

Package: scicalc (via r-universe)

September 18, 2026

Title Scientific Calculations for Quantitative Clinical Pharmacology
and Pharmacometrics Analysis

Version 0.4.2

Description Utility functions helpful for reproducible scientific
calculations.

License MIT + file LICENSE

Encoding UTF-8

Roxygen list(markdown = TRUE)

Depends R ($\geq 4.1.0$)

Imports arrow, checkmate, cli, digest, dplyr, fs, haven, lifecycle,
readr, readxl, rlang, stats, stringr, tibble, tidyr, tools

Suggests knitr, rmarkdown, testthat ($\geq 3.0.0$), ggplot2, here, purrr,
pzfx, withr

Config/testthat/edition 3

VignetteBuilder knitr

URL <https://a2-ai.github.io/scicalc-docs>,
<https://github.com/a2-ai/scicalc>

Config/roxygen2/version 8.0.0

Repository <https://a2-ai.r-universe.dev>

Date/Publication 2026-09-18 14:39:21 UTC

RemoteUrl <https://github.com/a2-ai/scicalc>

RemoteRef HEAD

RemoteSha 273e06e0d2ffabd2e6a5f9d1d52aa17ae2647032

Contents

aegfr	3
agec	4
aibw	5

bmi	6
bmic	7
bsa	8
categorize	9
check_for_unique_units	10
convert_alb	11
convert_bili	11
convert_creat	12
cor_df	13
crcl	14
cv	15
egfr	16
ethnicn	17
geom_cv	18
geom_mean	19
geom_sd	19
get_unique_units_df	20
hfc	21
ibw	22
is_american_native	24
is_asian	24
is_black	25
is_female	26
is_hispanic_or_latino	26
is_missing_value	27
is_not_hispanic_or_latino	28
is_other	28
is_pacific_islander	29
is_unspecified	30
is_white	31
racen	31
read_csv_with_hash	32
read_excel_with_hash	33
read_file_with_hash	34
read_hashed_file	35
read_parquet_with_hash	36
read_pzfx_with_hash	36
read_sas_with_hash	37
read_xpt_with_hash	38
rfc	38
round_like	41
sexf	41
write_file_with_hash	42

`aegfr`*Calculate Absolute eGFR*

Description

Converts relative eGFR (normalized to 1.73 m²) to absolute eGFR using the patient's actual body surface area.

Usage

```
aegfr(egfr, bsa)
```

Arguments

<code>egfr</code>	estimated glomerular filtration rate (mL/min/1.73 m ²)
<code>bsa</code>	body surface area (m ²)

Details

Absolute eGFR is calculated as:

$$aGFR = \frac{eGFR \cdot BSA}{1.73}$$

where:

- $eGFR$ = relative eGFR (mL/min/1.73m²)
- BSA = body surface area (m²)

Value

Absolute eGFR (mL/min)

See Also

Other renal_function: [crcl\(\)](#), [egfr\(\)](#), [rfc\(\)](#)

Examples

```
aegfr(90, 1.9)

df <- data.frame(
  ID = c(1, 2, 3, 4),
  EGFR = c(80, 95, 70, 60),
  BSA = c(1.60, 1.85, 1.75, 2.00)
)

df <- df |>
  dplyr::group_by(ID) |>
  dplyr::mutate(AEGFR = aegfr(EGFR, BSA))
df
```

agec

*Categorize Age***Description**

Categorizes subjects into FDA-defined age groups for pediatric and geriatric drug development studies.

Usage

```
agec(age)
```

Arguments

age Numeric vector of baseline age in years

Details**FDA age categories:**

- 1: Neonate: 0 to <28 days
- 2: Infant: 28 days to <2 years
- 3: Child: 2 to <12 years
- 4: Adolescent: 12 to <18 years
- 5: Adult: 18 to <65 years
- 6: Elderly Adult: 65 years

Value

Integer vector of age categories (1-6). Returns the value of `getOption("scicalc.missing_value")` (default -999) for missing values. Includes a `category_standard` attribute set to "FDA".

Custom bands

Set `options(scicalc.agec_config = )` to a data frame of custom bands with columns `label`, `min` (lower bound), and `code` to replace the FDA bands entirely. Bands are half-open `[min, next min)` with the top band open-ended; ages below the lowest `min` return the missing value. The `category_standard` attribute is then "custom". For example, `data.frame(label = c("child", "adult", "senior"), min = c(0, 18, 65), code = c(1, 2, 3))`.

References

Pediatric Drug Development: Regulatory Considerations — Complying With the Pediatric Research Equity Act and Qualifying for Pediatric Exclusivity Under the Best Pharmaceuticals for Children Act Guidance for Industry

Guideline for Industry Studies in Support of Special Populations: Geriatrics

Examples

```
age_cat <- agec(24)

df <- data.frame(
  ID = 1:12,
  AGE = c(0.07, 28 / 365, 0.25, 1, 2, 4, 12, 16, 18, 24, 65, 70)
)
dplyr::mutate(df, AGE_C = agec(AGE))
```

aibw

Calculate Adjusted Ideal Body Weight

Description

Calculates adjusted ideal body weight, which accounts for excess body weight in obese patients.

Usage

```
aibw(
  weight,
  height,
  sexf,
  age,
  allow_ibw_lt_intercept = TRUE,
  allow_tbw_lt_ibw = TRUE
)
```

Arguments

<code>weight</code>	baseline weight of subject in kilograms
<code>height</code>	baseline height of subject in centimeters
<code>sexf</code>	0 = male, 1 = female, from <code>sexf()</code>
<code>age</code>	Numeric vector of baseline age in years.
<code>allow_ibw_lt_intercept</code>	logical indicating whether to allow ideal body weight to be lower than intercepts (default TRUE).
<code>allow_tbw_lt_ibw</code>	logical indicating whether to allow adjusted ideal body weight to be less than IBW (default TRUE).

Details

Adjusted ideal body weight is calculated as:

$$AIBW = IBW + 0.4 \cdot (TBW - IBW)$$

where:

- IBW = ideal body weight (kg)
- TBW = total (actual) body weight (kg)

Value

adjusted ideal body weight (kg)

See Also

Other body_composition: [bmi\(\)](#), [bmic\(\)](#), [bsa\(\)](#), [ibw\(\)](#)

Examples

```
df <- data.frame(
  ID = 1:6,
  HEIGHT = c(160, 170, 175, 165, 180, 150),
  WEIGHT = c(53, 71, 78, 55, 72, 43),
  SEX = c(1, 0, 0, 1, 0, 1),
  AGE = c(18, 27, 34, 33, 29, 30)
)
df <- dplyr::mutate(df, AIBW = aibw(WEIGHT, HEIGHT, SEX, AGE))
df
```

bmi

Calculate Body Mass Index

Description

Calculate Body Mass Index

Usage

```
bmi(weight, height)
```

Arguments

weight	weight of subject (kg)
height	height of subject (cm)

Details

BMI is calculated using the formula:

$$BMI = \frac{W}{(H/100)^2}$$

where:

- W = weight (kg)
- H = height (cm)

Value

the BMI value (kg/m^2)

See Also

Other body_composition: [aibw\(\)](#), [bmic\(\)](#), [bsa\(\)](#), [ibw\(\)](#)

Examples

```
b <- bmi(80.56, 167)

df <- data.frame(
  "WT" = c(80.56, 71.53, 81.04, 70.17),
  "HT" = c(167, 161, 163, 164)
)
df <- dplyr::mutate(df, bmi = bmi(WT, HT))
df
```

bmic

*Categorize Body Mass Index***Description**

Categorizes individuals based on their Body Mass Index (BMI) according to standard WHO obesity classification criteria. Validates age to ensure appropriate application of adult BMI categories.

Usage

```
bmic(bmi, age)
```

Arguments

bmi	Numeric vector of baseline Body Mass Index values (kg/m^2)
age	Numeric vector of baseline age in years. the function will issue warnings for individuals under 18 years old, as adult BMI categories may not be appropriate for pediatric populations

Details**BMI Categories:**

- 1: Underweight: BMI $< 18.5 \text{ kg}/\text{m}^2$
- 2: Normal weight: BMI 18.5 to $< 25 \text{ kg}/\text{m}^2$
- 3: Overweight: BMI 25 to $< 30 \text{ kg}/\text{m}^2$
- 4: Obese class 1: BMI 30 to $< 35 \text{ kg}/\text{m}^2$
- 5: Obese class 2: BMI 35 to $< 40 \text{ kg}/\text{m}^2$

- 6: Obese class 3: BMI ≥ 40 kg/m²

Age Considerations: The function will warn if any individuals are under 18 years old, as standard adult BMI categories may not be appropriate for children and adolescents.

Value

Integer vector of obesity categories (1-6). Returns the value of `getOption("scicalc.missing_value")` (default -999) for missing BMI values.

Custom bands

Set `options(scicalc.bmic_config = )` to a data frame of custom bands with columns `label`, `min` (lower bound), and `code` to replace the WHO bands entirely. Bands are half-open `[min, next min)` with the top band open-ended; BMI values below the lowest `min` return the missing value. The `category_standard` attribute is then "custom".

References

World Health Organization. <https://www.who.int/news-room/fact-sheets/detail/obesity-and-overweight>

See Also

[bmi](#) for calculating BMI from weight and height, [agec](#) for age categorization

Other body_composition: [aibw\(\)](#), [bmi\(\)](#), [bsa\(\)](#), [ibw\(\)](#)

Examples

```
patients <- data.frame(
  ID = 1:6,
  AGE = c(25, 45, 17, 55, 30, 65),
  WEIGHT = c(60, 80, 70, 90, 75, 85),
  HEIGHT = c(165, 175, 170, 180, 160, 175)
)

dplyr::mutate(
  patients,
  BMI = bmi(WEIGHT, HEIGHT),
  BMIC = bmic(BMI, AGE)
)
```

b sa

Calculate Body Surface Area

Description

Calculate Body Surface Area

Usage

```
bsa(weight, height, method = "Dubois")
```

Arguments

weight weight of a subject (kg)
height height of a subject (cm)
method equation to use: `bsa(method = "Dubois")` or `bsa(method = "Mosteller")`.

Value

`bsa` (m^2)

See Also

The method implementations: `bsa(method = "Dubois")`, `bsa(method = "Mosteller")`.
 Other body_composition: `aibw()`, `bmi()`, `bmic()`, `ibw()`

Examples

```
bsa(70, 170)
bsa(70, 170, method = "Mosteller")
bsa(70, 170, method = "Dubois")
```

categorize
Categorize Continuous Variable into Quantile Bins

Description

Categorize Continuous Variable into Quantile Bins

Usage

```
categorize(continuous_var, nbins = 4, units = "", type = 7, digits = 1)
```

Arguments

continuous_var continuous variable data
nbins number of bins to break data into, default is 4
units string, optional units string to add to labels of categorized data
type type argument for `stats::quantile`, default is 7
digits number of digits to round quantile breaks to for labels, default is 1

Value

a vector of categorized data as factor

Examples

```
x <- rnorm(1000, mean = 10, sd = 5)
categorize(x, nbins = 5)
```

```
check_for_unique_units
```

Check for Unique Units per Parameter

Description

Check for Unique Units per Parameter

Usage

```
check_for_unique_units(params, units)
```

Arguments

<code>params</code>	a column from a dataset with lab parameters
<code>units</code>	a column from a dataset with units associated with those parameters

Value

a boolean

See Also

Other unit_checking: [get_unique_units_df\(\)](#)

Examples

```
df <- data.frame(
  PARAM = c(
    "ALB", "ALT", "AST", "CR", "TBIL",
    "ALB", "CR", "TBIL", "ALT", "AST"),
  UNIT = c(
    "g/L", "U/L", "U/L", "umol/L", "umol/L",
    "U/L", " mol/L", " mol/L", "IU/L", "IU/L")
)
check_for_unique_units(df$PARAM, df$UNIT)
```

convert_alb	<i>Convert Albumin Concentration Units</i>
-------------	--

Description

Convert Albumin Concentration Units

Usage

```
convert_alb(alb)
```

Arguments

alb albumin concentration (g/L)

Value

Albumin concentration (g/dL)

See Also

Other unit_conversion: [convert_bili\(\)](#), [convert_creat\(\)](#)

Examples

```
convert_alb(40)

df <- data.frame(
  ID = c(1, 2, 3, 4),
  ALB = c(35, 40, 28, 45)
)

df <- df |>
  dplyr::group_by(ID) |>
  dplyr::mutate(ALBBL = convert_alb(ALB))
df
```

convert_bili	<i>Convert Bilirubin Concentration Units</i>
--------------	--

Description

Convert Bilirubin Concentration Units

Usage

```
convert_bili(bili)
```

Arguments

`bili` bilirubin concentration ($\mu\text{mol/L}$)

Value

Bilirubin concentration (mg/dL)

See Also

Other unit_conversion: [convert_alb\(\)](#), [convert_creat\(\)](#)

Examples

```
convert_bili(17.1) # 1 mg/dL

df <- data.frame(
  ID = c(1, 2, 3, 4),
  BILI = c(10, 15, 25, 40)
)

df <- df |>
  dplyr::group_by(ID) |>
  dplyr::mutate(BILIBL = convert_bili(BILI))
df
```

convert_creat	<i>Convert Serum Creatinine Concentration Units</i>
---------------	---

Description

Convert Serum Creatinine Concentration Units

Usage

```
convert_creat(creat)
```

Arguments

`creat` serum creatinine concentration ($\mu\text{mol/L}$)

Value

Serum Creatinine concentration (mg/dL)

See Also

Other unit_conversion: [convert_alb\(\)](#), [convert_bili\(\)](#)

Examples

```
convert_creat(88.42) # 1 mg/dL

df <- data.frame(
  ID = c(1, 2, 3, 4),
  CREAT = c(70, 90, 110, 130)
)

df <- df |>
  dplyr::group_by(ID) |>
  dplyr::mutate(CREATBL = convert_creat(CREAT))
df
```

cor_df

Convert Data Frame to Correlation Matrix

Description

Computes pairwise correlations between numeric columns and returns results in a tidy long format, sorted by absolute correlation.

Usage

```
cor_df(data, columns = NULL, use = "complete.obs", method = "pearson")
```

Arguments

data	A data.frame containing the variables to correlate
columns	Character vector of column names to include. If NULL, all numeric columns will be used.
use	Method for handling missing values, passed to cor(). Default is "complete.obs".
method	Correlation method, passed to cor(). Default is "pearson".

Value

A tibble with columns:

name1, name2 Variable pair names (lexicographically ordered)

CORR Correlation coefficient

ABSCORR Absolute correlation coefficient

Results are sorted by ABSCORR in descending order.

See Also

Other statistics: [cv\(\)](#), [geom_cv\(\)](#), [geom_mean\(\)](#), [geom_sd\(\)](#)

Examples

```
# Create sample data
set.seed(123)
df <- data.frame(
  A = rnorm(100, 5, 2),
  B = rnorm(100, 10, 3),
  C = rnorm(100, 15, 1),
  D = letters[1:100] # non-numeric
)
df$B <- df$A * 0.8 + rnorm(100, 0, 1) # Create some correlation

# All numeric columns
cor_df(df)

# Specific columns
cor_df(df, columns = c("A", "B", "C"))
```

crcl	<i>Calculate Creatinine Clearance Using Cockcroft-Gault Equation</i>
------	--

Description

Calculate Creatinine Clearance Using Cockcroft-Gault Equation

Usage

```
crcl(sexf, age, creat, weight)
```

Arguments

sexf	bool of sex of subject. Female: True, Male: False
age	age of subject (years)
creat	serum creatinine levels (mg/dL)
weight	weight of subject (kg)

Details

The Cockcroft-Gault equation:

$$CrCl = F \cdot \frac{(140 - A) \cdot W}{72 \cdot S_{cr}}$$

where:

- A = age (years)
- W = weight (kg)
- S_{cr} = serum creatinine (mg/dL)
- F = 0.85 (female) or 1 (male)

Value

CrCl (mL/min)

See Also

Other renal_function: [aegfr\(\)](#), [egfr\(\)](#), [rfc\(\)](#)

Examples

```
crcl(FALSE, 20, 10, 70)

df <- data.frame(
  "ID" = c(1, 1, 1, 1, 2, 2, 2, 2),
  "SEX" = c("F", "F", "F", "F", "M", "M", "M", "M"),
  "RACE" = c("WHITE", "WHITE", "WHITE", "WHITE", "BLACK", "BLACK", "BLACK", "BLACK"),
  "AGE" = c(24, 24, 24, 24, 22, 22, 22, 22),
  "CREAT" = c(1, 1, 1, 1, 4, 4, 4, 4),
  "WEIGHT" = c(70, 70, 70, 70, 65, 65, 65, 65)
)

df <- df |>
  dplyr::group_by(ID) |>
  dplyr::mutate(CRCL = crcl(is_female(SEX), AGE, CREAT, WEIGHT))
df
```

cv	<i>Calculate Coefficient of Variation</i>
----	---

Description

Calculate Coefficient of Variation

Usage

```
cv(x, na.rm = FALSE)
```

Arguments

x	Input vector to compute CV for.
na.rm	boolean to remove NA. default is FALSE

Details

The coefficient of variation is calculated as:

$$CV = \frac{\sigma}{\mu}$$

where:

- σ = standard deviation
- μ = mean

Value

CV of x. Standard deviation divided by mean. If you want % you'll need to multiply by 100

See Also

Other statistics: [cor_df\(\)](#), [geom_cv\(\)](#), [geom_mean\(\)](#), [geom_sd\(\)](#)

Examples

```
cv(c(1, 2, 1, 1, 2, 1, 2, 3))
```

egfr

Calculate Estimated Glomerular Filtration Rate

Description

Calculate Estimated Glomerular Filtration Rate

Usage

```
egfr(
  sexf,
  raceb,
  age,
  creat,
  ...,
  cystc = NULL,
  height = NULL,
  method = "CKDEPI 2021"
)
```

Arguments

sexf	a boolean representing if the patient is female.
raceb	a boolean representing if the patient is black.
age	the age of a patient in years.
creat	the serum creatinine levels in mg/dL.
...	reserved; must be empty. Arguments after creat must be named.
cystc	the cystatin C levels in mg/L - required for the "CKDEPI 2021 cystatin" method.
height	the height of a patient in cm - required for the "Schwartz" method.
method	equation to use, one of: egfr(method = "CKDEPI 2009") , egfr(method = "MDRD") , egfr(method = "CKDEPI 2021") , egfr(method = "CKDEPI 2021 cystatin") , egfr(method = "Schwartz") .

Value

the eGFR calculated based on method.

See Also

The method implementations: `egfr(method = "CKDEPI 2009")`, `egfr(method = "MDRD")`, `egfr(method = "CKDEPI 2021")`, `egfr(method = "CKDEPI 2021 cystatin")`, `egfr(method = "Schwartz")`.

Other renal_function: `aegfr()`, `crcl()`, `rfe()`

Examples

```
e <- egfr(TRUE, TRUE, 24, 1, method = "CKDEPI 2009")

df <- data.frame(
  "SEXF" = c(TRUE, FALSE, TRUE, FALSE),
  "RACEB" = c(FALSE, FALSE, TRUE, FALSE),
  "AGE" = c(24, 24, 23, 24),
  "CREAT" = c(1, 1, 2, 1)
)
df <- dplyr::mutate(df, egfr = egfr(SEXF, RACEB, AGE, CREAT, method = "CKDEPI 2009"))
df
```

ethnicn

Convert Ethnicity to Numeric Code

Description

Convert Ethnicity to Numeric Code

Usage

```
ethnicn(ethnicc)
```

Arguments

ethnicc Ethnic character

Value

the standard yspec numeric value for the inputted Ethnic character

See Also

Other demographics: `is_american_native()`, `is_asian()`, `is_black()`, `is_female()`, `is_hispanic_or_latino()`, `is_not_hispanic_or_latino()`, `is_other()`, `is_pacific_islander()`, `is_unspecified()`, `is_white()`, `racen()`, `sexf()`

Examples

```
ethn1cn("HISPANIC OR LATINO") # 1

ethn1cn("NOT HISPANIC OR LATINO") # 0

ethn1cn("UNKNOWN") # default missing value
```

geom_cv*Calculate Geometric Coefficient of Variation*

Description

Calculate Geometric Coefficient of Variation

Usage

```
geom_cv(x, na.rm = FALSE)
```

Arguments

x vector of data you want the geometric CV of.

na.rm boolean to remove NA from vector. Default is FALSE

Details

The geometric coefficient of variation is calculated as:

$$GCV = \sqrt{\exp(\text{Var}[\log(x)]) - 1}$$

Value

the geometric CV of the input vector x

See Also

Other statistics: [cor_df\(\)](#), [cv\(\)](#), [geom_mean\(\)](#), [geom_sd\(\)](#)

Examples

```
geom_cv(c(1, 2, 3, 2, 1))
```

geom_mean	Calculate Geometric Mean
-----------	--------------------------

Description

Calculate Geometric Mean

Usage

```
geom_mean(x, na.rm = FALSE)
```

Arguments

x	vector to compute geometric mean of
na.rm	boolean to remove NA from vector in calculation. Default is False

Details

The geometric mean is calculated as:

$$GM = \exp(\mathbb{E}[\log(x)])$$

Value

geometric mean of input vector x

See Also

Other statistics: [cor_df\(\)](#), [cv\(\)](#), [geom_cv\(\)](#), [geom_sd\(\)](#)

Examples

```
geom_mean(c(1, 2, 3, 2, 1))
```

geom_sd	Calculate Geometric Standard Deviation
---------	--

Description

Calculate Geometric Standard Deviation

Usage

```
geom_sd(x, na.rm = FALSE)
```

Arguments

x The vector of data you want the geometric sd of.

na.rm a boolean to remove NA values. Default is False

Details

The geometric standard deviation is calculated as:

$$GSD = \exp\left(\sqrt{\text{Var}[\log(x)]}\right)$$

Value

the geometric standard deviation of x

See Also

Other statistics: [cor_df\(\)](#), [cv\(\)](#), [geom_cv\(\)](#), [geom_mean\(\)](#)

Examples

```
geom_sd(c(1, 2, 3, 2, 1))
```

`get_unique_units_df` *Get Unique Parameter-Unit Combinations*

Description

Get Unique Parameter-Unit Combinations

Usage

```
get_unique_units_df(params, units)
```

Arguments

params a column from a dataset with lab parameters

units a column from a dataset with units associated with those parameters

Value

a dataframe with distinct units and parameters with IU replaced to U and mu replaced with u

See Also

Other unit_checking: [check_for_unique_units\(\)](#)

Examples

```
df <- data.frame(
  PARAM = c(
    "ALB", "ALT", "AST", "CR", "TBIL",
    "ALB", "CR", "TBIL", "ALT", "AST"),
  UNIT = c(
    "g/L", "U/L", "U/L", "umol/L", "umol/L",
    "U/L", "mol/L", "mol/L", "IU/L", "IU/L")
)
get_unique_units_df(df$PARAM, df$UNIT)
```

hfc

Categorize Hepatic Function

Description

This function categorizes hepatic function impairment using the National Cancer Institute Organ Dysfunction Working Group (NCI-ODWG) criteria. It evaluates aspartate aminotransferase (AST) and bilirubin levels relative to their upper limits of normal to determine hepatic impairment severity. The function handles edge cases where bilirubin values are very close to category boundaries using floating-point tolerant comparisons.

Usage

```
hfc(ast, ulnast, bili, ulnbili)
```

Arguments

ast	Numeric vector of aspartate aminotransferase concentrations (IU/L)
ulnast	Numeric vector of upper limit of normal AST values (IU/L). Typically 33 IU/L for most laboratories
bili	Numeric vector of total bilirubin concentrations (mg/dL)
ulnbili	Numeric vector of upper limit of normal bilirubin values (mg/dL). Typically 1.2 mg/dL for most laboratories

Details

The NCI-ODWG hepatic function categories are defined as:

- 1: Normal: AST ≤ ULN AND bilirubin ≤ ULN
- 2: Mild impairment: AST > ULN OR bilirubin > ULN but ≤ 1.5 × ULN
- 3: Moderate impairment: Bilirubin > 1.5 × ULN but ≤ 3 × ULN
- 4: Severe impairment: Bilirubin > 3 × ULN

Special handling: The function uses `dplyr::near()` for boundary comparisons when bilirubin values are very close to 1.5 × ULN or 3 × ULN to handle floating-point precision issues that can occur with calculated thresholds.

Value

Integer vector of hepatic function categories (1-4). Returns the value of `getOption("scicalc.missing_value")` (default `-999`) for missing values.

References

National Cancer Institute Organ Dysfunction Working Group criteria for hepatic impairment

Examples

```
# Single patient with normal hepatic function
hfc(ast = 15, ulnast = 33, bili = 0.6, ulnbili = 1.2)

# Multiple patients with different impairment levels
hfc(ast = c(25, 45, 30, 20),
     ulnast = c(33, 33, 33, 33),
     bili = c(0.8, 1.0, 2.5, 4.0),
     ulnbili = c(1.2, 1.2, 1.2, 1.2))

# Edge case: bilirubin exactly at boundary
hfc(ast = 25, ulnast = 33, bili = 1.8, ulnbili = 1.2) # 1.8 = 1.5 * 1.2

# Pipeline example with realistic data
library(dplyr)

patients <- data.frame(
  ID = 1:6,
  AST = c(15, 45, 28, 35, 22, 30),
  ULNAST = 33,
  BILI = c(0.8, 1.0, 2.2, 4.5, 1.8, 0.9),
  ULNBILI = 1.2
)

patients |>
  mutate(BHFC = hfc(AST, ULNAST, BILI, ULNBILI))
```

ibw

Calculate Ideal Body Weight

Description

Calculates ideal body weight using the Devine equation. By default, applies intercepts for individuals shorter than 5 feet (152.4 cm).

Usage

```
ibw(height, sexf, age, allow_ibw_lt_intercept = TRUE)
```

Arguments

height	baseline height of subject in centimeters
sexf	0 = male, 1 = female, from sexf()
age	Numeric vector of baseline age in years.
allow_ibw_lt_intercept	logical indicating whether to apply intercepts for heights < 152.4 cm (5 feet). When TRUE (default), ideal weight is set to intercept weight (50 kg for males, 45.5 kg for females)

Details

The Devine equation for ideal body weight:

$$IBW = I + 2.3 \cdot (H_{in} - 60)$$

where:

- $I = 50$ kg (male) or 45.5 kg (female)
- H_{in} = height in inches (60 inches = 5 feet = 152.4 cm)

Value

ideal body weight (kg)

References

The Origin of the "Ideal" Body Weight Equations, by Pai and Paloucek. The annals of Pharmacotherapy 2000 september, volumn 34

See Also

Other body_composition: [aibw\(\)](#), [bmi\(\)](#), [bmic\(\)](#), [bsa\(\)](#)

Examples

```
df <- data.frame(
  ID = 1:6,
  HEIGHT = c(160, 170, 175, 165, 180, 150),
  SEX = c(1, 0, 0, 1, 0, 1),
  AGE = c(18, 27, 34, 33, 29, 30)
)
df <- dplyr::mutate(df, IBW = ibw(HEIGHT, SEX, AGE))
df
```

is_american_native	<i>Check if Race is American Native</i>
--------------------	---

Description

Check if Race is American Native

Usage

```
is_american_native(x)
```

Arguments

x	input character representing race
---	-----------------------------------

Value

boolean representing Race == American Native

See Also

Other demographics: [ethnicon\(\)](#), [is_asian\(\)](#), [is_black\(\)](#), [is_female\(\)](#), [is_hispanic_or_latino\(\)](#), [is_not_hispanic_or_latino\(\)](#), [is_other\(\)](#), [is_pacific_islander\(\)](#), [is_unspecified\(\)](#), [is_white\(\)](#), [racen\(\)](#), [sexf\(\)](#)

Examples

```
is_american_native("AMERICAN INDIAN OR ALASKA NATIVE")

is_american_native("BLACK")

is_american_native(4)
```

is_asian	<i>Check if Race is Asian</i>
----------	-------------------------------

Description

Check if Race is Asian

Usage

```
is_asian(x)
```

Arguments

x	input character representing race
---	-----------------------------------

Value

boolean representing Race == Asian

See Also

Other demographics: [ethnicon\(\)](#), [is_american_native\(\)](#), [is_black\(\)](#), [is_female\(\)](#), [is_hispanic_or_latino\(\)](#), [is_not_hispanic_or_latino\(\)](#), [is_other\(\)](#), [is_pacific_islander\(\)](#), [is_unspecified\(\)](#), [is_white\(\)](#), [racen\(\)](#), [sexf\(\)](#)

Examples

```
is_asian("ASIAN")

is_asian("BLACK")

is_asian(3)
```

is_black	<i>Check if Race is Black</i>
----------	-------------------------------

Description

Check if Race is Black

Usage

```
is_black(x)
```

Arguments

x input character representing race

Value

boolean representing Race == Black

See Also

Other demographics: [ethnicon\(\)](#), [is_american_native\(\)](#), [is_asian\(\)](#), [is_female\(\)](#), [is_hispanic_or_latino\(\)](#), [is_not_hispanic_or_latino\(\)](#), [is_other\(\)](#), [is_pacific_islander\(\)](#), [is_unspecified\(\)](#), [is_white\(\)](#), [racen\(\)](#), [sexf\(\)](#)

Examples

```
is_black("WHITE")

is_black(c("AFRICAN AMERICAN", "BLACK"))

is_black(2)
```

is_female	<i>Check if Sex is Female</i>
-----------	-------------------------------

Description

Check if Sex is Female

Usage

```
is_female(x)
```

Arguments

x input character representing female or male

Value

boolean representing female

See Also

Other demographics: [ethnics\(\)](#), [is_american_native\(\)](#), [is_asian\(\)](#), [is_black\(\)](#), [is_hispanic_or_latino\(\)](#), [is_not_hispanic_or_latino\(\)](#), [is_other\(\)](#), [is_pacific_islander\(\)](#), [is_unspecified\(\)](#), [is_white\(\)](#), [racen\(\)](#), [sexf\(\)](#)

Examples

```
is_female("F")

is_female(c("MALE", "FEMALE"))

is_female(c(1, 0, -999))
```

is_hispanic_or_latino	<i>Check if Ethnicity is Hispanic or Latino</i>
-----------------------	---

Description

Check if Ethnicity is Hispanic or Latino

Usage

```
is_hispanic_or_latino(x)
```

Arguments

x input character representing ethnicity

Value

boolean representing Ethnic == "Hispanic or Latino"

See Also

Other demographics: `ethnicn()`, `is_american_native()`, `is_asian()`, `is_black()`, `is_female()`, `is_not_hispanic_or_latino()`, `is_other()`, `is_pacific_islander()`, `is_unspecified()`, `is_white()`, `racen()`, `sexf()`

Examples

```
is_hispanic_or_latino("HISPANIC OR LATINO")

is_hispanic_or_latino("NOT HISPANIC OR LATINO")

is_hispanic_or_latino("UNKNOWN")

is_hispanic_or_latino(1)
```

is_missing_value	<i>Test for the Missing Value Indicator</i>
------------------	---

Description

Returns TRUE for elements equal to the missing value indicator (`getOption("scicalc.missing_value")`, default -999), e.g. `dplyr::filter(df, !is_missing_value(NTFD))`.

NA elements return FALSE (they are a separate representation of missing, not the sentinel).

Usage

```
is_missing_value(x, missing_value = getOption("scicalc.missing_value", -999))
```

Arguments

x a numeric vector.

missing_value the sentinel to test against. Defaults to `getOption("scicalc.missing_value", -999)`.

Value

a logical vector the same length as **x**.

Examples

```
is_missing_value(c(1, -999, 3))
```

is_not_hispanic_or_latino*Check if Ethnicity is Not Hispanic or Latino*

Description

Check if Ethnicity is Not Hispanic or Latino

Usage

```
is_not_hispanic_or_latino(x)
```

Arguments

x input character representing ethnicity

Value

boolean representing Ethnic == "Not Hispanic or Latino"

See Also

Other demographics: [ethnicsn\(\)](#), [is_american_native\(\)](#), [is_asian\(\)](#), [is_black\(\)](#), [is_female\(\)](#), [is_hispanic_or_latino\(\)](#), [is_other\(\)](#), [is_pacific_islander\(\)](#), [is_unspecified\(\)](#), [is_white\(\)](#), [racen\(\)](#), [sexf\(\)](#)

Examples

```
is_not_hispanic_or_latino("HISPANIC OR LATINO")  
  
is_not_hispanic_or_latino("NOT HISPANIC OR LATINO")  
  
is_not_hispanic_or_latino("UNKNOWN")  
  
is_not_hispanic_or_latino(0)
```

is_other*Check if Race is Other*

Description

Check if Race is Other

Usage

```
is_other(x)
```

Arguments

`x` input character representing race

Value

boolean representing Race == Other

See Also

Other demographics: `ethnicn()`, `is_american_native()`, `is_asian()`, `is_black()`, `is_female()`, `is_hispanic_or_latino()`, `is_not_hispanic_or_latino()`, `is_pacific_islander()`, `is_unspecified()`, `is_white()`, `racen()`, `sexf()`

Examples

```
is_other("OTHER")  
  
is_other("MULTIPLE")  
  
is_other("BLACK")  
  
is_other(6)
```

`is_pacific_islander` *Check if Race is Pacific Islander*

Description

Check if Race is Pacific Islander

Usage

```
is_pacific_islander(x)
```

Arguments

`x` input character representing race

Value

boolean representing Race == Pacific Islander

See Also

Other demographics: `ethnicn()`, `is_american_native()`, `is_asian()`, `is_black()`, `is_female()`, `is_hispanic_or_latino()`, `is_not_hispanic_or_latino()`, `is_other()`, `is_unspecified()`, `is_white()`, `racen()`, `sexf()`

Examples

```
is_pacific_islander("NATIVE HAWAIIAN OR OTHER PACIFIC ISLANDER")

is_pacific_islander("BLACK")

is_pacific_islander(5)
```

is_unspecified	<i>Check if Race is Unspecified</i>
----------------	-------------------------------------

Description

TRUE for a race value that is present but matches none of the known categories (`is_white`, `is_black`, `is_asian`, `is_american_native`, `is_pacific_islander`, `is_other`). "unknown" and NA are treated as missing (not unspecified) and return FALSE. When unspecified values are found a warning is emitted naming the distinct offenders.

Usage

```
is_unspecified(x, known = character(0))
```

Arguments

x	input character representing race
known	optional character vector of additional recognized values to treat as specified (used by <code>racen()</code> to honor <code>scicalc.racen_config</code>).

Value

boolean representing an unrecognized race value

See Also

Other demographics: `ethnicn()`, `is_american_native()`, `is_asian()`, `is_black()`, `is_female()`, `is_hispanic_or_latino()`, `is_not_hispanic_or_latino()`, `is_other()`, `is_pacific_islander()`, `is_white()`, `racen()`, `sexf()`

Examples

```
is_unspecified("ROMAN LATIN")

is_unspecified("WHITE")

is_unspecified("UNKNOWN")
```

is_white	Check if Race is White
----------	------------------------

Description

Check if Race is White

Usage

```
is_white(x)
```

Arguments

x input character representing race

Value

boolean representing Race == White

See Also

Other demographics: [ethnicon\(\)](#), [is_american_native\(\)](#), [is_asian\(\)](#), [is_black\(\)](#), [is_female\(\)](#), [is_hispanic_or_latino\(\)](#), [is_not_hispanic_or_latino\(\)](#), [is_other\(\)](#), [is_pacific_islander\(\)](#), [is_unspecified\(\)](#), [racen\(\)](#), [sexf\(\)](#)

Examples

```
is_white("WHITE")

is_white("BLACK")

is_white(1)
```

racen	Convert Race to Numeric Code
-------	------------------------------

Description

Convert Race to Numeric Code

Usage

```
racen(racec)
```

Arguments

racec Race character

Details

Default mapping: White = 1, Black = 2, Asian = 3, American Native = 4, Pacific Islander = 5, Other = 6. "UNKNOWN" and NA map to the missing value indicator (`getOption("scicalc.missing_value", -999)`). Any other value is treated as unspecified: it maps to NA and triggers a warning.

The `scicalc.racen_config` option (a named numeric vector) customizes the mapping. A name matching a built-in category (`"white"`, `"black"`, `"asian"`, `"american native"`, `"pacific islander"`, `"other"`) overrides that category's code (synonyms included). Any other name adds a new exact-match category. For example `options(scicalc.racen_config = c("WHITE" = 2, "JAPANESE" = 7))`.

Value

the standard yspec numeric value for the inputted Race character

See Also

Other demographics: `ethnicsn()`, `is_american_native()`, `is_asian()`, `is_black()`, `is_female()`, `is_hispanic_or_latino()`, `is_not_hispanic_or_latino()`, `is_other()`, `is_pacific_islander()`, `is_unspecified()`, `is_white()`, `sexf()`

Examples

```
racen("WHITE") # 1

racen("BLACK") # 2

racen("ASIAN") # 3

racen("AMERICAN INDIAN OR ALASKA NATIVE") # 4

racen("PACIFIC ISLANDER") # 5

racen("OTHER") # 6

racen("UNKNOWN") # default missing value
```

<code>read_csv_with_hash</code>	<i>Read CSV File with Hash Verification</i>
---------------------------------	---

Description

Read CSV File with Hash Verification

Usage

```
read_csv_with_hash(csv_file_path, ..., algo = "blake3")
```

Arguments

`csv_file_path` path to csv file to ingest
`...` additional arguments for `digest` or `read_csv`
`algo` hashing algorithm to use, default is "blake3"

Value

dataframe of data within file

Examples

```
## Not run:
read_csv_with_hash("data/derived/example_data.csv")

## End(Not run)
```

`read_excel_with_hash` *Read Excel File with Hash Verification*

Description

Read Excel File with Hash Verification

Usage

```
read_excel_with_hash(xlsx_file_path, ..., algo = "blake3")
```

Arguments

`xlsx_file_path`
 an xlsx/xls file to ingest
`...` additional arguments to `digest` or `read_excel`
`algo` hashing algorithm to use, default is "blake3"

Value

a dataframe(?) of data within file

Examples

```
## Not run:
read_excel_with_hash("data/source/example.xpt")

## End(Not run)
```

`read_file_with_hash` *Read Data File with Hash Verification*

Description

Read Data File with Hash Verification

Usage

```
read_file_with_hash(
  file_path,
  ...,
  algo = "blake3",
  reader = NULL,
  force = FALSE
)
```

Arguments

<code>file_path</code>	path to data file
<code>...</code>	additional arguments to <code>digest</code> , <code>read_csv</code> , <code>read_parquet</code> , <code>read_sas</code> , <code>read_pzfx</code> , <code>read_xpt</code> , or <code>reader</code>
<code>algo</code>	hashing algorithm to use, default is "blake3"
<code>reader</code>	optional function used to read the file, called as <code>reader(file_path, ...)</code> . Required for file extensions this function doesn't know how to read. For a known extension (csv, parquet, sas7bdat, xpt, pzfx, xlsx/xls/xlsm), <code>reader</code> is ignored unless <code>force = TRUE</code> .
<code>force</code>	if TRUE, use <code>reader</code> even for a known extension instead of the built-in reader. Must be FALSE (the default) when <code>reader</code> is not supplied.

Value

data within the supplied file, carrying `file_hash` and `data_hash` attributes

See Also

Other file_io: [read_hashed_file\(\)](#), [write_file_with_hash\(\)](#)

Examples

```
## Not run:
dat <- read_file_with_hash("data/derived/PK_data.parquet")
dat2 <- read_file_with_hash("data/source/data.csv")
dat3 <- read_file_with_hash("data/derived/pk.feather", reader = arrow::read_feather)

## End(Not run)
```

read_hashed_file	<i>Read File with Required Hash Match</i>
------------------	---

Description

Read File with Required Hash Match

Usage

```
read_hashed_file(
  file_path,
  hash,
  ...,
  algo = "blake3",
  reader = NULL,
  force = FALSE
)
```

Arguments

file_path	path to file with data you want to read
hash	hash you expect the file to have
...	additional arguments for digest or read_csv, parquet, sas, or reader
algo	hashing algorithm to use, default is "blake3"
reader	optional function used to read the file, called as reader(file_path, ...) once the hash check passes. Required for file extensions this function doesn't know how to read. For a known extension (csv, parquet, sas7bdat, xpt, pzf, xlsx/xls/xlsm), reader is ignored unless force = TRUE .
force	if TRUE, use reader even for a known extension instead of the built-in reader. Must be FALSE (the default) when reader is not supplied.

Value

data object of contents of file_path

See Also

Other file_io: [read_file_with_hash\(\)](#), [write_file_with_hash\(\)](#)

Examples

```
## Not run:
file_path <- "data/derived/example_pk.parquet"

hash <- 0cfd6da55e6c1e198effe1e584c26d79
read_hashed_file(file_path, hash)

## End(Not run)
```

```
read_parquet_with_hash
```

Read Parquet File with Hash Verification

Description

Read Parquet File with Hash Verification

Usage

```
read_parquet_with_hash(parquet_file_path, ..., algo = "blake3")
```

Arguments

```
parquet_file_path    path to parquet file to ingest
...                  additional arguments to digest or read_parquet
algo                  hashing algorithm to use, default is "blake3"
```

Value

a tibble of data within file

Examples

```
## Not run:
read_parquet_with_hash("data/derived/example_data.parquet")

## End(Not run)
```

```
read_pzfx_with_hash
```

Read Prism PZFX File with Hash Verification

Description

Read Prism PZFX File with Hash Verification

Usage

```
read_pzfx_with_hash(pzfx_file_path, ..., algo = "blake3")
```

Arguments

```
pzfx_file_path       path to pzfx file
...                  additional arguments to digest or read_pzfx
algo                  hashing algorithm to use, default is "blake3"
```

Value

data within the table of the pzfx file

Examples

```
## Not run:  
read_pzfx_with_hash("mydata.pzfx", table = "experiment1")  
  
## End(Not run)
```

read_sas_with_hash	<i>Read SAS File with Hash Verification</i>
--------------------	---

Description

Read SAS File with Hash Verification

Usage

```
read_sas_with_hash(sas_file_path, ..., algo = "blake3")
```

Arguments

<code>sas_file_path</code>	path to sas file to ingest
<code>...</code>	additional arguments to digest or read_sas
<code>algo</code>	hashing algorithm to use, default is "blake3"

Value

a dataframe(?) of data within file

Examples

```
## Not run:  
read_sas_with_hash("data/source/example.sas7bdat")  
  
## End(Not run)
```

<code>read_xpt_with_hash</code>	<i>Read XPT File with Hash Verification</i>
---------------------------------	---

Description

Read XPT File with Hash Verification

Usage

```
read_xpt_with_hash(xpt_file_path, ..., algo = "blake3")
```

Arguments

<code>xpt_file_path</code>	an xpt file to ingest
<code>...</code>	additional arguments to digest or read_xpt
<code>algo</code>	hashing algorithm to use, default is "blake3"

Value

a dataframe(?) of data within file

Examples

```
## Not run:
read_xpt_with_hash("data/source/example.xpt")

## End(Not run)
```

<code>rfc</code>	<i>Categorize Renal Function</i>
------------------	----------------------------------

Description

This function categorizes renal function based on estimated glomerular filtration rate (eGFR), creatinine clearance, or other renal function estimators. It supports both clinical and regulatory categorization standards and can convert between absolute (mL/min) and relative (mL/min/1.73m²) units using body surface area.

Usage

```
rfc(
  estimator = NULL,
  bsa = NULL,
  category_standard = c("regulatory", "clinical"),
  absolute_units = NULL
)
```

Arguments

estimator	Numeric vector of renal function estimator values (eGFR, CrCL, etc.)
bsa	Numeric vector of body surface area in m ² for unit conversion. Required when converting between absolute and relative units
category_standard	Character string specifying categorization standard: "regulatory" (default) or "clinical"
absolute_units	Logical: are estimator units mL/min (TRUE) or mL/min/1.73m ² (FALSE)? If NULL (default), inferred from the estimator's scicalc_units attribute.

Details

The function applies different categorization schemes based on the **category_standard**:

Regulatory categories (uses mL/min):

- 1: Normal: 90 mL/min
- 2: Mild impairment: 60-89 mL/min
- 3: Moderate impairment: 30-59 mL/min
- 4: Severe impairment: <30 mL/min

Clinical categories (uses mL/min/1.73m²):

- 1: Normal: 90 mL/min/1.73m²
- 2: Mild impairment: 60-89 mL/min/1.73m²
- 3: Moderate impairment: 30-59 mL/min/1.73m²
- 4: Severe impairment: 15-29 mL/min/1.73m²
- 5: End-stage: <15 mL/min/1.73m²

When unit conversion is required, the function uses:

- Absolute to relative: **relative** = 1.73 (**absolute** / **bsa**)
- Relative to absolute: **absolute** = **relative** (**bsa** / 1.73)

Value

Integer vector of renal impairment categories (1-4 for regulatory, 1-5 for clinical). Returns the value of **getOption("scicalc.missing_value")** (default -999) for missing values. Includes a **category_standard** attribute indicating the source ("FDA" or "KDIGO").

References

FDA Guidance for Industry: Pharmacokinetics in Patients with Impaired Renal Function. <https://www.fda.gov/media/78573/download>

KDIGO 2024 Clinical Practice Guideline for the Evaluation and Management of Chronic Kidney Disease. [https://www.kidney-international.org/action/showPdf?pii=S0085-2538\(23\)00766-4](https://www.kidney-international.org/action/showPdf?pii=S0085-2538(23)00766-4)

See Also

[egfr](#) for calculating eGFR, [crcl](#) for creatinine clearance, [bsa](#) for body surface area calculation

Other renal_function: [aegfr\(\)](#), [crcl\(\)](#), [egfr\(\)](#)

Examples

```
# Regulatory categories with absolute units (creatinine clearance)
rfc(estimator = c(95, 75, 45, 25), absolute_units = TRUE)

# Clinical categories with relative units (eGFR)
rfc(
  estimator = c(95, 75, 45, 25, 10),
  absolute_units = FALSE,
  category_standard = "clinical"
)

# Convert relative eGFR to regulatory categories
rfc(
  estimator = 65,
  absolute_units = FALSE,
  bsa = 1.8
)

# Pipeline example with realistic data
df <- data.frame(
  ID = 1:4,
  SEX = c("F", "M", "F", "M"),
  AGE = c(65, 45, 70, 50),
  CREAT = c(1.2, 0.9, 1.5, 1.1),
  WEIGHT = c(70, 80, 60, 85),
  HEIGHT = c(165, 175, 160, 180),
  RACE = c("WHITE", "BLACK", "OTHER", "ASIAN")
)

library(dplyr)
df <- df |>
  mutate(
    BSA = bsa(WEIGHT, HEIGHT, method = "Dubois"),
    EGFR = egfr(is_female(SEX), is_black(RACE), AGE, CREAT),
    AEGFR = aegfr(EGFR, BSA),
    # Clinical categories using relative eGFR directly
    BRFC_CLINICAL = rfc(EGFR, category_standard = "clinical"),
    # Regulatory categories - convert relative eGFR to absolute
    BRFC_REGULATORY_REL = rfc(EGFR, BSA),
    # Regulatory categories - AEGFR already absolute
    BRFC_REGULATORY_ABS = rfc(AEGFR)
  )
df
```

round_like	<i>Round to Significant Digits</i>
------------	------------------------------------

Description

Priority:

1. If `sdig` is provided, round `x` to that many significant digits.
2. If `sdig` is NULL but `ref` is provided, infer the number of significant digits from `ref`.
3. If both are NULL, return `x` unchanged.

Usage

```
round_like(x, ref = NULL, sdig = NULL)
```

Arguments

<code>x</code>	Numeric vector to round.
<code>ref</code>	Optional numeric vector used to infer the number of significant digits.
<code>sdig</code>	Optional integer specifying the number of significant digits to apply.

Value

A numeric vector rounded to the appropriate precision.

Examples

```
round_like(123.456, sdig = 2)      # -> 120
round_like(123.456, ref = 12.3)    # -> 123
round_like(123.456, ref = 12.34)   # -> 123.5
round_like(123.456)                # -> 123.456
```

sexf	<i>Convert Sex to Numeric Code</i>
------	------------------------------------

Description

Also returns numeric for single character Sex characters "F" and "M"

Usage

```
sexf(sex)
```

Arguments

<code>sex</code>	Sex character
------------------	---------------

Value

the standard yspec numeric value for the inputted Sex character

See Also

Other demographics: `ethnicn()`, `is_american_native()`, `is_asian()`, `is_black()`, `is_female()`, `is_hispanic_or_latino()`, `is_not_hispanic_or_latino()`, `is_other()`, `is_pacific_islander()`, `is_unspecified()`, `is_white()`, `racen()`

Examples

```
sexf("FEMALE") # 1
sexf("female") # 1
sexf("f") # 1

sexf("MALE") # 0

sexf("NOT SPECIFIED") # 0
```

`write_file_with_hash` *Write Data File with Hash Output*

Description

Write Data File with Hash Output

Usage

```
write_file_with_hash(
  data,
  path,
  overwrite = FALSE,
  ...,
  algo = "blake3",
  writer = NULL,
  force = FALSE
)
```

Arguments

<code>data</code>	the data object to write to file
<code>path</code>	the destination of the file (csv or parquet)
<code>overwrite</code>	boolean of whether to overwrite or not.
<code>...</code>	additional arguments for <code>digest</code> , <code>write_csv</code> , <code>write_parquet</code> , or <code>writer</code> .
<code>algo</code>	hashing algorithm to use, default is "blake3"

writer	optional function used to write the file, called as <code>writer(data, path, ...)</code> . Required for file extensions this function doesn't know how to write. For a known extension (csv, parquet), writer is ignored unless force = TRUE.
force	if TRUE, use writer even for a known extension instead of the built-in writer. Must be FALSE (the default) when writer is not supplied.

Value

Nothing, File is created and hash of created file is printed

See Also

Other file_io: [read_file_with_hash\(\)](#), [read_hashed_file\(\)](#)

Examples

```
## Not run:
df <- data.frame(
  "a" = c(1, 2, 3, 4)
  "b" = c("A", "B", "C", "D")
)
write_file_with_hash(df, "data.csv")
write_file_with_hash(df, "data.rds", writer = saveRDS)

## End(Not run)
```

Index

- * **body_composition**
 - aibw, [5](#)
 - bmi, [6](#)
 - bmic, [7](#)
 - bsa, [8](#)
 - ibw, [22](#)
- * **categorization**
 - agec, [4](#)
- * **demographics**
 - ethnrcn, [17](#)
 - is_american_native, [24](#)
 - is_asian, [24](#)
 - is_black, [25](#)
 - is_female, [26](#)
 - is_hispanic_or_latino, [26](#)
 - is_not_hispanic_or_latino, [28](#)
 - is_other, [28](#)
 - is_pacific_islander, [29](#)
 - is_unspecified, [30](#)
 - is_white, [31](#)
 - racen, [31](#)
 - sexf, [41](#)
- * **file_io**
 - read_file_with_hash, [34](#)
 - read_hashed_file, [35](#)
 - write_file_with_hash, [42](#)
- * **hepatic_function**
 - hfc, [21](#)
- * **renal_function**
 - aegfr, [3](#)
 - crcl, [14](#)
 - egfr, [16](#)
 - rfc, [38](#)
- * **statistics**
 - cor_df, [13](#)
 - cv, [15](#)
 - geom_cv, [18](#)
 - geom_mean, [19](#)
 - geom_sd, [19](#)
- * **unit_checking**
 - check_for_unique_units, [10](#)
 - get_unique_units_df, [20](#)
- * **unit_conversion**
 - convert_alb, [11](#)
 - convert_bili, [11](#)
 - convert_creat, [12](#)
- aegfr, [3](#)
- aegfr(), [15](#), [17](#), [40](#)
- agec, [4](#), [8](#)
- aibw, [5](#)
- aibw(), [7–9](#), [23](#)
- bmi, [6](#), [8](#)
- bmi(), [6](#), [8](#), [9](#), [23](#)
- bmic, [7](#)
- bmic(), [6](#), [7](#), [9](#), [23](#)
- bsa, [8](#), [40](#)
- bsa(), [6–8](#), [23](#)
- bsa(method = Dubois), [9](#)
- bsa(method = Mosteller), [9](#)
- categorize, [9](#)
- check_for_unique_units, [10](#)
- check_for_unique_units(), [20](#)
- convert_alb, [11](#)
- convert_alb(), [12](#)
- convert_bili, [11](#)
- convert_bili(), [11](#), [12](#)
- convert_creat, [12](#)
- convert_creat(), [11](#), [12](#)
- cor_df, [13](#)
- cor_df(), [16](#), [18–20](#)
- crcl, [14](#), [40](#)
- crcl(), [3](#), [17](#), [40](#)
- cv, [15](#)
- cv(), [13](#), [18–20](#)
- egfr, [16](#), [40](#)

egfr(), [3](#), [15](#), [40](#)
 egfr(method = CKDEPI 2009), [16](#), [17](#)
 egfr(method = CKDEPI 2021 cystatin),
 [16](#), [17](#)
 egfr(method = CKDEPI 2021), [16](#), [17](#)
 egfr(method = MDRD), [16](#), [17](#)
 egfr(method = Schwartz), [16](#), [17](#)
 ethnich, [17](#)
 ethnich(), [24–32](#), [42](#)

 geom_cv, [18](#)
 geom_cv(), [13](#), [16](#), [19](#), [20](#)
 geom_mean, [19](#)
 geom_mean(), [13](#), [16](#), [18](#), [20](#)
 geom_sd, [19](#)
 geom_sd(), [13](#), [16](#), [18](#), [19](#)
 get_unique_units_df, [20](#)
 get_unique_units_df(), [10](#)

 hfc, [21](#)

 ibw, [22](#)
 ibw(), [6–9](#)
 is_american_native, [24](#)
 is_american_native(), [17](#), [25–32](#), [42](#)
 is_asian, [24](#)
 is_asian(), [17](#), [24–32](#), [42](#)
 is_black, [25](#)
 is_black(), [17](#), [24–32](#), [42](#)
 is_female, [26](#)
 is_female(), [17](#), [24](#), [25](#), [27–32](#), [42](#)
 is_hispanic_or_latino, [26](#)
 is_hispanic_or_latino(), [17](#), [24–26](#),
 [28–32](#), [42](#)
 is_missing_value, [27](#)
 is_not_hispanic_or_latino, [28](#)
 is_not_hispanic_or_latino(), [17](#),
 [24–27](#), [29–32](#), [42](#)
 is_other, [28](#)
 is_other(), [17](#), [24–32](#), [42](#)
 is_pacific_islander, [29](#)
 is_pacific_islander(), [17](#), [24–32](#), [42](#)
 is_unspecified, [30](#)
 is_unspecified(), [17](#), [24–29](#), [31](#), [32](#), [42](#)
 is_white, [31](#)
 is_white(), [17](#), [24–30](#), [32](#), [42](#)

 racen, [31](#)
 racen(), [17](#), [24–31](#), [42](#)

 read_csv_with_hash, [32](#)
 read_excel_with_hash, [33](#)
 read_file_with_hash, [34](#)
 read_file_with_hash(), [35](#), [43](#)
 read_hashed_file, [35](#)
 read_hashed_file(), [34](#), [43](#)
 read_parquet_with_hash, [36](#)
 read_pzfx_with_hash, [36](#)
 read_sas_with_hash, [37](#)
 read_xpt_with_hash, [38](#)
 rfc, [38](#)
 rfc(), [3](#), [15](#), [17](#)
 round_like, [41](#)

 sexf, [41](#)
 sexf(), [17](#), [24–32](#)

 write_file_with_hash, [42](#)
 write_file_with_hash(), [34](#), [35](#)