# Package ‘PAutilities’

December 16, 2019

**Type** Package  
**Title** Streamline Physical Activity Research  
**Version** 0.3.1  
**Depends** R (>= 2.10)  

**Description** A collection of utilities that are useful for a broad range of tasks that are common in physical activity research, including the following: creation of Bland-Altman plots, formatted descriptive statistics, metabolic calculations (e.g. basal metabolic rate predictions) and conversions, demographic calculations (age and age-for-body-mass-index percentile), bout analysis of moderate-to-vigorous intensity physical activity, and analysis of bout detection algorithm performance.

**License** GPL-3  
**Encoding** UTF-8  
**LazyData** true  

**URL** [https://github.com/paulhibbing/PAutilities](https://github.com/paulhibbing/PAutilities)  

**BugReports** [https://github.com/paulhibbing/PAutilities/issues](https://github.com/paulhibbing/PAutilities/issues)  

**RoxygenNote** 6.1.1  

**Imports** clues (>= 0.6.1), dplyr (>= 0.7), equivalence, ggplot2 (>= 2.2), graphics, lazyeval (>= 0.2), lubridate (>= 1.7.4), magrittr (>= 1.5), methods, matchingMarkets (>= 1.0.1), reshape2, rlang (>= 0.3.1), stats, utils  

**Suggests** testthat, knitr, rmarkdown  

**VignetteBuilder** knitr  

**NeedsCompilation** no  

**Author** Paul R. Hibbing [aut, cre], Centers for Disease Control and Prevention [ctb]  

**Maintainer** Paul R. Hibbing <paulhibbing@gmail.com>  

**Repository** CRAN  

**Date/Publication** 2019-12-16 06:10:18 UTC
R topics documented:

as ................................................................. 2
ba_plot .......................................................... 2
bout_mvpa ......................................................... 4
descriptives ...................................................... 5
ex_data ............................................................ 6
get_age .............................................................. 7
get_BMI_percentile ............................................. 7
get_bmr ............................................................. 8
get_intensity ...................................................... 9
get_kcal_vo2_conversion ...................................... 10
index_runs ......................................................... 11
manage_procedure ............................................... 12
mean_sd .......................................................... 13
paired_equivalence_test.data.frame ......................... 14
PAutilities ........................................................ 15
plot.paired_equivalence ...................................... 16
plot.spurious_curve ............................................ 17
plot.transition ................................................... 18
rmr_sliding ....................................................... 18
spurious_curve .................................................. 19
summaryTransition-class .................................... 20
test_errors ....................................................... 20
weir_equation ................................................... 21

Index 23

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

Description

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

ba_plot Create a Bland-Altman plot

Description

Create a Bland-Altman plot

Usage

\[
\text{ba_plot(plotdata, x_var, y_var, x_name, y_name, shape = 16, ...)}
\]
**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**


**Examples**

```r
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
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

Note

output_var gives one or both of `is_MVPA` and `bout_tracker`, 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")
})

---

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

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")

---

ex_data Example data for calculating bouts of moderate-to-vigorous physical activity

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)

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

Axes activity counts for the device’s respective 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
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_BMI_percentile

Calculate youth BMI percentile from CDC standards

Description

Calculate youth BMI percentile from CDC standards

Usage

get_BMI_percentile(weight_kg, height_cm, age_yrs, age_mos = NULL, 
                   sex = c("M", "F"), output = c("percentile", "classification", "both"))

Arguments

weight_kg                  Weight in kilograms
height_cm                  height in centimeters
age_yrs                    age in years
age_mos                    age in months (optional)
sex                         Character scalar indicating participant’s sex
output                     What should be returned: raw percentile, weight status classification, or both?
get_bmr

Retrieve estimated basal metabolic rate for an individual

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?

Details

If age_mos is not provided, it will be calculated based on age_yrs, assuming 365.2425 days per year and 30.4375 days per month. Depending on how the initial age calculation was made, rounding error will occur. Thus, use of the get_age function is recommended. If age_mos is provided, age_yrs can be passed as NULL.

Value

One of: A numeric scalar giving the BMI percentile (for output = "percentile"); a factor scalar giving the weight status (for output = "classification"); or a list with the percentile and classification (for output = "both").

References

This function was developed with reference to public domain resources provided by the Centers for Disease Control and Prevention. For more information, see:
https://www.cdc.gov/obesity/childhood/defining.html

Examples

get_BMI_percentile(39.4, 144.5, 12.35, sex = "M")
### get_bmr

- **RER** numeric. The respiratory exchange ratio
- **equation** The equation to apply
- **kcal_table** The table to reference for converting kilocalories to oxygen consumption. See `get_kcal_vo2_conversion`
- **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

### References


### Examples

```r
# 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_intensity

**Classify activity intensity**

**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

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

Details

If breaks and labels arguments are not provided, default values are <= 1.5 METs for sedentary behavior, 1.51-2.99 METs for light physical activity, and >= 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


Examples

get_kcal_vo2_conversion(0.85, "both")

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
manage_procedure

Printing and timing utility for managing processes

Usage

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

generate_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

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

## Default starting message
```

Examples
mean_sd

manage_procedure("Start")

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

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

---

mean_sd

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

---

**Description**

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

**Usage**

```r
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`

**Examples**

```r
mean_sd(rnorm(1:100, 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

x numeric vector representing the (possibly surrogate) sample

y numeric vector representing the (possibly criterion) sample. Index paired with x

y_type classification of y for the purpose of analysis. Can be "criterion", "comparison", or "both".

alpha the alpha level for the test

na.rm logical. Omit mean values for mean calculations?

scale character specifying whether the test should occur on an absolute or relative scale

absolute_region_width the region width for use when scale = "absolute"

relative_region_width the region width for use when scale = "relative"

... 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

```r
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)
```

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`)
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

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

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

Plot the transitions and matchings from a transition object

Description

Plot the transitions and matchings from a transition object

Usage

```r
## 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

```r
predictions <- (sample(1:100)%%2)
references <- (sample(1:100)%%2)
window_size <- 7
transitions <- get_transition_info(predictions, references, window_size)
plot(transitions)
```

rmr_sliding

Calculate resting metabolic rate using a sliding window method

Description

Calculate resting metabolic rate using a sliding window method

Usage

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

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]
fake_breaths <- rnorm(length(fake_timestamps), 450, 0.5)
window_size <- 5

rmr_sliding(
    fake_breaths, fake_timestamps,
    fake_start_time, fake_stop_time,
    window_size
)

spurious_curve Perform a spurious curve analysis

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

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

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

test_errors

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

Description

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

Usage

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

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

```r
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)
```

weir_equation

Calculate energy expenditure using the Weir equation

Description

Calculate energy expenditure using the Weir equation

Usage

```r
weir_equation(VO2, VCO2, epochSecs)
```
weir_equation

Arguments

- VO2: Oxygen consumption
- VCO2: Carbon dioxide production
- epochSecs: The averaging window of the metabolic data, in seconds

Examples

weir_equation(3.5, 3.1, 60)
Index

*Topic **datasets**
  - ex_data, 6
  - as, 2
  - ba_plot, 2, 15
  - bout_mvpa, 4, 15
  - descriptives, 5, 15
  - difftime, 7
  - ex_data, 6
  - get_age, 7, 8, 15
  - get_BMI_percentile, 7, 15
  - get_bmr, 8, 15
  - get_duration (manage_procedure), 12
  - get_intensity, 9
  - get_kcal_vo2_conversion, 9, 10, 11, 15
  - get_transition_info, 15
  - index_runs, 11
  - manage_procedure, 12
  - mean_sd, 13

  - paired_equivalence_test
    (paired_equivalence_test.data.frame), 14
  - paired_equivalence_test.data.frame, 14
  - PAutilities, 15
  - PAutilities-package (PAutilities), 15
  - plot.paired_equivalence, 16
  - plot.spurious_curve, 17
  - plot.transition, 18
  - rmr_sliding, 18

  - shaded_equivalence_plot
    (plot.paired_equivalence), 16
  - spurious_curve, 17, 19

  - summaryTransition
    (summaryTransition-class), 20
  - summaryTransition-class, 20
  - test_errors, 20
  - unshaded_equivalence_plot
    (plot.paired_equivalence), 16
  - weir_equation, 15, 21