

# Package: PAutilities (via r-universe)

May 13, 2026

**Type** Package

**Title** Streamline Physical Activity Research

**Version** 1.3.0

**Depends** R (>= 3.5.0)

**Description** Functions that support a broad range of common tasks in physical activity research, including but not limited to creation of Bland-Altman plots (<[doi:10.1136/bmj.313.7049.106](https://doi.org/10.1136/bmj.313.7049.106)>), metabolic calculations such as basal metabolic rate predictions (<<https://europepmc.org/article/med/4044297/reloa>>), demographic calculations such as age-for-body-mass-index percentile (<[https://www.cdc.gov/growthcharts/cdc\\_charts.htm](https://www.cdc.gov/growthcharts/cdc_charts.htm)>), and analysis of bout detection algorithm performance (<<https://pubmed.ncbi.nlm.nih.gov/34258524/>>).

**License** GPL-3

**Encoding** UTF-8

**LazyData** true

**URL** <https://github.com/paulhibbing/PAutilities>

**BugReports** <https://github.com/paulhibbing/PAutilities/issues>

**RoxygenNote** 7.3.3

**Imports** dplyr (>= 0.7), equivalence, ggplot2 (>= 4.0), graphics, lazyeval (>= 0.2), lubridate (>= 1.7.4), magrittr (>= 1.5), methods, reshape2, rlang (>= 0.3.1), stats, utils, Rcpp

**Suggests** testthat, knitr, rmarkdown, matchingMarkets (>= 1.0.1)

**VignetteBuilder** knitr

**LinkingTo** Rcpp

**Config/pak/sysreqs** libicu-dev

**Repository** <https://paulhibbing.r-universe.dev>

**Date/Publication** 2026-03-14 17:44:45 UTC

**RemoteUrl** <https://github.com/paulhibbing/pautilities>

**RemoteRef** HEAD

**RemoteSha** bf4c74602dce8b4be8ac7f12222783f8d4eedc00

## Contents

as . . . . .	3
ba_analysis . . . . .	3
ba_plot . . . . .	4
bout_mvpa . . . . .	5
cvd_risk . . . . .	7
descriptives . . . . .	9
df_continuous . . . . .	10
df_reorder . . . . .	11
epoch_length_sec . . . . .	11
ex_data . . . . .	12
full_days . . . . .	13
get_age . . . . .	14
get_bmr . . . . .	15
get_indices . . . . .	16
get_intensity . . . . .	17
get_kcal_vo2_conversion . . . . .	18
get_ree . . . . .	19
index_runs . . . . .	20
manage_procedure . . . . .	21
mean_sd . . . . .	22
paired_equivalence_test.data.frame . . . . .	23
PAutilities . . . . .	25
plot.paired_equivalence . . . . .	26
plot.spurious_curve . . . . .	27
plot.transition . . . . .	28
reintegrate . . . . .	28
residual_adjust . . . . .	29
rmr_sliding . . . . .	30
rolling_groups . . . . .	31
spurious_curve . . . . .	32
summaryTransition-class . . . . .	32
test_errors . . . . .	33
weight_status . . . . .	34
weir_equation . . . . .	35

**Index**

**36**

---

as	<i>As("summaryTransition", "data.frame")</i>
----	--

---

**Description**

As("summaryTransition", "data.frame")

As("summaryTransition", "list")

---

ba_analysis	<i>Perform Bland-Altman analysis on a data frame</i>
-------------	--

---

**Description**

Perform Bland-Altman analysis on a data frame

**Usage**

```
ba_analysis(df, x_var, y_var, regress_against = c("Y", "XY_mean"), ...)
```

**Arguments**

df	the data frame on which to operate
x_var	character. The column name of the X variable
y_var	character. The column name of the Y variable (criterion measure, if applicable)
regress_against	character. One of "Y" (to regress bias against yvar) or "XY_mean" (to regress bias against rowMeans(x_var, y_var)).
...	optional arguments passed to data.frame, e.g. to give the output results a label

**Value**

A data frame that has various summaries (means, standard deviations, and missing data details) plus mean bias (mean\_bias column) and limits of agreement (lower\_LOA and upper\_LOA columns)

**Examples**

```
data(ex_data, package = "PAutilities")
ba_analysis(ex_data, "Axis1", "Vector.Magnitude", "XY_mean")
ba_analysis(
  ex_data, "Axis1", "Vector.Magnitude", "XY_mean",
  an_arbitrary_added_column = "Example of passing an extra column"
)
```

---

ba\_plot *Create a Bland-Altman plot*

---

### Description

Create a Bland-Altman plot

### Usage

```
ba_plot(plotdata, x_var, y_var, x_name, y_name, shape = 16, ...)
```

### Arguments

plotdata	dataframe from which to build the plot
x_var	character expression to evaluate for the x-axis
y_var	character expression to evaluate for the y-axis
x_name	axis label for the x-axis
y_name	axis label for the y-axis
shape	numeric. The point shape to display.
...	further arguments passed to theme

### Value

a Bland-Altman plot

### References

Bland, J. M., & Altman, D. G. (1986). Statistical methods for assessing agreement between two methods of clinical measurement. *lancet*, 1(8476), 307-310.

### See Also

[ba\\_analysis](#)

### Examples

```
data(ex_data, package = "PAutilities")

# Reduce the number of data points (for illustration purposes) by isolating
# the 150 largest cases

illustration_threshold <-
  quantile(ex_data$Axis1, probs = 1 - (150 / nrow(ex_data)))
ex_data <- ex_data[ex_data$Axis1 > illustration_threshold, ]

# Generate the plot
my_ba <- ba_plot(
```

```

    ex_data,
    "(Axis1 + Axis3) / 2",
    "Axis1 - Axis3",
    "mean(Axis1, Axis3)",
    "Axis1 - Axis3"
  )

my_ba

# You can add to the plot as you would a normal ggplot object
my_ba +
  ggplot2::geom_text(
    x = 2000, y = 9000, label = "A",
    size = 8, fontface = "bold", colour = "blue"
  )

# With caution, you can change some automatic options (e.g. color of
# regression line) by overwriting in a new layer

my_ba + ggplot2::geom_smooth(method = "lm", se = FALSE, colour = "blue")

```

---

 bout\_mvpa

*Classify moderate-to-vigorous physical activity in bouts of a specific minimum length*

---

## Description

Classify moderate-to-vigorous physical activity in bouts of a specific minimum length

## Usage

```

bout_mvpa(
  intensity,
  var_type = c("METs", "Intensity"),
  min_duration = 10,
  termination = 3,
  MoreArgs = list(breaks = c(-Inf, 1.51, 3, Inf), labels = c("SB", "LPA", "MVPA"), right
    = FALSE),
  ...,
  timestamps = NULL,
  output_var = c("is_MVPA", "bout_tracker")
)

```

**Arguments**

intensity	a vector of intensity classifications to be re-classified according to the bout definition
var_type	character scalar indicating whether the intensity variable is a numeric vector of metabolic equivalents, or a factor variable giving activity intensity classification
min_duration	numeric scalar: minimum duration of a qualifying bout, in minutes
termination	numeric scalar: consecutive minutes of non-MVPA required to terminate the bout
MoreArgs	required arguments passed to cut
...	optional arguments passed to cut for converting METs to intensity classification
timestamps	optional vector of POSIX-formatted timestamps. Must have same length as intensity
output_var	the output variable(s) to give

**Value**

based on the setting of `output_var`, one or both of `is_MVPA` and `bout_tracker` will be returned, the former being a vector of indicators (1 or 0) specifying whether a minute is part of a valid MVPA bout, and the latter being a collapsed data frame giving only the valid bouts of MVPA and the relevant information (i.e., duration of the bout, minutes of MVPA, and percentage of time spent in MVPA within the bout). If both are selected, they are returned in a list.

**Examples**

```
data(ex_data, package = "PAutilities")
ex_data$DateTime <- as.POSIXct(ex_data$DateTime, "UTC")

# Runs with a warning

bout_mvpa(ex_data$METs, "METs")

bout_mvpa(ex_data$METs, "METs", timestamps = ex_data$DateTime)

# Recommended usage
lapply(split(ex_data, strftime(ex_data$DateTime, "%Y-%m-%d", "UTC")),
function(x) {
  bout_mvpa(x$METs, "METs", timestamps = x$DateTime)
})

lapply(split(ex_data, strftime(ex_data$DateTime, "%Y-%m-%d", "UTC")),
function(x) {
  bout_mvpa(x$METs,
"METs",
timestamps = x$DateTime,
output_var = "is_MVPA")
```

```
  })

  lapply(split(ex_data, strftime(ex_data$DateTime, "%Y-%m-%d", "UTC")),
    function(x) {
      bout_mvpa(x$METs,
        "METs",
        timestamps = x$DateTime,
        output_var = "bout_tracker")
    })
}
```

---

cvd\_risk

*Calculate risk of cardiovascular disease*

---

### Description

Calculate risk of cardiovascular disease

### Usage

```
cvd_risk(
  x = NULL,
  method = "D'Agostino_2008",
  sex,
  age,
  total_cholesterol,
  hdl,
  systolic,
  bp_treated,
  diabetes,
  smoker,
  points = TRUE,
  ...
)
```

## Default S3 method:

```
cvd_risk(
  x = NULL,
  method = "D'Agostino_2008",
  sex,
  age,
  total_cholesterol,
  hdl,
  systolic,
  bp_treated,
  diabetes,
  smoker,
  points = TRUE,
```

```

    ...
)

## S3 method for class 'data.frame'
cvd_risk(
  x = NULL,
  method = "D'Agostino_2008",
  sex,
  age,
  total_cholesterol,
  hdl,
  systolic,
  bp_treated,
  diabetes,
  smoker,
  points = TRUE,
  combine = TRUE,
  ...
)

```

### Arguments

x	optional data frame. If provided, the other arguments will be taken as column names under the assumption that each row represents a separate person, and each column provides one of the requested pieces of information
method	character. Currently only method = "D'Agostino_2008" is supported.
sex	character scalar indicating either sex for one person (i.e., male or female), or a column name in x containing sex values for multiple people
age	either a numeric scalar indicating age for one person, or a character scalar indicating the name of the column in x that contains age information. Units are years
total_cholesterol	same as age, but for total cholesterol, in mg/dL
hdl	same as age, but for HDL, in mg/dL
systolic	same as age, but for systolic blood pressure, in mmHg
bp_treated	either a logical scalar indicating whether a person is taking blood pressure medication, or a character scalar pointing to the column in x that contains the same information for multiple people
diabetes	same as bp_treated, but for the presence of diabetes
smoker	same as bp_treated, but for smoking status
points	logical. Return as points (default) or risk percentage?
...	arguments passed to other methods
combine	logical. Give results as a list of risk_profile objects, or combine the list into an integer vector (default)?

**Value**

One or more risk profiles (for default method with `points = TRUE`, or for data frames with `combine = FALSE & points = TRUE`). Otherwise numeric risk percentage (for `points = FALSE`, scalars and data frames) or an integer vector (for data frames with `combine = TRUE & points = FALSE`)

**References**

[D'Agostino et al. \(2008\)](#)

**Examples**

```
cvd_risk(sex = "Female", age = 111, total_cholesterol = 111, systolic = 111,
hd1 = 11, bp_treated = FALSE, diabetes = TRUE, smoker = TRUE)
```

```
df <- data.frame(
  sex = sample(c("Male", "Female"), 5, TRUE),
  age = sample(30:100, 5, TRUE),
  tc = sample(150:300, 5, TRUE),
  hd1 = sample(30:70, 5, TRUE),
  sbp = sample(100:180, 5, TRUE),
  bpmed = sample(c(TRUE, FALSE), 5, TRUE),
  diabetes = sample(c(TRUE, FALSE), 5, TRUE),
  smoker = sample(c(TRUE, FALSE), 5, TRUE)
)
```

```
cvd_risk(
  df, sex = "sex", age = "age",
  total_cholesterol = "tc", hd1 = "hd1",
  systolic = "sbp", bp_treated = "bpmed",
  diabetes = "diabetes", smoker = "smoker",
  combine = FALSE
)
```

---

descriptives

*Compute descriptive statistics for a variable in the metabolic data set*

---

**Description**

Compute descriptive statistics for a variable in the metabolic data set

**Usage**

```
descriptives(dataset, variable, group = NULL)
```

**Arguments**

dataset	the dataset to analyze
variable	character scalar giving the variable name to summarize
group	character scalar giving an optional grouping variable for the summary

**Value**

a data frame of formatted summary statistics

**Examples**

```
data(ex_data, package = "PAutilities")
ex_data$group_var <- rep(
  c("One", "Two", "Three"),
  each = ceiling(nrow(ex_data)/3)
)[seq(nrow(ex_data))]
descriptives(ex_data, "Axis1", "group_var")
```

---

df\_continuous

*Check if a dataframe is continuous*


---

**Description**

Check if a dataframe is continuous

**Usage**

```
df_continuous(df, time_var = "Timestamp", digits = 6, ...)
```

**Arguments**

df	the input data frame
time_var	character scalar giving the column name of the variable containing timestamp information (either character or POSIXt format)
digits	see <a href="#">epoch_length_sec</a>
...	arguments passed to as.POSIXct, for use if time_var is a character rather than POSIXt variable

**Value**

a logical scalar indicating whether the dataframe reflects a continuous time series

**Examples**

```
data(ex_data, package = "PAutilities")
df_continuous(ex_data, "DateTime", tz = "UTC")
df_continuous(ex_data[-c(300:500), ], "DateTime", tz = "UTC")
```

---

df_reorder	<i>Reorder the columns of a data frame</i>
------------	--

---

**Description**

Reorder the columns of a data frame

**Usage**

```
df_reorder(df, columns, after)
```

**Arguments**

df	the data frame
columns	the column(s) to move (either as character names or numeric indices)
after	the column after which to insert columns (must be a scalar, either a character name or a numeric index)

**Value**

The reordered data frame

**Examples**

```
df <- data.frame(a = 1:10, b = 11:20, c = 21:30, d = 31:40)
df_reorder(df, 2:3, "d")
df_reorder(df, c("c", "d"), "a")
```

---

epoch_length_sec	<i>Determine epoch length in seconds</i>
------------------	--

---

**Description**

Determine epoch length in seconds

**Usage**

```
epoch_length_sec(timestamps, digits = 6)
```

**Arguments**

timestamps	POSIX-formatted input
digits	for rounding. See details

## Details

The function is designed to work even when the epoch length is less than one second (e.g., for raw accelerometry data). Thus, it is not possible to base the code on convenient `difftime` methods. Instead, numeric operations are performed after running `unclass` on the input. This sometimes results in minuscule fluctuations of the calculated epoch length (e.g., +/- 0.0000002). Thus, the code rounds everything to the precision indicated by `digits`. For most applications, the default value (`digits = 6`) should be well past the range of meaningful fluctuations and lead to a favorable outcome. But the `digits` argument can also be adjusted if greater assurance is needed.

After rounding, the code checks for the existence of multiple epoch lengths. If they are detected (e.g., due to a discontinuity in the file), a warning is issued and the most prevalent epoch length is returned. The warning will specify all the different epoch lengths that were detected, which may be useful information for troubleshooting.

## Value

The epoch length of the data, in seconds

## Examples

```
epoch_length_sec(Sys.time() + 0:5)
epoch_length_sec(Sys.time() + seq(0, 25, 5))
```

---

ex_data	<i>Example data for calculating bouts of moderate-to-vigorous physical activity</i>
---------	---

---

## Description

A dataset containing accelerometer data and predicted energy expenditure in metabolic equivalents (METs) that can be used to classify moderate-to-vigorous physical activity in continuous bouts.

## Usage

```
ex_data
```

## Format

A data frame with 10080 rows and 12 variables:

**FileID** character. Name of the file originating the data

**Date** character giving the date ("%m/%d/%Y")

**Time** character giving the time ("%H:%M:%S")

**DateTime** full timestamp (%Y-%m-%d %H:%M:%S) given as character

**dayofyear** numeric day of the year (i.e., julian date)

**minofday** numeric minute of the day (i.e., 0 for midnight and 1439 for 11:59)

**Axis1** activity counts for the device's first axis

**Axis2** activity counts for the device's second axis

**Axis3** activity counts for the device's third axis

**Steps** number of steps taken

**Vector.Magnitude** vector magnitude (Euclidian norm) of the activity counts from the three axes

**METs** predicted energy expenditure, in metabolic equivalents

---

full\_days

*Drop incomplete days from a dataset*

---

### Description

Drop incomplete days from a dataset

### Usage

```
full_days(
  df,
  time_var = "Timestamp",
  drop = c("all", "leading", "trailing", "label"),
  epoch_length_sec = NULL,
  label_name = "is_full_day",
  digits = 6,
  check_continuous = TRUE,
  discontinuous_action = c("stop", "warn"),
  ...
)
```

### Arguments

df	the input data frame
time_var	character scalar giving the column name of the variable containing timestamp information (either character or POSIXt format)
drop	character scalar indicating which incomplete days to drop. Can be all (default), leading (only day/s at the start of the file), trailing (only day/s at the end of the file), or label. If the latter is selected, the full dataset is returned with an additional column indicating whether each row of data corresponds with a complete day (useful for troubleshooting, among other things)
epoch_length_sec	optional. The epoch length of the data. If no value is passed, <a href="#">epoch_length_sec</a> is invoked on the time_var column
label_name	character scalar. Name to give the indicator column when drop == "label"
digits	see <a href="#">epoch_length_sec</a>

```

check_continuous      logical. Check the dataframe after dropping to see if it is continuous?
discontinuous_action  character scalar telling what to do if a discontinuity is expected when check_continuous
                      = TRUE. Can be either warn (the default) or stop
...                  arguments passed to as.POSIXct, for use if time_var is a character rather than
                      POSIXt variable

```

**Value**

an updated copy of df, in which incomplete days are addressed according to the selected value of drop.

**See Also**

[df\\_continuous](#)

**Examples**

```

data(ex_data, package = "PAutilities")
ex_data <- full_days(
  ex_data, "DateTime", "label", 60,
  "full_day_indicator", tz = "UTC"
)
head(ex_data)

```

---

get\_age

*Calculate age*

---

**Description**

Takes two Date objects and calculates age based on [difftime](#) (in days) divided by 365.2425 days per year (for age in years) or 30.4375 days per month (for age in months).

**Usage**

```
get_age(birthdate, current_date, units = c("years", "months"))
```

**Arguments**

```

birthdate      Date object giving the date of birth
current_date    Date object giving the date from which age is to be calculated
units          The units in which age should be reported

```

**Value**

Numeric value giving age in the specified units.

**Examples**

```
get_age(as.Date("2000-01-01"), Sys.Date(), "years")
```

---

get_bmr	<i>Retrieve estimated basal metabolic rate for an individual</i>
---------	--

---

**Description**

Retrieve estimated basal metabolic rate for an individual

**Usage**

```
get_bmr(
  Sex = c("M", "F"),
  Ht = NULL,
  Wt,
  Age,
  verbose = FALSE,
  RER = NULL,
  equation = c("ht_wt", "wt", "both"),
  kcal_table = c("Lusk", "Peronnet", "both"),
  method = c("Schofield", "FAO", "both"),
  MJ_conversion = c("thermochemical", "dry", "convenience", "all"),
  kcal_conversion = 5
)
```

**Arguments**

Sex	The individual's sex
Ht	The individual's height, in meters
Wt	The individual's weight, in kilograms
Age	The individual's age, in years
verbose	Logical. Should processing updates be printed?
RER	numeric. The respiratory exchange ratio
equation	The equation to apply
kcal_table	The table to reference for converting kilocalories to oxygen consumption. See <a href="#">get_kcal_vo2_conversion</a>
method	The calculation method to use
MJ_conversion	The value to use for converting megajoules to kilocalories. Defaults to thermochemical.
kcal_conversion	numeric. If RER is NULL (default), the factor to use for converting kilocalories to oxygen consumption

**Value**

a data frame containing predictions of basal metabolic rate in one column, along with additional columns that indicate how the predictions were obtained (e.g., which sources and conversions were applied)

**References**

Schofield, W. N. (1985). Predicting basal metabolic rate, new standards and review of previous work. *Human nutrition. Clinical nutrition*, 39, 5-41.

**Examples**

```
# Get BMR for an imaginary 900-year-old person (Age is only
# used to determine which equations to use, in this case the
# equations for someone older than 60)

get_bmr(
  Sex = "M", Ht = 1.5, Wt = 80, Age = 900, equation = "both",
  method = "both", RER = 0.865, kcal_table = "both",
  MJ_conversion = c("all")
)

get_bmr(
  Sex = "M", Ht = 1.5, Wt = 80, Age = 900, MJ_conversion = "all",
  kcal_conversion = 4.86
)

get_bmr(
  Sex = "M", Ht = 1.5, Wt = 80, Age = 900, method = "FAO",
  kcal_conversion = 4.86
)
```

---

get\_indices

*Retrieve indices for a rolling window analysis*


---

**Description**

Retrieve indices for a rolling window analysis

**Usage**

```
get_indices(y_var, window_size = 15L)
```

**Arguments**

**y\_var** NumericVector. Input on which to define the indices for each roll of the window  
**window\_size** int. The size of the window

**Value**

a list in which each element contains `window_size` consecutive integers that indicate which elements of `y_var` would be extracted for that roll of the window

**Note**

For this function, the output elements contain positions (i.e., indices) from `y_var`, whereas for [rolling\\_groups](#) the output elements contain the raw values found at each index

**See Also**

[rolling\\_groups](#)

**Examples**

```
result <- get_indices(1:100, 10)
head(result)
tail(result)
```

---

get_intensity	<i>Classify activity intensity</i>
---------------	------------------------------------

---

**Description**

Supports intensity classification via energy expenditure with or without additional posture requirements (i.e., for sedentary behavior to be in lying/seated posture)

**Usage**

```
get_intensity(mets, posture = NULL, ...)
```

**Arguments**

<code>mets</code>	numeric vector of metabolic equivalents to classify
<code>posture</code>	character vector of postures
<code>...</code>	further arguments passed to <code>cut</code>

**Details**

If `breaks` and `labels` arguments are not provided, default values are  $\leq 1.5$  METs for sedentary behavior, 1.51-2.99 METs for light physical activity, and  $\geq 3.0$  METs for moderate-to-vigorous physical activity.

It is expected for the elements of `posture` to be one of `c("lie", "sit", "stand", "other")`. The function will run (with a warning) if that requirement is not met, but the output will likely be incorrect.

**Value**

a factor giving intensity classifications for each element of mets

**Examples**

```
mets <- seq(1, 8, 0.2)
posture <- rep(
  c("lie", "sit", "stand", "other"), 9
)

intensity_no_posture <- get_intensity(mets)
intensity_posture <- get_intensity(mets, posture)
head(intensity_no_posture)
head(intensity_posture)
```

---

get\_kcal\_vo2\_conversion

*Retrieve conversion factors from kilocalories to oxygen consumption*

---

**Description**

Retrieve conversion factors from kilocalories to oxygen consumption

**Usage**

```
get_kcal_vo2_conversion(RER, kcal_table = c("Lusk", "Peronnet", "both"))
```

**Arguments**

RER                numeric. The respiratory exchange ratio  
kcal\_table        The table to reference for converting kilocalories to oxygen consumption. See [get\\_kcal\\_vo2\\_conversion](#)

**Details**

RER values are matched to the table entries based on the minimum absolute difference. If there is a tie, the lower RER is taken.

**Value**

numeric vector giving the conversion factor from the specified table(s)

**References**

Peronnet, F., & Massicotte, D. (1991). Table of nonprotein respiratory quotient: an update. *Can J Sport Sci*, 16(1), 23-29.  
Lusk, G. (1924). Analysis of the oxidation of mixtures of carbohydrate and fat: a correction. *Journal of Biological Chemistry*, 59, 41-42.

**Examples**

```
get_kcal_vo2_conversion(0.85, "both")
```

---

```
get_ree Calculate resting energy expenditure
```

---

**Description**

Calculate resting energy expenditure

**Usage**

```
get_ree(
  method = c("harris_benedict", "schofield_wt", "schofield_wt_ht", "fao", "muller_wt_ht",
             "muller_ffm"),
  sex,
  age_yr = NA,
  ...,
  output = c("default", "mj_day", "kcal_day", "vo2_ml_min"),
  calorie = c("thermochemical", "convenience", "dry"),
  RER = 0.86,
  kcal_table = c("Lusk", "Peronnet", "both"),
  df = NULL
)
```

**Arguments**

method	character. The equation(s) to use, chosen from "harris_benedict", "schofield_wt", "schofield_wt_ht", "fao", "muller_wt_ht", or "muller_ffm"
sex	character. The participant/patient sex, one of "female" or "male"
age_yr	numeric. The participant/patient age in years. Not used for method = "muller_ffm", but a value must still be given if a data frame is passed. (The value does not need to correspond with age, it is simply a placeholder to satisfy internal checks that are applied to all equations when making computations on a data frame.)
...	arguments (e.g. wt_kg or ht_cm) for calculations. An error message will clarify which variables need to be passed if they are missing
output	character. The desired output unit(s), chosen from "default", "mj_day", "kcal_day", or "vo2_ml_min"
calorie	character. The desired conversion factor(s) for calculating MJ from kcal, chosen from "thermochemical", "convenience", or "dry"
RER	numeric. The respiratory exchange ratio
kcal_table	character. The desired conversion table(s) to use for converting kcal to oxygen consumption, chosen from "Lusk", "Peronnet", or "both"
df	optional data frame. If passed, all prior arguments should be character scalars pointing to a column in df that contains the corresponding information is stored

**Value**

Calculated resting energy expenditure

**Examples**

```
get_ree("schofield_wt_ht", "female", 57.8, wt_kg = 80, ht_m = 1.50)
```

---

index\_runs

*Run length encoding with indices*

---

**Description**

Run length encoding with indices

**Usage**

```
index_runs(x, zero_index = FALSE)
```

**Arguments**

x                    vector of values on which to perform run length encoding  
zero\_index          logical. Should indices be indexed from zero (useful for Rcpp)?

**Value**

A data frame with information about the runs and start/stop indices

**Examples**

```
x <- c(
  FALSE, TRUE, FALSE, FALSE, FALSE, TRUE,
  FALSE, TRUE, TRUE, FALSE, TRUE, FALSE,
  FALSE, FALSE, FALSE, FALSE, TRUE, TRUE,
  FALSE, TRUE
)
head(index_runs(x))
```

---

manage_procedure	<i>Printing and timing utility for managing processes</i>
------------------	---

---

**Description**

Printing and timing utility for managing processes

**Usage**

```
manage_procedure(part = c("Start", "End"), ..., timer = NULL, verbose = TRUE)
get_duration(timer)
```

**Arguments**

part	character scalar, either Start or End.
...	character strings to print. Default messages will print if no arguments are provided.
timer	a proc_time object. Required for manage_procedure only if using the default message for part = "End" default message.
verbose	logical. Print to console?

**Value**

For part = "Start", a proc\_time object (i.e., a timer passable to an eventual part = "End" command); for part = "End", invisible

**Examples**

```
manage_procedure("Start", "String will be printed\n")
timer <- manage_procedure(
  "Start", "Printing a string is optional", verbose = FALSE
)

## Default starting message
manage_procedure("Start")

## Default ending message
manage_procedure("End", timer = timer)

## Other examples
get_duration(timer)
manage_procedure("End", "Custom ending message")
```

---

mean_sd	<i>Compute the mean and standard deviation of a vector, returning a formatted string containing the values as 'M +/- SD'</i>
---------	--

---

### Description

Compute the mean and standard deviation of a vector, returning a formatted string containing the values as 'M +/- SD'

### Usage

```
mean_sd(
  x = NULL,
  MoreArgs = NULL,
  give_df = TRUE,
  ...,
  mean_x = NULL,
  sd_x = NULL
)

## Default S3 method:
mean_sd(
  x = NULL,
  MoreArgs = NULL,
  give_df = TRUE,
  ...,
  mean_x = NULL,
  sd_x = NULL
)

## S3 method for class 'data.frame'
mean_sd(
  x = NULL,
  MoreArgs = NULL,
  give_df = TRUE,
  ...,
  mean_x = NULL,
  sd_x = NULL
)
```

### Arguments

x	numeric vector of values to summarize
MoreArgs	named list of arguments to pass to mean and sd
give_df	logical. Should mean, sd, and summary string be returned in a data frame?
...	additional arguments passed to format

mean\_x            an already-calculated mean value for x  
sd\_x              an already-calculated sd value for x

**Value**

either a formatted character scalar (if `give_df == FALSE`), or else a data frame containing columns for the mean value, standard deviation, and formatted character string combining the two.

**Examples**

```
mean_sd(rnorm(100, 50))
```

---

```
paired_equivalence_test.data.frame  
                                  Perform equivalence testing on paired samples
```

---

**Description**

Perform equivalence testing on paired samples

**Usage**

```
## S3 method for class 'data.frame'  
paired_equivalence_test(  
  x,  
  y,  
  y_type = c("both", "criterion", "comparison"),  
  alpha = 0.05,  
  na.rm = TRUE,  
  scale = c("relative", "absolute"),  
  absolute_region_width = NULL,  
  relative_region_width = NULL,  
  ...  
)
```

```
## Default S3 method:  
paired_equivalence_test(  
  x,  
  y,  
  y_type = c("both", "criterion", "comparison"),  
  alpha = 0.05,  
  na.rm = TRUE,  
  scale = c("relative", "absolute"),  
  absolute_region_width = NULL,  
  relative_region_width = NULL,  
  ...  
)
```

```

)

paired_equivalence_test(
  x,
  y,
  y_type = c("both", "criterion", "comparison"),
  alpha = 0.05,
  na.rm = TRUE,
  scale = c("relative", "absolute"),
  absolute_region_width = NULL,
  relative_region_width = NULL,
  ...
)

```

### Arguments

<code>x</code>	numeric vector representing the (possibly surrogate) sample
<code>y</code>	numeric vector representing the (possibly criterion) sample. Index paired with <code>x</code>
<code>y_type</code>	classification of <code>y</code> for the purpose of analysis. Can be "criterion", "comparison", or "both".
<code>alpha</code>	the alpha level for the test
<code>na.rm</code>	logical. Omit mean values for mean calculations?
<code>scale</code>	character specifying whether the test should occur on an absolute or relative scale. Must be one of "relative" (default) or "absolute".
<code>absolute_region_width</code>	the region width for use when <code>scale = "absolute"</code>
<code>relative_region_width</code>	the region width for use when <code>scale = "relative"</code>
<code>...</code>	further arguments passed to methods. Currently unused.

### Value

a 'paired\_equivalence' object summarizing the test input and results

### Note

If a value is not specified for the region width that corresponds with `scale`, a default value will be assigned with a warning.

### References

[Dixon et al.](#)

## Examples

```
set.seed(1544)
x <- data.frame(
  var1 = rnorm(500, 15, 4),
  var2 = rnorm(500, 23, 7.3)
)
y <- rnorm(500, 17.4, 9)

test_result <- paired_equivalence_test(
  x, y, relative_region_width = 0.25
)

lapply(test_result, head)
```

---

PAutilities

*PAutilities: Streamline Physical Activity Research*

---

## Description

A collection of utilities that are useful for a broad range of tasks that are common in physical activity research. The main features (with associated functions in parentheses) are:

## Details

\* Bland-Altman plots ([ba\\_plot](#)) \* Bout analysis for moderate-to-vigorous physical activity ([bout\\_mvpa](#))  
\* Formatted descriptive statistics [descriptives](#) \* Demographic calculations ([get\\_age](#) and [get\\_BMI\\_percentile](#))  
\* Metabolic calculations ([get\\_bmr](#), [weir\\_equation](#), and [get\\_kcal\\_vo2\\_conversion](#)) \* Analysis of bout detection algorithm performance ([get\\_transition\\_info](#) and associated methods, e.g. [summary](#) and [plot](#))

## Author(s)

**Maintainer:** Paul R. Hibbing <[paulhibbing@gmail.com](mailto:paulhibbing@gmail.com)>

Other contributors:

- Centers for Disease Control and Prevention [contributor]

## See Also

Useful links:

- <https://github.com/paulhibbing/PAutilities>
- Report bugs at <https://github.com/paulhibbing/PAutilities/issues>

---

`plot.paired_equivalence`*Plot the outcome of a paired equivalence test*

---

### Description

Plot the outcome of a paired equivalence test

### Usage

```
## S3 method for class 'paired_equivalence'  
plot(x, shade = "auto", ...)  
  
shaded_equivalence_plot(results, ...)  
  
unshaded_equivalence_plot(results, ...)
```

### Arguments

<code>x</code>	the object to be plotted
<code>shade</code>	logical. Should the results be plotted using a shaded equivalence region?
<code>...</code>	arguments passed to <code>ggplot2::theme</code> .
<code>results</code>	data frame. The results component of a <code>paired_equivalence</code> object

### Details

`shaded_equivalence_plot` plots the results of an equivalence test in which a single equivalence region applies to all variables. In that case, the equivalence region is displayed as a shaded region. `unshaded_equivalence_plot` plots the results of an equivalence test in which variables have unique equivalence regions. In that case, the equivalence regions are displayed as dodged "confidence intervals".

### Value

A plot of the equivalence test

### Examples

```
set.seed(1544)  
y <- rnorm(500, 17.4, 9)  
z <- data.frame(  
  var1 = rnorm(500, 15, 4),  
  var2 = rnorm(500, 23, 7.3)  
)  
  
# Optionally create artificial missing values to trigger unshaded plot  
missing_indices <- sample(seq(nrow(z)), 250)
```

```
z$var1[missing_indices] <- NA

x <- paired_equivalence_test(
  z, y, "criterion", scale = "relative",
  relative_region_width = 0.25
)

plot(x)
```

---

plot.spurious\_curve    *Plot a spurious curve*

---

### Description

Plot a spurious curve

### Usage

```
## S3 method for class 'spurious_curve'
plot(x, ...)
```

### Arguments

x                    a spurious\_curve object  
...                   further arguments (currently unused)

### Value

a plot of the object

### See Also

[spurious\\_curve](#)

### Examples

```
set.seed(100)
predictions <- (sample(1:100)%2)
references <- (sample(1:100)%2)

trans <- get_transition_info(
  predictions, references, 7
)
result <- spurious_curve(trans)
plot(result)
```

---

plot.transition	<i>Plot the transitions and matchings from a transition object</i>
-----------------	--

---

**Description**

Plot the transitions and matchings from a transition object

**Usage**

```
## S3 method for class 'transition'  
plot(x, ...)
```

**Arguments**

x	the object to plot
...	further methods passed to or from methods, currently unused

**Value**

A plot of the predicted and actual transitions in a transition object, as well as the matchings between them

**Examples**

```
predictions <- (sample(1:100)%2)  
references <- (sample(1:100)%2)  
window_size <- 7  
if (isTRUE(requireNamespace("matchingMarkets", quietly = TRUE))){  
  transitions <- get_transition_info(  
    predictions, references, window_size  
  )  
  plot(transitions)  
}
```

---

reintegrate	<i>Reintegrate accelerometer data</i>
-------------	---------------------------------------

---

**Description**

Reintegrate accelerometer data

**Usage**

```
reintegrate(df, target_sec, time_var = "Timestamp")
```

**Arguments**

df	a data frame to reintegrate
target_sec	the desired epoch length of the output. Starting epoch length will be determined automatically
time_var	The name of the column containing POSIX-formatted timestamp information

**Value**

A data frame of reintegrated data. Numeric variables will be summed, and others will be labeled with the first value.

**Examples**

```
##Example data in 60-second epochs
data(ex_data, package = "PAutilities")
ex_data$Timestamp <- as.POSIXct(
  paste(ex_data$Date, ex_data$Time),
  tz = "UTC",
  format = "%m/%d/%Y %H:%M:%S"
)

##Reintegrate to 120-second epochs
result <- reintegrate(ex_data, 120)

utils::head(result)
```

---

residual_adjust	<i>Perform residual adjustment on an epidemiologic variable</i>
-----------------	---

---

**Description**

Perform residual adjustment on an epidemiologic variable

**Usage**

```
residual_adjust(d, variable, confounder, label, verbose = FALSE)
```

**Arguments**

d	the input data frame on which to perform the adjustment
variable	character. Name of variable needing adjustment
confounder	character. Name of the confounder to adjust for
label	character. Name to give the adjusted variable
verbose	logical. Print updates to console?

**Value**

The original d object, with an extra column reflecting residual adjustments on the selected variable.

**Examples**

```
d <- data.frame(
  VARIABLE = rnorm(100, 10, 2),
  CONFOUNDER = rnorm(100, 3, 7)
)
result <- residual_adjust(d, "VARIABLE", "CONFOUNDER", "ADJUSTED")

head(d)
head(result)
```

---

rmr\_sliding

*Calculate resting metabolic rate using a sliding window method*

---

**Description**

Calculate resting metabolic rate using a sliding window method

**Usage**

```
rmr_sliding(
  vo2_values,
  vo2_timestamps,
  start_time,
  stop_time,
  window_size_minutes = 5
)
```

**Arguments**

vo2\_values        numeric vector of oxygen consumption values  
vo2\_timestamps    timestamps corresponding to each element of vo2\_values  
start\_time        the beginning time of the assessment period  
stop\_time         the ending time of the assessment period  
window\_size\_minutes    the size of the sliding window, in minutes

**Value**

A data frame giving the oxygen consumption from the lowest window, as well as the time difference from first to last breath in the same window.

**Examples**

```
set.seed(144)
fake_start_time <- Sys.time()
fake_stop_time <- fake_start_time + 1800
fake_timestamps <- fake_start_time + cumsum(sample(1:3, 500, TRUE))
fake_timestamps <- fake_timestamps[fake_timestamps <= fake_stop_time]
fakeBreaths <- rnorm(length(fake_timestamps), 450, 0.5)
window_size <- 5

rnr_sliding(
  fakeBreaths, fake_timestamps,
  fake_start_time, fake_stop_time,
  window_size
)
```

---

rolling_groups	<i>Loop along a vector, returning n elements at a time in a list</i>
----------------	--

---

**Description**

Loop along a vector, returning n elements at a time in a list

**Usage**

```
rolling_groups(values, n = 2L)
```

**Arguments**

values	IntegerVector. The vector to loop along
n	int. The number of elements to return in each element of the resulting list

**Value**

a list in which each element contains n elements from values

**Note**

For this function, the output elements contain raw values from values, whereas for [get\\_indices](#) the output elements contain the positions (i.e., indices) rather than the raw values

**See Also**

[get\\_indices](#)

**Examples**

```
groups <- rolling_groups(0:50, 3)
head(groups)
tail(groups)
```

---

spurious_curve	<i>Perform a spurious curve analysis</i>
----------------	--

---

**Description**

Assess performance using the Transition Pairing Method when the spurious pairing threshold is varied

**Usage**

```
spurious_curve(trans, predictions, references, thresholds = 1:20)
```

**Arguments**

trans	a transition object
predictions	vector of predictions indicating transition (1) or non-transition (2)
references	vector of criteria indicating transition (1) or non-transition (2)
thresholds	the threshold settings to test

**Value**

an object with class spurious\_curve

**Examples**

```
set.seed(100)
predictions <- (sample(1:100)%%2)
references <- (sample(1:100)%%2)

trans <- get_transition_info(
  predictions, references, 7
)
head(spurious_curve(trans))
```

---

summaryTransition-class

*An S4 class containing summary information about a transition object*

---

**Description**

An S4 class containing summary information about a transition object

**Slots**

result a data frame with the summary information

---

test_errors	<i>Compare numeric variables in a data frame based on root-squared differences</i>
-------------	--

---

### Description

Compare numeric variables in a data frame based on root-squared differences

### Usage

```
test_errors(  
  reference,  
  target,  
  vars,  
  tolerance = 0.001005,  
  return_logical = TRUE  
)
```

### Arguments

reference	a data frame giving reference data
target	a data frame giving target data
vars	character vector of variable names to compare in each data frame
tolerance	allowable difference between numeric values
return_logical	logical. Should result be given as a logical vector (indicating TRUE/FALSE equality within tolerance) or a data frame of error summary values?

### Value

If `return_logical = TRUE`, a named logical vector with one element per variable compared, indicating whether the maximum and root-mean-squared differences fall within the tolerance. If `return_logical = FALSE`, a data frame indicating the variables compared and the maximum and root-mean-squared differences.

### Note

It is assumed that reference and target have equal numbers of rows.

### Examples

```
reference <- data.frame(  
  a = 1:100, b = 75:174  
)
```

```
target <- data.frame(  
  a = 0.001 + (1:100),  
  b = 76:175
```

```
)
test_errors(reference, target, c("a", "b"))
test_errors(reference, target, c("a", "b"), return_logical = FALSE)
```

---

weight_status	<i>Determine weight status from body mass index</i>
---------------	---

---

## Description

Allows users to determine weight status from body mass index (BMI). The function is designed to classify adult weight status, with default settings yielding weight classes defined by the Centers for Disease Control and Prevention (see reference below). Alternatively, the function can be used as a wrapper for [get\\_BMI\\_percentile](#) to obtain classifications for youth.

## Usage

```
weight_status(BMI = NULL, breaks = c(-Inf, 18.5, 25, 30, 35, 40, Inf),
  labels = c("Underweight", "Healthy Weight", "Overweight", "Class 1 Obese",
    "Class 2 Obese", "Class 3 Obese"), right = FALSE, youth = FALSE, ...)

#get_BMI_percentile(weight_kg, height_cm, age_yrs = NULL, age_mos = NULL,
  #sex = c("Male", "Female"), BMI = NULL, df = NULL,
  #output = c("percentile", "classification", "both", "summary"))
```

## Arguments

BMI	numeric. The participant body mass index
breaks	numeric vector. The boundaries for each weight class; passed to <code>base::cut</code> , with warnings if <code>-Inf</code> and <code>Inf</code> are not included in the vector.
labels	character vector. The labels for each weight class; passed to <code>base::cut</code> , and should have a length one less than the length of breaks
right	logical. See <code>?base::cut</code>
youth	logical. Use function as a wrapper for <a href="#">get_BMI_percentile?</a>
...	Arguments passed to <a href="#">get_BMI_percentile</a>

## Value

a factor reflecting weight status

## References

<https://www.cdc.gov/bmi/adult-calculator/bmi-categories.html>

## Examples

```
weight_status(17:42)
```

---

weir_equation	<i>Calculate energy expenditure using the Weir equation</i>
---------------	---

---

**Description**

Calculate energy expenditure using the Weir equation

**Usage**

```
weir_equation(VO2, VC02, epochSecs)
```

**Arguments**

VO2	Oxygen consumption
VC02	Carbon dioxide production
epochSecs	The averaging window of the metabolic data, in seconds

**Value**

numeric scalar indicating predicted energy expenditure from the Weir equation, based on the inputs

**References**

De V Weir, J. B. (1949). New methods for calculating metabolic rate with special reference to protein metabolism. *The Journal of physiology*, 109(1-2), 1.

**Examples**

```
weir_equation(3.5, 3.1, 60)
```

# Index

- \* **datasets**
  - ex\_data, [12](#)
- \* **summaryTransition**
  - as, [3](#)
- as, [3](#)
- ba\_analysis, [3](#), [4](#)
- ba\_plot, [4](#), [25](#)
- bout\_mvpa, [5](#), [25](#)
- cvd\_risk, [7](#)
- descriptives, [9](#), [25](#)
- df\_continuous, [10](#), [14](#)
- df\_reorder, [11](#)
- difftime, [14](#)
- epoch\_length\_sec, [10](#), [11](#), [13](#)
- ex\_data, [12](#)
- full\_days, [13](#)
- get\_age, [14](#), [25](#)
- get\_BMI\_percentile, [25](#), [34](#)
- get\_bmr, [15](#), [25](#)
- get\_duration (manage\_procedure), [21](#)
- get\_indices, [16](#), [31](#)
- get\_intensity, [17](#)
- get\_kcal\_vo2\_conversion, [15](#), [18](#), [18](#), [25](#)
- get\_ree, [19](#)
- get\_transition\_info, [25](#)
- index\_runs, [20](#)
- manage\_procedure, [21](#)
- mean\_sd, [22](#)
- paired\_equivalence\_test
  - (paired\_equivalence\_test.data.frame), [23](#)
- paired\_equivalence\_test.data.frame, [23](#)
- PAutilities, [25](#)
- PAutilities-package (PAutilities), [25](#)
- plot.paired\_equivalence, [26](#)
- plot.spurious\_curve, [27](#)
- plot.transition, [28](#)
- reintegrate, [28](#)
- residual\_adjust, [29](#)
- rmr\_sliding, [30](#)
- rolling\_groups, [17](#), [31](#)
- shaded\_equivalence\_plot
  - (plot.paired\_equivalence), [26](#)
- spurious\_curve, [27](#), [32](#)
- summaryTransition
  - (summaryTransition-class), [32](#)
- summaryTransition-class, [32](#)
- test\_errors, [33](#)
- unshaded\_equivalence\_plot
  - (plot.paired\_equivalence), [26](#)
- weight\_status, [34](#)
- weir\_equation, [25](#), [35](#)