Package 'iglu'

June 14, 2024

Type Package

Title Interpreting Glucose Data from Continuous Glucose Monitors

Version 4.1.6

Description Implements a wide range of metrics for measuring glucose control and glucose variability based on continuous glucose monitoring data. The list of implemented metrics is summarized in Rodbard (2009) <doi:10.1089/dia.2009.0015>. Additional visualization tools include time-series plots, lasagna plots and ambulatory glucose profile report.

License GPL-2

Encoding UTF-8

LazyData true

RoxygenNote 7.3.1

Depends R (>= 3.5.0)

Imports caTools, dplyr, DT, ggplot2, ggpubr, gridExtra, hms, lubridate, magrittr, patchwork, pheatmap, scales, shiny, stats, tibble, tidyr, utils, zoo, gtable, grid, plotly

Suggests knitr, rmarkdown, testthat (>= 3.0.0)

VignetteBuilder knitr

URL https://irinagain.github.io/iglu/

BugReports https://github.com/irinagain/iglu/issues

Config/testthat/edition 3

NeedsCompilation no

Author Elizabeth Chun [aut],
Steve Broll [aut],
David Buchanan [aut],
John Muschelli [aut] (<https://orcid.org/0000-0001-6469-1750>),
Nathaniel Fernandes [aut] (<https://orcid.org/0000-0003-0485-0726>),
Jung Hoon Seo [ctb],
Johnathan Shih [ctb],
Jacek Urbanek [ctb],
John Schwenck [ctb],

Marielle Hicban [ctb],
Mary Martin [ctb],
Pratik Patel [ctb],
Meyappan Ashok [ctb],
Nhan Nguyen [ctb],
Irina Gaynanova [aut, cre] (<https://orcid.org/0000-0002-4116-0268>)

Maintainer Irina Gaynanova <irinagn@umich.edu>

Repository CRAN

Date/Publication 2024-06-14 14:20:11 UTC

Contents

`above_percent` *Calculate percentage of values above target thresholds*

Description

The function `above_percent` produces a tibble object with values equal to the percentage of glucose measurements above target values. The output columns correspond to the subject id followed by the target values, and the output rows correspond to the subjects. The values will be between 0 (no measurements) and 100 (all measurements).

Usage

```
above_percent(data, targets_above = c(140, 180, 250))
```

Arguments

`data`	DataFrame object with column names "id", "time", and "gl", or numeric vector of glucose values.
`targets_above`	**Default: (140, 180, 250).** Numeric vector of glucose thresholds. Glucose values from data argument will be compared to each value in the targets_above vector.

Details

A tibble object with 1 row for each subject, a column for subject id and column for each target value is returned. NA's will be omitted from the glucose values in calculation of percent.

Value

If a DataFrame object is passed, then a tibble object with a column for subject id and then a column for each target value is returned. If a vector of glucose values is passed, then a tibble object without the subject id is returned. Wrap 'as.numeric()' around the latter to output a numeric vector.

References

Rodbard (2009) Interpretation of continuous glucose monitoring data: glycemic variability and quality of glycemic control, *Diabetes Technology and Therapeutics* **11** .55-67, doi:10.1089/dia.2008.0132.

See Also

plot_ranges()

Examples

```
data(example_data_1_subject)

above_percent(example_data_1_subject)
above_percent(example_data_1_subject, targets_above = c(100, 150, 180))

data(example_data_5_subject)

above_percent(example_data_5_subject)
above_percent(example_data_5_subject, targets_above = c(70, 170))
```

`active_percent` *Calculate percentage of time CGM was active*

Description

The function 'active_percent' produces the

Usage

```
active_percent(data, dt0 = NULL)
```

Arguments

data	DataFrame object with column names "id", "time", and "gl".
dt0	The time frequency for interpolated aligned grid in minutes, the default will match the CGM meter's frequency (e.g. 5 min for Dexcom).

Details

The function 'active_percent' produces a tibble object with values equal to the percentage of time the CGM was active, the total number of observed days, the start date, and the end date. For example, if a CGM's (5 min frequency) times were 0, 5, 10, 15 and glucose values were missing at time 5, then percentage of time the CGM was active is 75 The output columns correspond to the subject id, the percentage of time for which the CGM was active, the number of days of measurements, the start date and the end date of measurements. The output rows correspond to the subjects. The values of 'active_percent' are always between 0

Value

A tibble object with five columns: subject id, corresponding active_percent value, duration of measurement period in days, start date, and end date.

Author(s)

Pratik Patel, Irina Gaynanova

References

Danne et al. (2017) International Consensus on Use of Continuous Glucose Monitoring *Diabetes Care* **40** .1631-1640, doi:10.2337/dc171600.

Examples

```
data(example_data_1_subject)

active_percent(example_data_1_subject)

data(example_data_5_subject)

active_percent(example_data_5_subject)
active_percent(example_data_5_subject, dt0 = 5)
```

adrr *Calculate average daily risk range (ADRR)*

Description

The function ‘adrr‘ produces ADRR values in a tibble object.

Usage

```
adrr(data)
```

Arguments

data	DataFrame object with column names "id", "time", and "gl".

Details

A tibble object with 1 row for each subject, a column for subject id and a column for ADRR values is returned. ‘NA‘ glucose values are omitted from the calculation of the ADRR values.

ADRR is the average sum of HBGI corresponding to the highest glucose value and LBGI corresponding to the lowest glucose value for each day, with the average taken over the daily sums. If there are no high glucose or no low glucose values, then 0 will be substituted for the HBGI value or the LBGI value, respectively, for that day.

Value

A tibble object with two columns: subject id and corresponding ADRR value.

References

Kovatchev et al. (2006) Evaluation of a New Measure of Blood Glucose Variability in, Diabetes *Diabetes care* **29** .2433-2438, doi:10.2337/dc061085.

Examples

```
data(example_data_1_subject)
adrr(example_data_1_subject)

data(example_data_5_subject)
adrr(example_data_5_subject)
```

agp *Display Ambulatory Glucose Profile (AGP) statistics for selected subject*

Description

Display Ambulatory Glucose Profile (AGP) statistics for selected subject

Usage

```
agp(data, maxd = 14, inter_gap = 45, dt0 = NULL, tz = "", daily = TRUE)
```

Arguments

data	DataFrame object with column names "id", "time", and "gl". Should only be data for 1 subject. In case multiple subject ids are detected, a warning is produced and only 1st subject is used.
maxd	**Default: 14.** Number of days to plot. If less than 'maxd' days of data are available, all days are plotted.
inter_gap	The maximum allowable gap (in minutes) for interpolation. The values will not be interpolated between the glucose measurements that are more than inter_gap minutes apart. The default value is 45 min.
dt0	The time frequency for interpolation in minutes, the default will match the CGM meter's frequency (e.g. 5 min for Dexcom).
tz	A character string specifying the time zone to be used. System-specific (see as.POSIXct), but " " is the current time zone, and "GMT" is UTC (Universal Time, Coordinated). Invalid values are most commonly treated as UTC, on some platforms with a warning.
daily	**Default: TRUE.** Logical indicator whether AGP should include separate daily plots.

Value

A plot displaying glucose measurements range, selected glucose statistics (average glucose, Glucose Management Indicator,

References

Johnson et al. (2019) Utilizing the Ambulatory Glucose Profile to Standardize and Implement Continuous Glucose Monitoring in Clinical Practice, *Diabetes Technology and Therapeutics* **21:S2** S2-17-S2-25, doi:10.1089/dia.2019.0034.

Examples

```
data(example_data_1_subject)
agp(example_data_1_subject, daily = FALSE)
```

agp_metrics *Calculate metrics for the Ambulatory Glucose Profile (AGP)*

Description

The function ‘agp_metrics‘ runs the following functions and combines them into a tibble object: ‘active_percent‘, ‘mean_glu‘, ‘gmi‘, ‘cv_glu‘, ‘below_percent‘, ‘in_range_percent‘, ‘above_percent‘.

Usage

```
agp_metrics(data, shinyformat = FALSE)
```

Arguments

data	DataFrame object with column names "id", "time", and "gl".
shinyformat	**Default: FALSE.** Boolean indicating whether the output should be formatted for the single subject AGP page in shiny.

Details

The function uses recommended cutoffs of 54, 70, 180, and 250 mg/dL for calculation.

If ‘shinyformat == FALSE‘ (default), returns a tibble object with 1 row for each subject, and 12 columns: a column for subject id (‘id‘), a column for start date (‘start_date‘), a column for end date (‘end_date‘), a column for number of days (‘ndays‘), a column for active percent (‘active_percent‘), a column for mean value (‘mean‘), a column for GMI value (‘GMI‘), a column for CV value (‘CV‘), a column for a column for a column for a column for a column for

If ‘shinyformat == TRUE‘, a tibble with 2 columns: metric and value, is returned. This output is used when generating the single subject AGP shiny page.

Value

By default, a tibble object with 1 row for each subject, and 13 columns is returned: a column for subject id, a column for start date, a column for end date, a column for number of days, a column for active_percent, a column for Mean value, a column for gmi value, a column for cv value, a column for below_54 value, a column for below_70 value, a column for in_range_70_180 value, a column for above_180 value, a column for above_250 value,

References

Johnson et al. (2019) Utilizing the Ambulatory Glucose Profile to Standardize and Implement Continuous Glucose Monitoring in Clinical Practice, *Diabetes Technology and Therapeutics* **21:S2** S2-17-S2-25, doi:10.1089/dia.2019.0034.

Examples

```
data(example_data_1_subject)
agp_metrics(example_data_1_subject)
```

`all_metrics` *Calculate all metrics in iglu*

Description

The function all_metrics runs all of the iglu metrics, and returns the results with one column per metric.

Usage

```
all_metrics(
  data,
  dt0 = NULL,
  inter_gap = 45,
  tz = "",
  timelag = 15,
  lag = 1,
  metrics_to_include = c("all", "consensus_only")
)
```

Arguments

`data`	DataFrame object with column names "id", "time", and "gl".
`dt0`	The time frequency for interpolation in minutes, the default will match the CGM meter's frequency (e.g. 5 min for Dexcom).
`inter_gap`	The maximum allowable gap (in minutes) for interpolation. The values will not be interpolated between the glucose measurements that are more than inter_gap minutes apart. The default value is 45 min.
`tz`	A character string specifying the time zone to be used. System-specific (see `as.POSIXct`), but " " is the current time zone, and "GMT" is UTC (Universal Time, Coordinated). Invalid values are most commonly treated as UTC, on some platforms with a warning.
`timelag`	Integer indicating the time period (# minutes) over which rate of change is calculated. Default is 15, e.g. rate of change is the change in glucose over the past 15 minutes divided by 15.
`lag`	Integer indicating which lag (# days) to use. Default is 1.
`metrics_to_include`	Returns all metrics computed by iglu or all on the consensus list (Battelino 2023)

Details

All iglu functions are calculated within the all_metrics function, and the resulting tibble is returned with one row per subject and a column for each metric. Time dependent functions are calculated together using the function optimized_iglu_functions with two exceptions: PGS and episodes are calculated within all_metrics because their structure does not align with optimized_iglu_functions.

Note that episodes related outputs included in all_metrics are only average episodes per day. To get the average duration and glucose, please use the standalone episodes function

For metric specific information, please see the corresponding function documentation.

Value

A tibble object with 1 row per subject and one column per metric is returned.

References

Battelino T, Alexander CM, Amiel SA, et al. Continuous glucose monitoring and metrics for clinical trials: an international consensus statement. *Lancet Diabetes Endocrinol.* **2023;11(1):**42-57. doi:10.1016/S22138587(22)003199.

Specify the meter frequency and change the interpolation gap to 30 min all_metrics(example_data_1_subject, dt0 = 5, inter_gap = 30)

Examples

```
data(example_data_1_subject)
all_metrics(example_data_1_subject)
```

---

auc *Calculate Area Under Curve AUC*

Description

The function auc produces hourly average AUC for each subject.

Usage

```
auc(data, tz="")
```

Arguments

data	DataFrame object with column names "id", "time", and "gl", or numeric vector of glucose values.
tz	String value of time zone.

Details

A tibble object with 1 row for each subject, a column for subject id and a column for hourly average AUC values is returned. NA glucose values are omitted from the calculation of the AUC.

AUC is calculated using the formula: (dt0/60) * ((gl[2:length(gl)] + gl[1:(length(gl)-1)])/2), where dt0/60 is the frequency of the cgm measurements in hours and gl are the glucose values.

This formula is based off the Trapezoidal Rule: (time[2]-time[1] * ((glucose[1]+glucose[2])/2)).

Value

If a data.frame object is passed, then a tibble object with two columns: subject id and corresponding hourly average AUC value is returned.

AUC is calculated for every hour using the trapezoidal rule, then hourly average AUC is calculated for each 24 hour period, then the mean of hourly average AUC across all 24 hour periods is returned as overall hourly average AUC.

References

Danne et al. (2017) International Consensus on Use of Continuous Glucose Monitoring, *Diabetes Care* **40** .1631-1640, doi:10.2337/dc171600.

Examples

```
data(example_data_1_subject)
auc(example_data_1_subject)
```

`below_percent` *Calculate percentage below targeted values*

Description

The function below_percent produces a tibble object with values equal to the percentage of glucose measurements below target values. The output columns correspond to the subject id followed by the target values and the output rows correspond to the subjects. The values will be between 0 (no measurements) and 100 (all measurements).

Usage

```
below_percent(data, targets_below = c(54, 70))
```

Arguments

`data`	DataFrame with column names ("id", "time", and "gl"), or numeric vector of glucose values.
`targets_below`	Numeric vector of glucose thresholds. Glucose values from data argument will be compared to each value in the targets_below vector. Default list is (54, 70).

Details

A tibble object with 1 row for each subject, a column for subject id and column for each target value is returned. NA's will be omitted from the glucose values in calculation of percent.

Value

If a data.frame object is passed, then a tibble object with a column for subject id and then a column for each target value is returned. If a vector of glucose values is passed, then a tibble object without the subject id is returned. as.numeric() can be wrapped around the latter to output a numeric vector.

References

Rodbard (2009) Interpretation of continuous glucose monitoring data: glycemic variability and quality of glycemic control, *Diabetes Technology and Therapeutics* **11** .55-67, doi:10.1089/dia.2008.0132.

See Also

plot_ranges()

Examples

```
data(example_data_1_subject)

below_percent(example_data_1_subject)
below_percent(example_data_1_subject, targets_below = c(50, 100, 180))

data(example_data_5_subject)

below_percent(example_data_5_subject)
below_percent(example_data_5_subject, targets_below = c(80, 180))
```

---

calculate_sleep_wake *Calculate metrics for values inside and/or outside a specified time range.*

Description

This function applies a given function to a subset of data filtered by time of day.

Usage

```
calculate_sleep_wake(
  data,
  FUN,
  sleep_start = 0,
  sleep_end = 6,
  calculate = c("sleep", "wake", "both"),
  ...
)
```

Arguments

data	DataFrame object with column names "id", "time", and "gl".
FUN	Function to be applied to the filtered data.
sleep_start	Numeric between 0-24 signifying the hour at which the time range should start.
sleep_end	Numeric between 0-24 signifying the hour at which the time range should end.
calculate	String determining whether FUN should be applied to values inside or outside the time range. Both separately is an option
...	Optional arguments which will be passed to FUN

Details

An object of the same return type as FUN, with the same column names as FUN will be returned. If calculate = "both", there will be columns for FUN applied to both inside and outside values, with either "in range" or "out of range" append to signify whether the statistic was calculated on values which were inside the time range or outside the range.

FUN is found by a call to match.fun and typically is either a function or a character string specifying a function to be searched for from the environment of the call to apply. Arguments in ... cannot have the same name as any of the other arguments, and care may be needed to avoid partial matching to FUN. FUN is applied to the data after the data is filtered based on whether its hour falls within the given range. If sleep_start is an integer, all times within that hour will be included in the range, but if sleep_end is an integer only times up to that hour will be included in the range. If sleep_start is after sleep_end, the data will be filtered to include all hours after sleep_start and all times before sleep_end.

Value

An object of the same return type as FUN, with columns corresponding to the values returned by FUN. Separated for values inside or outside the time range, if calculate = both.

Examples

```
data(example_data_1_subject)
calculate_sleep_wake(example_data_1_subject, sd_glu, calculate = "sleep")

data(example_data_5_subject)
calculate_sleep_wake(example_data_5_subject, cogi, targets = c(80, 150),
weights = c(.3,.2,.5), calculate = "wake")
calculate_sleep_wake(example_data_5_subject, sd_measures, sleep_start = 2,
sleep_end = 8, calculate = "both")
```

`CGMS2DayByDay` *Interpolate glucose value on an equally spaced grid from day to day*

Description

Interpolate glucose value on an equally spaced grid from day to day

Usage

```
CGMS2DayByDay(data, dt0 = NULL, inter_gap = 45, tz = "")
```

Arguments

`data`	DataFrame object with column names "id", "time", and "gl". Should only be data for 1 subject. In case multiple subject ids are detected, a warning is produced and only 1st subject is used.
`dt0`	The time frequency for interpolation in minutes, the default will match the CGM meter's frequency (e.g. 5 min for Dexcom).
`inter_gap`	The maximum allowable gap (in minutes) for interpolation. The values will not be interpolated between the glucose measurements that are more than inter_gap minutes apart. The default value is 45 min.
`tz`	A character string specifying the time zone to be used. System-specific (see `as.POSIXct`), but " " is the current time zone, and "GMT" is UTC (Universal Time, Coordinated). Invalid values are most commonly treated as UTC, on some platforms with a warning.

Value

A list with

`gd2d`	A matrix of glucose values with each row corresponding to a new day, and each column corresponding to time
`actual_dates`	A vector of dates corresponding to the rows of `gd2d`
`dt0`	Time frequency of the resulting grid, in minutes

Examples

```
CGMS2DayByDay(example_data_1_subject)
```

cogi *Calculate Continuous Glucose Monitoring Index (COGI) values*

Description

The function COGI produces cogi values in a tibble object.

Usage

```
cogi(data, targets = c(70, 180), weights = c(.5,.35,.15))
```

Arguments

data	DataFrame with column names ("id", "time", and "gl"), or numeric vector of glucose values.
targets	Numeric vector of two glucose values for threshold. Glucose values from data argument will be compared to each value in the targets vector to determine the time in range and time below range for COGI. The lower value will be used for determining time below range. Default list is (70, 180).
weights	Numeric vector of three weights to be applied to time in range, time below range and glucose variability, respectively. The default list is (.5,.35,.15)

Details

A tibble object with 1 row for each subject, a column for subject id and column for each target value is returned. NA's will be omitted from the glucose values in calculation of cogi.

Value

If a data.frame object is passed, then a tibble object with a column for subject id and then a column for each target value is returned. If a vector of glucose values is passed, then a tibble object without the subject id is returned. as.numeric() can be wrapped around the latter to output a numeric vector.

References

Leelarathna (2020) Evaluating Glucose Control With a Novel Composite Continuous Glucose Monitoring Index, *Diabetes Technology and Therapeutics* **14(2)** 277-284, doi:10.1177/1932296819838525.

Examples

```
data(example_data_1_subject)

cogi(example_data_1_subject)
cogi(example_data_1_subject, targets = c(50, 140), weights = c(.3,.6,.1))

data(example_data_5_subject)

cogi(example_data_5_subject)
```

```
cogi(example_data_5_subject, targets = c(80, 180), weights = c(.2,.4,.4))
```

---

conga *Continuous Overall Net Glycemic Action (CONGA)*

Description

The function conga produces CONGA values a tibble object for any n hours apart.

Usage

```
conga(data, n = 24, tz = "")
```

Arguments

data	DataFrame object with column names "id", "time", and "gl".
n	An integer specifying how many hours prior to an observation should be used in the CONGA calculation. The default value is set to n = 24 hours
tz	A character string specifying the time zone to be used. System-specific (see as.POSIXct), but " " is the current time zone, and "GMT" is UTC (Universal Time, Coordinated). Invalid values are most commonly treated as UTC, on some platforms with a warning.

Details

A tibble object with 1 row for each subject, a column for subject id and a column for the CONGA values is returned.

Missing values will be linearly interpolated when close enough to non-missing values.

CONGA is the standard deviation of the difference between glucose values that are exactly n hours apart. CONGA is computed by taking the standard deviation of differences in measurements separated by n hours.

Value

A tibble object with two columns: subject id and corresponding CONGA value.

References

McDonnell et al. (2005) : A novel approach to continuous glucose analysis utilizing glycemic variation *Diabetes Technology and Therapeutics* **7** .253-263, doi:10.1089/dia.2005.7.253.

Examples

```
data(example_data_1_subject)
conga(example_data_1_subject)

data(example_data_5_subject)
conga(example_data_5_subject)
```

---

cv_glu *Calculate Coefficient of Variation (CV) of glucose levels*

Description

The function cv_glu produces CV values in a tibble object.

Usage

```
cv_glu(data)
```

Arguments

data	DataFrame object with column names "id", "time", and "gl", or numeric vector of glucose values.

Details

A tibble object with 1 row for each subject, a column for subject id and a column for CV values is returned. NA glucose values are omitted from the calculation of the CV.

CV (Coefficient of Variation) is calculated by $100 * sd(G)/mean(G)$ Where G is the list of all Glucose measurements for a subject.

Value

If a data.frame object is passed, then a tibble object with two columns: subject id and corresponding CV value is returned. If a vector of glucose values is passed, then a tibble object with just the CV value is returned. as.numeric() can be wrapped around the latter to output just a numeric value.

References

Rodbard (2009) Interpretation of continuous glucose monitoring data: glycemic variability and quality of glycemic control, *Diabetes Technology and Therapeutics* **11** .55-67, doi:10.1089/dia.2008.0132.

Examples

```
data(example_data_1_subject)
cv_glu(example_data_1_subject)

data(example_data_5_subject)
cv_glu(example_data_5_subject)
```

`cv_measures` *Calculate Coefficient of Variation subtypes*

Description

The function cv_measures produces CV subtype values in a tibble object.

Usage

```
cv_measures(data, dt0 = NULL, inter_gap = 45, tz = "" )
```

Arguments

`data`	DataFrame object with column names "id", "time", and "gl". Should only be data for 1 subject. In case multiple subject ids are detected, a warning is produced and only 1st subject is used.
`dt0`	The time frequency for interpolation in minutes, the default will match the CGM meter's frequency (e.g. 5 min for Dexcom).
`inter_gap`	The maximum allowable gap (in minutes) for interpolation. The values will not be interpolated between the glucose measurements that are more than inter_gap minutes apart. The default value is 45 min.
`tz`	A character string specifying the time zone to be used. System-specific (see `as.POSIXct`), but " " is the current time zone, and "GMT" is UTC (Universal Time, Coordinated). Invalid values are most commonly treated as UTC, on some platforms with a warning.

Details

A tibble object with 1 row for each subject, a column for subject id and a column for each cv subtype values is returned.

Missing values will be linearly interpolated when close enough to non-missing values.

1. CVmean:
   Calculated by first taking the coefficient of variation of each day's glucose measurements, then taking the mean of all the coefficient of variation. That is, for x days we compute cv_1 ... cv_x daily coefficient of variations and calculate $1/x * \sum[(cv_i)]$
2. CVsd:
   Calculated by first taking the coefficient of variation of each day's glucose measurements, then taking the standard deviation of all the coefficient of variations. That is, for d days we compute cv_1 ... cv_d daily coefficient of variations and calculate SD([cv_1, cv_2, ... cv_d])

Value

When a data.frame object is passed, then a tibble object with three columns: subject id and corresponding CV subtype values is returned.

References

Umpierrez, et.al. (2018) Glycemic Variability: How to Measure and Its Clinical Implication for Type 2 Diabetes *The American Journal of Medical Sciences* **356** .518-527, doi:10.1016/j.amjms.2018.09.010.

Examples

```
data(example_data_1_subject)
cv_measures(example_data_1_subject)

data(example_data_5_subject)
cv_measures(example_data_5_subject)
```

---

ea1c *Calculate eA1C*

Description

The function ea1c produces eA1C values in a tibble object.

Usage

```
ea1c(data)
```

Arguments

`data`	DataFrame object with column names "id", "time", and "gl", or numeric vector of glucose values.

Details

A tibble object with 1 row for each subject, a column for subject id and a column for eA1C values is returned. NA glucose values are omitted from the calculation of the eA1C.

eA1C score is calculated by $(46.7 + mean(G))/28.7$ where G is the vector of Glucose Measurements (mg/dL).

Value

If a data.frame object is passed, then a tibble object with two columns: subject id and corresponding eA1C is returned. If a vector of glucose values is passed, then a tibble object with just the eA1C value is returned. as.numeric() can be wrapped around the latter to output just a numeric value.

Author(s)

Marielle Hicban

References

Nathan (2008) Translating the A1C assay into estimated average glucose values *Hormone and Metabolic Research* **31** .1473-1478, doi:10.2337/dc080545.

Examples

```
data(example_data_1_subject)
ea1c(example_data_1_subject)

data(example_data_5_subject)
ea1c(example_data_5_subject)
```

`epicalc_profile` *Display Episode Calculation statistics for selected subject*

Description

Display Episode Calculation statistics for selected subject

Usage

```
epicalc_profile(
  data,
  lv1_hypo = 70,
  lv2_hypo = 54,
  lv1_hyper = 180,
  lv2_hyper = 250,
  dur_length = 15,
  end_length = 15,
  subject = NULL,
  dt0 = NULL,
  inter_gap = 45,
  tz = ""
)
```

Arguments

`data`	DataFrame object with column names "id", "time", and "gl". Should only be data for 1 subject. In case multiple subject ids are detected, a warning is produced and only 1st subject is used.
`lv1_hypo`	Numeric value specifying a hypoglycemia threshold for level 1
`lv2_hypo`	Numeric value specifying a hypoglycemia threshold for level 2

lv1_hyper	Numeric value specifying a hyperglycemia threshold for level 1
lv2_hyper	Numeric value specifying a hyperglycemia threshold for level 2
dur_length	Numeric value specifying the minimum duration in minutes to be considered an episode. Note dur_length should be a multiple of the data recording interval otherwise the function will round up to the nearest multiple. Default is 15 minutes to match consensus.
end_length	Numeric value specifying the minimum duration in minutes of improved glycemia for an episode to end. Default is equal to dur_length to match consensus.
subject	String corresponding to subject id
dt0	The time frequency for interpolation in minutes, the default will match the CGM meter's frequency (e.g. 5 min for Dexcom).
inter_gap	The maximum allowable gap (in minutes) for interpolation. The values will not be interpolated between the glucose measurements that are more than inter_gap minutes apart. The default value is 45 min.
tz	A character string specifying the time zone to be used. System-specific (see as.POSIXct), but " " is the current time zone, and "GMT" is UTC (Universal Time, Coordinated). Invalid values are most commonly treated as UTC, on some platforms with a warning.

Value

A plot displaying (1) the statistics for the episodes and (2) the episodes colored by level.

Author(s)

Johnathan Shih, Jung Hoon Seo, Elizabeth Chun

See Also

episode_calculation()

Examples

```
epicalc_profile(example_data_1_subject)
```

---

episode_calculation *Calculates Hypo/Hyperglycemic episodes with summary statistics*

Description

The function determines episodes or events, calculates summary statistics, and optionally returns data with episode label columns added

Usage

```
episode_calculation(
  data,
  lv1_hypo = 70,
  lv2_hypo = 54,
  lv1_hyper = 180,
  lv2_hyper = 250,
  dur_length = 15,
  end_length = 15,
  return_data = FALSE,
  dt0 = NULL,
  inter_gap = 45,
  tz = ""
)
```

Arguments

data	DataFrame object with column names "id", "time", and "gl". Should only be data for 1 subject. In case multiple subject ids are detected, a warning is produced and only 1st subject is used.
lv1_hypo	Numeric value specifying a hypoglycemia threshold for level 1
lv2_hypo	Numeric value specifying a hypoglycemia threshold for level 2
lv1_hyper	Numeric value specifying a hyperglycemia threshold for level 1
lv2_hyper	Numeric value specifying a hyperglycemia threshold for level 2
dur_length	Numeric value specifying the minimum duration in minutes to be considered an episode. Note dur_length should be a multiple of the data recording interval otherwise the function will round up to the nearest multiple. Default is 15 minutes to match consensus.
end_length	Numeric value specifying the minimum duration in minutes of improved glycemia for an episode to end. Default is equal to dur_length to match consensus.
return_data	Boolean indicating whether to also return data with episode labels. Defaults to FALSE which means only episode summary statistics will be returned
dt0	The time frequency for interpolation in minutes, the default will match the CGM meter's frequency (e.g. 5 min for Dexcom).
inter_gap	The maximum allowable gap (in minutes) for interpolation. The values will not be interpolated between the glucose measurements that are more than inter_gap minutes apart. The default value is 45 min.
tz	A character string specifying the time zone to be used. System-specific (see as.POSIXct), but " " is the current time zone, and "GMT" is UTC (Universal Time, Coordinated). Invalid values are most commonly treated as UTC, on some platforms with a warning.

Details

We follow the definition of episodes given in the 2023 consensus by Battelino et al. Note we have classified lv2 as a subset of lv1 since we find the consensus to be slightly ambiguous. For lv1

exclusive of lv2, please see lv1_excl which summarises episodes that were exclusively lv1 and did not cross the lv2 threshold. Also note, hypo extended refers to episodes that are >120 consecutive minutes below lv1 hypo and ends with at least 15 minutes of normoglycemia. For more details on each category please see the reference below (Battelino et al 2023).

Value

If return_data is FALSE, a single dataframe with columns:

`id`	Subject id
`type`	Type of episode - either hypoglycemia or hyperglycemia
`level`	Level of episode - one of lv1, lv2, extended, lv1_excl
`avg_ep_per_day`	Average number of episodes per day calculated as (total # episodes)/(recording time in days (24hrs))
`avg_ep_duration`	Average duration of episodes in minutes
`avg_ep_gl`	Average glucose in the episode in mg/dL
`total_episodes`	Total number of episodes in the subject's glucose trace

If return_data is TRUE, returns a list where the first entry is the episode summary dataframe (see above) and the second entry is the input data with episode labels added. Note the data returned here has been interpolated using the CGMS2DayByDay() function. Mostly for use with epicalc_profile function. Format of the second list entry is:

`id`	Subject id
`time`	Interpolated timestamps
`gl`	glucose in mg/dL
`[episode_label]`	One column per episode label - i.e. lv1_hypo, lv2_hypo, lv1_hyper, lv2_hyper, ext_hypo. 0 means not this type of episode, a positive integer label is assigned to each episode. Note the labels are *not* unique by subject only unique by segment

Author(s)

Elizabeth Chun, Jung Hoon Seo, Johnathan Shih

References

Battelino et al. (2023): Continuous glucose monitoring and metrics for clinical trials: an international consensus statement *Lancet Diabetes & Endocrinology* **11(1)** .42-57, doi:10.1016/s2213-8587(22)003199.

See Also

epicalc_profile()

Examples

```
episode_calculation(example_data_5_subject, lv1_hypo=100, lv1_hyper= 120)
```

`example_data_1_subject` *Example CGM data for one subject with Type II diabetes*

Description

Dexcom G4 CGM measurements from 1 subject with Type II diabetes, this is a subset of example_data_5_subject.

Usage

```
example_data_1_subject
```

Format

A data.frame with 2915 rows and 3 columns, which are:

id identifier of subject

time 5-10 minute time value

gl glucose level

`example_data_5_subject` *Example CGM data for 5 subjects with Type II diabetes*

Description

Dexcom G4 CGM measurements for 5 subjects with Type II diabetes. These data are part of a larger study sample that consisted of patients with Type 2 diabetes recruited from the general community. To be eligible, patients with Type 2 diabetes, not using insulin therapy and with a glycosylated hemoglobin (HbA_{1c}) value at least 6.5

Usage

```
example_data_5_subject
```

Format

A data.frame with 13866 rows and 3 columns, which are:

id identifier of subject

time date and time stamp

gl glucose level as measured by CGM (mg/dL)

example_data_hall *Example data from Hall et al. (2018)*

Description

Dexcom G4 CGM measurements for 19 subjects from the Hall publicly available dataset. Chosen as a subset of all subjects to be only those with diabetes or pre-diabetes. Primarily intended for use with example_meals_hall

Usage

```
example_data_hall
```

Format

a data.frame with 34890 rows and 4 columns, which are:

id identifier of subject

time date and time stamp

gl glucose level as measured by CGM (mg/dL)

diagnosis character indicating diabetes diagnosis: diabetic or pre-diabetic

Details

This dataset can be used along with the example_meals_hall dataset in this package to calculate meal_metrics.

References

Hall et al. (2018) : Glucotypes reveal new patterns of glucose dysregulation *Plos Biology* **16** (7): 3:e2005143 doi:10.1371/journal.pbio.2005143.

example_meals_hall *Example mealtimes data from Hall et al. (2018)*

Description

Example of mealtimes data format for meal_metrics function, corresponds to example_data_hall data.

Usage

```
example_meals_hall
```

Format

A data.frame with 9 rows and 3 columns, which are:

id identifier of subject

meal meal type identifier

mealtime time of meal

Details

There are 3 types of meals available: Cereal Flakes (CF), Peanut Butter Sandwich (PB), and Protein Bar (Bar). The number after the abbreviation refers to the replication number for the original study. For more details on nutritional differences, please see the original study reference.

This dataset should be used along with example_data_hall to calculate meal_metrics.

References

Hall et al. (2018) : Glucotypes reveal new patterns of glucose dysregulation *Plos Biology* **16** (7): 3:e2005143 doi:10.1371/journal.pbio.2005143.

`gmi` *Calculate GMI*

Description

The function gmi produces GMI values in a tibble object.

Usage

```
gmi(data)
```

Arguments

`data`	DataFrame object with column names "id", "time", and "gl", or numeric vector of glucose values.

Details

A tibble object with 1 row for each subject, a column for subject id and a column for GMI values is returned. NA glucose values are omitted from the calculation of the GMI.

GMI score is calculated by $3.31 + (.02392 * mean(G))$ where G is the vector of Glucose Measurements (mg/dL).

Value

If a data.frame object is passed, then a tibble object with two columns: subject id and corresponding GMI is returned. If a vector of glucose values is passed, then a tibble object with just the GMI value is returned. as.numeric() can be wrapped around the latter to output just a numeric value.

References

Bergenstal (2018) Glucose Management Indicator (GMI): A New Term for Estimating A1C From Continuous Glucose Monitoring *Hormone and Metabolic Research* **41** .2275-2280, doi:10.2337/dc181581.

Examples

```
data(example_data_1_subject)
gmi(example_data_1_subject)

data(example_data_5_subject)
gmi(example_data_5_subject)
```

`grade` *Calculate mean GRADE score*

Description

The function grade produces GRADE score values in a tibble object.

Usage

```
grade(data)
```

Arguments

`data`	DataFrame object with column names "id", "time", and "gl", or numeric vector of glucose values.

Details

A tibble object with 1 row for each subject, a column for subject id and a column for GRADE values is returned. NA glucose values are omitted from the calculation of the GRADE.

GRADE score is calculated by $1/n * \sum[425 * (log(log(G_i/18)) + .16)^2]$ Where G_i is the ith Glucose measurement and n is the total number of measurements.

Value

If a data.frame object is passed, then a tibble object with two columns: subject id and corresponding GRADE value is returned. If a vector of glucose values is passed, then a tibble object with just the GRADE value is returned. as.numeric() can be wrapped around the latter to output just a numeric value.

References

Hill et al. (2007): A method for assessing quality of control from glucose profiles *Diabetic Medicine* **24** .753-758, doi:10.1111/j.14645491.2007.02119.x.

Examples

```
data(example_data_1_subject)
grade(example_data_1_subject)

data(example_data_5_subject)
grade(example_data_5_subject)
```

`grade_eugly` *Percentage of GRADE score attributable to target range*

Description

The function grade_eugly produces %GRADE euglycemia values in a tibble object.

Usage

```
grade_eugly(data, lower = 70, upper = 140)
```

Arguments

`data`	DataFrame object with column names "id", "time", and "gl", or numeric vector of glucose values.
`lower`	Lower bound used for hypoglycemia cutoff, in mg/dL. Default is 70
`upper`	Upper bound used for hyperglycemia cutoff, in mg/dL. Default is 140.

Details

A tibble object with 1 row for each subject, a column for subject id and a column for %GRADE euglycemia values is returned. NA glucose values are omitted from the calculation of the %GRADE euglycemia values.

%GRADE euglycemia is determined by calculating the percentage of GRADE score (see grade function) attributed to values in the target range, i.e. values not below hypoglycemic or above hyperglycemic cutoffs.

Value

If a data.frame object is passed, then a tibble object with two columns: subject id and corresponding %GRADE euglycemia value is returned. If a vector of glucose values is passed, then a tibble object with just the %GRADE euglycemia value is returned. as.numeric() can be wrapped around the latter to output just a numeric value.

References

Hill et al. (2007): A method for assessing quality of control from glucose profiles *Diabetic Medicine* **24** .753-758, doi:10.1111/j.14645491.2007.02119.x.

Examples

```
data(example_data_1_subject)
grade_eugly(example_data_1_subject)
grade_eugly(example_data_1_subject, lower = 80, upper = 180)

data(example_data_5_subject)
grade_eugly(example_data_5_subject)
grade_eugly(example_data_5_subject, lower = 80, upper = 160)
```

---

grade_hyper *Percentage of GRADE score attributable to hyperglycemia*

Description

The function grade_hyper produces %GRADE hyperglycemia values in a tibble object.

Usage

```
grade_hyper(data, upper = 140)
```

Arguments

data	DataFrame object with column names "id", "time", and "gl", or numeric vector of glucose values.
upper	Upper bound used for hyperglycemia cutoff, in mg/dL. Default is 140.

Details

A tibble object with 1 row for each subject, a column for subject id and a column for %GRADE hyperglycemia values is returned. NA glucose values are omitted from the calculation of the %GRADE hyperglycemia values.

%GRADE hyperglycemia is determined by calculating the percentage of GRADE score (see grade function) attributed to hyperglycemic glucose values.

Value

If a data.frame object is passed, then a tibble object with two columns: subject id and corresponding %GRADE hyperglycemia value is returned. If a vector of glucose values is passed, then a tibble object with just the %GRADE hyperglycemia value is returned. as.numeric() can be wrapped around the latter to output just a numeric value.

References

Hill et al. (2007): A method for assessing quality of control from glucose profiles *Diabetic Medicine* **24** .753-758, doi:10.1111/j.14645491.2007.02119.x.

Examples

```
data(example_data_1_subject)
grade_hyper(example_data_1_subject)
grade_hyper(example_data_1_subject, upper = 180)

data(example_data_5_subject)
grade_hyper(example_data_5_subject)
grade_hyper(example_data_5_subject, upper = 160)
```

---

grade_hypo *Percentage of GRADE score attributable to hypoglycemia*

Description

The function grade_hypo produces %GRADE hypoglycemia values in a tibble object.

Usage

```
grade_hypo(data, lower = 80)
```

Arguments

data	DataFrame object with column names "id", "time", and "gl", or numeric vector of glucose values.
lower	Lower bound used for hypoglycemia cutoff, in mg/dL. Default is 80

Details

A tibble object with 1 row for each subject, a column for subject id and a column for %GRADE hypoglycemia values is returned. NA glucose values are omitted from the calculation of the %GRADE hypoglycemia values.

%GRADE hypoglycemia is determined by calculating the percentage of GRADE score (see grade function) attributed to hypoglycemic glucose values.

Value

If a data.frame object is passed, then a tibble object with two columns: subject id and corresponding %GRADE hypoglycemia value is returned. If a vector of glucose values is passed, then a tibble object with just the %GRADE hypoglycemia value is returned. as.numeric() can be wrapped around the latter to output just a numeric value.

References

Hill et al. (2007): A method for assessing quality of control from glucose profiles *Diabetic Medicine* **24** .753-758, doi:10.1111/j.14645491.2007.02119.x.

Examples

```
data(example_data_1_subject)
grade_hypo(example_data_1_subject)
grade_hypo(example_data_1_subject, lower = 70)

data(example_data_5_subject)
grade_hypo(example_data_5_subject)
grade_hypo(example_data_5_subject, lower = 65)
```

`gri` *Calculate Glycemia Risk Index (GRI)*

Description

The function gri produces a tibble object with values equal to the glycemia risk index (GRI) metric. The output columns are subject id and GRI value. ' The output rows correspond to subjects.

Usage

```
gri(data)
```

Arguments

`data`	DataFrame object with column names "id", "time", and "gl", or numeric vector of glucose values.

Details

A tibble object with 1 row for each subject, a column for subject id and column for GRI is returned. The formula for GRI is as follows:

$(3.0VLow) + (2.4Low) + (1.6VHigh) + (0.8High)$

where VLow, Low, VHigh, and High correspond to the percent of glucose values in the ranges <54 mg/dL, 54-70 mg/dL, >250 mg/dL, and 180-250 mg/dL respectively. The maximum allowed value for GRI is 100%, any calculated values higher than 100 are capped.

Value

A tibble object with columns for subject id and GRI value. Rows correspond to individual subjects.

Author(s)

Elizabeth Chun

References

Klonoff et al. (2022) A Glycemia Risk Index (GRI) of Hypoglycemia and Hyperglycemia for Continuous Glucose Monitoring Validated by Clinician Ratings. *J Diabetes Sci Technol* doi:10.1177/19322968221085273.

Examples

```
data(example_data_1_subject)
gri(example_data_1_subject)

data(example_data_5_subject)
gri(example_data_5_subject)
```

---

gvp *Calculate Glucose Variability Percentage (GVP)*

Description

The function mad produces GVP values in a tibble object.

Usage

```
gvp(data)
```

Arguments

data	DataFrame object with column names "id", "time", and "gl"

Details

A tibble object with 1 row for each subject, a column for subject id and a column for GVP values is returned. NA glucose values are omitted from the calculation of the GVP.

GVP is calculated by dividing the total length of the line of the glucose trace by the length of a perfectly flat trace. The formula for this is $sqrt(diff^2 + dt0^2)/(n * dt0)$, where diff is the change in Glucose measurements from one reading to the next, dt0 is the time gap between measurements and n is the number of glucose readings

Value

A tibble object with two columns: subject id and corresponding GVP value.

Author(s)

David Buchanan, Mary Martin

References

Peyser et al. (2017) Glycemic Variability Percentage: A Novel Method for Assessing Glycemic Variability from Continuous Glucose Monitor Data. *Diabetes Technol Ther* **20**(1):6–16, doi:10.1089/dia.2017.0187.

Examples

```
data(example_data_1_subject)
gvp(example_data_1_subject)

data(example_data_5_subject)
gvp(example_data_5_subject)
```

`hbgi` *Calculate High Blood Glucose Index (HBGI)*

Description

The function hbgi produces HBGI values in a tibble object.

Usage

```
hbgi(data)
```

Arguments

`data`	DataFrame object with column names "id", "time", and "gl", or numeric vector of glucose values.

Details

A tibble object with 1 row for each subject, a column for subject id and a column for HBGI values is returned. NA glucose values are omitted from the calculation of the HBGI.

HBGI is calculated by $1/n * \sum(10 * fg_i^2)$, where $fg_i = max(0, 1.509 * (log(G_i)^{1.084} - 5.381)$, G_i is the ith Glucose measurement for a subject, and n is the total number of measurements for that subject.

Value

If a data.frame object is passed, then a tibble object with two columns: subject id and corresponding HBGI value is returned. If a vector of glucose values is passed, then a tibble object with just the HBGI value is returned. as.numeric() can be wrapped around the latter to output just a numeric value.

References

Kovatchev et al. (2006) Evaluation of a New Measure of Blood Glucose Variability in, Diabetes *Diabetes care* **29** .2433-2438, doi:10.2337/dc061085.

Examples

```
data(example_data_1_subject)
hbgi(example_data_1_subject)

data(example_data_5_subject)
hbgi(example_data_5_subject)
```

`hist_roc` *Plot histogram of Rate of Change values (ROC)*

Description

The function hist_roc produces a histogram plot of ROC values

Usage

```
hist_roc(data, subjects = NULL, timelag = 15, dt0 = NULL, inter_gap = 45, tz = "")
```

Arguments

`data`	DataFrame object with column names "id", "time", and "gl".
`subjects`	String or list of strings corresponding to subject names in 'id' column of data. Default is all subjects.
`timelag`	Integer indicating the time period (# minutes) over which rate of change is calculated. Default is 15, e.g. rate of change is the change in glucose over the past 15 minutes divided by 15.
`dt0`	The time frequency for interpolation in minutes, the default will match the CGM meter's frequency (e.g. 5 min for Dexcom).
`inter_gap`	The maximum allowable gap (in minutes) for interpolation. The values will not be interpolated between the glucose measurements that are more than inter_gap minutes apart. The default value is 45 min.
`tz`	A character string specifying the time zone to be used. System-specific (see `as.POSIXct`), but " " is the current time zone, and "GMT" is UTC (Universal Time, Coordinated). Invalid values are most commonly treated as UTC, on some platforms with a warning.

Details

For the default, a histogram is produced for each subject displaying the ROC values colored by ROC categories defined as follows. The breaks for the categories are: c(-Inf, -3, -2, -1, 1, 2, 3, Inf) where the glucose is in mg/dl and the ROC values are in mg/dl/min. A ROC of -5 mg/dl/min will thus be placed in the first category and colored accordingly.

Value

A histogram of ROC values per subject

Author(s)

Elizabeth Chun, David Buchanan

References

Clarke et al. (2009) Statistical Tools to Analyze Continuous Glucose Monitor Data, Diabetes *Diabetes Technology and Therapeutics* **11** S45-S54, doi:10.1089/dia.2008.0138.

See Also

plot_roc for reference paper on ROC categories.

Examples

```
data(example_data_1_subject)
hist_roc(example_data_1_subject)

data(example_data_5_subject)
hist_roc(example_data_5_subject)
hist_roc(example_data_5_subject, subjects = 'Subject 3')
```

---

hyper_index *Calculate Hyperglycemia Index*

Description

The function hyper_index produces Hyperglycemia Index values in a tibble object.

Usage

```
hyper_index(data, ULTR = 140, a = 1.1, c = 30)
```

Arguments

data	DataFrame object with column names "id", "time", and "gl", or numeric vector of glucose values.
ULTR	Upper Limit of Target Range, default value is 140 mg/dL.
a	Exponent, generally in the range from 1.0 to 2.0, default value is 1.1.
c	Scaling factor, to display Hyperglycemia Index, Hypoglycemia Index, and IGC on approximately the same numerical range as measurements of HBGI, LBGI and GRADE, default value is 30.

Details

A tibble object with 1 row for each subject, a column for subject id and a column for the Hyperglycemia Index values is returned. NA glucose values are omitted from the calculation of the Hyperglycemia Index values.

Hyperglycemia Index is calculated by $n/c * \sum[(hyperBG_j - ULTR)^a]$ Here n is the total number of Glucose measurements (excluding NA values), $hyperBG_j$ is the jth Glucose measurement above the ULTR cutoff, a is an exponent, and c is a scaling factor.

Value

If a data.frame object is passed, then a tibble object with two columns: subject id and corresponding Hyperglycemia Index value is returned. If a vector of glucose values is passed, then a tibble object with just the Hyperglycemia Index value is returned. as.numeric() can be wrapped around the latter to output just a numeric value.

References

Rodbard (2009) Interpretation of continuous glucose monitoring data: glycemic variability and quality of glycemic control, *Diabetes Technology and Therapeutics* **11** .55-67, doi:10.1089/dia.2008.0132.

Examples

```
data(example_data_1_subject)
hyper_index(example_data_1_subject)
hyper_index(example_data_1_subject, ULTR = 160)

data(example_data_5_subject)
hyper_index(example_data_5_subject)
hyper_index(example_data_5_subject, ULTR = 150)
```

---

hypo_index *Calculate Hypoglycemia Index*

Description

The function hypo_index produces Hypoglycemia index values in a tibble object.

Usage

```
hypo_index(data, LLTR = 80, b = 2, d = 30)
```

Arguments

data	DataFrame object with column names "id", "time", and "gl", or numeric vector of glucose values.
LLTR	Lower Limit of Target Range, default value is 80 mg/dL.
b	Exponent, generally in the range from 1.0 to 2.0, default value is 2.
d	Scaling factor,to display Hyperglycemia Index, Hypoglycemia Index, and IGC on approximately the same numerical range as measurements of HBGI, LBGI and GRADE, default value is 30.

Details

A tibble object with 1 row for each subject, a column for subject id and a column for the Hypoglycemia Index values is returned. NA glucose values are omitted from the calculation of the Hypoglycemia Index values.

Hypoglycemia Index is calculated by $n/d * \sum[(LLTR - hypoBG_j)^b]$ Here n is the total number of Glucose measurements (excluding NA values), and $hypoBG_j$ is the jth Glucose measurement below the LLTR cutoff, b is an exponent, and d is a scaling factor.

Value

If a data.frame object is passed, then a tibble object with two columns: subject id and corresponding Hypoglycemia Index value is returned. If a vector of glucose values is passed, then a tibble object with just the Hypoglycemia Index value is returned. as.numeric() can be wrapped around the latter to output just a numeric value.

References

Rodbard (2009) Interpretation of continuous glucose monitoring data: glycemic variability and quality of glycemic control, *Diabetes Technology and Therapeutics* **11** .55-67, doi:10.1089/dia.2008.0132.

Examples

```
data(example_data_1_subject)
hypo_index(example_data_1_subject, LLTR = 60)

data(example_data_5_subject)
hypo_index(example_data_5_subject)
hypo_index(example_data_5_subject, LLTR = 70)
```

igc *Calculate Index of Glycemic Control*

Description

The function igc produces IGC values in a tibble object.

Usage

```
igc(data, LLTR = 80, ULTR = 140, a = 1.1, b = 2, c = 30, d = 30)
```

Arguments

data	DataFrame object with column names "id", "time", and "gl", or numeric vector of glucose values.
LLTR	Lower Limit of Target Range, default value is 80 mg/dL.
ULTR	Upper Limit of Target Range, default value is 140 mg/dL.
a	Exponent, generally in the range from 1.0 to 2.0, default value is 1.1.
b	Exponent, generally in the range from 1.0 to 2.0, default value is 2.
c	Scaling factor, to display Hyperglycemia Index, Hypoglycemia Index, and IGC on approximately the same numerical range as measurements of HBGI, LBGI and GRADE, default value is 30.
d	Scaling factor,to display Hyperglycemia Index, Hypoglycemia Index, and IGC on approximately the same numerical range as measurements of HBGI, LBGI and GRADE, default value is 30.

Details

A tibble object with 1 row for each subject, a column for subject id and a column for the IGC values is returned.

IGC is calculated by taking the sum of the Hyperglycemia Index and the Hypoglycemia index. See hypo_index and hyper_index.

Value

A tibble object with two columns: subject id and corresponding IGC value.

References

Rodbard (2009) Interpretation of continuous glucose monitoring data: glycemic variability and quality of glycemic control, *Diabetes Technology and Therapeutics* **11** .55-67, doi:10.1089/dia.2008.0132.

Examples

```
data(example_data_1_subject)
igc(example_data_1_subject)
igc(example_data_1_subject, ULTR = 160)

data(example_data_5_subject)
igc(example_data_5_subject)
igc(example_data_5_subject, LLTR = 75, ULTR = 150)
```

---

iglu_shiny *Run IGLU Shiny App*

Description

Run IGLU Shiny App

Usage

```
iglu_shiny()
```

---

in_range_percent *Calculate percentage in targeted value ranges*

Description

The function in_range_percent produces a tibble object with values equal to the percentage of glucose measurements in ranges of target values. The output columns correspond to subject id followed by the target value ranges, and the rows correspond to the subjects. The values will be between 0 (no measurements) and 100 (all measurements).

Usage

```
in_range_percent(data, target_ranges = list(c(70, 180), c(63, 140)))
```

Arguments

data	DataFrame object with column names "id", "time", and "gl", or numeric vector of glucose values.
target_ranges	List of target value ranges wrapped in an r 'list' structure. Default list of ranges is ((70, 180), (63, 140)) mg/dL, where the range (70, 180) is recommended to assess glycemic control for subjects with type 1 or type 2 diabetes, and (63, 140) is recommended for assessment of glycemic control during pregnancy; see Battelino et al. (2019)

Details

A tibble object with 1 row for each subject, a column for subject id and column for each range of target values is returned. NA's will be omitted from the glucose values in calculation of percent.

in_range_percent will only work properly if the target_ranges argument is a list of paired values in the format list(c(a1,b1), c(a2,b2), ...). The paired values can be ordered (min, max) or (max, min). See the Examples section for proper usage.

Value

If a data.frame object is passed, then a tibble object with a column for subject id and then a column for each target value is returned. If a vector of glucose values is passed, then a tibble object without the subject id is returned. as.numeric() can be wrapped around the latter to output a numeric vector.

References

Rodbard (2009) Interpretation of continuous glucose monitoring data: glycemic variability and quality of glycemic control, *Diabetes Technology and Therapeutics* **11** .55-67, doi:10.1089/dia.2008.0132.

Battelino et al. (2019) Clinical targets for continuous glucose monitoring data interpretation: recommendations from the international consensus on time in range. *Diabetes Care* **42**(8):1593-603, doi:10.2337/dci190028

See Also

plot_ranges()

Examples

```
data(example_data_1_subject)

in_range_percent(example_data_1_subject)
in_range_percent(example_data_1_subject, target_ranges = list(c(50, 100), c(200,
300), c(80, 140)))

data(example_data_5_subject)

in_range_percent(example_data_5_subject)
in_range_percent(example_data_1_subject, target_ranges = list(c(60, 120), c(140,
250)))
```

---

iqr_glu *Calculate glucose level iqr*

Description

The function iqr_glu outputs the distance between the 25th percentile and the 25th percentile of the glucose values in a tibble object.

Usage

```
iqr_glu(data)
```

Arguments

data	DataFrame object with column names "id", "time", and "gl", or numeric vector of glucose values.

Details

A tibble object with 1 row for each subject, a column for subject id and a column for the IQR values is returned. NA glucose values are omitted from the calculation of the IQR.

Value

If a data.frame object is passed, then a tibble object with two columns: subject id and corresponding IQR value is returned. If a vector of glucose values is passed, then a tibble object with just the IQR value is returned. as.numeric() can be wrapped around the latter to output just a numeric value.

Examples

```
data(example_data_1_subject)
iqr_glu(example_data_1_subject)

data(example_data_5_subject)
iqr_glu(example_data_5_subject)
```

---

j_index *Calculate J-index*

Description

The function j_index produces J-Index values a tibble object.

Usage

```
j_index(data)
```

Arguments

data	DataFrame object with column names "id", "time", and "gl", or numeric vector of glucose values.

Details

A tibble object with 1 row for each subject, a column for subject id and a column for J-Index values is returned. NA glucose values are omitted from the calculation of the J-Index.

J-Index score is calculated by $.001 * [mean(G) + sd(G)]^2$ where G is the list of Glucose Measurements.

Value

If a data.frame object is passed, then a tibble object with two columns: subject id and corresponding J-Index value is returned. If a vector of glucose values is passed, then a tibble object with just the J-Index value is returned. as.numeric() can be wrapped around the latter to output just a numeric value.

References

Wojcicki (1995) "J"-index. A new proposition of the assessment of current glucose control in diabetic patients *Hormone and Metabolic Research* **27** .41-42, doi:10.1055/s2007979906.

Examples

```
data(example_data_1_subject)
j_index(example_data_1_subject)

data(example_data_5_subject)
j_index(example_data_5_subject)
```

`lbgi` *Calculate Low Blood Glucose Index (LBGI)*

Description

The function lbgi produces LBGI values in a tibble object.

Usage

```
lbgi(data)
```

Arguments

`data`	DataFrame object with column names "id", "time", and "gl", or numeric vector of glucose values.

Details

A tibble object with 1 row for each subject, a column for subject id and a column for LBGI values is returned. NA glucose values are omitted from the calculation of the LBGI.

LBGI is calculated by $1/n * \sum(10 * fbg_i^2)$, where $fbg_i = min(0, 1.509 * (log(G_i)^{1.084} - 5.381)$, G_i is the ith Glucose measurement for a subject, and n is the total number of measurements for that subject.

Value

If a data.frame object is passed, then a tibble object with two columns: subject id and corresponding LBGI value is returned. If a vector of glucose values is passed, then a tibble object with just the LBGI value is returned. as.numeric() can be wrapped around the latter to output just a numeric value.

References

Kovatchev et al. (2006) Evaluation of a New Measure of Blood Glucose Variability in, Diabetes *Diabetes care* **29** .2433-2438, doi:10.2337/dc061085.

Examples

```
data(example_data_1_subject)
lbgi(example_data_1_subject)

data(example_data_5_subject)
lbgi(example_data_5_subject)
```

---

mad_glu *Calculate Median Absolute Deviation (MAD)*

Description

The function mad produces MAD values in a tibble object.

Usage

```
mad_glu(data, constant = 1.4826)
```

Arguments

data	DataFrame object with column names "id", "time", and "gl", or numeric vector of glucose values.
constant	Numeric object which will be multipled by the MAD value. Defaults to 1.4826. Reasons for this default value can be seen in the details section of the documentation of r's base mad method

Details

A tibble object with 1 row for each subject, a column for subject id and a column for MAD values is returned. NA glucose values are omitted from the calculation of the MAD.

MAD is calculated by taking the median of the difference of the glucose readings from their median and multiplying it by a scaling factor $1.4826 * median(|gl - median(gl)|)$, where gl is the list of Glucose measurements.

Value

If a data.frame object is passed, then a tibble object with two columns: subject id and corresponding MAD value is returned. If a vector of glucose values is passed, then a tibble object with just the MAD value is returned. as.numeric() can be wrapped around the latter to output just a numeric value.

Author(s)

David Buchanan, Marielle Hicban

Examples

```
data(example_data_1_subject)
mad_glu(example_data_1_subject)

data(example_data_5_subject)
mad_glu(example_data_5_subject)
```

---

mag *Calculate the Mean Absolute Glucose (MAG)*

Description

The function mag calculates the mean absolute glucose or MAG.

Usage

```
mag(data, n = 60, dt0 = NULL, inter_gap = 45, tz = "")
```

Arguments

data	DataFrame object with column names "id", "time", and "gl".
n	Integer giving the desired interval in minutes over which to calculate the change in glucose. Default is 60 to have hourly (60 minutes) intervals.
dt0	The time frequency for interpolation in minutes, the default will match the CGM meter's frequency (e.g. 5 min for Dexcom).

inter_gap	The maximum allowable gap (in minutes) for interpolation. The values will not be interpolated between the glucose measurements that are more than inter_gap minutes apart. The default value is 45 min.
tz	A character string specifying the time zone to be used. System-specific (see as.POSIXct), but " " is the current time zone, and "GMT" is UTC (Universal Time, Coordinated). Invalid values are most commonly treated as UTC, on some platforms with a warning.

Details

A tibble object with a column for subject id and a column for MAG values is returned.

The glucose values are linearly interpolated over a time grid starting at the beginning of the first day of data and ending on the last day of data. Then, MAG is calculated as $\frac{|\Delta G|}{\Delta t}$ where $|\Delta G|$ is the sum of the absolute change in glucose calculated for each interval as specified by n, default n = 60 for hourly change in glucose. The sum is then divided by Δt which is the total time in hours.

Value

A tibble object with two columns: subject id and MAG value

Author(s)

Elizabeth Chun

References

Hermanides et al. (2010) Glucose Variability is Associated with Intensive Care Unit Mortality, *Critical Care Medicine* **38(3)** 838-842, doi:10.1097/CCM.0b013e3181cc4be9

Examples

```
data(example_data_1_subject)
mag(example_data_1_subject)

data(example_data_5_subject)
mag(example_data_5_subject)
```

---

mage *Calculate Mean Amplitude of Glycemic Excursions*

Description

The function calculates MAGE values and can optionally return a plot of the glucose trace.

Usage

```
mage(
  data,
  version = c("ma", "naive"),
  sd_multiplier = 1,
  short_ma = 5,
  long_ma = 32,
  return_type = c("num", "df"),
  direction = c("avg", "service", "max", "plus", "minus"),
  tz = "",
  inter_gap = 45,
  max_gap = 180,
  plot = FALSE,
  title = NA,
  xlab = NA,
  ylab = NA,
  show_ma = FALSE,
  show_excursions = TRUE
)
```

Arguments

data	DataFrame object with column names "id", "time", and "gl" OR numeric vector of glucose values.
version	Either 'ma' or 'naive'. **Default: 'ma'.** Chooses which version of the MAGE algorithm to use. 'ma' algorithm is more accurate, and is the default. Earlier versions of iglu package (<=2.0.0) used 'naive'.
sd_multiplier	A numeric value that can change the sd value used to determine size of glycemic excursions used in the calculation. This is the only parameter that can be specified for version = "naive", and it is ignored if version = "ma".
short_ma	**Default: 5.** Integer for period length of the short moving average. Must be positive and less than 'long_ma'. (Recommended <15)
long_ma	**Default: 32.** Integer for period length for the long moving average. Must be positive and greater than 'short_ma'. (Recommended >20)
return_type	**Default: "num".** One of ("num", "df"). Will return either a single number for the "MAGE over the entire trace" (weighted by segment length) or a DataFrame with the MAGE value for each segment (see the MAGE vignette for discussion of handling large gaps by splitting trace into multiple segments).
direction	**Default: "avg".** One of ("avg", "service", "max", "plus", or "minus"). Algorithm will calculate one of the following: MAGE+ (nadir to peak), MAGE- (peak to nadir), MAGEavg = avg(MAGE+, MAGE-), MAGEmax = max(MAGE+, MAGE-), or automatically choose MAGE+/MAGE- based on the first countable excursion (i.e., "service"). NOTE: the selection of peak-to-nadir or nadir-to-peak is chosen independently on each segment, thus MAGEservice may choose peak-to-nadir on one segment and nadir-to-peak on another, for example.

tz — A character string specifying the time zone to be used. System-specific (see as.POSIXct), but " " is the current time zone, and "GMT" is UTC (Universal Time, Coordinated). Invalid values are most commonly treated as UTC, on some platforms with a warning.

inter_gap — The maximum allowable gap (in minutes) for interpolation. The values will not be interpolated between the glucose measurements that are more than inter_gap minutes apart. The default value is 45 min.

max_gap — **Default: 180.** Integer for the maximum length of a gap in minutes before the trace is split into segments and MAGE is calculated on each segment independently.

plot — **Default: FALSE.** Boolean. If 'TRUE', returns a plot that visualizes all identified peaks and nadirs, excursions, and missing gaps. An interactive GUI can be loaded with 'static_or_gui = 'plotly''.

title — **Default: "Glucose Trace - Subject [ID]".** Title for the ggplot.

xlab — **Default: "Time".** Label for x-axis of ggplot.

ylab — **Default: "Glucose Level".** Label for y-axis of ggplot.

show_ma — **Default: FALSE.** Boolean. If TRUE, plots the moving average lines on the plot.

show_excursions — **Default: TRUE.** Boolean. If TRUE, shows identified excursions as arrows from peak-to-nadir/nadir-to-peak on the plot.

Details

If version 'ma' is selected, the function computationally emulates the manual method for calculating the mean amplitude of glycemic excursions (MAGE) first suggested in "Mean Amplitude of Glycemic Excursions, a Measure of Diabetic Instability", (Service, 1970). For this version, glucose values will be interpolated over a uniform time grid prior to calculation.

'ma' is a more accurate algorithm that uses the crosses of a short and long moving average to identify intervals where a peak/nadir might exist. Then, the height from one peak/nadir to the next nadir/peak is calculated from the _original_ (not moving average) glucose values. (Note: this function internally uses CGMS2DayByDay with dt0 = 5. Thus, all CGM data is linearly interpolated to 5 minute intervals. See the MAGE vignette for more details.)

'naive' algorithm calculates MAGE by taking the mean of absolute glucose differences (between each value and the mean) that are greater than the standard deviation. A multiplier can be added to the standard deviation using the sd_multiplier argument.

Value

A tibble object with two columns: the subject id and corresponding MAGE value. If a vector of glucose values is passed, then a tibble object with just the MAGE value is returned.

In version = "ma", if plot = TRUE, a list of ggplots will be returned with one plot per subject. To return an interactive plot, use iglu::mage_ma_single with static_or_gui='plotly' on each subject individually.

References

Service et al. (1970) Mean amplitude of glycemic excursions, a measure of diabetic instability *Diabetes* **19** .644-655, doi:10.2337/diab.19.9.644.

Fernandes, Nathaniel J., et al. "Open-source algorithm to calculate mean amplitude of glycemic excursions using short and long moving averages." Journal of diabetes science and technology 16.2 (2022): 576-577. doi:10.1177/19322968211061165

Examples

```
data(example_data_5_subject)
mage(example_data_5_subject, version = 'ma')
```

---

mage_ma_single *Calculates Mean Amplitude of Glycemic Excursions (see "mage")*

Description

This function is an internal function used ‘mage‘. The function will calculate the Mean Amplitude of Glycemic Excursions (MAGE) on **all** the values of the inputted data set regardless of subject. To calculate separate MAGE values for a group of subjects, use the ‘mage‘ function.

Usage

```
mage_ma_single(
  data,
  short_ma = 5,
  long_ma = 32,
  return_type = c("num", "df"),
  direction = c("avg", "service", "max", "plus", "minus"),
  tz = "",
  inter_gap = 45,
  max_gap = 180,
  plot = FALSE,
  title = NA,
  xlab = NA,
  ylab = NA,
  show_ma = FALSE,
  show_excursions = TRUE,
  static_or_gui = c("plotly", "ggplot")
)
```

Arguments

data	DataFrame object with column names "id", "time", and "gl". Should only be data for 1 subject. In case multiple subject ids are detected, a warning is produced and only 1st subject is used.

short_ma	**Default: 5.** Integer for period length of the short moving average. Must be positive and less than 'long_ma'. (Recommended <15)
long_ma	**Default: 32.** Integer for period length for the long moving average. Must be positive and greater than 'short_ma'. (Recommended >20)
return_type	**Default: "num".** One of ("num", "df"). Will return either a single number for the "MAGE over the entire trace" (weighted by segment length) or a DataFrame with the MAGE value for each segment (see the MAGE vignette for discussion of handling large gaps by splitting trace into multiple segments).
direction	**Default: "avg".** One of ("avg", "service", "max", "plus", or "minus"). Algorithm will calculate one of the following: MAGE+ (nadir to peak), MAGE- (peak to nadir), MAGEavg = avg(MAGE+, MAGE-), MAGEmax = max(MAGE+, MAGE-), or automatically choose MAGE+/MAGE- based on the first countable excursion (i.e., "service"). NOTE: the selection of peak-to-nadir or nadir-to-peak is chosen independently on each segment, thus MAGEservice may choose peak-to-nadir on one segment and nadir-to-peak on another, for example.
tz	A character string specifying the time zone to be used. System-specific (see as.POSIXct), but " " is the current time zone, and "GMT" is UTC (Universal Time, Coordinated). Invalid values are most commonly treated as UTC, on some platforms with a warning.
inter_gap	The maximum allowable gap (in minutes) for interpolation. The values will not be interpolated between the glucose measurements that are more than inter_gap minutes apart. The default value is 45 min.
max_gap	**Default: 180.** Integer for the maximum length of a gap in minutes before the trace is split into segments and MAGE is calculated on each segment independently.
plot	**Default: FALSE.** Boolean. If 'TRUE', returns a plot that visualizes all identified peaks and nadirs, excursions, and missing gaps. An interactive GUI can be loaded with 'static_or_gui = 'plotly''.
title	**Default: "Glucose Trace - Subject [ID]".** Title for the ggplot.
xlab	**Default: "Time".** Label for x-axis of ggplot.
ylab	**Default: "Glucose Level".** Label for y-axis of ggplot.
show_ma	**Default: FALSE.** Boolean. If TRUE, plots the moving average lines on the plot.
show_excursions	**Default: TRUE.** Boolean. If TRUE, shows identified excursions as arrows from peak-to-nadir/nadir-to-peak on the plot.
static_or_gui	**Default: "plotly".** One of "ggplot" or "plotly". Returns either a ggplot (static image) or Plotly chart (interactive GUI).

Details

See 'mage'.

Value

A ggplot or Plotly chart if plot = TRUE, depending on static_or_gui. Otherwise, a numeric MAGE value for the inputted glucose trace or a DataFrame with the MAGE values on each segment, depending on return_type.

Author(s)

Nathaniel J. Fernandes

Examples

```
data(example_data_1_subject)
mage_ma_single(
   example_data_1_subject,
   short_ma = 4,
   long_ma = 24,
   direction = 'plus')

mage_ma_single(
   example_data_1_subject,
   inter_gap = 300)

mage_ma_single(
   example_data_1_subject,
   plot=TRUE,
   static_or_gui='ggplot',
   title="Patient X",
   xlab="Time",
   ylab="Glucose Level (mg/dL)",
   show_ma=FALSE)
```

---

meal_metrics *Calculate Meal Metrics*

Description

The function meal_metrics calculates three simple glucose meal metrics

Usage

```
meal_metrics(data, mealtimes, before_win = 1, after_win = 3,
recovery_win = 1, interpolate = TRUE, adjust_mealtimes = TRUE, dt0 = NULL,
inter_gap = 45, tz = "", glucose_times = FALSE)
```

Arguments

data	DataFrame object with column names "id", "time", and "gl". Should only be data for 1 subject. In case multiple subject ids are detected, a warning is produced and only 1st subject is used.
mealtimes	Either a vector of mealtimes, corresponding to data being from a single subject, OR a dataframe with at least 2 columns labeled id and mealtime. Optionally the mealtimes dataframe can include a column labeled meal, giving the meal type (helps to compensate for overlapping meals)
before_win	integer specifying number of hours to extend window before meal

`after_win` integer specifying number of hours to extend window after meal

`recovery_win` interger specifying number of hours for recovery beyond after window

`interpolate` Boolean to indicate if CGM data should be interpolated or not. Default set to FALSE due to time intensive nature of interpolation. Parameters dt0, inter_gap, and tz will only be used if interpolate is set to TRUE.

`adjust_mealtimes` Boolean to indicate if function should attempt to align mealtimes with CGM data times. This is important if mealtimes and CGM data times are not exactly aligned, because the function will return NA's for mealtimes that don't match with a corresponding CGM time stamp.

`dt0` The time frequency for interpolation in minutes, the default will match the CGM meter's frequency (e.g. 5 min for Dexcom).

`inter_gap` The maximum allowable gap (in minutes) for interpolation. The values will not be interpolated between the glucose measurements that are more than inter_gap minutes apart. The default value is 45 min.

`tz` A character string specifying the time zone to be used. System-specific (see `as.POSIXct`), but " " is the current time zone, and "GMT" is UTC (Universal Time, Coordinated). Invalid values are most commonly treated as UTC, on some platforms with a warning.

`glucose_times` Boolean to indicate if underlying glucose and times should be returned - e.g. baseline glucose value used to calculate $DeltaG$. Intended for use in plotting function

Details

A tibble object is returned with three metrics calculated for each mealtime. The last three columns of the output correspond to the three metrics: deltag refers to ΔG, deltat is ΔT, and basereco is % Baseline recovery. If no meal column is given in the original data, one will be automatically generated with a unique number for each meal.

Value

By default tibble object with 6 columns will be returned: id, time, meal, deltag, deltat, and basereco. If glucose_times = TRUE then 5 more columns are returned for baseline glucose (basegl), peak glucose (peakgl), recover glucose (recovergl), peak timestamp (peaktime), and recovery timestamp (recovertime)

References

Service, F. John. (2013) Glucose Variability, *Diabetes* **62(5)**: 1398-1404, doi:10.2337/db121396

See Also

plot_meals()

Examples

```
data(example_data_hall)
data(example_meals_hall)
meal_metrics(example_data_hall, example_meals_hall)
```

`mean_glu` *Calculate mean glucose level*

Description

The function mean_glu is a wrapper for the base function mean(). Output is a tibble object with subject id and mean values.

Usage

```
mean_glu(data)
```

Arguments

`data`	DataFrame object with column names "id", "time", and "gl", or numeric vector of glucose values.

Details

A tibble object with 1 row for each subject, a column for subject id and a column for the mean values is returned. NA glucose values are omitted from the calculation of the mean.

Value

If a data.frame object is passed, then a tibble object with two columns: subject id and corresponding mean value is returned. If a vector of glucose values is passed, then a tibble object with just the mean value is returned. as.numeric() can be wrapped around the latter to output just a numeric value.

Examples

```
data(example_data_1_subject)
mean_glu(example_data_1_subject)

data(example_data_5_subject)
mean_glu(example_data_5_subject)
```

median_glu *Calculate median glucose level*

Description

The function median_glu is a wrapper for the base function median(). Output is a tibble object with subject id and median values.

Usage

```
median_glu(data)
```

Arguments

data	DataFrame object with column names "id", "time", and "gl", or numeric vector of glucose values.

Details

A tibble object with 1 row for each subject, a column for subject id and a column for the median values is returned. NA glucose values are omitted from the calculation of the median.

Value

If a data.frame object is passed, then a tibble object with two columns: subject id and corresponding median value is returned. If a vector of glucose values is passed, then a tibble object with just the median value is returned. as.numeric() can be wrapped around the latter to output just a numeric value.

Examples

```
data(example_data_1_subject)
median_glu(example_data_1_subject)

data(example_data_5_subject)
median_glu(example_data_5_subject)
```

`metrics_heatmap` *Create a heatmap of metric values by subject based on hierarchical clustering order*

Description

Create a heatmap of metric values by subject based on hierarchical clustering order

Usage

```
metrics_heatmap(
  data = NULL,
  metrics = NULL,
  metric_cluster = 6,
  clustering_method = "complete",
  clustering_distance_metrics = "correlation",
  clustering_distance_subjects = "correlation"
)
```

Arguments

`data`	DataFrame object with column names "id", "time", and "gl".
`metrics`	precalculated metric values, with first column corresponding to subject id. If 'NULL', the metrics are calculated from supplied 'data' using `all_metrics`
`metric_cluster`	number of visual metric clusters, default value is 6
`clustering_method`	the agglomeration method for hierarchical clustering, accepts same values as `hclust`, default value is 'complete'
`clustering_distance_metrics`	the distance measure for metrics clustering, accepts same values as `dist`, default value is 'correlation' distance
`clustering_distance_subjects`	the distance measure for subjects clustering, accepts same values as `dist`, default value is 'correlation' distance

Value

A heatmap of metrics by subjects generated via `pheatmap`

Examples

```
# Using pre-calculated sd metrics only rather than default (all metrics)
mecs = sd_measures(example_data_5_subject)
metrics_heatmap(metrics = mecs)
```

modd *Calculate mean difference between glucose values obtained at the same time of day (MODD)*

Description

The function modd produces MODD values in a tibble object.

Usage

```
modd(data, lag = 1, tz = "")
```

Arguments

data	DataFrame object with column names "id", "time", and "gl".
lag	Integer indicating which lag (# days) to use. Default is 1.
tz	A character string specifying the time zone to be used. System-specific (see as.POSIXct), but " " is the current time zone, and "GMT" is UTC (Universal Time, Coordinated). Invalid values are most commonly treated as UTC, on some platforms with a warning.

Details

A tibble object with 1 row for each subject, a column for subject id and a column for the MODD values is returned.

Missing values will be linearly interpolated when close enough to non-missing values.

MODD is calculated by taking the mean of absolute differences between measurements at the same time 1 day away, or more if lag parameter is set to an integer > 1.

Value

A tibble object with two columns: subject id and corresponding MODD value.

References

Service, F. J. & Nelson, R. L. (1980) Characteristics of glycemic stability. *Diabetes care* **3** .58-62, doi:10.2337/diacare.3.1.58.

Examples

```
data(example_data_1_subject)
modd(example_data_1_subject)
modd(example_data_1_subject, lag = 2)

data(example_data_5_subject)
modd(example_data_5_subject, lag = 2)
```

`m_value` *Calculate the M-value*

Description

Calculates the M-value of Schlichtkrull et al. (1965) for each subject in the data, where the M-value is the mean of the logarithmic transformation of the deviation from a reference value. Produces a tibble object with subject id and M-values.

Usage

```
m_value(data, r = 90)
```

Arguments

`data`	DataFrame object with column names "id", "time", and "gl", or numeric vector of glucose values.
`r`	A reference value corresponding to basal glycemia in normal subjects; default is 90 mg/dL.

Details

A tibble object with 1 row for each subject, a column for subject id and a column for the M-values is returned. NA glucose values are omitted from the calculation of the M-value.

M-value is computed by averaging the transformed glucose values, where each transformed value is equal to $|10 * log_10(glucose/r)|^3$, where r is the specified reference value.

Value

If a data.frame object is passed, then a tibble object with two columns: subject id and corresponding M-value is returned. If a vector of glucose values is passed, then a tibble object with just the M-value is returned. as.numeric() can be wrapped around the latter to output just a numeric value.

References

Schlichtkrull J, Munck O, Jersild M. (1965) The M-value, an index of blood-sugar control in diabetics. *Acta Medica Scandinavica* **177** .95-102. doi:10.1111/j.09546820.1965.tb01810.x.

Examples

```
data(example_data_5_subject)

m_value(example_data_5_subject)
m_value(example_data_5_subject, r = 100)
```

`optimized_iglu_functions` *Optimized Calculations of Time Dependent iglu Metrics*

Description

The function optimized_iglu_functions optimizes the calculation of all time dependent iglu metrics by extracting the CGMS2DayByDay calculation and passing the result into each function.

Usage

```
optimized_iglu_functions(data, dt0 = NULL, inter_gap = 45, tz = "", timelag = 15, lag = 1)
```

Arguments

`data`	DataFrame object with column names "id", "time", and "gl".
`dt0`	The time frequency for interpolation in minutes, the default will match the CGM meter's frequency (e.g. 5 min for Dexcom).
`inter_gap`	The maximum allowable gap (in minutes) for interpolation. The values will not be interpolated between the glucose measurements that are more than inter_gap minutes apart. The default value is 45 min.
`tz`	A character string specifying the time zone to be used. System-specific (see `as.POSIXct`), but " " is the current time zone, and "GMT" is UTC (Universal Time, Coordinated). Invalid values are most commonly treated as UTC, on some platforms with a warning.
`timelag`	Integer indicating the time period (# minutes) over which rate of change is calculated. Default is 15, e.g. rate of change is the change in glucose over the past 15 minutes divided by 15.
`lag`	Integer indicating which lag (# days) to use. Default is 1.

Details

Returns a tibble object with 1 row for each subject and a column for each metric. This function includes time dependent iglu metrics only. For metric specific information, please see the corresponding function documentation.

Value

If a data.frame object is passed, then a tibble object with 1 row for each subject and one column for each metric is returned.

Examples

```
data(example_data_1_subject)
optimized_iglu_functions(example_data_1_subject)

# Pass some arguments to possibly change the defaults
optimized_iglu_functions(example_data_1_subject, dt0 = 5, inter_gap = 30)

data(example_data_5_subject)
optimized_iglu_functions(example_data_5_subject)
```

---

pgs *Calculate Personal Glycemic State (PGS)*

Description

The function mad produces PGS values in a tibble object.

Usage

```
pgs(data, dur_length = 20, end_length = 30)
```

Arguments

data	DataFrame object with column names "id", "time", and "gl". Should only be data for 1 subject. In case multiple subject ids are detected, a warning is produced and only 1st subject is used.
dur_length	Numeric value specifying the minimum duration in minutes to be considered an episode. Note dur_length should be a multiple of the data recording interval otherwise the function will round up to the nearest multiple. Default is 20 minutes to match the original PGS definition.
end_length	Numeric value specifying the minimum duration in minutes of improved glycemia for an episode to end. Default is 30 minutes to match original PGS definition.

Details

A tibble object with 1 row for each subject, a column for subject id and a column for PGS values is returned. NA glucose values are omitted from the calculation. The formula for PGS is as follows, where GVP = glucose variability percentage, MG = mean glucose, PTIR = percent time in range, and N54, N70 are the number of hypoglycemic episodes per week in the ranges <54 mg/dL and 54 to <70 mg/dL level respectively.

$$PGS = f(GVP) + g(MG) + h(PTIR) + j(N54, N70)$$

The component functions are listed below.

$$f(GVP) = 1 + \frac{9}{1 + \exp(-0.049(GVP - 65.47))} g(MG) = 1 + 9(\frac{1}{1 + \exp(0.1139(MG - 72.08))} + \frac{1}{1 + \exp(-0.09195(M}$$

and b(N70) is defined such that b(N70) = $0.5714N70 + 0.625$ if N70 <= 7.65, and b(N70) = 5 otherwise.

Note that the duration thresholds for episodes are NOT the same as the episode_calculation defaults. The defaults chosen for PGS are those that match the original PGS paper definition, while the episode_calculation defaults match the consensus.

Value

A tibble object with two columns: subject id and corresponding PGS value.

Author(s)

Elizabeth Chun

References

Hirsch et al. (2017): A Simple Composite Metric for the Assessment of Glycemic Status from Continuous Glucose Monitoring Data: Implications for Clinical Practice and the Artificial Pancreas *Diabetes Technol Ther* **19(S3)** .S38-S48, doi:10.1089/dia.2017.0080.

See Also

episode_calculation()

Examples

```
data(example_data_1_subject)
pgs(example_data_1_subject)
```

---

plot_agp *Plot Ambulatory Glucose Profile (AGP) modal day*

Description

The function plot_agp produces an AGP plot that collapses all data into a single 24 hr "modal day".

Usage

```
plot_agp(data, LLTR = 70, ULTR = 180, smooth = TRUE, span = 0.3, dt0 = NULL,
inter_gap = 45, tz = "", title = FALSE)
```

Arguments

data	DataFrame object with column names "id", "time", and "gl". Should only be data for 1 subject. In case multiple subject ids are detected, a warning is produced and only 1st subject is used.
LLTR	**Default: 70.** Lower Limit of Target Range in mg/dL.
ULTR	**Default: 180.** Upper Limit of Target Range in mg/dL.
smooth	Boolean indicating whether quantiles should be smoothed before plotting, default is TRUE
span	Optional parameter indicating span for loess smoothing. Default is 0.3, larger values result in more smoothing, recommended to choose between 0.1 to 0.7.
dt0	The time frequency for interpolation in minutes, the default will match the CGM meter's frequency (e.g. 5 min for Dexcom).
inter_gap	The maximum allowable gap (in minutes) for interpolation. The values will not be interpolated between the glucose measurements that are more than inter_gap minutes apart. The default value is 45 min.
tz	A character string specifying the time zone to be used. System-specific (see as.POSIXct), but " " is the current time zone, and "GMT" is UTC (Universal Time, Coordinated). Invalid values are most commonly treated as UTC, on some platforms with a warning.
title	Indicator whether the title of the plot should display the subject ID. The default is FALSE (no title).

Details

Only a single subject's data may be plotted. If smooth = TRUE, then the quantiles are loess smoothed with the specified span before plotting. The horizontal green lines represent the target range, default is 70-180 mg/dL. The black line is the median glucose value for each time of day. The dark blue shaded area represents 50% of glucose values - those between the 25th and 75 quantiles. The light blue shaded area shows 90% of the glucose values - those between the 5th and 95th quantiles. Additionally, the percents shown on the right hand side of the plot show which quantiles each line refers to - e.g. the line ending at 95% is the line corresponding to the 95th quantiles of glucose values.

Value

Plot of a 24 hr modal day collapsing all data to a single day.

Author(s)

Elizabeth Chun

References

Johnson et al. (2019) Utilizing the Ambulatory Glucose Profile to Standardize and Implement Continuous Glucose Monitoring in Clinical Practice, *Diabetes Technology and Therapeutics* **21:S2** S2-17-S2-25, doi:10.1089/dia.2019.0034.

Examples

```
data(example_data_1_subject)
plot_agp(example_data_1_subject)
```

`plot_daily` *Plot daily glucose profiles*

Description

The function ‘plot_daily‘ plots daily glucose time series profiles for a single subject.

Usage

```
plot_daily(data, maxd = 14, LLTR = 70, ULTR = 180, inter_gap = 45, tz = "")
```

Arguments

`data`	DataFrame with column names ("id", "time", and "gl").
`maxd`	**Default: 14.** Number of days to plot. If less than ‘maxd‘ days of data are available, all days are plotted.
`LLTR`	**Default: 70.** Lower Limit of Target Range in mg/dL.
`ULTR`	**Default: 180.** Upper Limit of Target Range in mg/dL.
`inter_gap`	**Default: 45**. The maximum allowable gap (in minutes). Gaps larger than this will not be connected in the time series plot.
`tz`	A character string specifying the time zone to be used. System-specific (see `as.POSIXct`), but " " is the current time zone, and "GMT" is UTC (Universal Time, Coordinated). Invalid values are most commonly treated as UTC, on some platforms with a warning.

Details

Only a single subject’s data may be plotted. The black line shows the glucose values. The shaded gray area shows the target range, default 70-180 mg/dL. Areas of the curve above the ULTR are shaded yellow, while areas below the LLTR are shaded red.

Value

Daily glucose time series plots for a single subject

Author(s)

Elizabeth Chun

References

Johnson et al. (2019) Utilizing the Ambulatory Glucose Profile to Standardize and Implement Continuous Glucose Monitoring in Clinical Practice, *Diabetes Technology and Therapeutics* **21:S2** S2-17-S2-25, doi:10.1089/dia.2019.0034.

Examples

```
data(example_data_1_subject)
plot_daily(example_data_1_subject)
plot_daily(example_data_1_subject, LLTR = 100, ULTR = 140)
```

`plot_glu` *Plot time series and lasagna plots of glucose measurements*

Description

The function ‘plot_glu‘ supports several plotting methods for both single and multiple subject data.

Usage

```
plot_glu(
  data,
  plottype = c("tsplot", "lasagna"),
  datatype = c("all", "average", "single"),
  lasagnatype = c("unsorted", "timesorted"),
  LLTR = 70,
  ULTR = 180,
  subjects = NULL,
  inter_gap = 45,
  tz = "",
  color_scheme = c("blue-red", "red-orange"),
  log = F,
  static_or_gui = c("ggplot", "plotly")
)
```

Arguments

`data`	DataFrame with column names ("id", "time", and "gl").
`plottype`	**Default: "tsplot".** One of (’tsplot’, ’lasagna’). String corresponding to the desired plot type. Options are ’tsplot’ for a time series plot and ’lasagna’ for a lasagna plot. See the ‘lasagnatype‘ parameter for further options corresponding to the ’lasagna’ ‘plottype‘.
`datatype`	String corresponding to data aggregation used for plotting, currently supported options are ’all’ which plots all glucose measurements within the first `maxd` days for each subject, and ’average’ which plots average 24 hour glucose values across days for each subject

lasagnatype	String corresponding to plot type when using datatype = "average", currently supported options are 'unsorted' for an unsorted lasagna plot, 'timesorted' for a lasagna plot with glucose values sorted within each time point across subjects, and ''subjectsorted'' for a lasagna plot with glucose values sorted within each subject across time points.
LLTR	**Default: 70.** Lower Limit of Target Range in mg/dL.
ULTR	**Default: 180.** Upper Limit of Target Range in mg/dL.
subjects	String or list of strings corresponding to subject names in 'id' column of data. Default is all subjects.
inter_gap	**Default: 45**. The maximum allowable gap (in minutes). Gaps larger than this will not be connected in the time series plot.
tz	A character string specifying the time zone to be used. System-specific (see as.POSIXct), but " " is the current time zone, and "GMT" is UTC (Universal Time, Coordinated). Invalid values are most commonly treated as UTC, on some platforms with a warning.
color_scheme	**Default: "blue-red".** String corresponding to the chosen color scheme when the 'plottype' is 'lasagna'. By default, 'blue-red' scheme is used, with the values below 'LLTR' colored in shades of blue, and values above 'ULTR' colored in shades of red. The alternative 'red-orange' scheme mimics AGP output from agp with low values colored in red, in-range values colored in green, and high values colored in yellow and orange.
log	**Default: FALSE.** Boolean indicating whether 'log10' of glucose values should be taken. When 'log = TRUE', the glucose values, LLTR, and ULTR will all be log transformed, and time series plots will be on a semilogarithmic scale.
static_or_gui	**Default: "ggplot".** One of ("ggplot", "plotly"). Returns either a ggplot (static image) or Plotly chart (interactive GUI).

Details

For the default option 'plottype = tsplot', a time series graph for each subject is produced with hypo- and hyperglycemia cutoffs shown as horizontal red lines. The time series plots for all subjects chosen (all by default) are displayed on a grid.

The 'lasagna' plot type works best when the datatype argument is set to average.

Value

Any output from the plot object

Examples

```
data(example_data_1_subject)
plot_glu(example_data_1_subject)

data(example_data_5_subject)
plot_glu(example_data_5_subject, subjects = 'Subject 2')
plot_glu(example_data_5_subject, plottype = 'tsplot', tz = 'EST', LLTR = 70, ULTR = 150)
plot_glu(example_data_5_subject, plottype = 'lasagna', lasagnatype = 'timesorted')
```

plot_lasagna *Lasagna plot of glucose values for multiple subjects*

Description

Lasagna plot of glucose values for multiple subjects

Usage

```
plot_lasagna(
  data,
  datatype = c("all", "average"),
  lasagnatype = c("unsorted", "timesorted", "subjectsorted"),
  maxd = 14,
  limits = c(50, 500),
  midpoint = 105,
  LLTR = 70,
  ULTR = 180,
  dt0 = NULL,
  inter_gap = 45,
  tz = "",
  color_scheme = c("blue-red", "red-orange"),
  log = F,
  static_or_gui = c("ggplot", "plotly")
)
```

Arguments

data	DataFrame object with column names "id", "time", and "gl".
datatype	String corresponding to data aggregation used for plotting, currently supported options are 'all' which plots all glucose measurements within the first maxd days for each subject, and 'average' which plots average 24 hour glucose values across days for each subject
lasagnatype	String corresponding to plot type when using datatype = "average", currently supported options are 'unsorted' for an unsorted lasagna plot, 'timesorted' for a lasagna plot with glucose values sorted within each time point across subjects, and ''subjectsorted'' for a lasagna plot with glucose values sorted within each subject across time points.
maxd	For datatype "all", maximal number of days to be plotted from the study. The default value is 14 days (2 weeks).
limits	The minimal and maximal glucose values for coloring grid which is gradient from blue (minimal) to red (maximal), see scale_fill_gradient2)
midpoint	The glucose value serving as midpoint of the diverging gradient scale (see scale_fill_gradient2). The default value is 105 mg/dL. The values above are colored in red, and below in blue in the default color_scheme, which can be adjusted.

LLTR	Lower Limit of Target Range, default value is 70 mg/dL.
ULTR	Upper Limit of Target Range, default value is 180 mg/dL.
dt0	The time frequency for interpolated aligned grid in minutes, the default will match the CGM meter's frequency (e.g. 5 min for Dexcom).
inter_gap	The maximum allowable gap (in minutes) for interpolation of NA glucose values. The values will not be interpolated between the glucose measurements that are more than inter_gap minutes apart. The default value is 45 min.
tz	A character string specifying the time zone to be used. System-specific (see as.POSIXct), but " " is the current time zone, and "GMT" is UTC (Universal Time, Coordinated). Invalid values are most commonly treated as UTC, on some platforms with a warning.
color_scheme	String corresponding to the chosen color scheme. By default, 'blue-red' scheme is used, with the values below 'LLTR' colored in shades of blue, and values above 'ULTR' colored in shades of red. The alternative 'red-orange' scheme mimics AGP output from agp with low values colored in red, in-range values colored in green, and high values colored in yellow and orange.
log	Logical value indicating whether log10 of glucose values should be taken, default value is FALSE. When log = TRUE the glucose values, limits, midpoint, LLTR, and ULTR will all be log transformed.
static_or_gui	One of "ggplot" or "plotly". **Default: "plotly".** Returns either a ggplot (static image) or Plotly chart (interactive GUI).

Value

A ggplot object corresponding to lasagna plot

References

Swihart et al. (2010) Lasagna Plots: A Saucy Alternative to Spaghetti Plots, *Epidemiology* **21**(5), 621-625, doi:10.1097/EDE.0b013e3181e5b06a

Examples

```
plot_lasagna(example_data_5_subject, datatype = "average", lasagnatype = 'timesorted', tz = "EST")
plot_lasagna(example_data_5_subject, lasagnatype = "subjectsorted", LLTR = 100, tz = "EST")
```

---

plot_lasagna_1subject *Lasagna plot of glucose values for 1 subject aligned across times of day*

Description

Lasagna plot of glucose values for 1 subject aligned across times of day

Usage

```
plot_lasagna_1subject(
  data,
  lasagnatype = c("unsorted", "timesorted", "daysorted"),
  limits = c(50, 500),
  midpoint = 105,
  LLTR = 70,
  ULTR = 180,
  dt0 = NULL,
  inter_gap = 45,
  tz = "",
  color_scheme = c("blue-red", "red-orange"),
  log = F,
  static_or_gui = c("ggplot", "plotly")
)
```

Arguments

data	DataFrame object with column names "id", "time", and "gl".
lasagnatype	**Default: "unsorted".** String corresponding to plot type, currently supported options are 'unsorted' for an unsorted single-subject lasagna plot, 'timesorted' for a lasagna plot with glucose values sorted within each time point across days, and 'daysorted' for a lasagna plot with glucose values sorted within each day across time points.
limits	The minimal and maximal glucose values for coloring grid which is gradient from blue (minimal) to red (maximal), see scale_fill_gradient2)
midpoint	The glucose value serving as midpoint of the diverging gradient scale (see scale_fill_gradient2). The default value is 105 mg/dL. The values above are colored in red, and below in blue in the default color_scheme, which can be adjusted.
LLTR	Lower Limit of Target Range, default value is 70 mg/dL.
ULTR	Upper Limit of Target Range, default value is 180 mg/dL.
dt0	The time frequency for interpolated aligned grid in minutes, the default will match the CGM meter's frequency (e.g. 5 min for Dexcom).
inter_gap	The maximum allowable gap (in minutes) for interpolation of NA glucose values. The values will not be interpolated between the glucose measurements that are more than inter_gap minutes apart. The default value is 45 min.
tz	A character string specifying the time zone to be used. System-specific (see as.POSIXct), but " " is the current time zone, and "GMT" is UTC (Universal Time, Coordinated). Invalid values are most commonly treated as UTC, on some platforms with a warning.
color_scheme	String corresponding to the chosen color scheme. By default, 'blue-red' scheme is used, with the values below 'LLTR' colored in shades of blue, and values above 'ULTR' colored in shades of red. The alternative 'red-orange' scheme mimics AGP output from agp with low values colored in red, in-range values colored in green, and high values colored in yellow and orange.

log	Logical value indicating whether log of glucose values should be taken, default values is FALSE. When log = TRUE the glucose values, limits, midpoint, LLTR, and ULTR will all be log transformed.
static_or_gui	One of "ggplot" or "plotly". **Default: "plotly".** Returns either a ggplot (static image) or Plotly chart (interactive GUI).

Value

A ggplot object corresponding to lasagna plot

References

Swihart et al. (2010) Lasagna Plots: A Saucy Alternative to Spaghetti Plots, *Epidemiology* **21**(5), 621-625, doi:10.1097/EDE.0b013e3181e5b06a

Examples

```
plot_lasagna_1subject(example_data_1_subject)
plot_lasagna_1subject(example_data_1_subject, color_scheme = 'red-orange')
plot_lasagna_1subject(example_data_1_subject, lasagnatype = 'timesorted')
plot_lasagna_1subject(example_data_1_subject, lasagnatype = 'daysorted')
plot_lasagna_1subject(example_data_1_subject, log = TRUE)
```

---

plot_meals *Plot meal metrics visualization*

Description

The function plot_meals produces a visual for meals data

Usage

```
plot_meals(data, mealtimes, plot_type=c('ggplot','plotly'))
```

Arguments

data	DataFrame object with column names "id", "time", and "gl". Should only be data for 1 subject. In case multiple subject ids are detected, a warning is produced and only 1st subject is used.
mealtimes	Either a vector of mealtimes, corresponding to data being from a single subject, OR a dataframe with at least 2 columns labeled id and mealtime. Optionally the mealtimes dataframe can include a column labeled meal, giving the meal type (helps to compensate for overlapping meals)
plot_type	Default: "ggplot". One of 'ggplot', 'plotly'. Determines whether the function returns a static publication-ready image or an interactive GUI.

Details

Only a single subject's data may be plotted. The solid black line is the glucose trace. Vertical dashed red lines show the mealtimes, and the horizontal blue lines show the baseline for each meal. Purple triangles are plotted to illustrate the 3 meal_metrics Namely the three vertices show the baseline, peak, and 1hr post-peak recovery. If plot_type = 'plotly', plotly is used to display an interactive visual that allows one to zoom into specific areas of the plot.

Value

Plot to visualize meals data.

Author(s)

Elizabeth Chun

References

Service, F. John. (2013) Glucose Variability, *Diabetes* **62(5)**: 1398-1404, doi:10.2337/db121396

See Also

meal_metrics()

Examples

```
select_subject = example_data_hall[example_data_hall$id == "2133-018", ]
select_meals = example_meals_hall[example_meals_hall$id == "2133-018", ]
plot_meals(select_subject, select_meals)
```

---

plot_ranges *Plot Time in Ranges as a bar plot*

Description

The function plot_ranges produces a barplot showing the percent of time in glucose ranges.

Usage

```
plot_ranges(data)
```

Arguments

data	DataFrame object with column names "id", "time", and "gl". Should only be data for 1 subject. In case multiple subject ids are detected, a warning is produced and only 1st subject is used.

Details

Only a single subject's data may be used. There are four ranges: very low (below 54 mg/dL), low (54-69 mg/dL), target range (70-180 mg/dL), high (181-250 mg/dL), and very high (above 250 mg/dL). This plot is meant to be used as part of the Ambulatory Glucose Profile (AGP)

Value

Single subject bar chart showing percent in different glucose ranges.

Author(s)

Elizabeth Chun

References

Johnson et al. (2019) Utilizing the Ambulatory Glucose Profile to Standardize and Implement Continuous Glucose Monitoring in Clinical Practice, *Diabetes Technology and Therapeutics* **21:S2** S2-17-S2-25, doi:10.1089/dia.2019.0034.

Examples

```
data(example_data_1_subject)
plot_ranges(example_data_1_subject)
```

`plot_roc` *Plot time series of glucose colored by rate of change*

Description

The function plot_roc produces a time series plot of glucose values colored by categorized rate of change values

Usage

```
plot_roc(data, subjects = NULL, timelag = 15, dt0 = NULL, inter_gap = 45, tz = "")
```

Arguments

`data`	DataFrame object with column names "id", "time", and "gl".
`subjects`	String or list of strings corresponding to subject names in 'id' column of data. Default is all subjects.
`timelag`	Integer indicating the time period (# minutes) over which rate of change is calculated. Default is 15, e.g. rate of change is the change in glucose over the past 15 minutes divided by 15.
`dt0`	The time frequency for interpolation in minutes, the default will match the CGM meter's frequency (e.g. 5 min for Dexcom).

inter_gap	The maximum allowable gap (in minutes) for interpolation. The values will not be interpolated between the glucose measurements that are more than inter_gap minutes apart. The default value is 45 min.
tz	A character string specifying the time zone to be used. System-specific (see as.POSIXct), but " " is the current time zone, and "GMT" is UTC (Universal Time, Coordinated). Invalid values are most commonly treated as UTC, on some platforms with a warning.

Details

For the default, a time series is produced for each subject in which the glucose values are plotted and colored by ROC categories defined as follows. The breaks for the categories are: c(-Inf, -3, -2, -1, 1, 2, 3, Inf) where the glucose is in mg/dl and the ROC values are in mg/dl/min. A ROC of -5 mg/dl/min will thus be placed in the first category and colored accordingly. The breaks for the categories come from the reference paper below.

Value

A time series of glucose values colored by ROC categories per subject

Author(s)

Elizabeth Chun, David Buchanan

References

Klonoff, D. C., & Kerr, D. (2017) A Simplified Approach Using Rate of Change Arrows to Adjust Insulin With Real-Time Continuous Glucose Monitoring. *Journal of Diabetes Science and Technology* **11(6)** 1063-1069, doi:10.1177/1932296817723260.

Examples

```
data(example_data_1_subject)
plot_roc(example_data_1_subject)

data(example_data_5_subject)
plot_roc(example_data_5_subject, subjects = 'Subject 5')
```

---

process_data *Data Pre-Processor*

Description

A helper function to assist in pre-processing the user-supplied input data for use with other functions. Typically, this function will process the data and return another DataFrame. This function ensures that the returned data will be compatible with every function within the iglu package. All NAs will be removed. See Vignette for further details.

Usage

```
process_data(data, id, timestamp, glu, time_parser = as.POSIXct)
```

Arguments

data	User-supplied dataset containing continuous glucose monitor data. Must contain data for time and glucose readings at a minimum. Accepted formats are dataframe and tibble.
id	Optional column name (character string) corresponding to subject id column. If no value is passed, an id of 1 will be assigned to the data.
timestamp	Required column name (character string) corresponding to time values in data. The dates can be in any format parsable by as.POSIXct, or any format accepted by the parser passed to time_parser. See time_parser param for an explanation on how to handle arbitrary formats.
glu	Required column name (character string) corresponding to glucose values, mg/dL
time_parser	Optional function used to convert datetime strings to time objects. Defaults to as.POSIXct. If your times are in a format not parsable by as.POSIXct, you can parse a custom format by passing function(time_string) {strptime(time_string, format = <format string>)} as the time_parser parameter.

Details

A dataframe with the columns "id", "time", and "gl" will be returned. All NAs will be removed.

If "mmol/l" in the glucose column name, the glucose values will be multipled by 18 to convert to mg/dL.

Based on John Schwenck's data_process for his bp package "https://github.com/johnschwenck/bp".

Value

A processed DataFrame object that cooperates with every other function within the iglu package - all column names and formats comply.

Author(s)

David Buchanan, John Schwenck

Examples

```
data("example_data_1_subject")

# Process example data
processed <- process_data(example_data_1_subject, id = "id", timestamp = "time", glu = "gl")

processed

data("example_data_5_subject")

# Process example data
```

```
processed_5subj <- process_data(example_data_5_subject, id = "id", timestamp = "time", glu = "gl")

processed_5subj
```

---

quantile_glu *Calculate glucose level quantiles*

Description

The function quantile_glu is a wrapper for the base function quantile(). Output is a tibble object with columns for subject id and each of the quantiles.

Usage

```
quantile_glu(data, quantiles = c(0, 25, 50, 75, 100))
```

Arguments

data	DataFrame object with column names "id", "time", and "gl", or numeric vector of glucose values.
quantiles	List of quantile values between 0 and 100.

Details

A tibble object with 1 row for each subject, a column for subject id and a column for each quantile is returned. NA glucose values are omitted from the calculation of the quantiles.

The values are scaled from 0-1 to 0-100 to be consistent in output with above_percent, below_percent, and in_range_percent.

The command quantile_glu(...) / 100 will scale each element down from 0-100 to 0-1.

Value

If a DataFrame object is passed, then a tibble object with a column for subject id and then a column for each quantile value is returned. If a vector of glucose values is passed, then a tibble object without the subject id is returned. as.numeric() can be wrapped around the latter to output a numeric vector.

Examples

```
data(example_data_1_subject)

quantile_glu(example_data_1_subject)
quantile_glu(example_data_1_subject, quantiles = c(0, 33, 66, 100))

data(example_data_5_subject)

quantile_glu(example_data_5_subject)
quantile_glu(example_data_5_subject, quantiles = c(0, 10, 90, 100))
```

range_glu *Calculate glucose level range*

Description

The function range_glu outputs the distance between minimum and maximum glucose values per subject in a tibble object.

Usage

```
range_glu(data)
```

Arguments

data	DataFrame object with column names "id", "time", and "gl", or numeric vector of glucose values.

Details

A tibble object with 1 row for each subject, a column for subject id and a column for the range values is returned. NA glucose values are omitted from the calculation of the range.

Value

If a DataFrame object is passed, then a tibble object with two columns: subject id and corresponding range value is returned. If a vector of glucose values is passed, then a tibble object with just the range value is returned. as.numeric() can be wrapped around the latter to output just a numeric value.

Examples

```
data(example_data_1_subject)
range_glu(example_data_1_subject)

data(example_data_5_subject)
range_glu(example_data_5_subject)
```

read_raw_data *Read raw data from a variety of common sensors.*

Description

Helper function to assist in reading data directly from sensor outputs. Should return a dataframe in correct format for use with the rest of the iglu package. Assumes all data will be readable with base R read.csv() function.

Usage

```
read_raw_data(
  filename,
  sensor = c("dexcom", "libre", "librepro", "asc", "ipro"),
  id = "filename",
  tz = ""
)
```

Arguments

filename	String matching the name of the data to be read. Assumed to be .csv
sensor	**Default: "dexcom".** String naming the type of sensor the data was exported from. Must be one of "dexcom", "libre", "librepro", "asc", or "ipro".
id	**Default: "filename".** String indicating subject id. A value of "read" will cause the program to attempt to read the subject id from the file. A value of "filename" will cause the program to use the basename of the filename (i.e. filename without any directory information) with .csv removed, as subject id. A value of "default" will cause the program to use whichever of "read" or "filename" that is default for that specific sensor. Any other string will be treated as the unique id for the entire file. Note the asc reader currently does not support id="read"
tz	A character string specifying the time zone to be used. System-specific (see as.POSIXct), but " " is the current time zone, and "GMT" is UTC (Universal Time, Coordinated). Invalid values are most commonly treated as UTC, on some platforms with a warning.

Details

A DataFrame object with the columns "id", "time" and "gl" and one row per reading will be returned. For the libre reader, if the phrase "mmol/l" is found in the column names, the glucose values will be multiplied by 18.

Assumes .csv format for all data.

Sensor formats change with ongoing development, so these functions may become deprecated. If any issues are encountered, contact the package maintainer: this is currently Irina Gaynanova, who can be reached at <irinagn@umich.edu>.

Note: this function is heavily derived from the readers avaiable in the cgmanalysis package's `cleandata` function.

Value

A dataframe containing the data read from the named file.

Author(s)

David Buchanan

References

Vigers et al. (2019) cgmanalysis: An R package for descriptive analysis of continuous glucose monitor data *PLoS ONE* **14(10)**: e0216851, doi:10.1371/journal.pone.0216851

`roc` *Calculate the Rate of Change at each time point (ROC)*

Description

The function `roc` produces rate of change values in a tibble object.

Usage

```
roc(data, timelag = 15, dt0 = NULL, inter_gap = 45, tz = "")
```

Arguments

`data`	DataFrame object with column names "id", "time", and "gl".
`timelag`	Integer indicating the time period (# minutes) over which rate of change is calculated. Default is 15, e.g. rate of change is the change in glucose over the past 15 minutes divided by 15.
`dt0`	The time frequency for interpolation in minutes, the default will match the CGM meter's frequency (e.g. 5 min for Dexcom).
`inter_gap`	The maximum allowable gap (in minutes) for interpolation. The values will not be interpolated between the glucose measurements that are more than inter_gap minutes apart. The default value is 45 min.
`tz`	A character string specifying the time zone to be used. System-specific (see `as.POSIXct`), but " " is the current time zone, and "GMT" is UTC (Universal Time, Coordinated). Invalid values are most commonly treated as UTC, on some platforms with a warning.

Details

A tibble object with a column for subject id and a column for ROC values is returned. A ROC value is returned for each time point for all the subjects. Thus multiple rows are returned for each subject. If the rate of change cannot be calculated, the function will return NA for that point.

The glucose values are linearly interpolated over a time grid starting at the beginning of the first day of data and ending on the last day of data. Because of this, there may be many NAs at the beginning and the end of the roc values for each subject. These NAs are a result of interpolated time points that do not have recorded glucose values near them because recording had either not yet begun for the day or had already ended.

The ROC is calculated as $\frac{G(t_i)-G(t_{i-1})}{t_i-t_{i-1}}$ where G_i is the Glucose measurement at time t_i and G_{i-1} is the Glucose measurement at time t_{i-1}. The time difference between the points, $t_i - t_{i-1}$, is selectable and set at a default of 15 minutes.

Value

A tibble object with two columns: subject id and rate of change values

Author(s)

Elizabeth Chun, David Buchanan

References

Clarke et al. (2009) Statistical Tools to Analyze Continuous Glucose Monitor Data, Diabetes *Diabetes Technology and Therapeutics* **11** S45-S54, doi:10.1089/dia.2008.0138.

Examples

```
data(example_data_1_subject)
roc(example_data_1_subject)
roc(example_data_1_subject, timelag = 10)

data(example_data_5_subject)
roc(example_data_5_subject)
```

---

sd_glu *Calculate sd glucose level*

Description

The function sd_glu is a wrapper for the base function sd(). Output is a tibble object with subject id and sd values.

Usage

```
sd_glu(data)
```

Arguments

data	DataFrame object with column names "id", "time", and "gl", or numeric vector of glucose values.

Details

A tibble object with 1 row for each subject, a column for subject id and a column for the sd values is returned. NA glucose values are omitted from the calculation of the sd.

Value

If a data.frame object is passed, then a tibble object with two columns: subject id and corresponding sd value is returned. If a vector of glucose values is passed, then a tibble object with just the sd value is returned. as.numeric() can be wrapped around the latter to output just a numeric value.

Examples

```
data(example_data_1_subject)
sd_glu(example_data_1_subject)

data(example_data_5_subject)
sd_glu(example_data_5_subject)
```

---

sd_measures *Calculate SD subtypes*

Description

The function sd_measures produces SD subtype values in a tibble object with a row for each subject and columns corresponding to id followed by each SD subtype.

Usage

```
sd_measures(data,dt0 = NULL, inter_gap = 45, tz = "")
```

Arguments

data	DataFrame object with column names "id", "time", and "gl".
dt0	The time frequency for interpolation in minutes, the default will match the CGM meter's frequency (e.g. 5 min for Dexcom).
inter_gap	The maximum allowable gap (in minutes) for interpolation. The values will not be interpolated between the glucose measurements that are more than inter_gap minutes apart. The default value is 45 min.
tz	A character string specifying the time zone to be used. System-specific (see as.POSIXct), but " " is the current time zone, and "GMT" is UTC (Universal Time, Coordinated). Invalid values are most commonly treated as UTC, on some platforms with a warning.

Details

A tibble object with 1 row for each subject, a column for subject id and a column for each SD subtype values is returned.

Missing values will be linearly interpolated when close enough to non-missing values.

1. SDw - vertical within days:

   Calculated by first taking the standard deviation of each day's glucose measurements, then taking the mean of all the standard deviations. That is, for d days we compute $SD_1 ... SD_d$ daily standard deviations and calculate $1/d * \sum[(SD_i)]$

2. SDhhmm - between time points:

   Also known as SDhh:mm. Calculated by taking the mean glucose values at each time point in the grid across days, and taking the standard deviation of those mans. That is, for t time points we compute X_t means for each time point and then compute $SD([X_1, X_2, ... X_t])$.

3. SDwsh - within series:

   Also known as SDws h. Calculated by taking the hour-long intervals starting at every point in the interpolated grid, computing the standard deviation of the points in each hour-long interval, and then finding the mean of those standard deviations. That is, for n time points compute $SD_1 ... SD_n$, where SD_i is the standard deviation of the glucose values $[X_i, X_{i+1}, ... X_{i+k}]$ corresponding to hour-long window starting at observation X_i, the number of observations in the window k depends on CGM meter frequency. Then, take $1/n * \sum[(SD_i)]$.

4. SDdm - horizontal sd:

   Calculated by taking the daily mean glucose values, and then taking the standard deviation of those daily means. That is, for d days we take $X_1 ... X_d$ daily means, and then compute $SD([X_1, X_2, ... X_d])$.

5. SDb - between days, within timepoints:

   Calculated by taking the standard deviation of the glucose values across days for each time point, and then taking the mean of those standard deviations. That is, for t time points take $SD_1 ... SD_t$ standard deviations, and then compute $1/t * \sum[(SD_i)]$

6. SDbdm - between days, within timepoints, corrected for changes in daily means:

   Also known as SDb // dm. Calculated by subtracting the daily mean from each glucose value, then taking the standard deviation of the corrected glucose values across days for each time point, and then taking the mean of those standard deviations. That is, for t time points take $SD_1 ... SD_t$ standard deviations, and then compute $1/t * \sum[(SD_i)]$. where SD_i is the standard deviation of d daily values at the 1st time point, where each value is the dth measurement for the ith time point subtracted by the mean of all glucose values for day d.

Value

A tibble object with a column for id and a column for each of the six SD subtypes.

References

Rodbard (2009) New and Improved Methods to Characterize Glycemic Variability Using Continuous Glucose Monitoring *Diabetes Technology and Therapeutics* **11** .551-565, doi:10.1089/dia.2009.0015.

Examples

```
data(example_data_1_subject)
sd_measures(example_data_1_subject)
```

---

sd_roc *Calculate the standard deviation of the rate of change*

Description

The function sd_roc produces the standard deviation of the rate of change values in a tibble object.

Usage

```
sd_roc(data, timelag = 15, dt0 = NULL, inter_gap = 45, tz = "")
```

Arguments

data	DataFrame object with column names "id", "time", and "gl".
timelag	Integer indicating the time period (# minutes) over which rate of change is calculated. Default is 15, e.g. rate of change is the change in glucose over the past 15 minutes divided by 15.
dt0	The time frequency for interpolation in minutes, the default will match the CGM meter's frequency (e.g. 5 min for Dexcom).
inter_gap	The maximum allowable gap (in minutes) for interpolation. The values will not be interpolated between the glucose measurements that are more than inter_gap minutes apart. The default value is 45 min.
tz	A character string specifying the time zone to be used. System-specific (see as.POSIXct), but " " is the current time zone, and "GMT" is UTC (Universal Time, Coordinated). Invalid values are most commonly treated as UTC, on some platforms with a warning.

Details

A tibble object with one row for each subject, a column for subject id and a column for the standard deviation of the rate of change.

When calculating rate of change, missing values will be linearly interpolated when close enough to non-missing values.

Calculated by taking the standard deviation of all the ROC values for each individual subject. NA rate of change values are omitted from the standard deviation calculation.

Value

A tibble object with two columns: subject id and standard deviation of the rate of change values for each subject.

Author(s)

Elizabeth Chun, David Buchanan

References

Clarke et al. (2009) Statistical Tools to Analyze Continuous Glucose Monitor Data, Diabetes *Diabetes Technology and Therapeutics* **11** S45-S54, doi:10.1089/dia.2008.0138.

Examples

```
data(example_data_1_subject)
sd_roc(example_data_1_subject)
sd_roc(example_data_1_subject, timelag = 10)

data(example_data_5_subject)
sd_roc(example_data_5_subject)
sd_roc(example_data_5_subject, timelag = 10)
```

---

summary_glu *Calculate summary glucose level*

Description

The function ‘summary_glu‘ is a wrapper for the base function ‘summary()‘. Output is a tibble object with subject id and the summary value: Minimum, 1st Quantile, Median, Mean, 3rd Quantile and Max.

Usage

```
summary_glu(data)
```

Arguments

data	DataFrame object with column names "id", "time", and "gl", or numeric vector of glucose values.

Details

A tibble object with 1 row for each subject, a column for subject id and a column for each of summary values is returned. ‘NA‘ glucose values are omitted from the calculation of the summary values.

Value

If a DataFrame object is passed, then a tibble object with a column for subject id and then a column for each summary value is returned. If a vector of glucose values is passed, then a tibble object without the subject id is returned. ‘as.numeric()‘ can be wrapped around the latter to output a numeric vector with values in order of Min, 1st Quantile, Median, Mean, 3rd Quantile and Max.

Examples

```
data(example_data_1_subject)
summary_glu(example_data_1_subject)

data(example_data_5_subject)
summary_glu(example_data_5_subject)
```

Index