Package 'mirmodels'

April 6, 2021

Title Models Built by the Informatics Team at Mirvie

Version 0.1.2

Maintainer Rory Nolan <rory@mirvie.com>

Description We needed a home for our high-quality, reusable, peer-reviewed models. This is where the R models live. The Python models live in the models module in the 'mirpy' package.

License file LICENSE

URL https://gitlab.com/Mirvie/mirmodels

BugReports https://gitlab.com/Mirvie/mirmodels/-/issues

Imports arrow (>= 3.0),
BiocParallel,
broom,
car,
checkmate,
clipr,
DescTools,
DESeq2,
dials,
doFuture,
doParallel,
dplyr,
edgeR,
embed,
forcats,
foreach,
furrr,
future,
generics,
ggplot2,
ggpmisc,
ggrepel,
ggthemes,
glmnet,
glue,
janitor,
magrittr,
MASS,

matrixStats,
methods,
mlbench,
parsnip,
pROC,
purrr,
RcppRoll,
recipes,
rlang,
rrcov,
rsample,
scales,
stats,
strex,
stringr,
tibble,
tidyr,
tune,
usethis,
vip,
withr,
workflows,
xgboost,
yardstick,
zeallot,
zoo

Suggests datasets,
fs,
knitr,
mirmisc,
mockery,
pacman,
readr,
rmarkdown,
spelling,
testthat (>= 3.0)

VignetteBuilder knitr

Config/testthat/edition 3

Config/testthat/parallel true

Encoding UTF-8

Language en-US

LazyData true

Roxygen list(markdown = TRUE)

RoxygenNote 7.1.1

R topics documented:

`autoplot.mirvie_learning_curve` *Produce a learning curve plot from a mirvie_learning_curve object.*

Description

This is a method for `ggplot2::autoplot()`.

Usage

```
## S3 method for class 'mirvie_learning_curve'
autoplot(object, metric = NULL, smooth = FALSE, meansd = FALSE, ...)
```

Arguments

`object`	A mirvie_learning_curve object (i.e. the output of a call to `learn_curve()`).
`metric`	A string. The metric used to evaluate the performance.
`smooth`	A flag. Use a loess smoothed line instead of joining the dots?
`meansd`	A flag. If there are multiple repeats, rather than plotting all of them, plot means with standard deviation error bars?
`...`	Arguments passed to `ggplot2::autoplot()`. Safe to ignore.

Value

A `ggplot2::ggplot()`.

Examples

```
data("BostonHousing", package = "mlbench")
bh <- dplyr::select_if(BostonHousing, is.numeric)
model_evaluate <- function(training_data, testing_data) {
  trained_mod <- lm(medv ~ ., training_data)
  training_preds <- predict(trained_mod, newdata = training_data)
  preds <- predict(trained_mod, newdata = testing_data)
  c(
    train = yardstick::mae_vec(training_data$medv, training_preds),
    test = yardstick::mae_vec(testing_data$medv, preds)
  )
}
mlc <- mlc0 <- suppressWarnings(
  learn_curve(model_evaluate, bh, "medv",
    training_fracs = c(seq(0.1, 0.7, 0.2), 0.85),
    testing_frac = c(0.25, 0.5), repeats = 8,
    strata = "medv"
  )
)
suppressWarnings(print(autoplot(mlc, metric = "mae")))
suppressWarnings(print(autoplot(mlc, metric = "mae", smooth = TRUE)))
suppressWarnings(print(autoplot(mlc, metric = "mae", meansd = TRUE)))
suppressWarnings(
  print(autoplot(mlc, metric = "mae", smooth = TRUE, meansd = TRUE))
)
```

```
mlc <- dplyr::filter(mlc0, testing_frac == 0.25)
suppressWarnings(print(autoplot(mlc, metric = "mae")))
suppressWarnings(print(autoplot(mlc, metric = "mae", smooth = TRUE)))
suppressWarnings(print(autoplot(mlc, metric = "mae", meansd = TRUE)))
suppressWarnings(
  print(autoplot(mlc, metric = "mae", smooth = TRUE, meansd = TRUE))
)
mlc <- dplyr::filter(mlc0, rep == 1)
suppressWarnings(print(autoplot(mlc, metric = "mae")))
suppressWarnings(print(autoplot(mlc, metric = "mae", smooth = TRUE)))
mlc <- dplyr::filter(mlc0, rep == 1, testing_frac == 0.25)
suppressWarnings(print(autoplot(mlc, metric = "mae")))
suppressWarnings(print(autoplot(mlc, metric = "mae", smooth = TRUE)))
bh_split <- rsample::initial_split(bh, strata = medv)
bh_training <- rsample::training(bh_split)
bh_testing <- rsample::testing(bh_split)
mlc <- learn_curve(model_evaluate,
  training_data = bh_training,
  outcome = "medv", testing_data = bh_testing,
  strata = "medv"
)
suppressWarnings(print(autoplot(mlc)))
suppressWarnings(print(autoplot(mlc, smooth = TRUE)))
```

`autoplot.mirvie_learning_curve_cv` *Produce a learning curve plot from a mirvie_learning_curve object.*

Description

This is a method for `ggplot2::autoplot()`.

Usage

```
## S3 method for class 'mirvie_learning_curve_cv'
autoplot(object, metric = NULL, smooth = FALSE, meansd = FALSE, ...)
```

Arguments

`object`	A mirvie_learning_curve_cv object (i.e. the output of a call to `learn_curve_cv()`).
`metric`	A string. The metric used to evaluate the performance.
`smooth`	A flag. Use a loess smoothed line instead of joining the dots?
`meansd`	A flag. If there are multiple repeats, rather than plotting all of them, plot means with standard deviation error bars?
`...`	Arguments passed to `ggplot2::autoplot()`. Safe to ignore.

Value

A `ggplot2::ggplot()`.

Examples

```
data("BostonHousing", package = "mlbench")
bh <- dplyr::select_if(BostonHousing, is.numeric)
mod <- parsnip::linear_reg(penalty = 0, mixture = 0) %>%
  parsnip::set_engine("lm")
wf <- workflows::workflow() %>%
  workflows::add_formula(medv ~ .) %>%
  workflows::add_model(mod)
metric_calculator <- ~ yardstick::mae(., medv, .pred)$.estimate
lccv <- suppressWarnings(
  learn_curve_cv(bh, wf, 2:9, 8, metric_calculator, n_cores = 4)
)
autoplot(lccv, metric = "mae")
autoplot(lccv, metric = "mae", smooth = TRUE)
autoplot(lccv, metric = "mae", meansd = TRUE)
autoplot(lccv, metric = "mae", smooth = TRUE, meansd = TRUE)
```

---

compute_pcas *Compute principal components.*

Description

Computation can be done using the recipes package (`robust = FALSE`, uses regular PCA) or with the rrcov package which uses `rrcov::PcaGrid()`, a robust PCA computation with built-in outlier identification.

Usage

```
compute_pcas(
  df,
  num_comp = 5,
  subset = NULL,
  normalize = TRUE,
  robust = FALSE,
  crit_pca_distances = 0.999
)
```

Arguments

`df`	A data frame.
`num_comp`	A positive integer. The number of PCA components to compute.
`subset`	A character vector. The subset of columns to use for PCA calculation. Default is all columns. Columns in `df` outside of this subset will be kept in the result.
`normalize`	A flag. Center and scale before PCA calculation?
`robust`	A flag. Use robust PCA (from `rrcov::PcaGrid()`)?
`crit_pca_distances`	A number between 0 and 1. Outlier identification parameter. See `rrcov::PcaGrid()`. The larger this number, the fewer outliers identified.

Value

A data frame with the principal components with attributes

- `var_exp`: A numeric vector. The variance explained by each component.
- `loadings`: A data frame. The contribution of each variable to each principal component.
- `outlier`: A logical vector with length `nrow(df)`. `TRUE` for outlier, `FALSE` otherwise. This is present with `robust = TRUE` only.

Examples

```
if (rlang::is_installed("mirmisc")) {
  ga_data <- get_ga_data(
    log2 = TRUE,
    gene_predicate = ~ median(.) > 0
  )
  pca_reg <- compute_pcas(ga_data,
    subset = intersect(
      mirmisc::get_gene_names(),
      names(ga_data)
    ),
    robust = FALSE
  )
  pca_rob <- compute_pcas(ga_data,
    subset = intersect(
      mirmisc::get_gene_names(),
      names(ga_data)
    ),
    robust = TRUE
  )
  ggplot2::ggplot(
    pca_reg,
    ggplot2::aes(PC1, PC2, color = meta_collectionga)
  ) +
    ggplot2::geom_point() +
    ggplot2::scale_color_viridis_c() +
    ggplot2::ggtitle("GAPPS regular")
  ggplot2::ggplot(
    pca_rob,
    ggplot2::aes(PC1, PC2, color = meta_collectionga)
  ) +
    ggplot2::geom_point() +
    ggplot2::scale_color_viridis_c() +
    ggplot2::ggtitle("GAPPS robust")
}
```

---

compute_umap *Compute UMAPs.*

Description

Computation is be done using the `embed` package.

Usage

```
compute_umap(df, num_comp = 2, subset = NULL)
```

Arguments

df	A data frame.
num_comp	A positive integer. The number of UMAP components to compute.
subset	A character vector. The subset of columns to use for UMAP calculation. Default is all columns. Columns in df outside of this subset will be kept in the result.

Value

A data frame with the UMAP components.

Examples

```
if (rlang::is_installed("mirmisc")) {
  ga_data <- get_ga_data(
    log2 = TRUE,
    gene_predicate = ~ median(.) > 0
  )
  genes <- dplyr::intersect(names(ga_data), mirmisc::get_gene_names())
  umap <- compute_umap(ga_data, subset = genes)
  ggplot2::ggplot(
    umap,
    ggplot2::aes(umap_1, umap_2, color = meta_collectionga)
  ) +
    ggplot2::geom_point() +
    ggplot2::scale_color_viridis_c()
}
```

---

conv_traintest_lst_to_rsplit

Convert a list of training_data *and* testing_data *to an* rsplit.

Description

A lot of functions in this package return lists with two elements that are data frames called training_data and testing_data. You might prefer these to be in the form of an rsplit object, such as the return of rsample::initial_split() so that you can access the training and testing parts with rsample::training() and rsample::testing().

Usage

```
conv_traintest_lst_to_rsplit(traintest_lst)
```

Arguments

traintest_lst	A list with two elements training_data and testing_data which are data frames with the same column names.

Value

An `rsplit` object. Something like the return of `rsample::initial_split()`.

`cor_de` *Conduct a differential expression analysis using correlation tests.*

Description

This function is designed to ingest data in the form output by functions like `get_bw_data()`.

Usage

```
cor_de(
  data,
  condition,
  genes = NULL,
  method = c("spearman", "kendall", "pearson"),
  padj_method = stats::p.adjust.methods[1]
)
```

Arguments

`data`	A data frame where rows are samples and columns are genes and metadata.
`condition`	A string. The name of the column in `data` which contains the condition on which the differential expression is premised. This column should be logical, a factor with two levels or a numeric vector. The first level (or `FALSE`) is the baseline and the second level (or `TRUE`) is the disease/treatment. If the condition is numeric, then columns `log2fc`, `base_med`, `case_med`, `base_mean` and `case_mean` will be absent from the result.
`genes`	A character vector. The column names that contain the genes for the differential expression. If `NULL`, any genes in data found in `mirmisc::get_gene_names()` are used.
`method`	A string. The correlation test method. Must be `"spearman"`, `"kendall"` or `"pearson"`.
`padj_method`	A string. The method of adjusting the p-values for multiple hypothesis testing. Must be one of `stats::p.adjust.methods()`.

Details

This function has no way of dealing with batch effects. To allow for these, you should correct your data (with e.g. `linear_correct()`) beforehand.

Value

A tibble with 8 columns:

- `gene`: The gene name.
- `log2fc`: The log2 fold-change between case and control.
- `pvalue`: The p-value for that gene being differentially expressed.

- padj: Adjusted p-values.
- base_med: The median raw value for the baseline samples.
- case_med: The median raw value for the case samples.
- base_mean: The mean raw value for the baseline samples.
- case_mean: The mean raw value for the case samples. For method = "pearson" the log2fc is of the means of the two groups. Otherwise, it is of the medians.

See Also

Other differential expression methods: deseq(), edger()

Examples

```
if (requireNamespace("mirmisc", quietly = TRUE)) {
  bwms_data <- get_combined_cohort_data(
    c("bw", "ms"),
    log2 = TRUE, tot_counts = TRUE,
    gene_predicate = ~ stats::median(.) > 0,
  ) %>%
    dplyr::filter(!is.na(meta_pre_eclampsia)) %>%
    dplyr::mutate(tot_counts = log2(tot_counts + 1))
  genes <- dplyr::intersect(mirmisc::get_gene_names(), names(bwms_data))
  bwms_corr <- linear_correct(
    bwms_data,
    correct_cols = genes,
    correct_for_cols = c("log2_tot_counts", "cohort",
                         "meta_q_pcr_actb_ct", "meta_q_pcr_ercc_ct"),
    keep_effect_cols = "meta_pre_eclampsia",
    cohort_col = "cohort"
  )[[1]]
  cor_de(bwms_corr, "meta_pre_eclampsia")
}
```

---

deseq *Conduct a differential expression analysis using DESeq2::DESeq().*

Description

This function is designed to ingest data in the form output by functions like get_bw_data().

Usage

```
deseq(
  data,
  condition,
  batch = NULL,
  genes = NULL,
  shrink = FALSE,
  padj_method = stats::p.adjust.methods[1],
  quick = FALSE,
  n_cores = 1,
  quiet = FALSE
)
```

Arguments

data	A data frame where rows are samples and columns are genes and metadata.
condition	A string. The name of the column in data which contains the condition on which the differential expression is premised. This column should be logical or a factor with two levels. The first level (or FALSE) is the baseline and the second level (or TRUE) is the disease/treatment.
batch	A string. The name of the column in data specifying the batches that you'd like the model to account for. A common choice is a cohort column.
genes	A character vector. The column names that contain the genes for the differential expression. If NULL, any genes in data found in mirmisc::get_gene_names() are used.
shrink	A flag. If FALSE (the default), DESeq2::results() will be used. Otherwise, DESeq2::lfcShrink() will be used.
padj_method	A string. The method of adjusting the p-values for multiple hypothesis testing. Must be one of stats::p.adjust.methods().
quick	A flag. If TRUE, the base_med, case_med, base_mean and case_mean columns are omitted from the return and the lack of need to calculate these medians offers a small speedup.
n_cores	The number of cores for parallel processing.
quiet	A flag. Suppresses messages.

Value

A tibble with 8 columns:

- gene: The gene name.
- log2fc: The log2 fold-change between case and control.
- pvalue: The p-value for that gene being differentially expressed.
- padj: Adjusted p-values.
- base_med: The median raw value for the baseline samples.
- case_med: The median raw value for the case samples.
- base_mean: The mean raw value for the baseline samples.
- case_mean: The mean raw value for the case samples.

See Also

Other differential expression methods: cor_de(), edger()

Examples

```
bwms_data <- get_combined_cohort_data(
  c("bw", "ms"),
  gene_predicate = ~ stats::median(.) > 0
) %>%
  dplyr::filter(!is.na(meta_pre_eclampsia))
deseq(bwms_data, "meta_pre_eclampsia", batch = "cohort")
```

edger *Conduct a differential expression analysis using* DESeq2::DESeq().

Description

This function is designed to ingest data in the form output by functions like get_bw_data().

Usage

```
edger(
  data,
  condition,
  batch = NULL,
  genes = NULL,
  quick = FALSE,
  padj_method = stats::p.adjust.methods[1]
)
```

Arguments

data	A data frame where rows are samples and columns are genes and metadata.
condition	A string. The name of the column in data which contains the condition on which the differential expression is premised. This column should be logical or a factor with two levels. The first level (or FALSE) is the baseline and the second level (or TRUE) is the disease/treatment.
batch	A string. The name of the column in data specifying the batches that you'd like the model to account for. A common choice is a cohort column.
genes	A character vector. The column names that contain the genes for the differential expression. If NULL, any genes in data found in mirmisc::get_gene_names() are used.
quick	A flag. If TRUE, the base_med, case_med, base_mean and case_mean columns are omitted from the return and the lack of need to calculate these medians offers a small speedup.
padj_method	A string. The method of adjusting the p-values for multiple hypothesis testing. Must be one of stats::p.adjust.methods().

Value

A tibble with 8 columns:

- gene: The gene name.
- log2fc: The log2 fold-change between case and control.
- pvalue: The p-value for that gene being differentially expressed.
- padj: Adjusted p-values.
- base_med: The median raw value for the baseline samples.
- case_med: The median raw value for the case samples.
- base_mean: The mean raw value for the baseline samples.
- case_mean: The mean raw value for the case samples.

See Also

Other differential expression methods: `cor_de()`, `deseq()`

Examples

```
bw_data <- get_bw_data()
edger(bw_data, "meta_pre_eclampsia")
```

`get_bw_data` *Get the Brigham and Women's Hospital data.*

Description

Get the Brigham and Women's Hospital gene count data and associated metadata.

Usage

```
get_bw_data(
  cpm = FALSE,
  log2 = FALSE,
  tot_counts = FALSE,
  remove_bads = TRUE,
  remove_noncoding = TRUE,
  remove_pseudo = TRUE,
  gene_predicate = NULL,
  feather_dir = NULL
)
```

Arguments

`cpm`	A flag. Gene counts in counts per million?
`log2`	A flag. Gene counts on log2 scale? If both `cpm` and `log2` are `TRUE`, then the `log2` calculation comes after the `cpm` calculation. The transformation actually used is `x <-log2(x + 1)`.
`tot_counts`	A flag. Add a column for the total number of counts for this sample? If `log2 = TRUE`, a column `log2_tot_counts` will also be added with `log2_tot_counts = log2(tot_counts + 1)`.
`remove_bads`	A flag. Remove samples that have been flagged as bad for whatever reason. The default is to do so.
`remove_noncoding`	A flag. Remove non-coding genes?
`remove_pseudo`	A flag. Remove psuedogenes?
`gene_predicate`	A predicate function which takes a single numeric argument. Only gene columns returning `TRUE` will be in the output. This will be passed through `rlang::as_function()`, so formula-style lambdas are permitted. For example, to keep only genes whose medians are positive, you can use `gene_predicate = function(x) {median(x) > 0}` or `gene_predicate = ~median(.) > 0`. This calculation is done in the transformed space, so if you select cpm = FALSE, log2 = TRUE, gene_predicate = ~median(.) > 2, you're actually selecting genes with medians greater than 4 in raw space (since `log2(4) == 2`).
`feather_dir`	A string. The path to a directory containing feather files.

Details

- meta_collectionga: The gestational age of the baby at the time of the blood draw (determined by ultrasound).
- meta_deliveryga: The gestational age of the baby at delivery (according to ultrasound).
- meta_weeks_to_delivery: The time to delivery in weeks.
- meta_sample_type: Case or control.
- meta_study_purpose: The purpose of the study. Pre-eclampsia or preterm birth.
- meta_major_race: The race of the mother.
- meta_mom_id: Unique identifier for each mother.
- meta_collection_num: The collection number.
- meta_case_control_match_id: Cases and controls are pairwise-matched on this ID.
- meta_minutes_before_spin: Minutes between blood draw and spin.
- meta_mom_age: The mother's age.
- meta_mom_bmi: The mother's pre-pregnancy BMI.
- meta_sga: Was the baby small for its gestational age at birth?
- meta_pre_eclampsia: Did the mother develop pre-eclampsia?
- meta_baby_weight_g: The weight of the newborn in grams. For twins this is the mean of the two.
- meta_batch_num: The batch number.
- meta_sample_volume_ul: The sample volume in microlitres.
- meta_q_pcr_actb_ct: The qPCR cycling threshold for ACTB.
- meta_q_pcr_ercc_ct: The qPCR cycling threshold for ERCCs.
- meta_q_pcr_ver: The qPCR protocol version.
- meta_lab_qc_passed: Did the sample pass lab QC?
- meta_pca_outlier: Is the sample a PCA outlier (found via robust PCA)?

There is an ID column called mirvie_id, a column called cohort which is just the first two characters in mirvie_id, and the rest of the data is gene counts.

See Also

Other cohort data: get_dp_data(), get_ga_data(), get_io_data(), get_kl_data(), get_ms_data(), get_ot_data(), get_pf_data(), get_pg_data(), get_pi_data(), get_pm_data(), get_pt_data(), get_rs_data(), get_st_data(), get_up_data(), get_vg_data()

---

get_combined_cohort_data

Retrieve the data from many cohorts combined.

Description

Only columns present in all of the cohorts will be kept.

Usage

```
get_combined_cohort_data(
  cohorts = c("bw", "ga", "io", "kl", "ms", "ot", "pf", "pg", "pi", "pm", "pt", "rs",
    "st", "up", "vg"),
  cpm = FALSE,
  log2 = FALSE,
  tot_counts = FALSE,
  remove_bads = TRUE,
  col_combine = c("intersect", "union"),
  remove_noncoding = TRUE,
  remove_pseudo = TRUE,
  gene_predicate = NULL,
  feather_dir = NULL,
  n_cores = 1
)
```

Arguments

cohorts	A character vector. A subset of c("bw","ga","io","kl","ms","ot","pf","pg","pi","pm","pt'
cpm	A flag. Gene counts in counts per million?
log2	A flag. Gene counts on log2 scale? If both cpm and log2 are TRUE, then the log2 calculation comes after the cpm calculation. The transformation actually used is x <-log2(x + 1).
tot_counts	A flag. Add a column for the total number of counts for this sample? If log2 = TRUE, a column log2_tot_counts will also be added with log2_tot_counts = log2(tot_counts + 1).
remove_bads	A flag. Remove samples that have been flagged as bad for whatever reason. The default is to do so.
col_combine	A string. Either "intersect" (the default) or "union". Different cohorts have different metadata columns. This parameter specifies how to deal with this. With "intersect", only columns present in all cohorts are kept, whereas with "union", all columns are kept, using NAs to fill.
remove_noncoding	A flag. Remove non-coding genes?
remove_pseudo	A flag. Remove psuedogenes?
gene_predicate	A predicate function which takes a single numeric argument. Only gene columns returning TRUE will be in the output. This will be passed through rlang::as_function(), so formula-style lambdas are permitted. For example, to keep only genes whose medians are positive, you can use gene_predicate = function(x) {median(x) > 0} or gene_predicate = ~median(.) > 0. This calculation is done in the transformed space, so if you select cpm = FALSE, log2 = TRUE, gene_predicate = ~median(.) > 2, you're actually selecting genes with medians greater than 4 in raw space (since log2(4) == 2).
feather_dir	A string. The path to a directory containing feather files.
n_cores	A positive integer. Number of cores to read each of the cohorts in parallel.

Value

A data frame.

get_dp_data *Get the DTPR data.*

Description

Get the DTPR gene count data and associated metadata.

Usage

```
get_dp_data(
  cpm = FALSE,
  log2 = FALSE,
  tot_counts = FALSE,
  remove_bads = TRUE,
  remove_noncoding = TRUE,
  remove_pseudo = TRUE,
  gene_predicate = NULL,
  feather_dir = NULL
)
```

Arguments

cpm	A flag. Gene counts in counts per million?
log2	A flag. Gene counts on log2 scale? If both cpm and log2 are TRUE, then the log2 calculation comes after the cpm calculation. The transformation actually used is x <-log2(x + 1).
tot_counts	A flag. Add a column for the total number of counts for this sample? If log2 = TRUE, a column log2_tot_counts will also be added with log2_tot_counts = log2(tot_counts + 1).
remove_bads	A flag. Remove samples that have been flagged as bad for whatever reason. The default is to do so.
remove_noncoding	A flag. Remove non-coding genes?
remove_pseudo	A flag. Remove psuedogenes?
gene_predicate	A predicate function which takes a single numeric argument. Only gene columns returning TRUE will be in the output. This will be passed through rlang::as_function(), so formula-style lambdas are permitted. For example, to keep only genes whose medians are positive, you can use gene_predicate = function(x) {median(x) > 0} or gene_predicate = ~median(.) > 0. This calculation is done in the transformed space, so if you select cpm = FALSE, log2 = TRUE, gene_predicate = ~median(.) > 2, you're actually selecting genes with medians greater than 4 in raw space (since log2(4) == 2).
feather_dir	A string. The path to a directory containing feather files.

Details

- meta_collectionga: The gestational age of the baby at the time of the blood draw (determined by ultrasound).

There is an ID column called mirvie_id, a column called cohort which is just the first two characters in mirvie_id, and the rest of the data is gene counts.

See Also

Other cohort data: get_bw_data(), get_ga_data(), get_io_data(), get_kl_data(), get_ms_data(), get_ot_data(), get_pf_data(), get_pg_data(), get_pi_data(), get_pm_data(), get_pt_data(), get_rs_data(), get_st_data(), get_up_data(), get_vg_data()

`get_ga_data` *Get the GAPPS data.*

Description

Get the GAPPS gene count data and associated metadata.

Usage

```
get_ga_data(
  cpm = FALSE,
  log2 = FALSE,
  tot_counts = FALSE,
  remove_bads = TRUE,
  remove_noncoding = TRUE,
  remove_pseudo = TRUE,
  gene_predicate = NULL,
  feather_dir = NULL
)
```

Arguments

cpm	A flag. Gene counts in counts per million?
log2	A flag. Gene counts on log2 scale? If both cpm and log2 are TRUE, then the log2 calculation comes after the cpm calculation. The transformation actually used is x <-log2(x + 1).
tot_counts	A flag. Add a column for the total number of counts for this sample? If log2 = TRUE, a column log2_tot_counts will also be added with log2_tot_counts = log2(tot_counts + 1).
remove_bads	A flag. Remove samples that have been flagged as bad for whatever reason. The default is to do so.
remove_noncoding	A flag. Remove non-coding genes?
remove_pseudo	A flag. Remove psuedogenes?
gene_predicate	A predicate function which takes a single numeric argument. Only gene columns returning TRUE will be in the output. This will be passed through rlang::as_function(), so formula-style lambdas are permitted. For example, to keep only genes whose medians are positive, you can use gene_predicate = function(x) {median(x) > 0} or gene_predicate = ~median(.) > 0. This calculation is done in the transformed space, so if you select cpm = FALSE, log2 = TRUE, gene_predicate = ~median(.) > 2, you're actually selecting genes with medians greater than 4 in raw space (since log2(4) == 2).
feather_dir	A string. The path to a directory containing feather files.

Details

- meta_mom_id: Unique identifier for each mother.
- meta_major_race: The race of the mother.
- meta_ethnicity: The mother's ethnicity.
- meta_mom_age: The mother's age.
- meta_deliveryga: The gestational age of the baby at delivery (according to ultrasound).
- meta_parity: The number of previous pregnancies that lasted at least 24 months.
- meta_prom: Was there a premature rupture of membrane?
- meta_pprom: Was there preterm premature rupture of membranes?
- meta_iugr: Was there intrauterine growth restriction?
- meta_pre_eclampsia: Did the mother develop pre-eclampsia?
- meta_smoker: Is the mother a smoker?
- meta_labor_spontaneous: Was the labor spontaneous?
- meta_labor_augmented: Was the labor augmented?
- meta_labor_induced_complication: Was the labor induced due to a complication?
- meta_labor_induced_elective: Was the labor electively induced?
- meta_ivf: Was it an IVF conception?
- meta_chronic_hypertension: Did the mother develop chronic hypertension?
- meta_collectionga: The gestational age of the baby at the time of the blood draw (determined by ultrasound).
- meta_minutes_before_spin: Minutes between blood draw and spin.
- meta_n_prior_term_deliveries: The number of prior term deliveries that the mother has had.
- meta_n_prior_preterm_deliveries: The number of prior preterm deliveries that the mother has had.
- meta_baby_sex: The sex of the baby.
- meta_baby_weight_g: The weight of the newborn in grams. For twins this is the mean of the two.
- meta_weeks_to_delivery: The time to delivery in weeks.
- meta_sample_volume_ul: The sample volume in microlitres.
- meta_q_pcr_actb_ct: The qPCR cycling threshold for ACTB.
- meta_q_pcr_ercc_ct: The qPCR cycling threshold for ERCCs.
- meta_pca_outlier: Is the sample a PCA outlier (found via robust PCA)?

There is an ID column called mirvie_id, a column called cohort which is just the first two characters in mirvie_id, and the rest of the data is gene counts.

This contains samples from the GA, PG (preterm GAPPS) and VG (validation GAPPS) cohorts.

See Also

Other cohort data: get_bw_data(), get_dp_data(), get_io_data(), get_kl_data(), get_ms_data(), get_ot_data(), get_pf_data(), get_pg_data(), get_pi_data(), get_pm_data(), get_pt_data(), get_rs_data(), get_st_data(), get_up_data(), get_vg_data()

get_io_data *Get the Iowa data.*

Description

Get the Iowa gene count data and associated metadata.

Usage

```
get_io_data(
  cpm = FALSE,
  log2 = FALSE,
  tot_counts = FALSE,
  remove_bads = TRUE,
  remove_noncoding = TRUE,
  remove_pseudo = TRUE,
  gene_predicate = NULL,
  feather_dir = NULL
)
```

Arguments

cpm	A flag. Gene counts in counts per million?
log2	A flag. Gene counts on log2 scale? If both cpm and log2 are TRUE, then the log2 calculation comes after the cpm calculation. The transformation actually used is x <-log2(x + 1).
tot_counts	A flag. Add a column for the total number of counts for this sample? If log2 = TRUE, a column log2_tot_counts will also be added with log2_tot_counts = log2(tot_counts + 1).
remove_bads	A flag. Remove samples that have been flagged as bad for whatever reason. The default is to do so.
remove_noncoding	A flag. Remove non-coding genes?
remove_pseudo	A flag. Remove psuedogenes?
gene_predicate	A predicate function which takes a single numeric argument. Only gene columns returning TRUE will be in the output. This will be passed through rlang::as_function(), so formula-style lambdas are permitted. For example, to keep only genes whose medians are positive, you can use gene_predicate = function(x) {median(x) > 0} or gene_predicate = ~median(.) > 0. This calculation is done in the transformed space, so if you select cpm = FALSE, log2 = TRUE, gene_predicate = ~median(.) > 2, you're actually selecting genes with medians greater than 4 in raw space (since log2(4) == 2).
feather_dir	A string. The path to a directory containing feather files.

Details

- meta_mom_id: Unique identifier for each mother.
- meta_sample_type: Case or control.
- meta_major_race: The race of the mother.

- meta_mom_bmi: The mother's pre-pregnancy BMI.
- meta_smoker: Is the mother a smoker?
- meta_pre_eclampsia: Did the mother develop pre-eclampsia?
- meta_parity: The number of previous pregnancies that lasted at least 24 months.
- meta_labor_spontaneous: Was the labor spontaneous?
- meta_baby_weight_g: The weight of the newborn in grams. For twins this is the mean of the two.
- meta_baby_length_cm: The length of the baby in centimetres. For twins this is the mean of the two.
- meta_stillbirth: Was it a stillbirth?
- meta_collectionga: The gestational age of the baby at the time of the blood draw (determined by ultrasound).
- meta_deliveryga: The gestational age of the baby at delivery (according to ultrasound).
- meta_weeks_to_delivery: The time to delivery in weeks.
- meta_mom_age: The mother's age.
- meta_caesarean: Was the birth by C-section?
- meta_q_pcr_actb_ct: The qPCR cycling threshold for ACTB.
- meta_q_pcr_ercc_ct: The qPCR cycling threshold for ERCCs.
- meta_pca_outlier: Is the sample a PCA outlier (found via robust PCA)?

There is an ID column called mirvie_id, a column called cohort which is just the first two characters in mirvie_id, and the rest of the data is gene counts.

See Also

Other cohort data: get_bw_data(), get_dp_data(), get_ga_data(), get_kl_data(), get_ms_data(), get_ot_data(), get_pf_data(), get_pg_data(), get_pi_data(), get_pm_data(), get_pt_data(), get_rs_data(), get_st_data(), get_up_data(), get_vg_data()

---

get_kl_data *Get the King's College London data.*

Description

Get the King's College London gene count data and associated metadata.

Usage

```
get_kl_data(
  cpm = FALSE,
  log2 = FALSE,
  tot_counts = FALSE,
  remove_bads = TRUE,
  remove_noncoding = TRUE,
  remove_pseudo = TRUE,
  gene_predicate = NULL,
  feather_dir = NULL
)
```

Arguments

cpm	A flag. Gene counts in counts per million?
log2	A flag. Gene counts on log2 scale? If both cpm and log2 are TRUE, then the log2 calculation comes after the cpm calculation. The transformation actually used is x <-log2(x + 1).
tot_counts	A flag. Add a column for the total number of counts for this sample? If log2 = TRUE, a column log2_tot_counts will also be added with log2_tot_counts = log2(tot_counts + 1).
remove_bads	A flag. Remove samples that have been flagged as bad for whatever reason. The default is to do so.
remove_noncoding	A flag. Remove non-coding genes?
remove_pseudo	A flag. Remove psuedogenes?
gene_predicate	A predicate function which takes a single numeric argument. Only gene columns returning TRUE will be in the output. This will be passed through rlang::as_function(), so formula-style lambdas are permitted. For example, to keep only genes whose medians are positive, you can use gene_predicate = function(x) {median(x) > 0} or gene_predicate = ~median(.) > 0. This calculation is done in the transformed space, so if you select cpm = FALSE, log2 = TRUE, gene_predicate = ~median(.) > 2, you're actually selecting genes with medians greater than 4 in raw space (since log2(4) == 2).
feather_dir	A string. The path to a directory containing feather files.

Details

- meta_mom_id: Unique identifier for each mother.
- meta_major_race: The race of the mother.
- meta_sample_type: Case or control.
- meta_case_control_match_id: Cases and controls are pairwise-matched on this ID.
- meta_prev_pprom: Has the mother had a previous preterm premature rupture of membranes?
- meta_prev_late_miscarriage: Has the mother had a previous late miscarriage?
- meta_prev_cervical_surgery: Has the mother had a previous cervical surgery?
- meta_low_risk_at_enrollment: Was the pregnancy classed as low-risk at enrollment?
- meta_type_2_diabetes: Does the mother have type 2 diabetes?
- meta_autoimmune_disease: Does the mother have an auto-immune disease?
- meta_chronic_viral_infection: Does the mother have a chronic viral infection?
- meta_antihypertensives: Is the mother taking antihypertensives?
- meta_immunosuppressants: Is the mother taking immunosuppressants?
- meta_mom_height_cm: The mother's pre-pregnancy height in centimetres.
- meta_mom_weight_kg: The mother's pre-pregnancy weight in kilograms.
- meta_mom_bmi: The mother's pre-pregnancy BMI.
- meta_mom_age: The mother's age.
- meta_ethnicity: The mother's ethnicity.
- meta_bv_history: Does the mother have a history of bacterial vaginosis?

- meta_smoker: Is the mother a smoker? Levels are "current", "ex" and "never".
- meta_first_pregnancy: Is this the mother's first pregnancy?
- meta_n_prior_preg_0_13: Number of prior pregnancies lasting 0-13 weeks.
- meta_n_prior_preg_14_23: Number of prior pregnancies lasting 14-23 weeks.
- meta_n_prior_preg_24_plus: Number of prior pregnancies lasting at least 24 weeks.
- meta_n_prior_preg_37_plus: Number of prior pregnancies lasting at least 37 weeks.
- meta_tocolysis: Is the mother on medication to suppress early labor?
- meta_steroids_fetal_lung: Is the mother taking steroids for fetal lung development?
- meta_antibiotics: Is the mother on antibiotics?
- meta_progesterone: Is the mother on progesterone supplements?
- meta_emergengy_cerclage: Did the mother receive emergency cerclage (an effort to prolong pregnancy).
- meta_mag_sulph: Did the mother take magnesium sulphate (it's prescribed during pregnancy to prevent seizures in women with pre-eclampsia).
- meta_maternal_pyrexia: Did the mother develop a fever during pregnancy?
- meta_crp_highest_value: CRP is a marker for inflammation.
- meta_wcc_highest_value: White cell count in g/dL.
- meta_blood_loss_ml: Blood loss during labor.
- meta_threatened_preterm_labor: Did labor threaten prematurely?
- meta_pprom: Was there preterm premature rupture of membranes?
- meta_obstetric_cholestasis: Obstetric cholestasis is a disorder affecting the liver during pregnancy.
- meta_antepartum_haemorrhage: Was there antepartum haemorrhaging?
- meta_vital_status: The birth status ("Live birth", "Stillbirth" or "Late miscarriage").
- meta_prev_late_miscarriage: Did the pregnancy end in miscarriage?
- meta_stillbirth: Was it a stillbirth?
- meta_baby_sex: The sex of the baby.
- meta_baby_weight_g: The weight of the newborn in grams. For twins this is the mean of the two.
- meta_apgar: Health rating of the baby at birth. Scale: 1-10. Higher is better.
- meta_sbcu_or_nicu: Did the baby go into some sort of ICU?
- meta_neonatal_inpatient_nights: Number of neonatal inpatient nights.
- meta_hrs_rom_to_delivery: Hours between rupture of membrane and delivery.
- meta_minutes_before_spin: Minutes between blood draw and spin.
- meta_collectionga: The gestational age of the baby at the time of the blood draw (determined by ultrasound).
- meta_deliveryga: The gestational age of the baby at delivery (according to ultrasound).
- meta_weeks_to_delivery: The time to delivery in weeks.
- meta_study_purpose: The purpose of the study. IUGR, pre-eclampsia or preterm birth.
- meta_mom_gdm: Did the mother develop gestational diabetes?
- meta_pre_eclampsia: Did the mother develop pre-eclampsia?

- meta_chronic_hypertension: Did the mother develop chronic hypertension?
- meta_mom_gbs: Was the mother infected with group B strep?
- meta_non_gdm_diabetic: Is the mother a non-GDM diabetic?
- meta_normal_delivery: Was it a normal delivery?
- meta_labor_spontaneous: Was the labor spontaneous?
- meta_n_prior_preterm_deliveries: The number of prior preterm deliveries that the mother has had.
- meta_q_pcr_actb_ct: The qPCR cycling threshold for ACTB.
- meta_q_pcr_ercc_ct: The qPCR cycling threshold for ERCCs.
- meta_q_pcr_ver: The qPCR protocol version.
- meta_lab_qc_passed: Did the sample pass lab QC?
- meta_pca_outlier: Is the sample a PCA outlier (found via robust PCA)?

There is an ID column called mirvie_id, a column called cohort which is just the first two characters in mirvie_id, and the rest of the data is gene counts.

See Also

Other cohort data: get_bw_data(), get_dp_data(), get_ga_data(), get_io_data(), get_ms_data(), get_ot_data(), get_pf_data(), get_pg_data(), get_pi_data(), get_pm_data(), get_pt_data(), get_rs_data(), get_st_data(), get_up_data(), get_vg_data()

---

get_ms_data *Get the Michigan State University data.*

Description

Get the MSU gene count data and associated metadata.

Usage

```
get_ms_data(
  cpm = FALSE,
  log2 = FALSE,
  tot_counts = FALSE,
  remove_bads = TRUE,
  remove_noncoding = TRUE,
  remove_pseudo = TRUE,
  gene_predicate = NULL,
  feather_dir = NULL
)
```

Arguments

cpm	A flag. Gene counts in counts per million?
log2	A flag. Gene counts on log2 scale? If both cpm and log2 are TRUE, then the log2 calculation comes after the cpm calculation. The transformation actually used is x <-log2(x + 1).

tot_counts	A flag. Add a column for the total number of counts for this sample? If log2 = TRUE, a column log2_tot_counts will also be added with log2_tot_counts = log2(tot_counts + 1).
remove_bads	A flag. Remove samples that have been flagged as bad for whatever reason. The default is to do so.
remove_noncoding	A flag. Remove non-coding genes?
remove_pseudo	A flag. Remove psuedogenes?
gene_predicate	A predicate function which takes a single numeric argument. Only gene columns returning TRUE will be in the output. This will be passed through rlang::as_function(), so formula-style lambdas are permitted. For example, to keep only genes whose medians are positive, you can use gene_predicate = function(x) {median(x) > 0} or gene_predicate = ~median(.) > 0. This calculation is done in the transformed space, so if you select cpm = FALSE, log2 = TRUE, gene_predicate = ~median(.) > 2, you're actually selecting genes with medians greater than 4 in raw space (since log2(4) == 2).
feather_dir	A string. The path to a directory containing feather files.

Details

- meta_batch_num: The batch number.
- meta_mom_id: Unique identifier for each mother.
- meta_major_race: The race of the mother.
- meta_sample_type: Case or control.
- meta_case_control_match_id: Cases and controls are pairwise-matched on this ID.
- meta_study_purpose: The purpose of the study.
- meta_collectionga: The gestational age of the baby at the time of the blood draw (determined by ultrasound).
- meta_deliveryga: The gestational age of the baby at delivery (according to ultrasound).
- meta_weeks_to_delivery: The time to delivery in weeks.
- meta_education: The education level of the mother.
- meta_pe_htn: Information about pre-eclampsia and hypertension on a sliding scale. Levels are 0_none, 1_chronic_htn, 2_severe_chronic_htn, 3_pi_htn, 4_severe_pi_htn, 5_pe, 6_severe_pe, 7_pe_and_chronic_htn, 8_pe_and_severe_chronic_htn.
- meta_gravida: The number of previous pregnancies.
- meta_parity: The number of previous pregnancies that lasted at least 24 months.
- meta_n_prior_term_deliveries: The number of prior term deliveries that the mother has had.
- meta_n_prior_preterm_deliveries: The number of prior preterm deliveries that the mother has had.
- meta_baby_weight_g: The weight of the newborn in grams. For twins this is the mean of the two.
- meta_prenatal_screening_ga: The gestational age at the time of prenatal screening.
- meta_mom_bmi: The mother's pre-pregnancy BMI.
- meta_mom_age: The mother's age.
- meta_baby_sex: The sex of the baby.

- meta_ga_method: The method of determining the gestational age.
- meta_conception_spontaneous: Was the conception medically assisted?
- meta_sga: Was the baby small for its gestational age at birth?
- meta_mom_gdm: Did the mother develop gestational diabetes?
- meta_lupus: Did the mother develop lupus during the pregnancy?
- meta_chronic_hypertension: Did the mother develop chronic hypertension?
- meta_pre_eclampsia: Did the mother develop pre-eclampsia?
- meta_antibiotics: Is the mother on antibiotics?
- meta_stillbirth: Was it a stillbirth?
- meta_caesarean: Was the birth by C-section?
- meta_pprom: Was there preterm premature rupture of membranes?
- meta_labor_spontaneous: Was the labor spontaneous?
- meta_labor_induced_complication: Was the labor induced due to a complication?
- meta_sample_volume_ul: The sample volume in microlitres.
- meta_q_pcr_actb_ct: The qPCR cycling threshold for ACTB.
- meta_q_pcr_ercc_ct: The qPCR cycling threshold for ERCCs.
- meta_lab_qc_passed: Did the sample pass lab QC?
- meta_pca_outlier: Is the sample a PCA outlier (found via robust PCA)?

There is an ID column called mirvie_id, a column called cohort which is just the first two characters in mirvie_id, and the rest of the data is gene counts.

See Also

Other cohort data: get_bw_data(), get_dp_data(), get_ga_data(), get_io_data(), get_kl_data(), get_ot_data(), get_pf_data(), get_pg_data(), get_pi_data(), get_pm_data(), get_pt_data(), get_rs_data(), get_st_data(), get_up_data(), get_vg_data()

---

get_ot_data *Get the First Trimester Gestational Age data.*

Description

Get the First Trimester Gestational Age gene count data and associated metadata.

Usage

```
get_ot_data(
  cpm = FALSE,
  log2 = FALSE,
  tot_counts = FALSE,
  remove_bads = TRUE,
  remove_noncoding = TRUE,
  remove_pseudo = TRUE,
  gene_predicate = NULL,
  feather_dir = NULL
)
```

Arguments

cpm	A flag. Gene counts in counts per million?
log2	A flag. Gene counts on log2 scale? If both cpm and log2 are TRUE, then the log2 calculation comes after the cpm calculation. The transformation actually used is x <-log2(x + 1).
tot_counts	A flag. Add a column for the total number of counts for this sample? If log2 = TRUE, a column log2_tot_counts will also be added with log2_tot_counts = log2(tot_counts + 1).
remove_bads	A flag. Remove samples that have been flagged as bad for whatever reason. The default is to do so.
remove_noncoding	A flag. Remove non-coding genes?
remove_pseudo	A flag. Remove psuedogenes?
gene_predicate	A predicate function which takes a single numeric argument. Only gene columns returning TRUE will be in the output. This will be passed through rlang::as_function(), so formula-style lambdas are permitted. For example, to keep only genes whose medians are positive, you can use gene_predicate = function(x) {median(x) > 0} or gene_predicate = ~median(.) > 0. This calculation is done in the transformed space, so if you select cpm = FALSE, log2 = TRUE, gene_predicate = ~median(.) > 2, you're actually selecting genes with medians greater than 4 in raw space (since log2(4) == 2).
feather_dir	A string. The path to a directory containing feather files.

Details

- meta_batch_num: The batch number.
- meta_mom_id: Unique identifier for each mother.
- meta_collectionga: The gestational age of the baby at the time of the blood draw (determined by ultrasound).
- meta_sample_volume_ul: The sample volume in microlitres.
- meta_q_pcr_actb_ct: The qPCR cycling threshold for ACTB.
- meta_q_pcr_ercc_ct: The qPCR cycling threshold for ERCCs.
- meta_lab_qc_passed: Did the sample pass lab QC?
- meta_pca_outlier: Is the sample a PCA outlier (found via robust PCA)?

There is an ID column called mirvie_id, a column called cohort which is just the first two characters in mirvie_id, and the rest of the data is gene counts.

See Also

Other cohort data: get_bw_data(), get_dp_data(), get_ga_data(), get_io_data(), get_kl_data(), get_ms_data(), get_pf_data(), get_pg_data(), get_pi_data(), get_pm_data(), get_pt_data(), get_rs_data(), get_st_data(), get_up_data(), get_vg_data()

`get_pf_data` *Get the Pittsburgh Four data.*

Description

Get the Pittsburgh Four gene count data and associated metadata.

Usage

```
get_pf_data(
  cpm = FALSE,
  log2 = FALSE,
  tot_counts = FALSE,
  remove_bads = TRUE,
  remove_noncoding = TRUE,
  remove_pseudo = TRUE,
  gene_predicate = NULL,
  feather_dir = NULL
)
```

Arguments

`cpm`	A flag. Gene counts in counts per million?
`log2`	A flag. Gene counts on log2 scale? If both `cpm` and `log2` are `TRUE`, then the `log2` calculation comes after the `cpm` calculation. The transformation actually used is `x <-log2(x + 1)`.
`tot_counts`	A flag. Add a column for the total number of counts for this sample? If `log2 = TRUE`, a column `log2_tot_counts` will also be added with `log2_tot_counts = log2(tot_counts + 1)`.
`remove_bads`	A flag. Remove samples that have been flagged as bad for whatever reason. The default is to do so.
`remove_noncoding`	A flag. Remove non-coding genes?
`remove_pseudo`	A flag. Remove psuedogenes?
`gene_predicate`	A predicate function which takes a single numeric argument. Only gene columns returning `TRUE` will be in the output. This will be passed through `rlang::as_function()`, so formula-style lambdas are permitted. For example, to keep only genes whose medians are positive, you can use `gene_predicate = function(x) {median(x) > 0}` or `gene_predicate = ~median(.) > 0`. This calculation is done in the transformed space, so if you select cpm = FALSE, log2 = TRUE, gene_predicate = ~median(.) > 2, you're actually selecting genes with medians greater than 4 in raw space (since `log2(4) == 2`).
`feather_dir`	A string. The path to a directory containing feather files.

Details

- `meta_batch_num`: The batch number.
- `meta_collectionga`: The gestational age of the baby at the time of the blood draw (determined by ultrasound).

- meta_deliveryga: The gestational age of the baby at delivery (according to ultrasound).
- meta_weeks_to_delivery: The time to delivery in weeks.
- meta_sample_type: Case or control.
- meta_study_purpose: The purpose of the study.
- meta_mom_id: Unique identifier for each mother.
- meta_case_control_match_id: Cases and controls are pairwise-matched on this ID.
- meta_gravida: The number of previous pregnancies.
- meta_parity: The number of previous pregnancies that lasted at least 24 months.
- meta_mom_bmi: The mother's pre-pregnancy BMI.
- meta_mom_age: The mother's age.
- meta_pre_eclampsia: Did the mother develop pre-eclampsia?
- meta_sample_volume_ul: The sample volume in microlitres.
- meta_q_pcr_actb_ct: The qPCR cycling threshold for ACTB.
- meta_q_pcr_ercc_ct: The qPCR cycling threshold for ERCCs.
- meta_pca_outlier: Is the sample a PCA outlier (found via robust PCA)?

There is an ID column called mirvie_id, a column called cohort which is just the first two characters in mirvie_id, and the rest of the data is gene counts.

See Also

Other cohort data: get_bw_data(), get_dp_data(), get_ga_data(), get_io_data(), get_kl_data(), get_ms_data(), get_ot_data(), get_pg_data(), get_pi_data(), get_pm_data(), get_pt_data(), get_rs_data(), get_st_data(), get_up_data(), get_vg_data()

---

get_pg_data *Get the Preterm GAPPS data.*

Description

Get the Preterm GAPPS gene count data and associated metadata.

Usage

```
get_pg_data(
  cpm = FALSE,
  log2 = FALSE,
  tot_counts = FALSE,
  remove_bads = TRUE,
  remove_noncoding = TRUE,
  remove_pseudo = TRUE,
  gene_predicate = NULL,
  feather_dir = NULL
)
```

Arguments

cpm	A flag. Gene counts in counts per million?
log2	A flag. Gene counts on log2 scale? If both cpm and log2 are TRUE, then the log2 calculation comes after the cpm calculation. The transformation actually used is x <-log2(x + 1).
tot_counts	A flag. Add a column for the total number of counts for this sample? If log2 = TRUE, a column log2_tot_counts will also be added with log2_tot_counts = log2(tot_counts + 1).
remove_bads	A flag. Remove samples that have been flagged as bad for whatever reason. The default is to do so.
remove_noncoding	A flag. Remove non-coding genes?
remove_pseudo	A flag. Remove psuedogenes?
gene_predicate	A predicate function which takes a single numeric argument. Only gene columns returning TRUE will be in the output. This will be passed through rlang::as_function(), so formula-style lambdas are permitted. For example, to keep only genes whose medians are positive, you can use gene_predicate = function(x) {median(x) > 0} or gene_predicate = ~median(.) > 0. This calculation is done in the transformed space, so if you select cpm = FALSE, log2 = TRUE, gene_predicate = ~median(.) > 2, you're actually selecting genes with medians greater than 4 in raw space (since log2(4) == 2).
feather_dir	A string. The path to a directory containing feather files.

Details

- meta_major_race: The race of the mother.
- meta_ethnicity: The mother's ethnicity.
- meta_deliveryga: The gestational age of the baby at delivery (according to ultrasound).
- meta_mom_age: The mother's age.
- meta_prom: Was there a premature rupture of membrane?
- meta_pprom: Was there preterm premature rupture of membranes?
- meta_normal_delivery: Was it a normal delivery?
- meta_caesarean: Was the birth by C-section?
- meta_smoker: Is the mother a smoker?
- meta_labor_spontaneous: Was the labor spontaneous?
- meta_baby_sex: The sex of the baby.
- meta_collectionga: The gestational age of the baby at the time of the blood draw (determined by ultrasound).
- meta_weeks_to_delivery: The time to delivery in weeks.
- meta_minutes_before_spin: Minutes between blood draw and spin.
- meta_sample_volume_ul: The sample volume in microlitres.
- meta_q_pcr_actb_ct: The qPCR cycling threshold for ACTB.
- meta_q_pcr_ercc_ct: The qPCR cycling threshold for ERCCs.
- meta_pca_outlier: Is the sample a PCA outlier (found via robust PCA)?

There is an ID column called mirvie_id, a column called cohort which is just the first two characters in mirvie_id, and the rest of the data is gene counts.

See Also

Other cohort data: get_bw_data(), get_dp_data(), get_ga_data(), get_io_data(), get_kl_data(), get_ms_data(), get_ot_data(), get_pf_data(), get_pi_data(), get_pm_data(), get_pt_data(), get_rs_data(), get_st_data(), get_up_data(), get_vg_data()

`get_pi_data` *Get the Pittsburgh Imminent Delivery data.*

Description

Get the Pittsburgh Imminent Delivery gene count data and associated metadata.

Usage

```
get_pi_data(
  cpm = FALSE,
  log2 = FALSE,
  tot_counts = FALSE,
  remove_bads = TRUE,
  remove_noncoding = TRUE,
  remove_pseudo = TRUE,
  gene_predicate = NULL,
  feather_dir = NULL
)
```

Arguments

cpm	A flag. Gene counts in counts per million?
log2	A flag. Gene counts on log2 scale? If both cpm and log2 are TRUE, then the log2 calculation comes after the cpm calculation. The transformation actually used is x <-log2(x + 1).
tot_counts	A flag. Add a column for the total number of counts for this sample? If log2 = TRUE, a column log2_tot_counts will also be added with log2_tot_counts = log2(tot_counts + 1).
remove_bads	A flag. Remove samples that have been flagged as bad for whatever reason. The default is to do so.
remove_noncoding	A flag. Remove non-coding genes?
remove_pseudo	A flag. Remove psuedogenes?
gene_predicate	A predicate function which takes a single numeric argument. Only gene columns returning TRUE will be in the output. This will be passed through rlang::as_function(), so formula-style lambdas are permitted. For example, to keep only genes whose medians are positive, you can use gene_predicate = function(x) {median(x) > 0} or gene_predicate = ~median(.) > 0. This calculation is done in the transformed space, so if you select cpm = FALSE, log2 = TRUE, gene_predicate = ~median(.) > 2, you're actually selecting genes with medians greater than 4 in raw space (since log2(4) == 2).
feather_dir	A string. The path to a directory containing feather files.

Details

- meta_mom_id: Unique identifier for each mother.
- meta_draw: The draw number. Most mothers have only one draw, some have two.
- meta_major_race: The race of the mother.
- meta_mom_weight_kg: The mother's pre-pregnancy weight in kilograms.
- meta_mom_height_cm: The mother's pre-pregnancy height in centimetres.
- meta_mom_bmi: The mother's pre-pregnancy BMI.
- meta_former_smoker: Is the mother a former smoker?
- meta_mom_age: The mother's age.
- meta_parity: The number of previous pregnancies that lasted at least 24 months.
- meta_baby_sex: The sex of the baby.
- meta_baby_weight_g: The weight of the newborn in grams. For twins this is the mean of the two.
- meta_minutes_before_spin: Minutes between blood draw and spin.
- meta_collectionga: The gestational age of the baby at the time of the blood draw (determined by ultrasound).
- meta_deliveryga: The gestational age of the baby at delivery (according to ultrasound).
- meta_weeks_to_delivery: The time to delivery in weeks.
- meta_q_pcr_actb_ct: The qPCR cycling threshold for ACTB.
- meta_q_pcr_ercc_ct: The qPCR cycling threshold for ERCCs.
- meta_lab_qc_passed: Did the sample pass lab QC?
- meta_pca_outlier: Is the sample a PCA outlier (found via robust PCA)?

There is an ID column called mirvie_id, a column called cohort which is just the first two characters in mirvie_id, and the rest of the data is gene counts.

See Also

Other cohort data: get_bw_data(), get_dp_data(), get_ga_data(), get_io_data(), get_kl_data(), get_ms_data(), get_ot_data(), get_pf_data(), get_pg_data(), get_pm_data(), get_pt_data(), get_rs_data(), get_st_data(), get_up_data(), get_vg_data()

---

get_pm_data *Get the Pemba data.*

Description

Get the Pemba gene count data and associated metadata.

Usage

```
get_pm_data(
  cpm = FALSE,
  log2 = FALSE,
  tot_counts = FALSE,
  remove_bads = TRUE,
  remove_noncoding = TRUE,
  remove_pseudo = TRUE,
  gene_predicate = NULL,
  feather_dir = NULL
)
```

Arguments

cpm	A flag. Gene counts in counts per million?
log2	A flag. Gene counts on log2 scale? If both cpm and log2 are TRUE, then the log2 calculation comes after the cpm calculation. The transformation actually used is x <-log2(x + 1).
tot_counts	A flag. Add a column for the total number of counts for this sample? If log2 = TRUE, a column log2_tot_counts will also be added with log2_tot_counts = log2(tot_counts + 1).
remove_bads	A flag. Remove samples that have been flagged as bad for whatever reason. The default is to do so.
remove_noncoding	A flag. Remove non-coding genes?
remove_pseudo	A flag. Remove psuedogenes?
gene_predicate	A predicate function which takes a single numeric argument. Only gene columns returning TRUE will be in the output. This will be passed through rlang::as_function(), so formula-style lambdas are permitted. For example, to keep only genes whose medians are positive, you can use gene_predicate = function(x) {median(x) > 0} or gene_predicate = ~median(.) > 0. This calculation is done in the transformed space, so if you select cpm = FALSE, log2 = TRUE, gene_predicate = ~median(.) > 2, you're actually selecting genes with medians greater than 4 in raw space (since log2(4) == 2).
feather_dir	A string. The path to a directory containing feather files.

Details

- meta_collectionga: The gestational age of the baby at the time of the blood draw (determined by ultrasound).
- meta_deliveryga: The gestational age of the baby at delivery (according to ultrasound).
- meta_weeks_to_delivery: The time to delivery in weeks.
- meta_sample_type: Case or control.
- meta_study_purpose: The purpose of the study. Pre-eclampsia or preterm birth.
- meta_major_race: The race of the mother.
- meta_ethnicity: The mother's ethnicity.
- meta_iugr: Was there intrauterine growth restriction?
- meta_sga: Was the baby small for its gestational age at birth?

- `meta_ivf`: Was it an IVF conception?
- `meta_chronic_hypertension`: Did the mother develop chronic hypertension?
- `meta_lupus`: Did the mother develop lupus during the pregnancy?
- `meta_caesarean`: Was the birth by C-section?
- `meta_stillbirth`: Was it a stillbirth?
- `meta_pre_eclampsia`: Did the mother develop pre-eclampsia?
- `meta_smoker`: Is the mother a smoker?
- `meta_education`: The education level of the mother.
- `meta_mom_bmi`: The mother's pre-pregnancy BMI.
- `meta_multiples`: Twins or more?
- `meta_labor_spontaneous`: Was the labor spontaneous?
- `meta_labor_augmented`: Was the labor augmented?
- `meta_mom_id`: Unique identifier for each mother.
- `meta_baby_sex`: The sex of the baby.
- `meta_baby_weight_g`: The weight of the newborn in grams. For twins this is the mean of the two.
- `meta_batch_num`: The batch number.
- `meta_sample_volume_ul`: The sample volume in microlitres.
- `meta_q_pcr_actb_ct`: The qPCR cycling threshold for ACTB.
- `meta_q_pcr_ercc_ct`: The qPCR cycling threshold for ERCCs.
- `meta_q_pcr_ver`: The qPCR protocol version.
- `meta_pca_outlier`: Is the sample a PCA outlier (found via robust PCA)?

There is an ID column called `mirvie_id`, a column called `cohort` which is just the first two characters in `mirvie_id`, and the rest of the data is gene counts.

See Also

Other cohort data: `get_bw_data()`, `get_dp_data()`, `get_ga_data()`, `get_io_data()`, `get_kl_data()`, `get_ms_data()`, `get_ot_data()`, `get_pf_data()`, `get_pg_data()`, `get_pi_data()`, `get_pt_data()`, `get_rs_data()`, `get_st_data()`, `get_up_data()`, `get_vg_data()`

`get_pt_data` *Get the Pittsburgh data.*

Description

Get the Pittsburgh gene count data and associated metadata.

Usage

```
get_pt_data(
  cpm = FALSE,
  log2 = FALSE,
  tot_counts = FALSE,
  remove_bads = TRUE,
  remove_noncoding = TRUE,
  remove_pseudo = TRUE,
  gene_predicate = NULL,
  feather_dir = NULL
)
```

Arguments

cpm	A flag. Gene counts in counts per million?
log2	A flag. Gene counts on log2 scale? If both cpm and log2 are TRUE, then the log2 calculation comes after the cpm calculation. The transformation actually used is x <-log2(x + 1).
tot_counts	A flag. Add a column for the total number of counts for this sample? If log2 = TRUE, a column log2_tot_counts will also be added with log2_tot_counts = log2(tot_counts + 1).
remove_bads	A flag. Remove samples that have been flagged as bad for whatever reason. The default is to do so.
remove_noncoding	A flag. Remove non-coding genes?
remove_pseudo	A flag. Remove psuedogenes?
gene_predicate	A predicate function which takes a single numeric argument. Only gene columns returning TRUE will be in the output. This will be passed through rlang::as_function(), so formula-style lambdas are permitted. For example, to keep only genes whose medians are positive, you can use gene_predicate = function(x) {median(x) > 0} or gene_predicate = ~median(.) > 0. This calculation is done in the transformed space, so if you select cpm = FALSE, log2 = TRUE, gene_predicate = ~median(.) > 2, you're actually selecting genes with medians greater than 4 in raw space (since log2(4) == 2).
feather_dir	A string. The path to a directory containing feather files.

Details

- meta_mom_id: Unique identifier for each mother.
- meta_draw: The draw number. Most mothers have only one draw, some have two.
- meta_major_race: The race of the mother.
- meta_mom_weight_kg: The mother's pre-pregnancy weight in kilograms.
- meta_mom_height_cm: The mother's pre-pregnancy height in centimetres.
- meta_mom_bmi: The mother's pre-pregnancy BMI.
- meta_former_smoker: Is the mother a former smoker?
- meta_mom_age: The mother's age.
- meta_parity: The number of previous pregnancies that lasted at least 24 months.
- meta_baby_sex: The sex of the baby.

- meta_baby_weight_g: The weight of the newborn in grams. For twins this is the mean of the two.
- meta_minutes_before_spin: Minutes between blood draw and spin.
- meta_collectionga: The gestational age of the baby at the time of the blood draw (determined by ultrasound).
- meta_deliveryga: The gestational age of the baby at delivery (according to ultrasound).
- meta_weeks_to_delivery: The time to delivery in weeks.
- meta_q_pcr_actb_ct: The qPCR cycling threshold for ACTB.
- meta_q_pcr_ercc_ct: The qPCR cycling threshold for ERCCs.
- meta_q_pcr_ver: The qPCR protocol version.
- meta_lab_qc_passed: Did the sample pass lab QC?
- meta_pca_outlier: Is the sample a PCA outlier (found via robust PCA)?

There is an ID column called mirvie_id, a column called cohort which is just the first two characters in mirvie_id, and the rest of the data is gene counts.

See Also

Other cohort data: get_bw_data(), get_dp_data(), get_ga_data(), get_io_data(), get_kl_data(), get_ms_data(), get_ot_data(), get_pf_data(), get_pg_data(), get_pi_data(), get_pm_data(), get_rs_data(), get_st_data(), get_up_data(), get_vg_data()

---

get_rs_data *Get the Roskilde data.*

Description

Get the Roskilde gene count data and associated metadata.

Usage

```
get_rs_data(
  cpm = FALSE,
  log2 = FALSE,
  tot_counts = FALSE,
  remove_bads = TRUE,
  remove_noncoding = TRUE,
  remove_pseudo = TRUE,
  gene_predicate = NULL,
  feather_dir = NULL
)
```

Arguments

cpm	A flag. Gene counts in counts per million?
log2	A flag. Gene counts on log2 scale? If both cpm and log2 are TRUE, then the log2 calculation comes after the cpm calculation. The transformation actually used is x <-log2(x + 1).

tot_counts	A flag. Add a column for the total number of counts for this sample? If log2 = TRUE, a column log2_tot_counts will also be added with log2_tot_counts = log2(tot_counts + 1).
remove_bads	A flag. Remove samples that have been flagged as bad for whatever reason. The default is to do so.
remove_noncoding	A flag. Remove non-coding genes?
remove_pseudo	A flag. Remove psuedogenes?
gene_predicate	A predicate function which takes a single numeric argument. Only gene columns returning TRUE will be in the output. This will be passed through rlang::as_function(), so formula-style lambdas are permitted. For example, to keep only genes whose medians are positive, you can use gene_predicate = function(x) {median(x) > 0} or gene_predicate = ~median(.) > 0. This calculation is done in the transformed space, so if you select cpm = FALSE, log2 = TRUE, gene_predicate = ~median(.) > 2, you're actually selecting genes with medians greater than 4 in raw space (since log2(4) == 2).
feather_dir	A string. The path to a directory containing feather files.

Details

- meta_mom_id: Unique identifier for each mother.
- meta_draw: The draw number. Most mothers have four draws.
- meta_collectionga: The gestational age of the baby at the time of the blood draw (determined by ultrasound).
- meta_deliveryga: The gestational age of the baby at delivery (according to ultrasound).
- meta_weeks_to_delivery: The time to delivery in weeks.
- meta_major_race: The race of the mother.
- meta_smoker: Is the mother a smoker?
- meta_conception_spontaneous: Was the conception spontaneous?
- meta_conception_spontaneous: Was the conception medically assisted?
- meta_ivf: Was it an IVF conception?
- meta_mom_age: The mother's age.
- meta_mom_weight_kg: The mother's pre-pregnancy weight in kilograms.
- meta_mom_height_cm: The mother's pre-pregnancy height in centimetres.
- meta_mom_bmi: The mother's pre-pregnancy BMI.
- meta_parity: The number of previous pregnancies that lasted at least 24 months.
- meta_baby_sex: The sex of the baby.
- meta_baby_weight_g: The weight of the newborn in grams. For twins this is the mean of the two.
- meta_baby_head_circumference_cm: The circumference of the baby's head in centimetres. For twins this is the mean of the two.
- meta_baby_length_cm: The length of the baby in centimetres. For twins this is the mean of the two.
- meta_labor_spontaneous: Was the labor spontaneous?
- meta_labor_augmented: Was the labor augmented?

- meta_q_pcr_actb_ct: The qPCR cycling threshold for ACTB.
- meta_q_pcr_ercc_ct: The qPCR cycling threshold for ERCCs.
- meta_q_pcr_ver: The qPCR protocol version.
- meta_pca_outlier: Is the sample a PCA outlier (found via robust PCA)?

There is an ID column called mirvie_id, a column called cohort which is just the first two characters in mirvie_id, and the rest of the data is gene counts.

See Also

Other cohort data: get_bw_data(), get_dp_data(), get_ga_data(), get_io_data(), get_kl_data(), get_ms_data(), get_ot_data(), get_pf_data(), get_pg_data(), get_pi_data(), get_pm_data(), get_pt_data(), get_st_data(), get_up_data(), get_vg_data()

---

get_st_data *Get the Stanford data.*

Description

Get the Stanford gene count data and associated metadata.

Usage

```
get_st_data(
  cpm = FALSE,
  log2 = FALSE,
  tot_counts = FALSE,
  remove_bads = TRUE,
  remove_noncoding = TRUE,
  remove_pseudo = TRUE,
  gene_predicate = NULL,
  feather_dir = NULL
)
```

Arguments

cpm	A flag. Gene counts in counts per million?
log2	A flag. Gene counts on log2 scale? If both cpm and log2 are TRUE, then the log2 calculation comes after the cpm calculation. The transformation actually used is x <-log2(x + 1).
tot_counts	A flag. Add a column for the total number of counts for this sample? If log2 = TRUE, a column log2_tot_counts will also be added with log2_tot_counts = log2(tot_counts + 1).
remove_bads	A flag. Remove samples that have been flagged as bad for whatever reason. The default is to do so.
remove_noncoding	A flag. Remove non-coding genes?
remove_pseudo	A flag. Remove psuedogenes?

gene_predicate A predicate function which takes a single numeric argument. Only gene columns returning TRUE will be in the output. This will be passed through rlang::as_function(), so formula-style lambdas are permitted. For example, to keep only genes whose medians are positive, you can use gene_predicate = function(x) {median(x) > 0} or gene_predicate = ~median(.) > 0. This calculation is done in the transformed space, so if you select cpm = FALSE, log2 = TRUE, gene_predicate = ~median(.) > 2, you're actually selecting genes with medians greater than 4 in raw space (since log2(4) == 2).

feather_dir A string. The path to a directory containing feather files.

Details

- meta_collectionga: The gestational age of the baby at the time of the blood draw (determined by ultrasound).
- meta_deliveryga: The gestational age of the baby at delivery (according to ultrasound).
- meta_weeks_to_delivery: The time to delivery in weeks.
- meta_pre_eclampsia: Did the mother develop pre-eclampsia?
- meta_pca_outlier: Is the sample a PCA outlier (found via robust PCA)?

There is an ID column called mirvie_id, a column called cohort which is just the first two characters in mirvie_id, and the rest of the data is gene counts.

See Also

Other cohort data: get_bw_data(), get_dp_data(), get_ga_data(), get_io_data(), get_kl_data(), get_ms_data(), get_ot_data(), get_pf_data(), get_pg_data(), get_pi_data(), get_pm_data(), get_pt_data(), get_rs_data(), get_up_data(), get_vg_data()

---

get_up_data *Get the UPenn data.*

Description

Get the UPenn gene count data and associated metadata.

Usage

```
get_up_data(
  cpm = FALSE,
  log2 = FALSE,
  tot_counts = FALSE,
  remove_bads = TRUE,
  remove_noncoding = TRUE,
  remove_pseudo = TRUE,
  gene_predicate = NULL,
  feather_dir = NULL
)
```

Arguments

cpm	A flag. Gene counts in counts per million?
log2	A flag. Gene counts on log2 scale? If both cpm and log2 are TRUE, then the log2 calculation comes after the cpm calculation. The transformation actually used is x <-log2(x + 1).
tot_counts	A flag. Add a column for the total number of counts for this sample? If log2 = TRUE, a column log2_tot_counts will also be added with log2_tot_counts = log2(tot_counts + 1).
remove_bads	A flag. Remove samples that have been flagged as bad for whatever reason. The default is to do so.
remove_noncoding	A flag. Remove non-coding genes?
remove_pseudo	A flag. Remove psuedogenes?
gene_predicate	A predicate function which takes a single numeric argument. Only gene columns returning TRUE will be in the output. This will be passed through rlang::as_function(), so formula-style lambdas are permitted. For example, to keep only genes whose medians are positive, you can use gene_predicate = function(x) {median(x) > 0} or gene_predicate = ~median(.) > 0. This calculation is done in the transformed space, so if you select cpm = FALSE, log2 = TRUE, gene_predicate = ~median(.) > 2, you're actually selecting genes with medians greater than 4 in raw space (since log2(4) == 2).
feather_dir	A string. The path to a directory containing feather files.

Details

- meta_collectionga: The gestational age of the baby at the time of the blood draw (determined by ultrasound).
- meta_deliveryga: The gestational age of the baby at delivery (according to ultrasound).
- meta_pca_outlier: Is the sample a PCA outlier (found via robust PCA)?

There is an ID column called mirvie_id, a column called cohort which is just the first two characters in mirvie_id, and the rest of the data is gene counts.

See Also

Other cohort data: get_bw_data(), get_dp_data(), get_ga_data(), get_io_data(), get_kl_data(), get_ms_data(), get_ot_data(), get_pf_data(), get_pg_data(), get_pi_data(), get_pm_data(), get_pt_data(), get_rs_data(), get_st_data(), get_vg_data()

---

get_vg_data *Get the Validation GAPPS data.*

Description

Get the Validation GAPPS gene count data and associated metadata.

Usage

```
get_vg_data(
  cpm = FALSE,
  log2 = FALSE,
  tot_counts = FALSE,
  remove_bads = TRUE,
  remove_noncoding = TRUE,
  remove_pseudo = TRUE,
  gene_predicate = NULL,
  feather_dir = NULL
)
```

Arguments

cpm	A flag. Gene counts in counts per million?
log2	A flag. Gene counts on log2 scale? If both cpm and log2 are TRUE, then the log2 calculation comes after the cpm calculation. The transformation actually used is x <-log2(x + 1).
tot_counts	A flag. Add a column for the total number of counts for this sample? If log2 = TRUE, a column log2_tot_counts will also be added with log2_tot_counts = log2(tot_counts + 1).
remove_bads	A flag. Remove samples that have been flagged as bad for whatever reason. The default is to do so.
remove_noncoding	A flag. Remove non-coding genes?
remove_pseudo	A flag. Remove psuedogenes?
gene_predicate	A predicate function which takes a single numeric argument. Only gene columns returning TRUE will be in the output. This will be passed through rlang::as_function(), so formula-style lambdas are permitted. For example, to keep only genes whose medians are positive, you can use gene_predicate = function(x) {median(x) > 0} or gene_predicate = ~median(.) > 0. This calculation is done in the transformed space, so if you select cpm = FALSE, log2 = TRUE, gene_predicate = ~median(.) > 2, you're actually selecting genes with medians greater than 4 in raw space (since log2(4) == 2).
feather_dir	A string. The path to a directory containing feather files.

Details

- meta_mom_id: Unique identifier for each mother.
- meta_major_race: The race of the mother.
- meta_ethnicity: The mother's ethnicity.
- meta_mom_age: The mother's age.
- meta_collectionga: The gestational age of the baby at the time of the blood draw (determined by ultrasound).
- meta_deliveryga: The gestational age of the baby at delivery (according to ultrasound).
- meta_smoker: Is the mother a smoker?
- meta_labor_spontaneous: Was the labor spontaneous?
- meta_baby_sex: The sex of the baby.

- `meta_baby_weight_g`: The weight of the newborn in grams. For twins this is the mean of the two.
- `meta_weeks_to_delivery`: The time to delivery in weeks.
- `meta_q_pcr_actb_ct`: The qPCR cycling threshold for ACTB.
- `meta_q_pcr_ercc_ct`: The qPCR cycling threshold for ERCCs.
- `meta_pca_outlier`: Is the sample a PCA outlier (found via robust PCA)?

There is an ID column called `mirvie_id`, a column called `cohort` which is just the first two characters in `mirvie_id`, and the rest of the data is gene counts.

See Also

Other cohort data: `get_bw_data()`, `get_dp_data()`, `get_ga_data()`, `get_io_data()`, `get_kl_data()`, `get_ms_data()`, `get_ot_data()`, `get_pf_data()`, `get_pg_data()`, `get_pi_data()`, `get_pm_data()`, `get_pt_data()`, `get_rs_data()`, `get_st_data()`, `get_up_data()`

---

learn_curve *Get the learning curve of a model as training data quantity increases.*

Description

Given a training and test set, fit a model on increasing fractions of the training set, up to the full set, with a constant test set per repeat (each repeat will have a different test set). The default is to use 10%, 20%, 30%, . . ., 90%, 100%. Care is taken to make sure each fraction is a subset of the last e.g. all samples present in the 10% will be present in the 20% to simulate the addition of more data, as opposed to a random sample of more data. Optionally, you can pass all of your data in as the `training_data` and then get the function to do the splitting for you.

Usage

```
learn_curve(
  model_evaluate,
  training_data,
  outcome,
  testing_data = NULL,
  testing_frac = NULL,
  training_fracs = seq(0.1, 1, by = 0.1),
  repeats = 1,
  strata = NULL,
  n_cores = 1
)
```

Arguments

`model_evaluate`	A function with exactly two arguments: `training_data` and `testing_data` that trains the model of choice on `training_data` and then produces predictions on `testing_data`, finally evaluating those predictions and outputting a length two numeric vector with names "cv" and "test" giving the cross-validation and test scores from the evaluation.

training_data	A data frame. Subsets of this will be used for training. If testing_data is NULL and testing_frac is not, this will be split into training and testing sets, with testing_frac used for testing.
outcome	A string. The name of the outcome variable. This must be a column in training_data.
testing_data	A data frame. The trained models will all be tested against this constant test set.
testing_frac	A numeric vector with values between 0 and 1/3.The fraction of training_data to use for the test set. This can only be used if testing_data is NULL. To try many different fractions, specify all of them as a numeric vector.
training_fracs	A numeric vector. Fractions of the training data to use. This must be a positive, increasing vector of real numbers ending in 1.
repeats	A positive integer. The number of times to repeat the sampling for each proportion in testing_frac. This can be greater than 1 only if testing_data is NULL and testing_frac is not NULL. For each repeat, a different subsetting of testing_data remains takes place.
strata	A string. Variable to stratify on when splitting data.
n_cores	A positive integer. The cross-validation can optionally be done in parallel. Specify the number of cores for parallel processing here.

Value

A data frame with the following columns.

- rep: The repeat number.
- testing_frac: The fraction of training_data that is set aside for testing. If the testing_data argument is specified, testing_frac will be 0, because none of training_data is set aside for testing.
- training_frac: The fraction of the (post train/test split)training data used for learning.
- testing_indices: The row indices of the training_data argument that were set aside for testing. If testing_data is specified (and hence none of training_data needs to be set aside for testing, this will be a vector of NAs with length equal to the number of rows in testing_data.
- training_indices: The row indices of the training_data that were used for learning.
- cv: The cross-validation score.
- test: The test score.

See Also

autoplot.mirvie_learning_curve()

Examples

```
data("BostonHousing", package = "mlbench")
bh <- dplyr::select_if(BostonHousing, is.numeric)
model_evaluate <- function(training_data, testing_data) {
  trained_mod <- lm(medv ~ ., training_data)
  training_preds <- predict(trained_mod, newdata = training_data)
  preds <- predict(trained_mod, newdata = testing_data)
  c(
    train = yardstick::mae_vec(training_data$medv, training_preds),
    test = yardstick::mae_vec(testing_data$medv, preds)
```

```
  )
}
mlc <- mlc0 <- suppressWarnings(
  learn_curve(model_evaluate, bh, "medv",
    training_fracs = c(seq(0.1, 0.7, 0.2), 0.85),
    testing_frac = c(0.25, 0.5), repeats = 8,
    strata = "medv", n_cores = 4
  )
)
```

`learn_curve_cv` *Create a cross-validation learning curve.*

Description

This function needs a workflows::workflow ready for tune::fit_resamples. It does different fold cross-validation to vary the training set sizes and then collects the predictions and scores them.

Usage

```
learn_curve_cv(
  data,
  wf,
  folds,
  repeats,
  metric_calculator,
  strata = NULL,
  pkgs = c("mirmodels"),
  n_cores = 1
)
```

Arguments

`data`	A data frame. The data to be used for the modelling.
`wf`	A `workflows::workflow()`. Should have been constructed using `data`.
`folds`	An integer vector. Different v to use in `rsample::vfold_cv()`.
`repeats`	The number of times to repeat each cross-validation.
`metric_calculator`	A function which takes a single data frame argument and returns a double. The data frame that will be passed to this function is the output of `tune::collect_predictions()` which will be run on the output of `tune::fit_resamples(save_preds = TRUE)`. See the example below.
`strata`	A string. Variable to stratify on when splitting for cross-validation.
`pkgs`	A character vector. Passed to `tune::control_resamples()`.
`n_cores`	A positive integer. The cross-validation can optionally be done in parallel. Specify the number of cores for parallel processing here.

Value

A tibble with 2 columns:

- `training_samples`: The number of samples used in training.
- `score`: The score computed by `metric_calculator()`.

Examples

```
data("BostonHousing", package = "mlbench")
bh <- dplyr::select_if(BostonHousing, is.numeric)
mod <- parsnip::linear_reg(penalty = 0, mixture = 0) %>%
  parsnip::set_engine("lm")
wf <- workflows::workflow() %>%
  workflows::add_formula(medv ~ .) %>%
  workflows::add_model(mod)
metric_calculator <- ~ yardstick::mae(., medv, .pred)$.estimate
lccv <- suppressWarnings(
  learn_curve_cv(bh, wf, 2:9, 3, metric_calculator, n_cores = 4)
)
```

---

linear_correct *Correct data for the effects of selected covariates.*

Description

This function uses linear models to regress away the effects of selected covariates on selected columns of a data frame. One may optionally specify variables whose effects are considered *real* or *of interest* and their effects will not be regressed away (only effects orthogonal to those will be regressed away).

Usage

```
linear_correct(
  training_data,
  testing_data = NULL,
  correct_cols,
  correct_for_cols,
  keep_effect_cols = NULL,
  robust = TRUE
)
```

Arguments

`training_data`	A data frame containing one sample per row. The correction is learned and applied on this data.
`testing_data`	A data frame. The testing counterpart to `training_data`. The correction that is learned on `training_data` is applied here. It's fine to pass this argument as `NULL`, in which case no testing data correction takes place.
`correct_cols`	A character vector. The names of the columns that are to be altered (corrected). These columns must all be numeric.

correct_for_cols
: A character vector. The names of the columns that are to be corrected *for*. These columns must all be numeric, factor or logical.

keep_effect_cols
: A character vector. The names of the column specifying variables whose effects should *not* be regressed away. These columns must all be numeric, factor or logical. If there are no columns whose effects should *not* be regressed away, pass this argument as NULL.

robust
: A flag. Use robust linear model MASS::rlm()? Can only be used with type = 1.

Details

If keep_effect_cols is NULL, this function is just a wrapper around multi_lm() and multi_resids() with reset_mean_med = TRUE. That is, for each variable in correct_cols, a linear model is fit with the variables correct_for_cols as explanatory variables. Then the residuals from this model (reset about their original mean or median) are kept as the *corrected* values of those variables.

If keep_effect_cols is not NULL, then first, for each variable in correct_cols, a linear model is fit with the variables keep_effect_cols as explanatory variables. The fitted variables from these models are remembered as the effects of these keep_effect_cols on correct_cols. The residuals from these models are then components of correct_cols which can't be explained by keep_effect_cols. With these residuals, the effects of correct_for_cols are regressed away, and what remains is added onto the fitted values from the modelling of correct_cols with keep_effect_cols.

Columns in training_data that are not specified in correct_cols, correct_for_cols, or keep_effect_cols will be returned unchanged.

Value

A list with elements named training_data and testing_data. The corrected data.

Examples

```
if (rlang::is_installed("mirmisc")) {
  data <- get_combined_cohort_data(
    c("bw", "ga", "io", "kl", "pm", "pt", "rs"),
    cpm = FALSE, log2 = TRUE, tot_counts = TRUE,
    gene_predicate = ~ median(.) > 0
  )
  res <- linear_correct(
    data,
    correct_cols = mirmisc::get_df_gene_names(data),
    correct_for_cols  = c("log2_tot_counts"),
    keep_effect_cols = "meta_collectionga"
  )
}
```

---

multi_aov *Perform multiple analyses of variance on linear model objects.*

Description

Given a list of linear models (the results of calls to `stats::lm()` or `MASS::rlm()`), all of which used the same explanatory variables with the same dataset (most likely the output of a call to `multi_lm()`), perform analyses of variance on all of them and arrange the result into a single data frame.

Usage

```
multi_aov(lms, lms_data = NULL, type = 1)
```

Arguments

`lms`	A list of fitted linear models (the results of calls to `stats::lm()` or `MASS::rlm()`). Most likely, the output of a call to `multi_lm()`.
`lms_data`	The data frame that was passed as the `df` argument to `multi_lm()`, if `multi_lm()` was used to create `lms`.
`type`	The type of sum of squares to use. Types I and II are currently supported. Type I can be used with either default or robust linear models, but type II cannot be used with robust linear models.

Details

`lms_data` is needed to ensure that the elements of the `y` column in the return match the input. You should provide this argument if you can at all.

Value

An object of class `mirmodels_multi_aov_df`. A long data frame with columns:

- `y`: The name of the response variable
- `x`: The name of the explanatory variable.
- `pctvarexp`: The percent of variance explained by `x`.
- `pval`: The p-value for x explaining a non-zero amount of y's variance.

Examples

```
if (requireNamespace("mirmisc")) {
  gars_data <- get_combined_cohort_data(c("ga", "rs"), log2 = TRUE) %>%
    dplyr::mutate(
      cohort = factor(
        dplyr::if_else(startsWith(mirvie_id, "RS"), "RS", "GA")
      ),
      meta_major_race = forcats::fct_drop(meta_major_race)
    ) %>%
    dplyr::filter(!is.na(meta_collectionga), !is.na(meta_major_race))
  xs <- c("cohort", "meta_major_race", "meta_collectionga")
  ys <- gars_data %>%
    dplyr::select(dplyr::any_of(mirmisc::get_gene_names())) %>%
    purrr::map_dbl(sum) %>%
    sort() %>%
    tail(100) %>%
    names() # ys are the highest expressed 100 genes
  lms <- multi_lm(gars_data, xs, ys, robust = TRUE)
```

```
  aovs <- multi_aov(lms)
  summary(aovs)
}
```

`multi_fitteds` *Get the fitted values from multiple linear models.*

Description

Get the fitted values from multiple linear models.

Usage

```
multi_fitteds(lms, lms_data = NULL, new_data = NULL)
```

Arguments

`lms`	A list of fitted linear models (the results of calls to `stats::lm()` or `MASS::rlm()`). Most likely, the output of a call to `multi_lm()`.
`lms_data`	The data frame that was passed as the `df` argument to `multi_lm()`, if `multi_lm()` was used to create `lms`.
`new_data`	Rather than calculating the residuals on the data where the model was fit, you can pass a new dataset and calculate the residuals there.

Value

A data frame of the fitted values The variables in this data frame will have their original names. If `lms_data` is given, other columns (ones for which residuals were not calculated) of that data frame will be returned as is.

Examples

```
if (rlang::is_installed("mirmisc")) {
  gars_data <- get_combined_cohort_data(c("ga", "rs"), log2 = TRUE) %>%
    dplyr::mutate(
      cohort = factor(
        dplyr::if_else(startsWith(mirvie_id, "RS"), "RS", "GA")
      ),
      meta_major_race = forcats::fct_drop(meta_major_race)
    ) %>%
    dplyr::filter(!is.na(meta_major_race), !is.na(meta_collectionga))
  xs <- c("cohort", "meta_major_race", "meta_collectionga")
  ys <- gars_data %>%
    dplyr::select(dplyr::any_of(mirmisc::get_gene_names())) %>%
    purrr::map_dbl(sum) %>%
    sort() %>%
    tail(100) %>%
    names() # ys are the highest expressed 100 genes
  lms <- multi_lm(gars_data, xs, ys)
  fitteds <- multi_fitteds(lms, gars_data)
}
```

multi_lm *Fit multiple linear models.*

Description

For a constant set of explanatory variables (xs) and several dependent variables (ys), fit a linear model y ~ xs for each y in ys.

Usage

```
multi_lm(df, xs, ys, robust = TRUE)
```

Arguments

df	A data frame containing explanatory variables xs and dependent variables ys.
xs	A character vector. The explanatory variables. These must be the names of columns of df that are either numeric, factor or logical.
ys	A character vector. The dependent variables. These must be the names of numeric columns of df that are either numeric or factors.
robust	A flag. Use robust linear model MASS::rlm()? Can only be used with type = 1.

Value

A list of fitted linear models (the results of calls to stats::lm() or MASS::rlm()).

Examples

```
if (rlang::is_installed("mirmisc")) {
  gars_data <- get_combined_cohort_data(c("ga", "rs"),
    gene_predicate = ~ median(.) > 0, log2 = TRUE
  ) %>%
    dplyr::mutate(
      cohort = factor(
        dplyr::if_else(startsWith(mirvie_id, "RS"), "RS", "GA")
      )
    ) %>%
    dplyr::filter(!is.na(meta_major_race), !is.na(meta_collectionga))
  xs <- c("cohort", "meta_major_race", "meta_collectionga")
  ys <- mirmisc::get_df_gene_names(gars_data)
  res <- multi_lm(gars_data, xs, ys)
}
```

`multi_resids` *Get the residuals from multiple linear models.*

Description

There is the option to add the mean (or median for robust models) to the residuals.

Usage

```
multi_resids(lms, lms_data = NULL, reset_mean_med = FALSE, new_data = NULL)
```

Arguments

`lms`	A list of fitted linear models (the results of calls to `stats::lm()` or `MASS::rlm()`). Most likely, the output of a call to `multi_lm()`.
`lms_data`	The data frame that was passed as the `df` argument to `multi_lm()`, if `multi_lm()` was used to create `lms`.
`reset_mean_med`	A flag. If `TRUE`, for each variable for which the residuals are calculated the mean (or in the case of robust linear models, the median) of the original values of these variables is added to the residuals, such that the output is again centred on this value.
`new_data`	Rather than calculating the residuals on the data where the model was fit, you can pass a new dataset and calculate the residuals there.

Value

A data frame of the residuals. The variables in this data frame will have their original names. If `lms_data` is given, other columns (ones for which residuals were not calculated) of that data frame will be returned as is.

Examples

```
if (rlang::is_installed("mirmisc")) {
  gars_data <- get_combined_cohort_data(c("ga", "rs"), log2 = TRUE) %>%
    dplyr::mutate(
      cohort = dplyr::if_else(startsWith(mirvie_id, "RS"), "RS", "GA"),
      cohort = factor(cohort),
      meta_major_race = forcats::fct_drop(meta_major_race)
    ) %>%
    dplyr::filter(!is.na(meta_major_race), !is.na(meta_collectionga))
  xs <- c("cohort", "meta_major_race", "meta_collectionga")
  ys <- gars_data %>%
    dplyr::select(dplyr::any_of(mirmisc::get_gene_names())) %>%
    purrr::map_dbl(sum) %>%
    sort() %>%
    tail(100) %>%
    names() # ys are the highest expressed 100 genes
  lms <- multi_lm(gars_data, xs, ys)
  resids <- multi_resids(lms, gars_data, reset_mean_med = TRUE)
}
```

`new_mirvie_learning_curve` *Construct a* `mirvie_learning_curve` *object.*

Description

A `mirvie_learning_curve` object is what is output by `learn_curve()`.

Usage

```
new_mirvie_learning_curve(tib)
```

Arguments

`tib`	A tibble with columns `rep` (int), `testing_frac` (dbl), `training_frac` (dbl), `testing_indices` (list), `training_indices` (list), `cv` (dbl) and `test` (dbl).

Details

This just tacks `"mirvie_learning_curve"` onto the front of the `class` attribute of an appropriate object. This should only be used inside of `learn_curve()`.

Value

A `mirvie_learning_curve` object.

`new_mirvie_learning_curve_cv` *Construct a* `mirvie_learning_curve` *object.*

Description

A `mirvie_learning_curve_cv` object is what is output by `learn_curve_cv()`.

Usage

```
new_mirvie_learning_curve_cv(tib)
```

Arguments

`tib`	A tibble with columns `training_samples` (integer, the number of training samples on that iteration) and `score` (double, a model metric score).

Details

This just tacks `"mirvie_learning_curve_cv"` onto the front of the `class` attribute of an appropriate object. This should only be used inside of `learn_curve_cv()`.

Value

A `mirvie_learning_curve_cv` object.

padj_cutoff *Adjusted p-value cutoff*

Description

Used for specification of `padj_cutoff` values in `step_select_genes()`.

Usage

```
padj_cutoff(range = c(0, 1), trans = NULL)
```

Arguments

`range`	A two-element vector holding the *defaults* for the smallest and largest possible values, respectively.
`trans`	A `trans` object from the `scales` package, such as `scales::log10_trans()` or `scales::reciprocal_trans()`. If not provided, the default is used which matches the units used in `range`. If no transformation, `NULL`.

plot_roc *Plot ROC curves with confidence intervals.*

Description

Plot ROC curves for several different models (or just one) with confidence intervals calculated by `pROC::ci.se()` and `pROC::ci.sp()`.

Usage

```
plot_roc(
  rocdf,
  conf_level = 0.95,
  aes_opts = list(conf_alpha = 0.8, roc_line_size = 1, roc_line_color = "black",
    null_line_size = 1, null_line_color = "blue", null_line_alpha = 0.5),
  quick = TRUE,
  parallel = TRUE
)
```

Arguments

`rocdf`	A data frame with columns `response` and `predictor` (see `pROC::roc()`. To plot several models, provide a `model` column with the model names; one ROC will be calculated and plot per model name.
`conf_level`	A number in [0, 1]. This is passed as the `conf.level` argument to `pROC::ci.se()` and `pROC::ci.sp()`. Use `conf_level = NA` to disable plotting of confidence intervals.
`aes_opts`	A named list of aesthetic options for the plot. • `conf_alpha`: Transparency of the confidence interval lines.

- roc_line_size: Thickness of the ROC line.
- roc_line_color: Color of the ROC line.
- null_line_size: Thickness of the null line.
- null_line_color: Color of the null line.
- null_line_alpha: Transparency of the null line

quick — A flag. With quick = FALSE, ROC confidence intervals are calculated with a 2,000-round bootstrap. quick = TRUE will use 100 rounds instead. quick = TRUE is fine for exploration.

parallel — A flag. Calculate the sensitivity and specificity confidence intervals simultaneously? This is a small speedup (<2x).

Details

Inside this function, pROC::roc() is called with direction = "<". The easiest thing is to have rocdf$response be a vector of 0s and 1s and have rocdf$predictor be a vector of probabilities where each probability is the probability of a response of 1. If rocdf$predictor is logical, then these rocdf$predictor is the probability of a response of TRUE.

Value

A ggplot2::ggplot().

Examples

```
rocdf1 <- dplyr::tibble( # decent model
  predictor = seq(0.01, 0.99, length.out = 1000),
  response = purrr::rbernoulli(length(predictor), predictor)
)
rocdf2 <- dplyr::tibble( # bad model
  predictor = seq(0.01, 0.99, length.out = 88),
  response = purrr::rbernoulli(length(predictor), 0.5)
)
rocdf12 <- dplyr::bind_rows(
  dplyr::bind_cols(rocdf1, model = "decent"),
  dplyr::bind_cols(rocdf2, model = "bad")
)
plot_roc(rocdf1,
  quick = TRUE, parallel = FALSE,
  aes_opts = list(conf_alpha = 0.3)
)
plot_roc(rocdf2,
  conf_level = NA, quick = TRUE,
  aes_opts = list(roc_line_size = 2, roc_line_color = "red")
)
plot_roc(rocdf12,
  conf_level = NA, quick = TRUE,
  aes_opts = list(null_line_size = 2, null_line_alpha = 1)
)
plot_roc(rocdf12, conf_level = 0.99, quick = TRUE)
```

qqplot_pvals *QQ-plot a vector of p-values.*

Description

QQ-plot a vector of p-values.

Usage

```
qqplot_pvals(
  x,
  labels = NULL,
  controls = NULL,
  cases = NULL,
  lambda_correct = FALSE,
  aes_opts = list(null_alpha = 1, fit_alpha = 1)
)
```

Arguments

x	A numeric vector with elements in (0, 1).
labels	A character vector of length at most length(x). Labels for the smallest p-values.
controls	Number of controls in the tests.
cases	Number of cases in the tests.
lambda_correct	A flag. Apply a correction to make the slope of the fitted line equal to 1.
aes_opts	A list. Aesthetic options for the plot. The list has two named elements. • null_alpha: Transparency of the x = y line. • fit_alpha: Transparency of the fit line.

Value

A ggplot2::ggplot().

Examples

```
set.seed(1)
pvals <- runif(999) ^ (4 / 3)
qqplot_pvals(pvals,
  controls = 500, cases = 499,
  aes_opts = list(null_alpha = 0.5, fit_alpha = 0.5)
)
qqplot_pvals(pvals,
  labels = c("a", "b", "c"),
  controls = 500, cases = 499,
  lambda_correct = TRUE
)
```

step_select_genes *Gene selection by differential expression analysis.*

Description

step_select_genes() creates a *specification* of a recipe step that will select genes by differential expression analysis, discarding those that don't pass a certain p-value threshold. Currently, cor_de() is used.

Usage

```
step_select_genes(
  recipe,
  ...,
  role = NA,
  trained = FALSE,
  condition = NULL,
  genes_pass = NULL,
  padj_cutoff = 0.05,
  max_n_genes = NULL,
  min_n_genes = NULL,
  options = list(method = "spearman", padj_method = stats::p.adjust.methods[1]),
  skip = FALSE,
  id = recipes::rand_id("select_genes")
)
```

Arguments

recipe	A recipe object. The step will be added to the sequence of operations for this recipe.
...	One or more selector functions to choose which variables will be used to compute the components. See selections() for more details. For the tidy method, these are not currently used.
role	For model terms created by this step, what analysis role should they be assigned?. By default, the function assumes that the new principal component columns created by the original variables will be used as predictors in a model.
trained	A logical to indicate if the quantities for preprocessing have been estimated.
condition	The condition for the differential expression. See cor_de().
genes_pass	This should not be specified by the end user. Information about which genes do and don't pass the p-value threshold in the differential expression analysis is stored here.
padj_cutoff	Genes with an adjusted p-value less than or equal to padj_cutoff in the differential expression analysis are kept. The rest are discarded.
max_n_genes	A positive integer. The maximum number of genes selected by this step.
min_n_genes	A positive integer. The minimum number of genes selected. This guarantees that even if no genes pass padj_cutoff, there will be this many returned.
options	A list with two elements named method and padj_method. Both are passed to cor_de(). padj_cutoff is the threshold for keeping genes.

skip	A logical. Should the step be skipped when the recipe is baked by bake.recipe()? While all operations are baked when prep.recipe() is run, some operations may not be able to be conducted on new data (e.g. processing the outcome variable(s)). Care should be taken when using skip = TRUE as it may affect the computations for subsequent operations
id	A character string that is unique to this step to identify it.

Value

An updated version of recipe with the new step added to the sequence of existing steps (if any).

---

summary.mirmodels_multi_aov_df *Summarize the output of* multi_aov().

Description

Group by x and get the mean and median percent variance explained for all of the ys.

Usage

```
## S3 method for class 'mirmodels_multi_aov_df'
summary(object, ..., na_rm = TRUE)
```

Arguments

object	The output of multi_aov().
...	Optional. Positive integers. Quantiles of variance explained. See example in multi_aov().
na_rm	A flag. Remove NAs in the summary calculations?

Value

A data frame.

---

train_gbm *Train a gradient boosted model.*

Description

Train a gradient boosted model, selecting hyperparameters trees, tree_depth and learn_rate by cross-validation.

Usage

```
train_gbm(
  training_data,
  outcome,
  metric,
 hyper_param_grid = list(trees = c(5, 10, 20, 40), tree_depth = c(1, 2), learn_rate =
    c(0.01, 0.1, 0.2, 0.5)),
  cv_nfolds = 5,
  cv_nreps = 1,
  id_col = NULL,
  strata = NULL,
  selection_method = "Breiman",
  simplicity_params = NULL,
  include_nullmod = TRUE,
  err_if_nullmod = FALSE,
  warn_if_nullmod = TRUE,
  n_cores = 1
)
```

Arguments

training_data	A data frame. The data used to train the model.
outcome	A string. The name of the outcome variable. This must be a column in training_data.
metric	A string. The probability metric to choose the best model. and select the best one. Common choices are "mn_log_loss", "roc_auc" and "accuracy". This should be a metric that is available in the yardstick package, but use e.g. "mae" and not "yardstick::mae" in this argument. If you specify this as a multi-element character vector, the first element will be used to select the best model; subsequent metrics will also be reported for that model in the cv_performance attribute of the the returned object.)‘
hyper_param_grid	A data frame with one row per hyperparameter combination. The column names give the hyper parameter names. Can optionally be passed as a list which is made into a tibble by tidyr::expand_grid().
cv_nfolds	A positive integer. The number of folds for cross-validation.
cv_nreps	A positive integer. The number of repeated rounds in the cross-validation.
id_col	A string. If there is a sample identifier column, specify it here to tell the model not to use it as a predictor.
strata	A string. Variable to stratify on when splitting for cross-validation.
selection_method	A string. How to select the best model. There are two options: "Breiman" and "absolute". "absolute" selects the best model by selecting the model with the best mean performance according to the chosen metric. "Breiman" selects the simplest model that comes within one standard deviation of the best score. The idea being that simple models generalize better, so it's better to select a simple model that had near-best performance.
simplicity_params	A character vector. For selection_method = "Breiman". These are passed directly to tune::select_by_one_std_err() and used to sort hyper_param_grid

by simplicity. To sort descending, put a minus in front of the parameter. For example, to sort ascending on "x" and then descending on "y", use `simplicity_params = c("x","-y")`. See `tune::select_by_one_std_err()` for details.

`include_nullmod`
: A bool. Include the null model (predicts mean or most common class every time) in the model comparison? This is recommended. If the null model comes within a standard deviation of the otherwise best model, the null model is chosen instead.

`err_if_nullmod`
: A bool. If the null model is chosen, throw an error rather than returning the null model.

`warn_if_nullmod`
: A bool. Warn if returning the null model?

`n_cores`
: A positive integer. The cross-validation can optionally be done in parallel. Specify the number of cores for parallel processing here.

Value

A parsnip::model_fit object. To use this fitted model `mod` to make predictions on some new data `df_new`, use `predict(mod,new_data = df_new)`.

See Also

Other model trainers: `train_glm()`, `train_lm()`

Examples

```
iris_data <- janitor::clean_names(datasets::iris)
iris_data_split <- rsample::initial_split(iris_data, strata = species)
iris_training_data <- rsample::training(iris_data_split)
iris_testing_data <- rsample::testing(iris_data_split)
mod <- train_gbm(
  training_data = iris_training_data, outcome = "species",
  metric = "mn_log_loss",
  hyper_param_grid = list(
    trees = c(20, 50),
    tree_depth = c(1, 2),
    learn_rate = c(0.01, 0.1)
  ),
  simplicity_params = c("trees", "learn_rate"),
  strata = c("species"),
  n_cores = 5
)
preds <- predict(mod, new_data = iris_testing_data, type = "prob")
dplyr::bind_cols(preds, truth = iris_testing_data$species)
yardstick::mn_log_loss_vec(
  truth = iris_testing_data$species,
  estimate = as.matrix(preds)
)
```

`train_glm` *Train a Generalized Lasso linear model.*

Description

Train a generalized Lasso linear model. The training routine automatically selects the best lambda parameter using `glmnet::cv.glmnet()`.

Usage

```
train_glm(
  training_data,
  outcome,
  metric = "mn_log_loss",
  na_action = c("medianimpute", "knnimpute"),
  lambda = NULL,
  cv_nfolds = 10,
  id_col = NULL,
  strata = NULL,
  selection_method = "Breiman",
  include_nullmod = TRUE,
  err_if_nullmod = FALSE,
  warn_if_nullmod = TRUE,
  n_cores = 1
)
```

Arguments

`training_data`	A data frame. The data used to train the model.
`outcome`	A string. The name of the outcome variable. This must be a column in `training_data`.
`metric`	A string. The probability metric to choose the best model. and select the best one. Common choices are `"mn_log_loss"`, `"roc_auc"` and `"accuracy"`. This should be a metric that is available in the `yardstick` package, but use e.g. "mae" and not "yardstick::mae" in this argument. If you specify this as a multi-element character vector, the first element will be used to select the best model; subsequent metrics will also be reported for that model in the `cv_performance` attribute of the the returned object.)‘
`na_action`	A string. How to impute missing data in explanatory variables`"medianimpute"` or `"knnimpute"`. See `recipes::step_medianimpute()` and `recipes::step_knnimpute()`. Default is `"medianimpute"`.
`lambda`	A numeric vector. Optional. A grid of lambdas for tuning the Lasso. If you leave this as `NULL`, recommended, a sensible grid is chosen for you.
`cv_nfolds`	A positive integer. The number of folds for cross-validation.
`id_col`	A string. If there is a sample identifier column, specify it here to tell the model not to use it as a predictor.
`strata`	A string. Variable to stratify on when splitting for cross-validation.

`selection_method`
: A string. How to select the best model. There are two options: "Breiman" and "absolute". "absolute" selects the best model by selecting the model with the best mean performance according to the chosen metric. "Breiman" selects the simplest model that comes within one standard deviation of the best score. The idea being that simple models generalize better, so it's better to select a simple model that had near-best performance.

`include_nullmod`
: A bool. Include the null model (predicts mean or most common class every time) in the model comparison? This is recommended. If the null model comes within a standard deviation of the otherwise best model, the null model is chosen instead.

`err_if_nullmod`
: A bool. If the null model is chosen, throw an error rather than returning the null model.

`warn_if_nullmod`
: A bool. Warn if returning the null model?

`n_cores`
: A positive integer. The cross-validation can optionally be done in parallel. Specify the number of cores for parallel processing here.

Details

The final model will be evaluated with the metric of your choice, but the hyperparameter tuning will be done using deviance. This is necessary to use `glmnet::cv.glmnet()`.

Value

A parsnip::model_fit object. To use this fitted model `mod` to make predictions on some new data `df_new`, use `predict(mod,new_data = df_new)`.

See Also

Other model trainers: `train_gbm()`, `train_lm()`

Examples

```
iris_data <- janitor::clean_names(datasets::iris)
iris_data_split <- rsample::initial_split(iris_data, strata = species)
mod <- train_glm(
  training_data = rsample::training(iris_data_split),
  outcome = "species",
  metric = c("mn_log_loss", "roc_auc"),
  n_cores = 5
)
preds <- predict(mod, new_data = rsample::testing(iris_data_split)) %>%
  magrittr::set_names("pred") %>%
  dplyr::mutate(truth = rsample::testing(iris_data_split)$species)
yardstick::accuracy(preds, truth, pred)
preds_prob <- predict(mod,
  new_data = rsample::testing(iris_data_split),
  type = "prob"
)
dplyr::bind_cols(preds_prob,
  truth = rsample::testing(iris_data_split)$species
)
```

```
yardstick::mn_log_loss_vec(
  truth = rsample::testing(iris_data_split)$species,
  estimate = as.matrix(preds_prob)
)
```

`train_lm` *Train a Lasso linear model.*

Description

Train a Lasso linear model. The training routine automatically selects the best lambda parameter using `glmnet::cv.glmnet()`.

Usage

```
train_lm(
  training_data,
  outcome,
  metric = c("rmse", "mae"),
  na_action = c("medianimpute", "knnimpute"),
  lambda = NULL,
  cv_nfolds = 10,
  id_col = NULL,
  strata = NULL,
  selection_method = "Breiman",
  include_nullmod = TRUE,
  err_if_nullmod = FALSE,
  warn_if_nullmod = TRUE,
  n_cores = 1
)
```

Arguments

`training_data`	A data frame. The data used to train the model.
`outcome`	A string. The name of the outcome variable. This must be a column in `training_data`.
`metric`	A string. `"rmse"` or `"mae"`.
`na_action`	A string. How to impute missing data in explanatory variables`"medianimpute"` or `"knnimpute"`. See `recipes::step_medianimpute()` and `recipes::step_knnimpute()`. Default is `"medianimpute"`.
`lambda`	A numeric vector. Optional. A grid of lambdas for tuning the Lasso. If you leave this as `NULL`, recommended, a sensible grid is chosen for you.
`cv_nfolds`	A positive integer. The number of folds for cross-validation.
`id_col`	A string. If there is a sample identifier column, specify it here to tell the model not to use it as a predictor.
`strata`	A string. Variable to stratify on when splitting for cross-validation.
`selection_method`	A string. How to select the best model. There are two options: "Breiman" and "absolute". "absolute" selects the best model by selecting the model with the best mean performance according to the chosen metric. "Breiman" selects the

simplest model that comes within one standard deviation of the best score. The idea being that simple models generalize better, so it's better to select a simple model that had near-best performance.

include_nullmod
: A bool. Include the null model (predicts mean or most common class every time) in the model comparison? This is recommended. If the null model comes within a standard deviation of the otherwise best model, the null model is chosen instead.

err_if_nullmod
: A bool. If the null model is chosen, throw an error rather than returning the null model.

warn_if_nullmod
: A bool. Warn if returning the null model?

n_cores
: A positive integer. The cross-validation can optionally be done in parallel. Specify the number of cores for parallel processing here.

Value

A parsnip::model_fit object. To use this fitted model `mod` to make predictions on some new data `df_new`, use `predict(mod,new_data = df_new)`.

See Also

Other model trainers: `train_gbm()`, `train_glm()`

Examples

```
iris_data <- janitor::clean_names(datasets::iris)
iris_data_split <- rsample::initial_split(iris_data, strata = species)
mod <- train_lm(
  training_data = rsample::training(iris_data_split),
  outcome = "petal_length",
  metric = "mae",
  n_cores = 5
)
preds <- predict(mod, new_data = rsample::testing(iris_data_split))
dplyr::bind_cols(preds,
  truth = rsample::testing(iris_data_split)$petal_length
)
yardstick::mae_vec(
  truth = rsample::testing(iris_data_split)$petal_length,
  estimate = preds[[1]]
)
```

---

train_on_grid *Train several models with different hyperparameters and select the best one.*

Description

Tune a model based on a provided hyperparameter grid by cross-validation. Select the best one, optionally including the null model in the comparison.

Usage

```
train_on_grid(
  mod_spec,
  hyper_param_grid,
  mod_rec,
  training_data,
  outcome,
  cv_nfolds,
  cv_nreps = 1,
  strata = NULL,
  id_col = NULL,
  metric,
  selection_method = "Breiman",
  simplicity_params = NULL,
  include_nullmod = TRUE,
  err_if_nullmod = FALSE,
  warn_if_nullmod = TRUE,
  n_cores = 1
)
```

Arguments

mod_spec	A parsnip model specification. It must include the model mode and engine. See parsnip::set_mode() and parsnip::set_engine().
hyper_param_grid	A data frame with one row per hyperparameter combination. The column names give the hyper parameter names. Can optionally be passed as a list which is made into a tibble by tidyr::expand_grid().
mod_rec	The recipe for preparing the data for this model. See recipes::recipe().
training_data	A data frame. The data used to train the model.
outcome	A string. The name of the outcome variable. This must be a column in training_data.
cv_nfolds	A positive integer. The number of folds for cross-validation.
cv_nreps	A positive integer. The number of repeated rounds in the cross-validation.
strata	A string. Variable to stratify on when splitting for cross-validation.
id_col	A string. If there is a sample identifier column, specify it here to tell the model not to use it as a predictor.
metric	A string. The metric to use to evaluate the models and select the best one. Common choices are "rmse", "mae", "roc_auc", "accuracy", "mn_log_loss". This should be a metric that is available in the yardstick package, but use e.g. "mae" and not "yardstick::mae" in this argumentIf you specify this as a multi-element character vector, the first element will be used to select the best model; subsequent metrics will also be reported for that model in the cv_performance attribute of the the returned object.)‘
selection_method	A string. How to select the best model. There are two options: "Breiman" and "absolute". "absolute" selects the best model by selecting the model with the best mean performance according to the chosen metric. "Breiman" selects the simplest model that comes within one standard deviation of the best score. The idea being that simple models generalize better, so it's better to select a simple model that had near-best performance.

simplicity_params
: A character vector. For `selection_method = "Breiman"`. These are passed directly to `tune::select_by_one_std_err()` and used to sort `hyper_param_grid` by simplicity. To sort descending, put a minus in front of the parameter. For example, to sort ascending on "x" and then descending on "y", use `simplicity_params = c("x","-y")`. See `tune::select_by_one_std_err()` for details.

include_nullmod
: A bool. Include the null model (predicts mean or most common class every time) in the model comparison? This is recommended. If the null model comes within a standard deviation of the otherwise best model, the null model is chosen instead.

err_if_nullmod
: A bool. If the null model is chosen, throw an error rather than returning the null model.

warn_if_nullmod
: A bool. Warn if returning the null model?

n_cores
: A positive integer. The cross-validation can optionally be done in parallel. Specify the number of cores for parallel processing here.

Value

A `parsnip model_fit` object with a `predict()` method and `recipe` and `cv_performance` attributes.

---

use_mirvie_gbm *Set up a GBM.*

Description

Copy boilerplate for fitting a GBM to the clipboard.

Usage

```
use_mirvie_gbm(mode, outcome)
```

Arguments

mode
: A string. `"classification"` or `"regression"`.

outcome
: A string. The name of the outcome variable in your data.

Value

The character vector copied to clipboard, invisibly.

use_mirvie_glm *Set up a GLM.*

Description

Copy boilerplate for fitting a GLM to the clipboard.

Usage

```
use_mirvie_glm(outcome)
```

Arguments

outcome	A string. The name of the outcome variable in your data.

Value

The character vector copied to clipboard, invisibly.

vimp *Get variable importances from a model or list of models.*

Description

This is a wrapper around `vip::vi()`. It returns variables in descending order of importance. For a list of models, it assumes that the models have all modelled the same problem with the same variables and the variables are summarized by median importance.

Usage

```
vimp(mod)
```

Arguments

mod	A `workflow` or `model_fit` object, or a list thereof.

Value

A tibble.

Examples

```
iris_data <- janitor::clean_names(datasets::iris)
iris_data_split <- rsample::initial_split(iris_data, strata = species)
mod <- train_glm(
  training_data = rsample::training(iris_data_split),
  outcome = "species",
  metric = "mn_log_loss",
  cv_nfolds = 5,
  n_cores = 5
)
vimp(mod)
vimp(list(mod, mod))
```

Index