE,
  freq = NULL,
  size_freq = 8,
  legend.size = 0.5,
  legend.size.text = 10,
  add_sum = TRUE,
  print_result = TRUE,
  round.num = 2
)
```

**Arguments**

<code>input</code>	Input data frame.
<code>x</code>	Name of the x-axis variable.
<code>y</code>	Name of the y-axis (grouping) variable.
<code>subset.x</code>	Optional subset of x-axis values.
<code>color</code>	Optional color palette.
<code>palette</code>	Optional palette type.
<code>title</code>	Optional plot title.
<code>axis_angle</code>	Angle for axis labels (0-90). Default is 0.
<code>coord_flip</code>	Logical to flip coordinates. Default is FALSE.
<code>add_freq</code>	Logical to add frequency count. Default is TRUE.
<code>freq</code>	Name of frequency column.
<code>size_freq</code>	Size of frequency labels. Default is 8.
<code>legend.size</code>	Size of legend. Default is 0.5.
<code>legend.size.text</code>	Size of legend text. Default is 10.
<code>add_sum</code>	Logical to add sum to x-axis labels. Default is TRUE.
<code>print_result</code>	Logical to print result data frame. Default is TRUE.
<code>round.num</code>	Decimal places for proportion. Default is 2.

**Value**

A ggplot object.

**Author(s)**

Dongqiang Zeng

**Examples**

```
sig_stad <- load_data("sig_stad")
percent_bar_plot(
  input = sig_stad, x = "Subtype", y = "Lauren",
  axis_angle = 60
)
```

---

`pie_chart`*Create Pie or Donut Charts*

---

**Description**

Generates a pie chart or donut chart from input data.

**Usage**

```
pie_chart(  
  input,  
  var,  
  var2 = NULL,  
  type = 2,  
  show_freq = FALSE,  
  color = NULL,  
  palette = "jama",  
  title = NULL,  
  text_size = 10,  
  title_size = 20,  
  add_sum = FALSE  
)
```

**Arguments**

<code>input</code>	Input dataframe.
<code>var</code>	Variable for the chart.
<code>var2</code>	Secondary variable for donut chart (type = 3).
<code>type</code>	Chart type: 1 (pie), 2 (donut), 3 (PieDonut via webr).
<code>show_freq</code>	Logical to show frequencies. Default is FALSE.
<code>color</code>	Optional color palette.
<code>palette</code>	Color palette name. Default is "jama".
<code>title</code>	Plot title. Default is NULL.
<code>text_size</code>	Text size. Default is 10.
<code>title_size</code>	Title size. Default is 20.
<code>add_sum</code>	Logical to add sum to labels. Default is FALSE.

**Value**

A ggplot object.

**Author(s)**

Dongqiang Zeng

## Examples

```
sig_stad <- load_data("sig_stad")
pie_chart(input = sig_stad, var = "Subtype", palette = "jama")
pie_chart(input = sig_stad, var = "Subtype", type = 2)
```

---

PlotAUC

*Plot AUC ROC Curves*

---

## Description

Generates ROC curves for model evaluation comparing training and testing performance at both  $\lambda_{\min}$  and  $\lambda_{1se}$ . Creates a ggplot visualization with AUC values in the legend.

## Usage

```
PlotAUC(
  train.x,
  train.y,
  test.x,
  test.y,
  model,
  modelname,
  cols = NULL,
  palette = "jama"
)
```

## Arguments

<code>train.x</code>	Training predictors matrix.
<code>train.y</code>	Training outcomes (binary factor).
<code>test.x</code>	Testing predictors matrix.
<code>test.y</code>	Testing outcomes (binary factor).
<code>model</code>	Fitted <code>cv.glmnet</code> model.
<code>modelname</code>	Character string for plot title.
<code>cols</code>	Optional color vector for ROC curves.
<code>palette</code>	Color palette name from IOBR palettes. Default is "jama".

## Value

ggplot object of ROC curves.

**Examples**

```
if (requireNamespace("glmnet", quietly = TRUE)) {
  set.seed(123)
  train_data <- matrix(rnorm(100 * 5), ncol = 5)
  train_outcome <- rbinom(100, 1, 0.5)
  test_data <- matrix(rnorm(50 * 5), ncol = 5)
  test_outcome <- rbinom(50, 1, 0.5)
  fitted_model <- glmnet::cv.glmnet(train_data, train_outcome, family = "binomial", nfolds = 5)
  p <- PlotAUC(train_data, train_outcome, test_data, test_outcome, fitted_model, "MyModel")
  print(p)
}
```

---

plotPurity

*plotPurity*

---

**Description**

This function generates scatterplots of tumor purity based on ESTIMATE scores for given samples.

**Usage**

```
plotPurity(
  scores,
  samples = "all_samples",
  platform = c("affymetrix", "agilent", "illumina"),
  output.dir = NULL
)
```

**Arguments**

scores	A character string specifying the path to the input file containing ESTIMATE scores. The file should be a tab-separated table with appropriate headers.
samples	A character vector specifying the sample names to plot. The default is "all_samples", which plots all samples in the input file.
platform	A character string specifying the platform used for data collection. Can be "affymetrix", "agilent", or "illumina". Currently, only "affymetrix" is implemented.
output.dir	A character string specifying the directory to save the output plots. If 'NULL', plots are not saved. Default is 'NULL'.

**Value**

No return value. The function generates and saves scatterplots in the specified output directory.

**Examples**

```
# Create a sample ESTIMATE score matrix
scores_data <- data.frame(
  Sample1 = c(100, 200, 500, 0.80),
  Sample2 = c(120, 220, 450, 0.70),
  Sample3 = c(140, 240, 600, 0.90),
  row.names = c(
    "StromalScore", "ImmuneScore", "ESTIMATEScore",
    "TumorPurity"
  ),
  check.names = FALSE
)

# Write to a temporary GCT file
scores_file <- tempfile(fileext = ".gct")
outputGCT(scores_data, scores_file)
```

---

PlotTimeROC

*Plot Time-Dependent ROC Curves*


---

**Description**

Generates time-dependent ROC curves for evaluating prognostic accuracy of survival models. Plots training and testing ROC curves at the 90th percentile survival time.

**Usage**

```
PlotTimeROC(
  train.x,
  train.y,
  test.x,
  test.y,
  model,
  modelname,
  cols = NULL,
  palette = "jama"
)
```

**Arguments**

train.x	Matrix or data frame of training predictors.
train.y	Training survival outcomes (time and status).
test.x	Matrix or data frame of testing predictors.
test.y	Testing survival outcomes (time and status).
model	Fitted survival model object.
modelname	Character string for model identification.
cols	Optional vector of colors for plotting.
palette	Character string specifying color palette. Default is "jama".

**Value**

A ‘ggplot’ object representing the ROC curve plot.

**Author(s)**

Dongqiang Zeng

**Examples**

```
if (requireNamespace("glmnet", quietly = TRUE) &&
    requireNamespace("survival", quietly = TRUE) &&
    requireNamespace("timeROC", quietly = TRUE)) {
  library(survival)
  set.seed(123)
  train_x <- matrix(rnorm(100 * 5), ncol = 5)
  train_y <- data.frame(time = rexp(100), status = rbinom(100, 1, 0.5))
  test_x <- matrix(rnorm(50 * 5), ncol = 5)
  test_y <- data.frame(time = rexp(50), status = rbinom(50, 1, 0.5))
  fit <- glmnet::cv.glmnet(train_x, Surv(train_y$time, train_y$status), family = "cox")
  p <- PlotTimeROC(train_x, train_y, test_x, test_y, fit, "Cox Model")
  print(p)
}
```

---

ProcessingData

*Process Data for Model Construction*

---

**Description**

Preprocesses data for binomial or survival analysis. Aligns and filters data based on sample IDs, optionally scales data, and ensures appropriate data types. Handles missing values by removing columns with NA values.

**Usage**

```
ProcessingData(x, y, scale, type = c("binomial", "survival"))
```

**Arguments**

x	Data frame of predictors with first column as IDs.
y	Data frame of outcomes with first column as IDs. For survival, expects two additional columns for time and status.
scale	Logical indicating whether to scale predictors.
type	Character string: "binomial" or "survival".

**Value**

List containing:

**x\_scale** Processed predictor matrix

**y** Processed outcome variable

**x\_ID** Sample IDs

**Examples**

```
x <- data.frame(ID = 1:10, predictor1 = rnorm(10), predictor2 = rnorm(10))
y <- data.frame(ID = 1:10, outcome = sample(c(0, 1), 10, replace = TRUE))
result <- ProcessingData(x, y, scale = TRUE, type = "binomial")
```

---

 PrognosticAUC

*Calculate Time-Dependent AUC for Survival Models*


---

**Description**

Evaluates prognostic ability of a survival model by calculating time-dependent AUC at the 30th and 90th percentiles of survival time. These thresholds assess short-term and long-term predictive accuracy.

**Usage**

```
PrognosticAUC(model, newx, s, acture.y)
```

**Arguments**

<code>model</code>	A fitted survival model object capable of generating risk scores.
<code>newx</code>	A matrix or data frame of new data for prediction.
<code>s</code>	Lambda value for prediction. Can be numeric or <code>"lambda.min"/"lambda.1se"</code> .
<code>acture.y</code>	Data frame with <code>'time'</code> and <code>'status'</code> columns.

**Value**

A data frame with AUC values at 30th (`'probs.3'`) and 90th (`'probs.9'`) percentiles.

**Author(s)**

Dongqiang Zeng

**Examples**

```

if (requireNamespace("glmnet", quietly = TRUE) &&
    requireNamespace("survival", quietly = TRUE) &&
    requireNamespace("timeROC", quietly = TRUE)) {
  library(survival)
  set.seed(123)
  x <- matrix(rnorm(100 * 5), ncol = 5)
  y <- Surv(rexp(100), rbinom(100, 1, 0.5))
  fit <- glmnet::cv.glmnet(x, y, family = "cox")
  acture_y <- data.frame(time = y[, 1], status = y[, 2])
  auc_results <- PrognosticAUC(fit, newx = x, s = "lambda.min", acture.y = acture_y)
}

```

PrognosticModel

*Build Prognostic Models Using LASSO and Ridge Regression***Description**

Prepares data, splits it into training and testing sets, and fits LASSO and Ridge regression models for survival analysis. Evaluates model performance using cross-validation and optionally generates time-dependent ROC curves for visual assessment of predictive accuracy.

**Usage**

```

PrognosticModel(
  x,
  y,
  scale = FALSE,
  seed = 123456,
  train_ratio = 0.7,
  nfold = 10,
  plot = TRUE,
  palette = "jama",
  cols = NULL
)

```

**Arguments**

x	A matrix or data frame of predictor variables (features).
y	A data frame of survival outcomes with two columns: survival time and event status.
scale	Logical indicating whether to scale predictor variables. Default is 'FALSE'.
seed	Integer seed for random number generation to ensure reproducibility. Default is '123456'.
train_ratio	Numeric proportion of data for training (e.g., 0.7). Default is '0.7'.
nfold	Integer number of folds for cross-validation. Default is '10'.

<code>plot</code>	Logical indicating whether to plot ROC curves. Default is 'TRUE'.
<code>palette</code>	String specifying color palette. Default is "jama".
<code>cols</code>	Optional vector of colors for ROC curves. If 'NULL', uses default palette.

**Value**

A list containing:

**lasso\_result** Results from LASSO model including coefficients and AUC

**ridge\_result** Results from Ridge model including coefficients and AUC

**train.x** Training data with sample IDs

**Author(s)**

Dongqiang Zeng

**Examples**

```
if (requireNamespace("glmnet", quietly = TRUE) &&
    requireNamespace("survival", quietly = TRUE)) {
  library(survival)
  imvigor210_sig <- load_data("imvigor210_sig")
  imvigor210_pdata <- load_data("imvigor210_pdata")
  pdata_prog <- data.frame(
    ID = imvigor210_pdata$ID,
    OS_days = as.numeric(imvigor210_pdata$OS_days),
    OS_status = as.numeric(imvigor210_pdata$OS_status)
  )
  prognostic_result <- PrognosticModel(
    x = imvigor210_sig, y = pdata_prog,
    scale = TRUE, seed = 123456,
    train_ratio = 0.7, nfold = 10, plot = FALSE
  )
}
```

---

PrognosticResult

*Compute Prognostic Results for Survival Models*

---

**Description**

Computes and compiles prognostic results from a survival model fitted with 'glmnet'. Extracts model coefficients at optimal lambda values ('lambda.min' and 'lambda.1se') and calculates time-dependent AUC metrics for both training and testing datasets.

**Usage**

```
PrognosticResult(model, train.x, train.y, test.x, test.y)
```

**Arguments**

<code>model</code>	A fitted survival model object (e.g., from <code>'glmnet::cv.glmnet'</code> ).
<code>train.x</code>	Matrix or data frame of training predictors.
<code>train.y</code>	Training dataset survival outcomes (time and status).
<code>test.x</code>	Matrix or data frame of testing predictors.
<code>test.y</code>	Testing dataset survival outcomes (time and status).

**Value**

A list containing:

**model** The fitted model object

**coefs** Data frame of coefficients at `'lambda.min'` and `'lambda.1se'`

**AUC** Data frame with AUC values for train/test at both lambda values

**Author(s)**

Dongqiang Zeng

**Examples**

```
if (requireNamespace("glmnet", quietly = TRUE) &&
    requireNamespace("survival", quietly = TRUE) &&
    requireNamespace("timeROC", quietly = TRUE)) {
  library(survival)
  set.seed(123)
  train_x <- matrix(rnorm(100 * 10), ncol = 10)
  train_y <- data.frame(time = rexp(100), status = rbinom(100, 1, 0.5))
  test_x <- matrix(rnorm(50 * 10), ncol = 10)
  test_y <- data.frame(time = rexp(50), status = rbinom(50, 1, 0.5))
  fit <- glmnet::cv.glmnet(train_x, Surv(train_y$time, train_y$status), family = "cox")
  results <- PrognosticResult(
    model = fit, train.x = train_x, train.y = train_y,
    test.x = test_x, test.y = test_y
  )
}
```

---

random\_strata\_cells     *Stratified Random Sampling of Cells*

---

**Description**

Performs stratified random sampling of cells from single-cell data, ensuring proportional representation of each cell type while respecting minimum and maximum count constraints.

**Usage**

```
random_strata_cells(
  input,
  group,
  proportion = 0.1,
  minimum_count_include = 300,
  minimum_count = 200,
  maximum_count = 1000,
  sub_cluster = NULL,
  cell_type = NULL
)
```

**Arguments**

input	A data frame or Seurat object containing cell annotations.
group	Character string specifying the column name for cell type grouping.
proportion	Numeric value between 0 and 1 specifying the sampling proportion. Default is 0.1.
minimum_count_include	Integer specifying the minimum count threshold for a cell type to be included in sampling. Default is 300.
minimum_count	Integer specifying the minimum number of cells to sample per cell type. Default is 200.
maximum_count	Integer specifying the maximum number of cells to sample per cell type. Default is 1000.
sub_cluster	Optional character string specifying a sub-cluster column for filtering. Default is NULL.
cell_type	Optional character string specifying the cell type value to filter when 'sub_cluster' is provided. Default is NULL.

**Value**

A data frame containing the sampled cells with preserved cell type proportions.

**Examples**

```
## Not run:
# Sample cells from a data frame
sampled_cells <- random_strata_cells(
  input = cell_annotations,
  group = "cell_type",
  proportion = 0.1,
  minimum_count_include = 300,
  minimum_count = 200,
  maximum_count = 1000
)

# Sample cells from a Seurat object
```

```
sampled_cells <- random_strata_cells(  
  input = seurat_object,  
  group = "seurat_clusters",  
  proportion = 0.2  
)  
  
## End(Not run)
```

---

**rbind\_iobr***Row Bind Multiple Data Sets*

---

### Description

Combines two or three data frames or matrices vertically using 'rbind'. Ensures compatibility of input data before binding by aligning columns.

### Usage

```
rbind_iobr(data1, data2, data3 = NULL)
```

### Arguments

data1	Data frame or matrix. First dataset to combine.
data2	Data frame or matrix. Second dataset to combine.
data3	Data frame or matrix or 'NULL'. Optional third dataset. Default is 'NULL'.

### Value

Combined data frame resulting from row binding the input datasets.

### Author(s)

Dongqiang Zeng

### Examples

```
data1 <- data.frame(A = 1:5, B = letters[1:5])  
data2 <- data.frame(A = 6:10, B = letters[6:10])  
combined_data <- rbind_iobr(data1, data2)  
  
# With three datasets  
data3 <- data.frame(A = 11:15, B = letters[11:15])  
combined_data <- rbind_iobr(data1, data2, data3)
```

---

RegressionResult	<i>Regression Result Computation</i>
------------------	--------------------------------------

---

### Description

Computes regression results with coefficients at  $\lambda_{\min}$  and  $\lambda_{1se}$ , and evaluates AUC for binomial outcomes. Returns a comprehensive summary of model performance on both training and testing datasets.

### Usage

```
RegressionResult(train.x, train.y, test.x, test.y, model)
```

### Arguments

<code>train.x</code>	Training predictors matrix.
<code>train.y</code>	Training outcomes (binary factor).
<code>test.x</code>	Testing predictors matrix.
<code>test.y</code>	Testing outcomes (binary factor).
<code>model</code>	Fitted <code>cv.glmnet</code> model object.

### Value

List containing:

**model** The fitted `cv.glmnet` model

**coefs** Data frame with feature names and coefficients at  $\lambda_{\min}$  and  $\lambda_{1se}$

**AUC** Matrix of AUC values for train/test sets at both  $\lambda$  values

### Examples

```
if (requireNamespace("glmnet", quietly = TRUE)) {
  set.seed(123)
  train_data <- matrix(rnorm(100 * 10), ncol = 10)
  train_outcome <- rbinom(100, 1, 0.5)
  test_data <- matrix(rnorm(50 * 10), ncol = 10)
  test_outcome <- rbinom(50, 1, 0.5)
  fitted_model <- glmnet::cv.glmnet(train_data, train_outcome, family = "binomial", nfolds = 5)
  results <- RegressionResult(
    train.x = train_data, train.y = train_outcome,
    test.x = test_data, test.y = test_outcome, model = fitted_model
  )
}
```

---

RemoveBatchEffect	<i>Remove Batch Effect of Expression Set</i>
-------------------	--

---

### Description

Removes batch effects between two gene expression datasets, typically representing different sample types such as cancer cells and immune cells. Uses ComBat from the sva package for batch correction.

### Usage

```
RemoveBatchEffect(cancer.exp, immune.exp, immune.cellType)
```

### Arguments

cancer.exp	Matrix or data frame. Cancer cell expression data with genes as rows and samples as columns.
immune.exp	Matrix or data frame. Immune cell expression data with genes as rows and samples as columns.
immune.cellType	Vector. Cell type for each column in 'immune.exp'.

### Value

A list containing:

- 1 Batch effect corrected cancer expression data
- 2 Batch effect corrected immune expression data
- 3 Aggregated immune expression data (median per cell type)

### Author(s)

Bo Li

### Examples

```
set.seed(123)
gene_names <- paste0("Gene", 1:100)
sample_names_cancer <- paste0("CancerSample", 1:10)
cancer.exp <- matrix(runif(1000, 1, 1000),
  nrow = 100, ncol = 10,
  dimnames = list(gene_names, sample_names_cancer)
)

sample_names_immune <- paste0("ImmuneSample", 1:5)
immune.exp <- matrix(runif(500, 1, 1000),
  nrow = 100, ncol = 5,
  dimnames = list(gene_names, sample_names_immune)
)
```

```

)

immune.cellType <- c("T-cell", "B-cell", "T-cell", "NK-cell", "B-cell")
names(immune.cellType) <- sample_names_immune

result <- RemoveBatchEffect(cancer.exp, immune.exp, immune.cellType)

```

---

remove\_batcheffect      *Removing Batch Effect from Expression Sets*

---

## Description

Removes batch effects from expression datasets using `sva::ComBat` (for microarray/TPM data) or `sva::ComBat_seq` (for RNA-seq count data). Generates PCA plots to visualize data before and after correction.

## Usage

```

remove_batcheffect(
  eset1,
  eset2,
  eset3 = NULL,
  id_type,
  data_type = c("array", "count", "tpm"),
  cols = "normal",
  palette = "jama",
  log2 = TRUE,
  check_eset = TRUE,
  adjust_eset = TRUE,
  repel = FALSE,
  path = NULL
)

```

## Arguments

<code>eset1</code>	First expression set (matrix or data frame with genes as rows).
<code>eset2</code>	Second expression set.
<code>eset3</code>	Optional third expression set. Use 'NULL' if not available.
<code>id_type</code>	Type of gene ID in expression sets (e.g., "ensembl", "symbol"). Required for count data normalization.
<code>data_type</code>	Type of data: "array", "count", or "tpm". Default is "array".
<code>cols</code>	Color scale for PCA plot. Default is "normal".
<code>palette</code>	Color palette for PCA plot. Default is "jama".
<code>log2</code>	Whether to perform log2 transformation. Default is 'TRUE'. Ignored for count data.

check_eset	Whether to check expression sets for errors. Default is 'TRUE'.
adjust_eset	Whether to adjust expression sets by removing problematic features. Default is 'TRUE'.
repel	Whether to add repelling labels to PCA plot. Default is 'FALSE'.
path	Directory where results should be saved. Default is 'NULL' (display only).

**Value**

Expression matrix after batch correction.

**Author(s)**

Dongqiang Zeng

**References**

Zhang Y, et al. ComBat-seq: batch effect adjustment for RNA-seq count data. *NAR Genomics and Bioinformatics*. 2020;2(3):lqaa078. doi:10.1093/nargab/lqaa078

Leek JT, et al. The sva package for removing batch effects and other unwanted variation in high-throughput experiments. *Bioinformatics*. 2012;28(6):882-883.

**Examples**

```
eset_stad <- load_data("eset_stad")
eset_blca <- load_data("eset_blca")

eset_corrected <- remove_batcheffect(
  eset_stad[1:1000, 1:5], eset_blca[1:1000, 1:5],
  id_type = "ensembl",
  data_type = "count"
)
```

---

remove\_duplicate\_genes

*Remove Duplicate Gene Symbols in Gene Expression Data*

---

**Description**

This function addresses duplicate gene symbols in a gene expression dataset by selecting the highest-expressing instance among duplicates. Users can choose between mean, standard deviation, or sum as the ranking criterion for selection. This is useful for preparing data where duplicates can lead to issues in downstream analyses.

**Usage**

```
remove_duplicate_genes(eset, column_of_symbol, method = c("mean", "sd", "sum"))
```

**Arguments**

<code>eset</code>	A data frame or matrix representing gene expression data, with gene symbols as one of the columns.
<code>column_of_symbol</code>	The name of the column containing gene symbols in <code>'eset'</code> .
<code>method</code>	The ranking method to use for selecting among duplicate gene symbols: <code>"mean"</code> for mean expression, <code>"sd"</code> for standard deviation, or <code>"sum"</code> for sum of expression values. Default is <code>"mean"</code> .

**Value**

A modified version of `'eset'` where duplicate gene symbols have been reduced to a single entry (the highest-ranking one). The gene symbols are set as row names in the returned data frame.

**Note**

Important: This function performs selection, not aggregation. For duplicate genes, it retains only the highest-ranking instance (based on the specified method) and discards others.

**Author(s)**

Dongqiang Zeng

**Examples**

```
# Load and annotate expression data
eset_stad <- load_data("eset_stad")
anno_rnaseq <- load_data("anno_rnaseq")
eset_stad <- anno_eset(eset = eset_stad, annotation = anno_rnaseq)
eset_stad <- tibble::rownames_to_column(as.data.frame(eset_stad), var = "id")

# Create duplicate gene names for demonstration
eset_stad[2:3, "id"] <- "MT-C01"

# Check duplicates before
sum(duplicated(eset_stad$id))

# Remove duplicates using mean expression as ranking criterion
eset_stad <- remove_duplicate_genes(
  eset = eset_stad,
  column_of_symbol = "id",
  method = "mean"
)

# Check duplicates after
sum(duplicated(rownames(eset_stad)))
```

---

`remove_names`*Remove Patterns from Column Names or Variables*

---

### Description

Modifies column names or specified variables in a data frame by replacing specified patterns with empty strings or spaces.

### Usage

```
remove_names(  
  input_df,  
  variable = "colnames",  
  patterns_to_na = patterns_to_na,  
  patterns_space = NULL  
)
```

### Arguments

`input_df` Data frame. Input data to modify.

`variable` Character. Column to modify: "colnames" for column names, or a specific column name. Default is "colnames".

`patterns_to_na` Character vector. Patterns to replace with empty string. Default uses [patterns\_to\_na].

`patterns_space` Character vector or 'NULL'. Patterns to replace with spaces. Default is 'NULL'.

### Value

Modified data frame with patterns replaced.

### Author(s)

Dongqiang Zeng

### Examples

```
df <- data.frame(  
  "CellA_cybersort" = 1:5,  
  "CellB_xCell" = 6:10,  
  "CellC_TIMER" = 11:15  
)  
result <- remove_names(df, variable = "colnames", patterns_to_na = patterns_to_na)  
colnames(result)
```

---

reset_iobr_mirrors	<i>Reset Download Mirrors to Default</i>
--------------------	--

---

**Description**

Resets the download mirror list to the default values.

**Usage**

```
reset_iobr_mirrors()
```

**Value**

Invisibly returns the default mirror list.

**Examples**

```
reset_iobr_mirrors()
```

---

roc_time	<i>Time-dependent ROC Curve for Survival Analysis</i>
----------	---

---

**Description**

Generates a time-dependent Receiver Operating Characteristic (ROC) plot to evaluate the predictive performance of one or more variables in survival analysis. Calculates the Area Under the Curve (AUC) for each specified time point and variable, and creates a multi-line ROC plot with annotated AUC values.

**Usage**

```
roc_time(  
  input,  
  vars,  
  time = "time",  
  status = "status",  
  time_point = 12,  
  time_type = "month",  
  palette = "jama",  
  cols = "normal",  
  seed = 1234,  
  show_col = FALSE,  
  path = NULL,  
  main = "PFS",  
  index = 1,  
  fig.type = "pdf",
```

```

    width = 5,
    height = 5.2
  )

```

### Arguments

input	Data frame containing variables for analysis.
vars	Character vector. Variable(s) to be evaluated.
time	Character string. Name of the time variable. Default is "time".
status	Character string. Name of the status variable. Default is "status".
time_point	Integer or vector. Time point(s) for ROC analysis. Default is '12'.
time_type	Character string. Time unit ("day" or "month"). Default is "month".
palette	Character string. Color palette for the plot. Default is "jama".
cols	Character vector or string. Color specification: "normal", "random", or custom color vector.
seed	Integer. Random seed for reproducibility. Default is '1234'.
show_col	Logical indicating whether to display the color palette. Default is 'FALSE'.
path	Character string or 'NULL'. Path to save the plot. Default is 'NULL'.
main	Character string. Main title of the plot. Default is "PFS".
index	Integer. Index for plot filename. Default is '1'.
fig.type	Character string. Output file type (e.g., "pdf", "png"). Default is "pdf".
width	Numeric. Width of the plot. Default is '5'.
height	Numeric. Height of the plot. Default is '5.2'.

### Value

A ggplot object representing the time-dependent ROC plot.

### Author(s)

Dongqiang Zeng

### Examples

```

tcga_stad_sig <- load_data("tcga_stad_sig")
pdata_stad <- load_data("pdata_stad")
input <- merge(pdata_stad, tcga_stad_sig, by = "ID")
roc_time(
  input = input, vars = c("Pan_F_TBRs", "CD_8_T_effector", "Immune_Checkpoint"),
  time = "time", status = "OS_status", time_point = 12, path = NULL, main = "OS"
)

```

---

scale\_matrix                      *Scale and Manipulate a Matrix*

---

### Description

Scales a gene expression matrix, optionally applies logarithmic transformation, and performs feature manipulation to remove problematic variables (e.g., genes with zero variance or missing values).

### Usage

```
scale_matrix(matrix, log2matrix = TRUE, manipulate = TRUE)
```

### Arguments

matrix	Numeric matrix with genes as rows and samples as columns.
log2matrix	Logical indicating whether to apply log2 transformation using [log2eset()]. Default is 'TRUE'.
manipulate	Logical indicating whether to perform feature manipulation to remove problematic features. Default is 'TRUE'.

### Value

A scaled matrix (genes as rows, samples as columns).

### Author(s)

Dongqiang Zeng

### Examples

```
eset_gse62254 <- load_data("eset_gse62254")
eset2 <- scale_matrix(eset_gse62254, log2matrix = FALSE, manipulate = TRUE)
```

---

select\_method                      *Select a Signature Scoring Method Subset*

---

### Description

Filters an integrated signature score matrix to retain results from a specified method (PCA, ssGSEA, or zscore) and strips method suffixes from column names.

### Usage

```
select_method(data, method = c("ssGSEA", "PCA", "zscore"))
```

**Arguments**

data	Data frame or matrix. Integrated signature score matrix.
method	Character. One of "PCA", "ssGSEA", or "zscore" (case-insensitive). Default is "ssGSEA".

**Value**

Matrix or data frame containing only the selected method's scores.

**Author(s)**

Dongqiang Zeng

**Examples**

```
eset_stad <- load_data("eset_stad")
anno_grch38 <- load_data("anno_grch38")
hallmark <- load_data("hallmark")
eset <- anno_eset(eset = eset_stad, annotation = anno_grch38, probe = "id")
eset <- eset[1:5000, 1:10]
res <- calculate_sig_score(
  eset = eset,
  signature = hallmark[1:4],
  method = "integration"
)
select_method(res, method = "PCA")
```

---

signature\_collection *Gene signature collection for pathway and immune analysis*

---

**Description**

A named list of gene signatures used in the IOBR package for immune deconvolution, pathway scoring, functional annotation, and tumour microenvironment (TME) characterization. Each element corresponds to a predefined biological signature and contains a character vector of HGNC gene symbols.

**Usage**

```
data(signature_collection)
```

**Format**

A named list of length 323. Each element is a character vector of gene symbols. Representative entries include:

**CD\_8\_T\_effector** Markers of CD8<sup>+</sup> effector T cells.

**DDR** DNA damage response and repair genes.

**Immune\_Checkpoint** Immune checkpoint molecules.

**CellCycle\_Reg** Core regulators of cell-cycle progression.

**Mismatch\_Repair** Mismatch-repair pathway genes.

**TMEsocreA\_CIR** TME-related signature used in TMEscore analysis.

... Additional signatures are included in the list but are not individually listed here; all follow the same structure.

**Examples**

```
data(signature_collection)
head(signature_collection)
```

---

```
signature_score_calculation_methods
      Signature Score Calculation Methods
```

---

**Description**

A named vector of supported methods for calculating signature scores.

**Usage**

```
signature_score_calculation_methods
```

**Format**

Named character vector:

**PCA** Principal Component Analysis method ("pca")

**ssGSEA** Single-sample Gene Set Enrichment Analysis ("ssgsea")

**z-score** Z-score transformation method ("zscore")

**Integration** Integration of multiple methods ("integration")

**Examples**

```
signature_score_calculation_methods
signature_score_calculation_methods["PCA"]
```

---

**sigScore***Calculate Signature Score Using PCA, Mean, or Z-score Methods*

---

**Description**

Computes signature scores from gene expression data using Principal Component Analysis (PCA), mean-based, or z-score approaches.

**Usage**

```
sigScore(eset, methods = c("PCA", "mean", "zscore"))
```

**Arguments**

eset	Normalized expression matrix with genes (signature) as rows and samples as columns.
methods	Scoring method: "PCA" (default) for principal component 1, "mean" for mean expression, or "zscore" for z-score normalized mean.

**Value**

Numeric vector of length 'ncol(eset)'; a score summarizing the rows of 'eset'.

**Author(s)**

Dorothee Nickles, Dongqiang Zeng

**Examples**

```
# Load example data
eset_stad <- load_data("eset_stad")
eset <- count2tpm(eset_stad, idType = "ensembl")

# Get signature genes
signature_tme <- load_data("signature_tme")
genes <- signature_tme[["CD_8_T_effector"]]
genes <- genes[genes %in% rownames(eset)]

# Calculate scores (only if enough genes are available)
if (length(genes) >= 2) {
  score_pca <- sigScore(eset = eset[genes, ], methods = "PCA")
  score_mean <- sigScore(eset = eset[genes, ], methods = "mean")
  score_zscore <- sigScore(eset = eset[genes, ], methods = "zscore")
}
```

sig\_box

*Signature Box Plot with Statistical Comparisons***Description**

Creates box plots to visualize signature distributions across groups with optional statistical pairwise comparisons. Supports both data frames and Seurat objects for single-cell data visualization.

**Usage**

```
sig_box(
  data,
  signature,
  variable,
  palette = "nrc",
  cols = NULL,
  jitter = FALSE,
  point_size = 5,
  angle_x_text = 0,
  hjust = 0.5,
  show_pairwise_p = TRUE,
  show_overall_p = FALSE,
  return_stat_res = FALSE,
  size_of_pvalue = 6,
  size_of_font = 10,
  assay = NULL,
  slot = "scale.data",
  scale = FALSE
)
```

**Arguments**

data	Data frame or Seurat object containing the signature and grouping variable.
signature	Character string specifying the column name (or feature name in Seurat) for the signature values to plot on the y-axis.
variable	Character string specifying the grouping variable column name for the x-axis.
palette	Character string specifying the color palette name. Default is "nrc".
cols	Character vector of custom fill colors. If 'NULL', palette is used. Default is 'NULL'.
jitter	Logical indicating whether to add jittered points to the box plot. Default is 'FALSE'.
point_size	Numeric value specifying the size of jittered points. Default is 5.
angle_x_text	Numeric value specifying the rotation angle for x-axis labels (in degrees). Default is 0.

hjust	Numeric value specifying the horizontal justification of x-axis labels. Default is 0.5.
show_pairwise_p	Logical indicating whether to display pairwise comparison p-values between groups. Default is 'TRUE'.
show_overall_p	Logical indicating whether to display the overall group difference p-value. Default is 'FALSE'.
return_stat_res	Logical indicating whether to return statistical test results instead of the plot. Default is 'FALSE'.
size_of_pvalue	Numeric value specifying the font size for p-values. Default is 6.
size_of_font	Numeric value specifying the base font size. Default is 10.
assay	Character string specifying the assay name (for Seurat objects). Default is 'NULL'.
slot	Character string specifying the slot name (for Seurat objects). Default is "scale.data".
scale	Logical indicating whether to scale signature values (z-score transformation). Default is 'FALSE'.

**Value**

If 'return\_stat\_res = FALSE', returns a ggplot2 object. If 'return\_stat\_res = TRUE', returns a data frame containing statistical test results.

**Author(s)**

Dongqiang Zeng

**Examples**

```
tcga_stad_pdata <- load_data("tcga_stad_pdata")
sig_box(
  data = tcga_stad_pdata,
  signature = "TMEscore_plus",
  variable = "subtype",
  jitter = TRUE,
  palette = "jco"
)
```

---

sig\_box\_batch

*Batch Signature Box Plots for Group Comparisons*


---

**Description**

Generates multiple box plots for specified features (signatures) across groups in the input data. Supports customization of plot appearance, output path, statistical annotation, and compatibility with Seurat objects. Plots are saved to the specified directory or a default folder.

**Usage**

```
sig_box_batch(
  input,
  vars,
  groups,
  pattern_vars = FALSE,
  path = NULL,
  index = 0,
  angle_x_text = 0,
  hjust = 0.5,
  palette = "jama",
  cols = NULL,
  jitter = FALSE,
  point_size = 5,
  size_of_font = 8,
  size_of_pvalue = 4.5,
  show_pvalue = TRUE,
  return_stat_res = FALSE,
  assay = NULL,
  slot = "scale.data",
  scale = FALSE,
  height = 5,
  width = 3.5,
  fig_type = "pdf",
  max_count_feas = 30
)
```

**Arguments**

input	Data frame or Seurat object containing the data for analysis.
vars	Character vector. Features or variables to analyze. When 'pattern_vars = TRUE', these are treated as regular expression patterns.
groups	Character vector. Grouping variable(s) for comparison.
pattern_vars	Logical indicating whether to treat 'vars' as regular expression patterns for matching column names. Default is 'FALSE'.
path	Character string or 'NULL'. Directory to save plots. If 'NULL', plots are not saved. Default is 'NULL'.
index	Integer. Starting index for plot filenames. Default is '0'.
angle_x_text	Numeric. Angle of x-axis labels in degrees. Default is '0'.
hjust	Numeric. Horizontal justification of x-axis labels. Default is '0.5'.
palette	Character. Color palette for plots. Default is "'jama'".
cols	Character vector or 'NULL'. Custom colors for plot elements.
jitter	Logical indicating whether to add jittered points. Default is 'FALSE'.
point_size	Numeric. Size of points. Default is '5'.
size_of_font	Numeric. Base font size. Default is '8'.

size_of_pvalue	Numeric. Size of p-value text. Default is '4.5'.
show_pvalue	Logical indicating whether to display p-values. Default is 'TRUE'.
return_stat_res	Logical indicating whether to return statistical results instead of saving plots. Default is 'FALSE'.
assay	Character string or 'NULL'. Assay type for Seurat objects.
slot	Character. Data slot for Seurat objects. Default is "scale.data".
scale	Logical indicating whether to scale data before analysis. Default is 'FALSE'.
height	Numeric. Height of plots in inches. Default is '5'.
width	Numeric. Width of plots in inches. Default is '3.5'.
fig_type	Character. File format for plots (e.g., "pdf", "png"). Default is "pdf".
max_count_feats	Integer. Maximum number of features to analyze when 'pattern_vars = TRUE'. If matched variables exceed this limit, only the first 'max_count_feats' features are used. Default is '30'.

**Value**

If 'return\_stat\_res = TRUE', returns a data frame of statistical results; otherwise, invisibly returns the path to saved plots.

**Author(s)**

Dongqiang Zeng

**Examples**

```
tcga_stad_pdata <- load_data("tcga_stad_pdata")
sig_box_batch(
  input = tcga_stad_pdata,
  vars = c("TMEscore_plus", "GZMB"),
  groups = "subtype",
  jitter = TRUE,
  palette = "jco",
  path = tempdir()
)
```

---

sig\_forest

*Forest Plot for Survival Analysis Results*


---

**Description**

Generates a forest plot to visualize hazard ratios, confidence intervals, and p-values for gene signatures or features from survival analysis.

**Usage**

```
sig_forest(  
  data,  
  signature,  
  pvalue = "P",  
  HR = "HR",  
  CI_low_0.95 = "CI_low_0.95",  
  CI_up_0.95 = "CI_up_0.95",  
  n = 10,  
  max_character = 25,  
  discrete_width = 35,  
  color_option = 1,  
  cols = NULL,  
  text.size = 13  
)
```

**Arguments**

data	Data frame with survival analysis results including p-values, hazard ratios, and confidence intervals.
signature	Character string. Column name for signatures or feature names.
pvalue	Character string. Column name for p-values. Default is "P".
HR	Character string. Column name for hazard ratios. Default is "HR".
CI_low_0.95	Character string. Column name for lower CI bound. Default is "CI_low_0.95".
CI_up_0.95	Character string. Column name for upper CI bound. Default is "CI_up_0.95".
n	Integer. Maximum number of signatures to display. Default is '10'.
max_character	Integer. Maximum characters for labels before wrapping. Default is '25'.
discrete_width	Integer. Width for discretizing long labels. Default is '35'.
color_option	Integer. Color option for p-value gradient (1, 2, or 3). Default is '1'.
cols	Character vector. Custom colors for p-value gradient (low to high). Default is 'NULL'.
text.size	Numeric. Text size for y-axis labels. Default is '13'.

**Value**

A ggplot2 object of the forest plot.

**Author(s)**

Dongqiang Zeng

**Examples**

```
# Example with sample survival results
sample_results <- data.frame(
  ID = c("Sig1", "Sig2", "Sig3"),
  HR = c(1.5, 0.8, 2.0),
  P = c(0.01, 0.05, 0.001),
  CI_low_0.95 = c(1.1, 0.6, 1.5),
  CI_up_0.95 = c(2.0, 1.0, 2.8)
)
sig_forest(data = sample_results, signature = "ID")
```

sig\_group

*Grouped gene signatures for IOBR analysis***Description**

A named list that organizes gene signatures into functional or biological categories. Each element of the list is a character vector containing the names of gene signatures defined in `signature_collection`. A total of 43 signature groups are included, covering tumour intrinsic pathways, immune-related processes, stromal activity, TME characteristics and immuno-oncology biomarkers. These groups are used in IOBR to conveniently select sets of signatures for scoring and visualization.

**Usage**

```
data(sig_group)
```

**Format**

A named list of length 43. Each element is a character vector of signature names. Representative groups include:

**tumor\_signature** Signatures related to intrinsic tumour biology such as cell cycle, DNA damage repair and histone regulation.

**EMT** Epithelial–mesenchymal transition (EMT)–associated signatures.

**io\_biomarkers** Immuno-oncology biomarker–related signatures.

**immu\_microenvironment** Immune microenvironment–related signatures.

**immu\_suppression** Immune suppression–related signatures.

**immu\_exclusion** Signatures associated with immune exclusion and stromal barriers.

**TCR\_BCR** T-cell and B-cell receptor pathway signatures.

**tme\_signatures1** Tumour microenvironment signature panel (set 1).

**tme\_signatures2** Tumour microenvironment signature panel (set 2).

... Additional groups are included (43 total), but not listed individually here; all groups follow the same structure.

**Examples**

```
data(sig_group)
head(sig_group)
```

---

`sig_gsea`*Perform Gene Set Enrichment Analysis (GSEA)*

---

### Description

Conducts Gene Set Enrichment Analysis to identify significantly enriched gene sets from differential gene expression data. Supports MSigDB gene sets or custom gene signatures, and generates comprehensive visualizations and statistical results.

### Usage

```
sig_gsea(  
  deg,  
  genesets = NULL,  
  path = NULL,  
  gene_symbol = "symbol",  
  logfc = "log2FoldChange",  
  org = c("hsa", "mus"),  
  msigdb = TRUE,  
  category = "H",  
  subcategory = NULL,  
  palette_bar = "jama",  
  palette_gsea = 2,  
  cols_gsea = NULL,  
  cols_bar = NULL,  
  show_bar = 10,  
  show_col = FALSE,  
  show_plot = FALSE,  
  show_gsea = 8,  
  show_path_n = 20,  
  plot_single_sig = FALSE,  
  project = "custom_sig",  
  minGSSize = 10,  
  maxGSSize = 500,  
  verbose = TRUE,  
  seed = FALSE,  
  fig.type = "pdf",  
  print_bar = TRUE  
)
```

### Arguments

<code>deg</code>	Data frame containing differential expression results with gene symbols and log fold changes.
<code>genesets</code>	List of custom gene sets for enrichment analysis. If 'NULL', MSigDB gene sets are used based on 'org' and 'category'. Default is 'NULL'.

path	Character string specifying the directory path for saving results. Default is 'NULL'.
gene_symbol	Character string specifying the column name in 'deg' containing gene symbols. Default is "symbol".
logfc	Character string specifying the column name in 'deg' containing log fold change values. Default is "log2FoldChange".
org	Character string specifying the organism. Options are "hsa" (Homo sapiens) or "mus" (Mus musculus). Default is "hsa".
msigdb	Logical indicating whether to use MSigDB gene sets. Default is 'TRUE'.
category	Character string specifying the MSigDB category (e.g., "H" for Hallmark, "C2" for curated gene sets). Default is "H".
subcategory	Character string specifying the MSigDB subcategory to filter gene sets. Default is 'NULL'.
palette_bar	Character string or integer specifying the color palette for bar plots. Default is "jama".
palette_gsea	Integer specifying the color palette for GSEA plots. Default is '2'.
cols_gsea	Character vector specifying custom colors for GSEA enrichment plots. If 'NULL', colors are automatically generated. Default is 'NULL'.
cols_bar	Character vector specifying custom colors for the enrichment bar plot. If 'NULL', colors are automatically generated. Default is 'NULL'.
show_bar	Integer specifying the number of top enriched gene sets to display in the bar plot. Default is '10'.
show_col	Logical indicating whether to display color names in the bar plot. Default is 'FALSE'.
show_plot	Logical indicating whether to display GSEA enrichment plots. Default is 'FALSE'.
show_gsea	Integer specifying the number of top significant gene sets for which to generate GSEA plots. Default is '8'.
show_path_n	Integer specifying the number of pathways to display in GSEA plots. Default is '20'.
plot_single_sig	Logical indicating whether to generate separate plots for each significant gene set. Default is 'TRUE'.
project	Character string specifying the project name for output files. Default is "custom_sig".
minGSSize	Integer specifying the minimum gene set size for analysis. Default is '10'.
maxGSSize	Integer specifying the maximum gene set size for analysis. Default is '500'.
verbose	Logical indicating whether to display progress messages. Default is 'TRUE'.
seed	Logical indicating whether to set a random seed for reproducibility. Default is 'FALSE'.
fig.type	Character string specifying the file format for saving plots (e.g., "pdf", "png"). Default is "pdf".
print_bar	Logical indicating whether to save and print the bar plot. Default is 'TRUE'.

**Value**

List containing:

- up** Data frame of up-regulated enriched gene sets
- down** Data frame of down-regulated enriched gene sets
- all** Complete GSEA results
- plot\_top** GSEA enrichment plot for top gene sets

**Author(s)**

Dongqiang Zeng

**Examples**

```
set.seed(123)
genes <- c(
  "TP53", "BRCA1", "EGFR", "MYC", "KRAS", "PTEN", "APC", "RB1",
  "CDKN2A", "VHL", "ATM", "ATR", "CHEK2", "PALB2", "RAD51", "MDM2",
  "CDK4", "CDK6", "CCND1", "CCNE1", "CDK2", "E2F1", "E2F2", "E2F3",
  "ARF1", "ARF3", "ARF4", "ARF5", "ARF6", "GSK3B", "AKT1", "AKT2",
  "PIK3CA", "PIK3CB", "PIK3CD", "PIK3CG", "PIK3R1", "PIK3R2", "PIK3R3"
)
deg <- data.frame(
  symbol = genes,
  log2FoldChange = rnorm(length(genes), mean = 0, sd = 2),
  padj = runif(length(genes), 0, 0.1)
)
signature <- list(
  DNA_Repair = c(
    "TP53", "BRCA1", "ATM",
    "ATR", "CHEK2", "PALB2", "RAD51"
  ),
  Cell_Cycle = c(
    "TP53", "MYC",
    "RB1", "CDKN2A", "CDK4",
    "CDK6", "CCND1", "CCNE1",
    "CDK2", "E2F1", "E2F2", "E2F3"
  ),
  PI3K_AKT = c(
    "AKT1", "AKT2",
    "PIK3CA", "PIK3CB", "PIK3CD",
    "PIK3CG", "PIK3R1", "PIK3R2", "PIK3R3"
  )
)
res <- sig_gsea(
  deg = deg,
  genesets = signature,
  path = tempdir(),
  show_plot = FALSE,
  print_bar = FALSE
)
```

```
)  
print(names(res))
```

---

sig\_heatmap

*Signature Heatmap with Optional Annotations*

---

## Description

Generates a heatmap of selected features grouped by a categorical variable, with optional conditional (annotation) bars. Supports palette customization, scaling, size controls, and output saving.

## Usage

```
sig_heatmap(  
  input,  
  id = "ID",  
  features,  
  group,  
  condition = NULL,  
  id_condition = "vars",  
  col_condition = "condition",  
  cols_condition = NULL,  
  scale = FALSE,  
  palette = 2,  
  cols_heatmap = NULL,  
  palette_group = "jama",  
  show_col = FALSE,  
  show_palettes = FALSE,  
  cols_group = NULL,  
  show_plot = TRUE,  
  width = 8,  
  height = NULL,  
  size_col = 10,  
  size_row = 8,  
  angle_col = 90,  
  column_title = NULL,  
  row_title = NULL,  
  show_heatmap_col_name = FALSE,  
  path = NULL,  
  index = NULL  
)
```

## Arguments

input	Data frame containing ID, grouping variable, and feature columns.
id	Character string. Column name for sample identifier. Default is "ID".

features	Character vector. Feature (column) names to include in the heatmap.
group	Character string. Grouping variable column name.
condition	Data frame or 'NULL'. Optional annotation table with variable-condition mapping. Default is 'NULL'.
id_condition	Character string. Column name in 'condition' for feature IDs. Default is "vars".
col_condition	Character string. Column name in 'condition' for condition labels. Default is "condition".
cols_condition	Character vector. Colors for conditions.
scale	Logical indicating whether to scale values by row. Default is 'FALSE'.
palette	Integer or character. Palette index/name for heatmap colors. Default is '2'.
cols_heatmap	Character vector. Custom colors for heatmap gradient.
palette_group	Character string. Palette name for group colors. Default is "jama".
show_col	Logical indicating whether to display the color vector. Default is 'FALSE'.
show_palettes	Logical indicating whether to print palette options. Default is 'FALSE'.
cols_group	Character vector. Custom colors for groups.
show_plot	Logical indicating whether to print the heatmap. Default is 'TRUE'.
width	Numeric. Plot width in inches. Default is '8'.
height	Numeric or 'NULL'. Plot height in inches. Auto-calculated if 'NULL'.
size_col	Numeric. Font size for column labels. Default is '10'.
size_row	Numeric. Font size for row labels. Default is '8'.
angle_col	Numeric. Rotation angle for column labels in degrees. Default is '90'.
column_title	Character string or 'NULL'. Title for column annotation.
row_title	Character string or 'NULL'. Title for row annotation.
show_heatmap_col_name	Logical indicating whether to show column names. Default is 'FALSE'.
path	Character string or 'NULL'. Output directory for saving the heatmap.
index	Integer or 'NULL'. Index appended to filename. Default is 'NULL'.

**Value**

A tidyHeatmap object. Saves PDF only when 'path' is provided.

**Author(s)**

Dongqiang Zeng

**Examples**

```
tcga_stad_sig <- load_data("tcga_stad_sig")
tcga_stad_pdata <- load_data("tcga_stad_pdata")
input <- merge(tcga_stad_pdata, tcga_stad_sig, by = "ID")
feas <- grep("MCPcounter", colnames(input), value = TRUE)
sig_heatmap(input = input, features = feas, group = "subtype", scale = TRUE)
```

---

`sig_pheatmap`*Generate Heatmap for Signature Data*

---

### Description

Creates a heatmap from signature data with grouping variables, offering flexible options for colors, clustering, and output formats using ComplexHeatmap.

### Usage

```
sig_pheatmap(  
  input,  
  feas,  
  group,  
  group2 = NULL,  
  group3 = NULL,  
  ID = "ID",  
  path = NULL,  
  cols1 = "random",  
  cols2 = "random",  
  cols3 = "random",  
  seed = 54321,  
  show_col = FALSE,  
  palette1 = 1,  
  palette2 = 2,  
  palette3 = 3,  
  cluster_cols = TRUE,  
  palette_for_heatmap = 6,  
  scale.matrix = TRUE,  
  cellwidth = 1,  
  cellheight = 9,  
  show_colnames = FALSE,  
  fig.type = "pdf",  
  width = 6,  
  height = NULL,  
  file_name_prefix = 1  
)
```

### Arguments

<code>input</code>	Data frame with variables in columns.
<code>feas</code>	Character vector. Feature names (columns) to include in heatmap.
<code>group</code>	Character string. Column name for primary grouping variable.
<code>group2</code>	Character string or 'NULL'. Optional secondary grouping variable.
<code>group3</code>	Character string or 'NULL'. Optional tertiary grouping variable.

ID	Character string. Column name for sample identifiers. Default is "ID".
path	Character string or 'NULL'. Directory to save output files. If 'NULL', the heatmap is not saved. Default is 'NULL'.
cols1	Character vector or "random" or "normal". Colors for primary group. Default is "random".
cols2	Character vector or "random" or "normal". Colors for secondary group. Default is "random".
cols3	Character vector or "random" or "normal". Colors for tertiary group. Default is "random".
seed	Integer. Random seed for color generation. Default is '54321'.
show_col	Logical indicating whether to display colors. Default is 'FALSE'.
palette1	Integer. Palette for primary group. Default is '1'.
palette2	Integer. Palette for secondary group. Default is '2'.
palette3	Integer. Palette for tertiary group. Default is '3'.
cluster_cols	Logical indicating whether to cluster columns. Default is 'TRUE'.
palette_for_heatmap	Integer. Palette number for heatmap. Default is '6'.
scale.matrix	Logical indicating whether to scale the matrix. Default is 'TRUE'.
cellwidth	Numeric. Width of each cell in points. Default is '1'.
cellheight	Numeric. Height of each cell in points. Default is '9'.
show_colnames	Logical indicating whether to show column names. Default is 'FALSE'.
fig.type	Character string. File format for saving. Default is "pdf".
width	Numeric. Width of saved figure in inches. Default is '6'.
height	Numeric or 'NULL'. Height of saved figure in inches. Calculated if 'NULL'.
file_name_prefix	Character or numeric. Prefix for saved file name. Default is '1'.

### Value

A list containing:

- p\_anno** Annotation data frame
- p\_cols** List of cluster colors
- plot** ComplexHeatmap object
- eset** Transformed expression matrix

### Author(s)

Dongqiang Zeng

**Examples**

```

tcga_stad_sig <- load_data("tcga_stad_sig")
tcga_stad_pdata <- load_data("tcga_stad_pdata")
input <- merge(tcga_stad_pdata, tcga_stad_sig, by = "ID")
feas <- grep("MCPcounter", colnames(input), value = TRUE)
sig_pheatmap(
  input = input, feas = feas, group = "subtype",
  scale.matrix = TRUE, path = tempdir()
)

```

sig\_roc

*Plot ROC Curves and Compare Them***Description**

Generates Receiver Operating Characteristic (ROC) curves for multiple predictors and optionally performs statistical comparisons between them.

**Usage**

```

sig_roc(
  data,
  response,
  variables,
  fig.path = NULL,
  main = NULL,
  file.name = NULL,
  palette = "jama",
  cols = NULL,
  alpha = 1,
  compare = FALSE,
  smooth = TRUE,
  compare_method = "bootstrap",
  boot.n = 100
)

```

**Arguments**

data	Data frame containing the predictor variables and binary outcome.
response	Character. Name of the binary outcome variable in 'data'.
variables	Character vector. Names of predictor variables for ROC curves.
fig.path	Character or 'NULL'. Directory path to save output PDF. Default is 'NULL'.
main	Character or 'NULL'. Main title for the ROC plot. Default is 'NULL'.
file.name	Character or 'NULL'. Output PDF filename without extension. Default is "'0-ROC of multiple variables'".

palette	Character. Color palette for ROC curves. Default is "jama".
cols	Character vector or 'NULL'. Custom colors for ROC curves. Default is 'NULL'.
alpha	Numeric. Transparency level (1 = opaque, 0 = transparent). Default is '1'.
compare	Logical. Whether to perform statistical comparison of AUCs. Default is 'FALSE'.
smooth	Logical. Whether to smooth ROC curves. Default is 'TRUE'.
compare_method	Character. Method for comparing ROC curves. Default is "bootstrap".
boot.n	Integer. Number of bootstrap replications. Default is '100'.

### Value

A list containing:

**auc.out** Data frame with AUC values and confidence intervals

**legend.name** Vector of legend entries for the plot

**p.out** If 'compare = TRUE', data frame with p-values from comparisons

### Author(s)

Dongqiang Zeng

### Examples

```
tcga_stad_pdata <- load_data("tcga_stad_pdata")
sig_roc(
  data = tcga_stad_pdata, response = "OS_status",
  variables = c("TMEscore_plus", "GZMB", "GNLY")
)
```

---

sig\_surv\_plot

*Generate Kaplan-Meier Survival Plot for Signature*

---

### Description

Creates Kaplan-Meier survival plots for a given signature or gene, with automatic cutoff determination. Generates three types of plots: optimal cutoff (best cutoff), tertile-based (3 groups), and median split (2 groups).

### Usage

```
sig_surv_plot(
  input_pdata,
  signature,
  project = "KM",
  ID = "ID",
  time = "time",
```

```

    status = "status",
    time_type = "month",
    break_month = "auto",
    cols = NULL,
    palette = "jama",
    show_col = TRUE,
    mini_sig = "score",
    fig.type = "png",
    save_path = NULL,
    index = 1
  )

```

### Arguments

input_pdata	Data frame with survival data and signature scores.
signature	Character string. Column name of the target signature.
project	Character string. Project name for output. Default is "KM".
ID	Character string. Column name for sample IDs. Default is "ID".
time	Character string. Column name for survival time. Default is "time".
status	Character string. Column name for survival status. Default is "status".
time_type	Character string. Time unit ("month" or "day"). Default is "month".
break_month	Numeric or "auto". Time axis breaks. Default is "auto".
cols	Character vector. Optional custom colors.
palette	Character string. Color palette if 'cols' not provided. Default is "jama".
show_col	Logical indicating whether to show colors. Default is 'TRUE'.
mini_sig	Character string. Label for low score group. Default is "score".
fig.type	Character string. File format. Default is "png".
save_path	Character string or 'NULL'. Directory for saving plots. If 'NULL', plots are not saved. Default is 'NULL'.
index	Integer. Index for multiple plots. Default is '1'.

### Value

A list containing:

**data** Processed input data with group assignments

**plots** Combined survival plots

### Author(s)

Dongqiang Zeng

### Examples

```
tcga_stad_pdata <- load_data("tcga_stad_pdata")
sig_surv_plot(
  input_pdata = tcga_stad_pdata,
  signature = "TMEscore_plus",
  time = "time",
  status = "OS_status"
)
```

---

SplitTrainTest

*Split Data into Training and Testing Sets*

---

### Description

Divides dataset into training and testing sets using random sampling. Maintains data integrity for both binomial and survival analysis types.

### Usage

```
SplitTrainTest(x, y, train_ratio, type = c("binomial", "survival"), seed)
```

### Arguments

x	Predictor matrix or data frame.
y	Outcome vector (binomial) or matrix with time/status (survival).
train_ratio	Proportion for training (0-1). Default is '0.7'.
type	Analysis type: "binomial" or "survival".
seed	Random seed for reproducibility.

### Value

List containing:

**train.x** Training predictors matrix

**train.y** Training outcomes

**test.x** Testing predictors matrix

**test.y** Testing outcomes

**train\_sample** Indices of training samples

### Examples

```
data_matrix <- matrix(rnorm(200), ncol = 2)
outcome_vector <- rbinom(100, 1, 0.5)
split_data <- SplitTrainTest(
  data_matrix, outcome_vector,
  train_ratio = 0.7,
  type = "binomial", seed = 123
)
```

---

stad\_group

*Example Clinical Data for TCGA-STAD Gastric Cancer Analysis*

---

### Description

A small example dataset demonstrating the clinical data structure required for TCGA Stomach Adenocarcinoma (STAD) analysis using IOBR package functions. Contains simulated clinical variables, molecular subtypes, and survival data.

### Usage

```
data(stad_group)
```

### Format

A data frame with patient samples as rows and clinical variables as columns:

**ID** Unique patient identifier (TCGA barcode format)

**stage** AJCC pathological stage (Stage\_II, Stage\_III, Stage\_IV)

**status** Patient vital status (Alive, Dead, NA)

**Lauren** Lauren histological classification (Intestinal, Diffuse, Mixed)

**subtype** Molecular subtype classification (EBV, GS)

**EBV** Epstein-Barr virus status (Positive, Negative)

**time** Overall survival time in months

**OS\_status** Overall survival status (0=alive, 1=dead)

### Examples

```
data(stad_group)
head(stad_group)
```

---

subgroup\_data

*Example Dataset for Subgroup Survival Analysis*


---

### Description

An example dataset demonstrating the data structure required for subgroup survival analysis using the `subgroup_survival` function. Contains simulated clinical and biomarker data with survival outcomes.

### Usage

```
data(subgroup_data)
```

### Format

A data frame with clinical variables and biomarker scores:

**Patient\_ID** Unique patient identifier

**ProjectID** Study or dataset identifier (e.g., "Dataset1")

**AJCC\_stage** AJCC pathological stage (2, 3, 4)

**status** Event status (0=censored, 1=event)

**time** Follow-up time in months

**score** Continuous biomarker score (numeric values)

**score\_binary** Binary biomarker classification ("High", "Low")

### Examples

```
data(subgroup_data)
head(subgroup_data)
```

---

subgroup\_survival

*Subgroup Survival Analysis Using Cox Proportional Hazards Models*


---

### Description

Extracts hazard ratios (HR) and 95 proportional hazards models across specified subgroups.

### Usage

```
subgroup_survival(
  pdata,
  time_name = "time",
  status_name = "status",
  variables,
  object
)
```

**Arguments**

pdata	Data frame containing variables, follow-up time, and outcome.
time_name	Character. Column name of follow-up time. Default is "time".
status_name	Character. Column name of event status (1/0). Default is "status".
variables	Character vector. Subgrouping variables (each processed independently).
object	Character. Variable of interest used in Cox model. If it has levels "High" and "Low", recode "High" to 0 and "Low" to 1 before calling.

**Value**

Data frame summarizing subgroup Cox results (HR, CI, p-value).

**Author(s)**

Dongqiang Zeng

**Examples**

```

subgroup_data <- load_data("subgroup_data")
input <- subset(subgroup_data, time > 0 & !is.na(status) & !is.na(AJCC_stage))

# Binary variable example
res_bin <- subgroup_survival(
  pdata = input, time_name = "time", status_name = "status",
  variables = c("ProjectID", "AJCC_stage"), object = "score_binary"
)

# Continuous variable example
res_cont <- subgroup_survival(
  pdata = input, time_name = "time", status_name = "status",
  variables = c("ProjectID", "AJCC_stage"), object = "score"
)

```

---

surv\_group

*Generate Kaplan-Meier Survival Plots for Categorical Groups*


---

**Description**

Creates Kaplan-Meier survival plots for data grouped by a categorical variable. Handles both binary and multi-level categorical groups with customizable plot aesthetics.

**Usage**

```

surv_group(
  input_pdata,
  target_group,
  ID = "ID",
  levels = c("High", "Low"),
  reference_group = NULL,
  project = NULL,
  time = "time",
  status = "status",
  time_type = "month",
  break_month = "auto",
  cols = NULL,
  palette = "jama",
  mini_sig = "score",
  save_path = NULL,
  fig.type = "pdf",
  index = 1,
  width = 6,
  height = 6.5,
  font.size.table = 3
)

```

**Arguments**

<code>input_pdata</code>	Data frame containing survival data and grouping variables.
<code>target_group</code>	Name of column containing the grouping variable.
<code>ID</code>	Name of column with unique identifiers. Default is "ID".
<code>levels</code>	Names for levels of <code>target_group</code> (for binary groups). Default is <code>c("High", "Low")</code> .
<code>reference_group</code>	Reference level for binary comparison. Default is <code>NULL</code> .
<code>project</code>	Optional title for plot. Default is <code>NULL</code> .
<code>time</code>	Name of column with follow-up times. Default is "time".
<code>status</code>	Name of column with event indicators. Default is "status".
<code>time_type</code>	Units: "month" or "day". Default is "month".
<code>break_month</code>	X-axis break interval. If "auto", calculated automatically. Default is "auto".
<code>cols</code>	Color vector for plot lines. Default is <code>NULL</code> .
<code>palette</code>	Color palette name. Default is "jama".
<code>mini_sig</code>	Prefix label for variables. Default is "score".
<code>save_path</code>	Directory for saving plot. Default is <code>NULL</code> .
<code>fig.type</code>	File format: "pdf" or "png". Default is "pdf".
<code>index</code>	Identifier for file naming. Default is 1.

width	Plot width. Default is 6.
height	Plot height. Default is 6.5.
font.size.table	Font size for risk table. Default is 3.

**Value**

Kaplan-Meier plot object.

**Author(s)**

Dongqiang Zeng

**Examples**

```
tcga_stad_pdata <- load_data("tcga_stad_pdata")

surv_group(
  input_pdata = tcga_stad_pdata,
  target_group = "Lauren",
  time = "time",
  status = "OS_status"
)
```

---

tcga_rna_preps	<i>Preprocess TCGA RNA-seq Data</i>
----------------	-------------------------------------

---

**Description**

Preprocesses TCGA RNA-seq data by modifying sample types, transforming data, and annotating genes based on specified parameters.

**Usage**

```
tcga_rna_preps(
  eset,
  id_type = c("ensembl", "symbol"),
  input_type = c("log2count", "count"),
  output = c("tumor", "tumor_normal"),
  output_type = c("tpm", "log2tpm", "count"),
  annotation = TRUE
)
```

**Arguments**

eset	Matrix or data frame. RNA-seq gene expression matrix from TCGA.
id_type	Character. Gene identifier type: "ensembl" or "symbol". Default is "ensembl".
input_type	Character. Input data type: "log2count" or "count". Default is "log2count".
output	Character. Sample type: "tumor" or "tumor_normal". Default is "tumor".
output_type	Character. Output data type: "tpm", "log2tpm", or "count". Default is "tpm".
annotation	Logical. Whether to perform gene annotation. Default is TRUE.

**Value**

Preprocessed gene expression matrix.

**Author(s)**

Dongqiang Zeng

**Examples**

```
eset_stad <- load_data("eset_stad")
eset <- tcga_rna_preps(
  eset = eset_stad, id_type = "ensembl", input_type = "count",
  output = "tumor", output_type = "tpm", annotation = TRUE
)
```

---

tcga\_stad\_pdata

*TCGA-STAD Clinical and Molecular Annotation Data*

---

**Description**

Clinical, molecular, and signature score data for TCGA stomach adenocarcinoma (STAD) samples. Includes patient demographics, tumor characteristics, molecular subtypes, and pre-computed signature scores.

**Usage**

```
data(tcga_stad_pdata)
```

**Format**

A data frame with samples as rows and variables as columns:

- ID – TCGA sample barcode
- stage – tumor stage (Stage\_I, Stage\_II, etc.)
- status – vital status (Alive/Dead)
- Lauren – histological classification

- subtype – molecular subtype (EBV, MSI, GS, CN, ...)
- EBV – EBV infection status
- TMEscore\_plus – continuous tumor-micro-environment score
- TMEscore\_plus\_binary – High/Low TME classification
- time – follow-up time (months)
- OS\_status – 0 = alive, 1 = dead
- ARID1A, PIK3CA – driver-gene mutation status
- MALAT1 – lncRNA expression
- remaining columns – gene-expression values and additional clinical/molecular annotations

### Source

The Cancer Genome Atlas Stomach Adenocarcinoma (TCGA-STAD)

### References

Cancer Genome Atlas Research Network. Comprehensive molecular characterization of gastric adenocarcinoma. *Nature* 513, 202-209 (2014). doi:10.1038/nature13480

### Examples

```
data(tcga_stad_pdata)
head(tcga_stad_pdata)
```

---

test\_for\_infiltration *Test for Cell Population Infiltration*

---

### Description

Returns p-values for the null hypothesis that samples are not infiltrated by the corresponding cell population.

### Usage

```
test_for_infiltration(MCPcounterMatrix, platform = c("133P2", "133A", "HG1"))
```

### Arguments

MCPcounterMatrix  
Matrix, usually output from MCPcounter.estimate.

platform  
Expression platform: "133P2", "133A", or "HG1". Default is "133P2".

### Value

Matrix with samples in rows and cell populations in columns. Elements are p-values.

**Author(s)**

Etienne Becht

**Examples**

```
# This function requires null_models data which is loaded internally
# Create example data
scores <- matrix(runif(30), nrow = 3, ncol = 10)
rownames(scores) <- c("T cells", "B cells", "NK cells")
pvals <- test_for_infiltration(scores, platform = "133P2")
```

---

timer\_available\_cancers

*TIMER Available Cancer Types*

---

**Description**

Character vector of cancer types supported by TIMER deconvolution. TIMER signatures are cancer-specific.

**Usage**

```
timer_available_cancers
```

**Format**

An object of class character of length 32.

**Value**

Character vector of available cancer type abbreviations.

**Examples**

```
# List all available cancer types for TIMER
timer_available_cancers

# Check if a cancer type is supported
"brca" %in% timer_available_cancers
```

---

`timer_info`*Source code for the TIMER deconvolution method.*

---

**Description**

Formats and displays informational messages for timing or logging purposes. Useful for tracking progress or stages of execution within scripts.

**Usage**

```
timer_info(string)
```

**Arguments**

`string`            Character. Message to be displayed.

**Details**

This code is adapted from <https://github.com/hanfeisun/TIMER>, which again is an adapted version of the original TIMER source code from <http://cistrome.org/TIMER/download.html>.

The method is described in Li et al. *Genome Biology* 2016;17(1):174., PMID 27549193. Display Timer Information Messages

**Value**

None; used for its side effect of printing a message.

**Author(s)**

Bo Li

**Examples**

```
timer_info("Data processing started.")
```

---

`tme_cluster`*Identification of TME Cluster*

---

**Description**

Performs TME (Tumor Microenvironment) clustering analysis using various clustering methods. Supports feature selection, scaling, and automatic determination of optimal cluster number.

**Usage**

```
tme_cluster(  
  input,  
  features = NULL,  
  pattern = NULL,  
  id = NULL,  
  scale = TRUE,  
  method = "kmeans",  
  min_nc = 2,  
  max.nc = 6  
)
```

**Arguments**

input	Data frame containing the input dataset.
features	Vector of features to use for clustering. Default is NULL (uses all columns or pattern-selected columns).
pattern	Regular expression pattern for selecting features. Default is NULL.
id	Column name for identifiers. Default is NULL (uses row names).
scale	Logical indicating whether to scale features. Default is TRUE.
method	Clustering method. Default is "kmeans".
min_nc	Minimum number of clusters to evaluate. Default is 2.
max.nc	Maximum number of clusters to evaluate. Default is 6.

**Value**

Data frame with cluster assignments appended.

**Author(s)**

Dongqiang Zeng

**Examples**

```
set.seed(123)  
input_data <- data.frame(  
  ID = paste0("Sample", 1:20),  
  xCell_Tcells = rnorm(20),  
  xCell_Bcells = rnorm(20),  
  xCell_Macrophages = rnorm(20),  
  Other_feature = rnorm(20)  
)  
result <- tme_cluster(  
  input = input_data,  
  pattern = "xCell",  
  id = "ID",  
  method = "kmeans"  
)  
table(result$cluster)
```

---

`tme_deconvolution_methods`*TME Deconvolution Methods*

---

**Description**

A named vector of supported tumor microenvironment (TME) deconvolution methods in the IOBR package.

**Usage**

```
tme_deconvolution_methods
```

**Format**

A named character vector where names are display names and values are internal method names.

**Details**

The methods currently supported are:

- ‘mcpcounter’: MCP-counter for immune and stromal cell populations
- ‘epic’: EPIC for immune, stromal, and cancer cell fractions
- ‘xcell’: xCell for 64 immune and stromal cell types
- ‘cibersort’: CIBERSORT for 22 immune cell types
- ‘cibersort\_abs’: CIBERSORT in absolute mode
- ‘ips’: Immunophenoscore calculation
- ‘estimate’: ESTIMATE for stromal/immune/estimate scores
- ‘svr’: Support Vector Regression (custom reference)
- ‘lsei’: Least Squares with Equality/Inequality constraints
- ‘timer’: TIMER for cancer-specific immune estimation
- ‘quantiseq’: quanTIseq for RNA-seq immune deconvolution

**Value**

Named character vector of available deconvolution methods.

**Examples**

```
# List all available TME deconvolution methods
tme_deconvolution_methods

# Get internal method name for a specific method
tme_deconvolution_methods["MCPcounter"]
```

---

Top_probe	<i>Top Probe Selector</i>
-----------	---------------------------

---

### Description

Extracts the top ‘i’ probes based on their ordering in the provided data frame. If the number of rows is less than or equal to ‘i’, returns all probes.

### Usage

```
Top_probe(dat, i)
```

### Arguments

dat	Data frame containing a column named "probe".
i	Integer. Number of top probes to return.

### Value

Character vector containing the names of the top ‘i’ probes.

### Examples

```
dat <- data.frame(
  probe = c("Probe1", "Probe2", "Probe3", "Probe4", "Probe5"),
  value = c(5, 3, 2, 4, 1)
)
top_probes <- Top_probe(dat, 3)
print(top_probes)
```

---

transform_data	<i>Transform NA, Inf, or Zero Values in Data</i>
----------------	--

---

### Description

Replaces NA, Inf, or zero values in specified columns of a data frame with a user-defined value or the column mean.

### Usage

```
transform_data(data, feature, data_type = c("NA", "Inf", "zero"), into = 0)
```

**Arguments**

data	Data frame. Input data to be transformed.
feature	Character vector. Column names in 'data' to apply transformation.
data_type	Character. Type of value to replace: "NA", "Inf", or "zero".
into	Value to replace specified type with. Default is 0. If "mean", replaces with column mean (excluding NA/Inf values as appropriate).

**Value**

Data frame with specified transformations applied to selected features.

**Author(s)**

Dongqiang Zeng

**Examples**

```
data_matrix <- data.frame(
  A = c(1, 2, NA, 4, Inf),
  B = c(Inf, 2, 3, 4, 5),
  C = c(0, 0, 0, 1, 2)
)

# Replace NAs with 0
transform_data(data_matrix, feature = c("A", "B"), data_type = "NA")

# Replace Inf values with the mean of the column
transform_data(data_matrix,
  feature = c("A", "B"),
  data_type = "Inf", into = "mean"
)

# Replace zeros with -1 in column C
transform_data(data_matrix, feature = "C", data_type = "zero", into = -1)
```

# Index

## \* datasets

- imvigor210\_pdata, 90
  - null\_models, 112
  - patterns\_to\_na, 118
  - pdata\_stad, 118
  - sig\_group, 149
  - signature\_collection, 141
  - signature\_score\_calculation\_methods, 142
  - stad\_group, 161
  - subgroup\_data, 162
  - tcga\_stad\_pdata, 166
  - timer\_available\_cancers, 168
  - tme\_deconvolution\_methods, 171
- add\_iobr\_mirror, 5
- add\_riskscore, 6
- anno\_eset, 8
- assimilate\_data, 9
- 
- batch\_cor, 10
- batch\_kruskal, 11
- batch\_pcc, 12
- batch\_sig\_surv\_plot, 13
- batch\_surv, 15
- batch\_wilcoxon, 16
- best\_cutoff, 18
- best\_cutoff2, 19
- BinomialAUC, 20
- BinomialModel, 21
- 
- calculate\_break\_month, 24
- calculate\_sig\_score, 24
- calculate\_sig\_score\_integration, 26
- calculate\_sig\_score\_pca, 27
- calculate\_sig\_score\_ssgsea, 28
- calculate\_sig\_score\_zscore, 30
- CalculatePref, 22
- CalculateTimeROC, 23
- cell\_bar\_plot, 31
- 
- check\_cancer\_types, 32
- check\_eset, 33
- CIBERSORT, 34, 116
- clear\_iobr\_cache, 35
- combine\_pd\_eset, 36
- Construct\_con, 37
- ConvertRownameToLoci, 38
- CoreAlg, 38, 116
- count2tpm, 39
- creat\_folder, 41
- 
- deconvo\_cibersort, 44
- deconvo\_epic, 45
- deconvo\_estimate, 46
- deconvo\_ips, 46
- deconvo\_mcpcounter, 47
- deconvo\_quantiseq, 48
- deconvo\_ref, 49
- deconvo\_timer, 50
- deconvo\_tme, 51
- deconvo\_xcell, 53
- deconvolute\_quantiseq.default, 41
- deconvolute\_timer.default, 43
- design\_mytheme, 54
- doPerm, 55, 116
- download\_iobr\_data, 56
- DrawQQPlot, 57
- 
- Enet, 58
- enrichment\_barplot, 58
- EPIC, 60
- eset\_distribution, 62
- estimateScore, 63
- exact\_pvalue, 64
- extract\_sc\_data, 65
- 
- feature\_manipulation, 66
- feature\_select, 67
- filterCommonGenes, 68
- find\_markers\_in\_bulk, 69

- find\_mutations, 70
- find\_outlier\_samples, 72
- find\_variable\_genes, 73
- format\_msigdb, 74
- format\_signatures, 75
  
- generateRef, 76
- generateRef\_DEseq2, 77
- generateRef\_limma, 78
- generateRef\_rnaseq, 79
- generateRef\_seurat, 80
- get\_cols, 83
- get\_cor, 84
- get\_cor\_matrix, 86
- get\_sig\_sc, 88
- GetFractions.Abbas, 81
- getHRandCIfromCoxph, 82
- GetOutlierGenes, 82
  
- high\_var\_fea, 89
  
- imvigor210\_pdata, 90
- iobr\_cor\_plot, 91
- iobr\_deconvo\_pipeline, 94
- iobr\_deg, 95
- iobr\_pca, 97
- IPS\_calculation, 99
- ipsmap, 98
  
- lasso\_select, 100
- limma.dif, 101
- list\_github\_datasets, 102
- list\_iobr\_mirrors, 102
- load\_data, 103
- log2eset, 104
- LR\_cal, 104
  
- make\_mut\_matrix, 105
- mapbw, 107
- mapcolors, 108
- MCPcounter.estimate, 108
- merge\_duplicate, 109
- merge\_eset, 110
- mouse2human\_eset, 111
  
- null\_models, 112
  
- output\_sig, 114
- outputGCT, 113
  
- palettes, 114
- parallel\_doperm, 115
- ParseInputExpression, 117
- patterns\_to\_na, 118
- pdata\_stad, 118
- percent\_bar\_plot, 119
- pie\_chart, 121
- PlotAUC, 122
- plotPurity, 123
- PlotTimeROC, 124
- ProcessingData, 125
- PrognosticAUC, 126
- PrognosticModel, 127
- PrognosticResult, 128
  
- random\_strata\_cells, 129
- rbind\_iobr, 131
- RegressionResult, 132
- remove\_batcheffect, 134
- remove\_duplicate\_genes, 135
- remove\_names, 137
- RemoveBatchEffect, 133
- reset\_iobr\_mirrors, 138
- roc\_time, 138
  
- scale\_matrix, 140
- select\_method, 140
- sig\_box, 144
- sig\_box\_batch, 145
- sig\_forest, 147
- sig\_group, 149
- sig\_gsea, 150
- sig\_heatmap, 153
- sig\_pheatmap, 155
- sig\_roc, 157
- sig\_surv\_plot, 158
- signature\_collection, 141
- signature\_score\_calculation\_methods, 142
- sigScore, 143
- SplitTrainTest, 160
- stad\_group, 161
- subgroup\_data, 162
- subgroup\_survival, 162, 162
- surv\_group, 163
  
- tcga\_rna\_preps, 165
- tcga\_stad\_pdata, 166
- test\_for\_infiltration, 167

timer\_available\_cancers, [168](#)  
timer\_info, [169](#)  
tme\_cluster, [169](#)  
tme\_deconvolution\_methods, [171](#)  
Top\_probe, [172](#)  
transform\_data, [172](#)