

Package: certestats (via r-universe)

September 28, 2026

Title A Certe R Package for Statistical Modelling

Version 1.26.0

Description A Certe R Package for early-warning, applying statistical modelling (such as creating machine learning models), QC rules and distribution analysis. This package is part of the 'certedata' universe.

URL <https://certe-medical-epidemiology.github.io/certestats>,
<https://github.com/certe-medical-epidemiology/certestats>

Depends R (>= 4.1.0)

Imports certestyle, AMR (>= 3.0.1), broom (>= 1.0.0), cli (>= 3.6.0),
dials (>= 1.0.0), dplyr (>= 1.1.0), generics (>= 0.1.0),
ggplot2 (>= 3.4.0), hardhat (>= 1.2.0), lubridate (>= 1.9.0),
parsnip (>= 1.0.0), recipes (>= 1.0.0), rsample (>= 1.0.0),
tibble (>= 1.0.0), tidyr (>= 1.0.0), tune (>= 1.0.0), vctrs (>= 0.6.0), workflows (>= 1.0.0), yaml (>= 2.2.0), yardstick (>= 1.0.0)

Suggests censored (>= 0.1.0), certepplot2, kknn, mice (>= 3.7.5), nnet,
plot2, progress, ranger, rlang, rpart, rpart.plot, surveillance
(>= 1.25.0), survival, testthat (>= 2.0.0), xgboost

License GPL-2

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

Config/testthat/edition 2

LazyData true

Config/pak/sysreqs cmake make libicu-dev libuv1-dev libx11-dev

Repository <https://certe-medical-epidemiology.r-universe.dev>

Date/Publication 2026-08-29 15:58:39 UTC

RemoteUrl <https://github.com/certe-medical-epidemiology/certestats>

RemoteRef HEAD

RemoteSha 38fa578ca6b3b8f9d7aacfda7d8db98a69ee1070

Contents

as.binary	2
confusion_matrix	3
detect_biomarker_changes	4
detect_disease_clusters	7
detect_farrington	9
different_means	14
distribution_metrics	14
esbl_tests	17
impute	18
machine_learning	20
math_functions	30
moving_average	32
normality	33
qc_rules	34
regression	36
remove_outliers	37
row_function	38
weighted_mean	38
Index	40

as.binary	<i>Work with Binary Columns</i>
-----------	---------------------------------

Description

Transform [logicals](#) and [numerics](#) to a new class 'binary'. The function [try_binary\(\)](#) only coerces to binary if the input is numeric and consists of only zeroes and ones.

Usage

```
as.binary(x)

is.binary(x)

try_binary(x)
```

Arguments

x value(s) to convert to binary

Examples

```
as.binary(c(TRUE, FALSE))

as.binary(c(1, 0))
as.binary(c(1, 0)) + 3 # not binary anymore

try_binary(c(0, 1))
try_binary(c(2, 3))
try_binary(c("a", "b"))
```

confusion_matrix

Confusion Matrix Metrics and Interpretation

Description

Create a confusion matrix and calculate compatible yardstick metrics. Supports class, class-probability, and numeric metrics. For multiclass classification problems (`n_classes > 2`), this function computes both overall (macro/multiclass) metrics and one-vs-rest (OvR) metrics per class label. It also add a human-readable model interpretation.

Usage

```
confusion_matrix(data, ...)

## Default S3 method:
confusion_matrix(data, truth, estimate, na.rm = getOption("na.rm", FALSE), ...)
```

Arguments

<code>data</code>	Either a <code>data.frame</code> containing the columns specified by the <code>truth</code> and <code>estimate</code> arguments, or a <code>table/matrix</code> where the true class results should be in the columns of the table.
<code>...</code>	Not currently used.
<code>truth</code>	The column identifier for the true class results (that is a factor). This should be an unquoted column name although this argument is passed by expression and supports quasiquotation (you can unquote column names). For <code>_vec()</code> functions, a factor vector.
<code>estimate</code>	The column identifier for the predicted class results (that is also factor). As with <code>truth</code> this can be specified different ways but the primary method is to use an unquoted variable name. For <code>_vec()</code> functions, a factor vector.
<code>na.rm</code>	A logical indicating whether missing values should be removed.

Details

This is a function-agnostic wrapper around yardstick. It automatically discovers metric functions exported by yardstick, filters them by compatibility with the input type (class labels, class probabilities, or numeric regression), and applies all applicable metrics.

For multiclass classification (i.e., where the truth and estimate are factors with more than two levels), both macro/multiclass metrics and per-class OvR versions are computed. Each per-class column treats the class as the positive label and all others as negative.

In classification settings, this includes metrics such as accuracy, precision, recall, specificity, F1, MCC, Kappa, and others, depending on compatibility.

For binary classification or regression, output is unchanged and includes applicable metrics only.

Examples

```
# From aggregated counts
df <- data.frame(name = c("Yes", "No"),
                 Yes = c(123, 26),
                 No  = c(13, 834))
confusion_matrix(df)

# From predictions on known labels
iris |>
  ml_decision_trees(Species, quiet = TRUE) |>
  confusion_matrix()
```

detect_biomarker_changes

Detect Unexpected Changes in Biomarkers

Description

The detect_biomarker_changes function is designed to detect unexpected changes in biomarker values for individual patients based on clinical chemistry data. It will flag cases where patients' biomarker results deviate from defined thresholds or exhibit significant changes within a specified time window.

Usage

```
detect_biomarker_changes(
  df,
  testcode = NULL,
  ...,
  column_date = NULL,
  column_patientid = NULL,
  column_testcode = NULL,
  column_testresult = NULL,
  threshold_min = NULL,
```

```

    threshold_max = NULL,
    window_days = NULL,
    max_delta_absolute = NULL,
    max_delta_relative = NULL,
    direction = "any"
)

```

Arguments

df	A data frame containing clinical chemistry data with columns for patient ID, date, test code, and test result.
testcode	Value of the column containing test codes to filter on.
...	Filter arguments, e.g. testcode == "eGFR"
column_date	Name of the column to use for dates. If left blank, the first date column will be used.
column_patientid	Name of the column to use for patient IDs. If left blank, the first column resembling "patient patid" will be used.
column_testcode	Name of the column containing test codes.
column_testresult	Name of the column containing test results.
threshold_min	Minimum threshold for biomarkers.
threshold_max	Maximum threshold for biomarkers.
window_days	Number of days for the time window to check for changes.
max_delta_absolute	Maximum allowable absolute change in biomarkers.
max_delta_relative	Maximum allowable relative change in biomarkers.
direction	Direction of change to check ("up", "down", or "any").

Details

This whole function, including the documentation, was written by ChatGPT 3.5 in October 2023. Only minor changes were applied manually.

This function is particularly useful in early detection of anomalous biomarker results, which can be indicative of health issues or treatment response. By providing detailed flags, it allows healthcare professionals and researchers to take timely action, conduct further investigations, or make informed clinical decisions.

The output of this function can be utilised for:

- Generating patient-specific reports for healthcare providers.
- Identifying trends and patterns in biomarker changes for research purposes.
- Enhancing patient care by enabling proactive interventions when necessary.
- Supporting data-driven clinical epidemiology studies and research.

The `format()` function allows you to format the results of the `detect_biomarker_changes()` function for better readability and analysis. It organises the flag information into a structured data frame for easier inspection.

Value

A list with the following components:

- `flags`: A list of flags per patient, containing data frames for each patient with details on dates, test codes, test results, and flagging criteria. The structure of each data frame includes the following columns:
 - `patient`: Patient identifier.
 - `date`: Date of the biomarker measurement.
 - `testcode`: Code of the biomarker test.
 - `testresult`: Biomarker test result.
 - `delta_absolute`: Absolute change in biomarker values.
 - `delta_relative`: Relative change in biomarker values.
 - `threshold_min_flag`: A logical flag indicating if the threshold minimum is exceeded.
 - `threshold_max_flag`: A logical flag indicating if the threshold maximum is exceeded.
 - `delta_absolute_flag`: A logical flag indicating if the absolute change exceeds the threshold.
 - `delta_relative_flag`: A logical flag indicating if the relative change exceeds the threshold.
- `details`: A data frame containing all patient details and calculated flags, regardless of whether they meet the flagging criteria. The data frame includes the same columns as the individual patient data frames.

See Also

[detect_disease_clusters\(\)](#)

Examples

```
data <- data.frame(date = Sys.Date() + 1:10,
                  patient = "test",
                  value = c(10,12,14,15,13,21,22,19,14,12))

check <- data |> detect_biomarker_changes(window_days = 6, max_delta_absolute = 10)

check

unlist(check)
```

detect_disease_clusters*Detect Disease Clusters*

Description

Detect disease clusters with `detect_disease_clusters()`. Use `has_clusters()` to return TRUE or FALSE based on its output, or employ `format()` to format the result.

Usage

```
detect_disease_clusters(  
  df,  
  column_date = NULL,  
  column_patientid = NULL,  
  based_on_historic_maximum = FALSE,  
  period_length_months = 12,  
  minimum_cases = 5,  
  minimum_days = 0,  
  minimum_case_days = 2,  
  minimum_case_fraction_in_period = 0.02,  
  threshold_percentile = 97.5,  
  remove_outliers = TRUE,  
  remove_outliers_coefficient = 1.5,  
  moving_average_days = 7,  
  moving_average_side = "left",  
  case_free_days = 14,  
  ...  
)  
  
n_clusters(x)  
  
has_clusters(x, n = 1)  
  
has_ongoing_cluster(x, dates = Sys.Date() - 1)  
  
has_cluster_before(x, date)  
  
has_cluster_after(x, date)
```

Arguments

<code>df</code>	Data set: This must consist of only positive results . The minimal data set should include a date column and a patient column. Do not summarize on patient IDs; this will be handled automatically.
<code>column_date</code>	Name of the column to use for dates. If left blank, the first date column will be used.

column_patientid	Name of the column to use for patient IDs. If left blank, the first column resembling "patient patid" will be used.
based_on_historic_maximum	A logical to indicate whether the cluster detection should be based on the maximum of previous years. The default is FALSE, which uses all historic data points.
period_length_months	Number of months per period.
minimum_cases	Minimum number of <i>cases</i> that a cluster requires to be considered a cluster.
minimum_days	Minimum number of <i>days</i> that a cluster requires to be considered a cluster.
minimum_case_days	Minimum number of <i>days with cases</i> that a cluster requires to be considered a cluster.
minimum_case_fraction_in_period	Minimum fraction of <i>cluster cases in a period</i> that a cluster requires to be considered a cluster.
threshold_percentile	Threshold to set.
remove_outliers	A logical to indicate whether outliers should be removed before determining the threshold.
remove_outliers_coefficient	Coefficient used for outlier determination.
moving_average_days	Number of days to set in moving_average() . Defaults to a whole week (7).
moving_average_side	Side of days to set in moving_average() . Defaults to "left" for retrospective analysis.
case_free_days	Number of days to set in get_episode() .
...	not used at the moment
x	output of detect_disease_clusters()
n	number of clusters, defaults to 1
dates	date(s) to test whether any of the clusters currently has this date in it, defaults to yesterday.
date	date to test whether there are any clusters since or until this date.

Details

A (disease) cluster is defined as an unusually large aggregation of disease events in time or space ([ATSDR, 2008](#)). They are common, particularly in large populations. From a statistical standpoint, it is nearly inevitable that some clusters of chronic diseases will emerge within various communities, be it schools, church groups, social circles, or neighborhoods. Initially, these clusters are often perceived as products of specific, predictable processes rather than random occurrences in a particular location, akin to a coin toss.

Whether a (suspected) cluster corresponds to an actual increase of disease in the area, needs to be assessed by an epidemiologist or biostatistician ([ATSDR, 2008](#)).

The function `has_ongoing_cluster()` returns a **logical** vector with the same length as dates, so dates can have any length.

See Also

`detect_biomarker_changes()`

Examples

```
cases <- data.frame(date = sample(seq(as.Date("2015-01-01"),
                                     as.Date("2022-12-31"),
                                     "1 day"),
                                size = 300),
                    patient = sample(LETTERS, size = 300, replace = TRUE))

# -----

check <- detect_disease_clusters(cases, threshold_percentile = 0.99)

has_clusters(check)
check

check2 <- detect_disease_clusters(cases,
                                minimum_cases = 1,
                                threshold_percentile = 0.75)

check2
check2 |> format()

check2 |> n_clusters()
check2 |> has_clusters()
check2 |> has_clusters(n = 15)

check2 |> has_ongoing_cluster("2022-06-01")
check2 |> has_ongoing_cluster(c("2022-06-01", "2022-06-20"))
check2 |> has_cluster_before("2022-06-01")
check2 |> has_cluster_after("2022-06-01")

check2 |> unclass()

# plot the results
# check2 |> plot2()
```

Description

Detect aberrations (unexpected increases) in surveillance count data using the Farrington algorithm (Farrington et al., 1996) or its improved flexible variant (Noufaily et al., 2012). For time series without sufficient historic baselines, the EARS methods (C1, C2, C3) are available as a fallback via the method argument.

Usage

```
detect_farrington(
  df,
  column_date = NULL,
  column_patientid = NULL,
  population = NULL,
  method = "farrington",
  frequency = 52,
  years_back = 5,
  window_width = 7,
  reweight = TRUE,
  alpha = NULL,
  trend = TRUE,
  population_offset = FALSE,
  n_periods = 1,
  past_periods_ignored = NULL,
  threshold_method = "delta",
  case_free_days = 14,
  minimum_cases = 1,
  minimum_duration = 1,
  range = NULL,
  ...
)

n_farrington_clusters(x)

has_farrington_clusters(x, n = 1)

has_ongoing_farrington_cluster(x, dates = Sys.Date() - 1)

has_farrington_cluster_before(x, date)

has_farrington_cluster_after(x, date)

## S3 method for class 'farrington_clusters'
autoplot(object, ...)
```

Arguments

df Data set. This must consist of **only positive results**. The minimal data set should include a date column and a patient column. Do not summarise on patient IDs;

	deduplication to unique patient-dates is handled automatically.
column_date	Name of the column to use for dates. If left blank, the first date column will be used.
column_patientid	Name of the column to use for patient IDs. If left blank, the first column resembling "patient patid" will be used.
population	A data.frame with columns date and population, giving the denominator over time, such as patient-days per week for a hospital ward. Required when population_offset = TRUE. It is supplied separately from df because periods with zero cases still need a denominator, so it cannot be derived from the line list. Values are carried forward to each epoch of the aggregated time series.
method	Detection method to use. One of: • "farrington" (default): the improved Farrington Flexible method (Noufaily et al., 2012), suitable when ≥ 3 years of historic data are available. • "ears_c1", "ears_c2", "ears_c3": the CDC EARS methods, suitable as a fallback for short time series without years of baseline data.
frequency	Number of observations per year. Use 52 for weekly data (the default) or 12 for monthly data.
years_back	Number of years back in time to include for the baseline (Farrington only). Defaults to 5 for the flexible method.
window_width	Total width of the reference window around the current period in each reference year (Farrington only). The default of 7 means 3 periods before, the current period, and 3 periods after. Internally converted to the half-size $w = \text{round}((\text{window_width} - 1) / 2)$ before passing to surveillance::farringtonFlexible() .
reweight	A logical indicating whether to perform the reweighting step to down-weight past outbreaks (Farrington only). Defaults to TRUE, which is the Noufaily et al. (2012) recommendation.
alpha	Significance level for the one-sided prediction interval. Defaults to 0.05 for Farrington and 0.001 for EARS C1/C2, 0.025 for EARS C3.
trend	A logical indicating whether to include a time trend in the GLM (Farrington only). Defaults to TRUE.
population_offset	A logical indicating whether to include a population offset in the GLM (Farrington only). Defaults to FALSE. If TRUE, population must be provided.
n_periods	Number of reference periods in the factor variable for the baseline (Farrington Flexible only). Defaults to 1, which corresponds to the original Farrington et al. (1996) definition. Setting this to e.g. 10 expands the reference window, which can be useful for large regions with more data.
past_periods_ignored	Number of recent periods to exclude from the baseline to avoid influence of ongoing outbreaks (Farrington only). Defaults to NULL, which uses the value of w. Noufaily et al. (2012) advise 26 for weekly data.
threshold_method	Method to derive the upper bound. One of "delta" (Farrington et al., 1996, the default), "nbPlugin" (Noufaily et al., 2012), or "muan" (extended from Noufaily et al., 2012).

case_free_days	Number of case-free days to separate distinct aberration episodes. Passed to <code>AMR::get_episode()</code> . Defaults to 14.
minimum_cases	Minimum number of cases for an aberration episode to be retained. Defaults to 1.
minimum_duration	Minimum number of days (inclusive) for an aberration episode to be retained. Defaults to 1.
range	Index of timepoints to monitor. If NULL (the default), the last frequency time-points (i.e. the most recent year) are evaluated.
...	Additional arguments passed to <code>surveillance::farringtonFlexible()</code> or <code>surveillance::earsC()</code> .
x	output of <code>detect_farrington()</code>
n	minimum number of clusters, defaults to 1
dates	date(s) to test whether any cluster currently spans this date. Defaults to yesterday. Returns a <code>logical</code> vector with the same length as dates.
date	a single date to test whether there are any clusters before or after this date.
object	output of <code>detect_farrington()</code>

Details

The data are internally converted from a line list (one row per isolate or case) to an aggregated sts object from the **surveillance** package, which is then passed to `surveillance::farringtonFlexible()` or `surveillance::earsC()`.

Use `has_farrington_clusters()` to return TRUE or FALSE based on the output, or employ `format()` to format the result into a summary data frame. Use `autoplot()` for a ggplot2 visualisation, or `plot()` for the base graphics version.

Farrington (recommended):

The Farrington algorithm is the standard method for automated aberration detection in European infectious disease surveillance, used by ECDC, Public Health England, and the Robert Koch Institute (RKI), among others. For each evaluated time point, a quasi-Poisson GLM is fitted to reference counts from the same calendar period in previous years. The predicted count and its overdispersion are used to derive an upper threshold via a variance-stabilising transformation. An aberration is flagged when the observed count exceeds this threshold.

Farrington Flexible:

The improved "Flexible" variant (Noufaily et al., 2012) adds reweighting of past outbreaks so that a historic epidemic does not inflate the baseline and mask future events.

EARS (fallback for short series):

The Early Aberration Reporting System (EARS) methods from the CDC are Shewhart-type control charts that only require counts from the recent past (default: 7 time points). They are useful when insufficient historic data are available for the Farrington approach.

Aberration clusters:

After the surveillance algorithm flags individual time points as aberrations, consecutive (or near-consecutive) aberrations are grouped into clusters using `AMR::get_episode()` with the `case_free_days` parameter. This produces operationally useful clusters with start dates, end dates, case counts, and durations, analogous to `detect_disease_clusters()`.

References

- Farrington CP, Andrews NJ, Beale AD, Catchpole MA (1996). A statistical algorithm for the early detection of outbreaks of infectious disease. *J. R. Statist. Soc. A*, **159**, 547-563.
- Noufaily A, Enki DG, Farrington CP, Garthwaite PH, Andrews NJ, Charlett A (2012). An improved algorithm for outbreak detection in multiple surveillance systems. *Statistics in Medicine*, **32**(7), 1206-1222.
- Salmon M, Schumacher D, Hohle M (2016). Monitoring count time series in R: Aberration detection in public health surveillance. *Journal of Statistical Software*, **70**(10), 1-35.

See Also

[detect_disease_clusters\(\)](#), [surveillance::farringtonFlexible\(\)](#), [surveillance::earsC\(\)](#)

Examples

```
# generate example line list data spanning several years
set.seed(123)
cases <- data.frame(
  date = sample(seq(as.Date("2018-01-01"),
                    as.Date("2024-12-31"),
                    "1 day"),
                size = 500,
                replace = TRUE),
  patient = sample(LETTERS, size = 500, replace = TRUE)
)

# --- Farrington Flexible (default) ---
result <- detect_farrington(cases)
result
has_farrington_clusters(result)
n_farrington_clusters(result)
format(result)

# check for ongoing cluster
has_ongoing_farrington_cluster(result, Sys.Date() - 1)

# plot the results
plot(result)
if (require("ggplot2")) autoplot(result)

# --- EARS C2 (short baseline) ---
recent <- cases[cases$date >= as.Date("2024-06-01"), ]
result_ears <- detect_farrington(recent, method = "ears_c2")
result_ears
if (require("ggplot2")) autoplot(result_ears)

# --- Farrington with expanded reference window for large regions ---
result_large <- detect_farrington(cases, n_periods = 10,
                                past_periods_ignored = 26,
                                threshold_method = "nbPlugin")
result_large
```

```
if (require("ggplot2")) autoplot(result_large)
```

different_means *Different Means*

Description

Functions to determine harmonic and geometric means.

Usage

```
mean_harmonic(x, ..., na.rm = getOption("na.rm", FALSE))
```

```
mean_geometric(x, ..., na.rm = getOption("na.rm", FALSE))
```

Arguments

x	numeric vector
...	arguments passed on to base::mean()
na.rm	ignore empty values

Details

The harmonic mean can be expressed as the reciprocal of the arithmetic mean of the reciprocals.

The geometric mean is defined as the nth root of the product of n numbers.

distribution_metrics *Distribution Metrics*

Description

These are simple distribution metric functions.

Usage

```
se(x, na.rm = getOption("na.rm", FALSE))
```

```
ci(x, level = 0.95, na.rm = getOption("na.rm", FALSE))
```

```
sum_of_squares(x, correct_mean = TRUE, na.rm = getOption("na.rm", FALSE))
```

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

```
cqv(x, na.rm = getOption("na.rm", FALSE))
```

```

mse(actual, predicted, na.rm = getOption("na.rm", FALSE))
mape(actual, predicted, na.rm = getOption("na.rm", FALSE))
rmse(actual, predicted, na.rm = getOption("na.rm", FALSE))
mae(actual, predicted, na.rm = getOption("na.rm", FALSE))
z_score(x, na.rm = getOption("na.rm", FALSE))
midhinge(x, na.rm = getOption("na.rm", FALSE))
ewma(x, lambda, na.rm = getOption("na.rm", FALSE))
rr_ewma(x, lambda, na.rm = getOption("na.rm", FALSE))
normalise(n, n_ref, per = 1000)
normalize(n, n_ref, per = 1000)
scale_sd(x)
centre_mean(x)
percentiles(x, na.rm = getOption("na.rm", FALSE))
deciles(x, na.rm = getOption("na.rm", FALSE))

```

Arguments

x	values
na.rm	a logical to indicate whether empty must be removed from x
level	alpha level, defaults to 95%
correct_mean	with TRUE (the default) correct for the mean will be applied, by summing each square of x after the mean of x has been subtracted, so that this says something about x. With FALSE, all x^2 are simply added together, so this says something about x's location in the data.
actual	Vector of actual values
predicted	Vector of predicted values
lambda	smoothing parameter, a value between 0 and 1. A value of 0 is equal to x, a value of 1 equal to the <i>mean</i> of x. The EWMA looks back and has a delay - the rrEWMA takes the mean of a 'forward' and 'backward' EWMA.
n	number to be normalised
n_ref	reference to use for normalisation
per	normalisation factor

Details

These are the explanations of the functions:

- `se()` calculates the standard error: $sd / \text{square root of length}$
- `ci()` calculates the confidence intervals for a mean (defaults at 95%), which returns length 2
- `sum_of_squares()` calculates the sum of $(x - \text{mean}(x))^2$
- `cv()` calculates the coefficient of variation: $\text{standard deviation} / \text{mean}$
- `cqv()` calculates the coefficient of quartile variation: $(Q3 - Q1) / (Q3 + Q1)$
- `mse()` calculates the mean squared error
- `mape()` calculates the mean absolute percentage error
- `rmse()` calculates the root mean squared error
- `mae()` calculates the mean absolute error
- `z_score()` calculates the number of standard deviations from the mean: $(x - \text{mean}) / sd$
- `midhinge()` calculates the mean of interquartile range: $(Q1 + Q3) / 2$
- `ewma()` calculates the EWMA (exponentially weighted moving average)
- `rr_ewma()` calculates the rrEWMA (reversed-recombined exponentially weighted moving average)
- `normalise()` normalises the data based on a reference: $(n / \text{reference}) * \text{unit}$
- `scale_sd()` normalises the data to have a standard deviation of 1, while retaining the mean
- `centre_mean()` normalises the data to have a mean of 0, while retaining the standard deviation
- `percentiles()` and `deciles()` take a numeric vector as input, and return the lowest **percentiles** or **deciles** for each value

Default values of `na.rm`

This 'certestats' package supports a global default setting for `na.rm` in many mathematical functions. This can be set with `options(na.rm = TRUE)` or `options(na.rm = FALSE)`.

For `normality()`, `quantile()` and `IQR()`, this also applies to the `type` argument. The default, `type = 7`, is the default of base R. Use `type = 6` to comply with SPSS.

See Also

For the sum of squares: <https://www.thoughtco.com/sum-of-squares-formula-shortcut-3126266>

Examples

```
x <- c(0, 1, 2, 3, 4, 5, 5, 5, 5, 5, 5, 6)
percentiles(x)
deciles(x)

percentiles(rnorm(10))

library(dplyr, warn.conflicts = FALSE)
tib <- as_tibble(matrix(as.integer(runif(40, min = 1, max = 7)), ncol = 4),
                      .name_repair = function(...) LETTERS[1:4])
tib

# percentiles per column
tib |> mutate_all(percentiles)
```

esbl_tests

*Example Data Set with ESBL Test Outcomes***Description**

This data set contains phenotypic test outcomes of the presence of ESBL (Extended Spectrum Beta-Lactamase) genes, with the minimum inhibitory concentrations (MIC, in mg/L) for 17 antibiotic drugs.

Usage

```
esbl_tests
```

Format

A [tibble::tibble/data.frame](#) with 500 observations and 19 variables:

- esbl: Outcome of ESBL test (TRUE (n = 250), FALSE (n = 250))
- genus: Taxonomic genus of the bacteria (*Citrobacter* (n = 42), *Enterobacter* (n = 30), *Escherichia* (n = 301), *Klebsiella* (n = 70), *Morganella* (n = 24), *Proteus* (n = 33))
- AMC: MIC values of amoxicillin/clavulanic acid
- AMP: MIC values of ampicillin
- TZP: MIC values of piperacillin/tazobactam
- CXM: MIC values of cefuroxime
- FOX: MIC values of ceftaxime
- CTX: MIC values of cefotaxime
- CAZ: MIC values of ceftazidime
- GEN: MIC values of gentamicin
- TOB: MIC values of tobramycin
- TMP: MIC values of trimethoprim

- SXT: MIC values of trimethoprim/sulfamethoxazole
- NIT: MIC values of nitrofurantoin
- FOS: MIC values of fosfomycin
- CIP: MIC values of ciprofloxacin
- IPM: MIC values of imipenem
- MEM: MIC values of meropenem
- COL: MIC values of colistin

Examples

```
print(esbl_tests, n = 5)
```

impute	<i>Impute: Filling Missing Values</i>
--------	---------------------------------------

Description

Imputation is the process of replacing missing data with substituted values. This is done because of three main problems that missing data causes: missing data can introduce a substantial amount of bias, make the handling and analysis of the data more arduous, and create reductions in efficiency.

Usage

```
impute(
  .data,
  vars = everything(),
  algorithm = "mice",
  m = 10,
  method = NULL,
  FUN = median,
  info = TRUE,
  ...
)

is_imputed(.data)

get_mice(.data)
```

Arguments

.data	data set with missing values to impute
vars	variables of .data that must be imputed, defaults to everything() and supports the tidyselect language.
algorithm	algorithm to use for imputation, must be "mice" or "single-point", see <i>Details</i> . For the latter, FUN must be given.

<code>m</code>	number of multiple imputations if using MICE, see <code>mice::mice()</code> . The mean of all imputations will be used as result.
<code>method</code>	method to use if using MICE, see <code>mice::mice()</code>
<code>FUN</code>	function to use for single-point imputation (directly) or for MICE to summarise the results over all <code>m</code> iterations
<code>info</code>	print info about imputation
<code>...</code>	arguments to pass on to <code>mice::mice()</code>

Details

Imputation can be done using single-point, such as the mean or the median, or using [Multivariate Imputations by Chained Equations \(MICE\)](#). Using MICE is a lot more reliable, but also a lot slower, than single-point imputation.

The suggested and default method is MICE. The generated MICE object will be stored as an [attribute](#) with the data, and can be retrieved with `get_mice()`, containing all specifics about the imputation. MICE is also known as *fully conditional specification* and *sequential regression multiple imputation*. It was designed for data with randomly missing values, though there is simulation evidence to suggest that with a sufficient number of auxiliary variables it can also work on data that are missing not at random.

Use `is_imputed()` to get a [data.frame](#) with TRUEs for all values that were imputed.

Examples

```
iris2 <- dplyr::as_tibble(iris)
iris2[2, 2] <- NA
iris2[3, 3] <- NA
iris2[4, 5] <- NA
iris
iris2

result <- iris2 |> impute()
result

iris2 |> impute(algorithm = "single-point")
iris2 |>
  impute(vars = starts_with("Sepal"),
        algorithm = "single-point")
iris2 |>
  impute(vars = where(is.double),
        algorithm = "single-point",
        FUN = median)

result |> is_imputed()
result |> get_mice()
```

Description

These functions create machine learning models using different parsnip model specifications and engines, and provide a unified interface for training, evaluating, tuning, and applying those models.

The `ml_*()` functions are structured wrappers around the tidymodels ecosystem, most notably [parsnip](#), [recipes](#), [rsample](#), [tune](#), and [yardstick](#).

The goal is to reduce boilerplate while preserving access to the underlying modeling assumptions, parameters, and diagnostics.

Usage

```
ml_xg_boost(  
  .data,  
  outcome,  
  predictors = everything(),  
  training_fraction = 0.75,  
  strata = NULL,  
  na_threshold = 0.01,  
  correlation_threshold = 0.9,  
  centre = FALSE,  
  scale = FALSE,  
  engine = "xgboost",  
  mode = c("classification", "regression", "unknown"),  
  trees = 15,  
  ...,  
  quiet = FALSE  
)
```

```
ml_decision_trees(  
  .data,  
  outcome,  
  predictors = everything(),  
  training_fraction = 0.75,  
  strata = NULL,  
  na_threshold = 0.01,  
  correlation_threshold = 0.9,  
  centre = FALSE,  
  scale = FALSE,  
  engine = "rpart",  
  mode = c("classification", "regression", "unknown"),  
  tree_depth = 30,  
  ...,  
  quiet = FALSE
```

```
)

ml_random_forest(
  .data,
  outcome,
  predictors = everything(),
  training_fraction = 0.75,
  strata = NULL,
  na_threshold = 0.01,
  correlation_threshold = 0.9,
  centre = FALSE,
  scale = FALSE,
  engine = "ranger",
  mode = c("classification", "regression", "unknown"),
  trees = 500,
  ...,
  quiet = FALSE
)

ml_neural_network(
  .data,
  outcome,
  predictors = everything(),
  training_fraction = 0.75,
  strata = NULL,
  na_threshold = 0.01,
  correlation_threshold = 0.9,
  centre = TRUE,
  scale = TRUE,
  engine = "nnet",
  mode = c("classification", "regression", "unknown"),
  penalty = 0,
  epochs = 100,
  ...,
  quiet = FALSE
)

ml_nearest_neighbour(
  .data,
  outcome,
  predictors = everything(),
  training_fraction = 0.75,
  strata = NULL,
  na_threshold = 0.01,
  correlation_threshold = 0.9,
  centre = TRUE,
  scale = TRUE,
  engine = "kknn",
```

```
mode = c("classification", "regression", "unknown"),
neighbors = 5,
weight_func = "triangular",
...,
quiet = FALSE
)

ml_linear_regression(
  .data,
  outcome,
  predictors = everything(),
  training_fraction = 0.75,
  strata = NULL,
  na_threshold = 0.01,
  correlation_threshold = 0.9,
  centre = TRUE,
  scale = TRUE,
  engine = "lm",
  mode = "regression",
  ...,
  quiet = FALSE
)

ml_logistic_regression(
  .data,
  outcome,
  predictors = everything(),
  training_fraction = 0.75,
  strata = NULL,
  na_threshold = 0.01,
  correlation_threshold = 0.9,
  centre = TRUE,
  scale = TRUE,
  engine = "glm",
  mode = "classification",
  penalty = 0.1,
  ...,
  quiet = FALSE
)

## S3 method for class 'certestats_ml'
confusion_matrix(data, ...)

## S3 method for class 'certestats_ml'
predict(object, new_data, type = NULL, ...)

apply_model_to(
  object,
```

```
        new_data,
        only_prediction = FALSE,
        only_certainty = FALSE,
        correct_mistakes = TRUE,
        impute_algorithm = "mice",
        ...
    )

feature_importances(object, ...)

feature_importance_plot(object, ...)

roc_plot(object, ...)

gain_plot(object, ...)

tree_plot(object, ...)

correlation_plot(
  data,
  add_values = TRUE,
  cols = everything(),
  correlation_threshold = 0.9
)

get_metrics(object)

get_accuracy(object)

get_kappa(object)

get_recipe(object)

get_specification(object)

get_rows_testing(object)

get_rows_training(object)

get_original_data(object)

get_roc_data(object)

get_coefficients(object)

get_model_variables(object)

get_variable_weights(object)
```

```

tune_parameters(object, ..., only_params_in_model = FALSE, levels = 5, k = 10)

check_testing_predictions(object)

## S3 method for class 'certestats_ml'
autoplot(object, plot_type = "roc", ...)

## S3 method for class 'certestats_feature_importances'
autoplot(object, ...)

## S3 method for class 'certestats_tuning'
autoplot(object, type = c("marginals", "parameters", "performance"), ...)

```

Arguments

<code>.data</code>	A data set used for model training and internal testing. The data will be split into training and testing subsets using <code>rsample::initial_split()</code> .
<code>outcome</code>	Outcome variable, also called the <i>response</i> or <i>dependent</i> variable; the variable to be predicted. The value is evaluated using <code>dplyr::select()</code> and therefore supports the tidyselect language. For classification models, the outcome will be coerced to a <code>factor</code> if it is not already a factor or character.
<code>predictors</code>	Explanatory variables, also called the <i>predictors</i> or <i>independent variables</i>; the variables used to predict outcome. This argument supports the tidyselect language and defaults to <code>everything()</code>. Predictors are processed via a <code>recipes::recipe()</code>, including dummy encoding for nominal variables and optional centring, scaling, and correlation filtering.
<code>training_fraction</code>	Fraction of rows to be used for <i>training</i>, defaults to 0.75. The remaining rows are used for <i>testing</i>. If a value greater than 1 is supplied, it is interpreted as the absolute number of rows to include in the training set. When strata is supplied, the split is stratified accordingly.
<code>strata</code>	A variable in data (single character or name) used to conduct stratified sampling. When not NULL, each resample is created within the stratification variable. Numeric strata are binned into quartiles.
<code>na_threshold</code>	Maximum fraction of NA values (defaults to 0.01) of the predictors before they are removed from the model, using <code>recipes::step_rm()</code>
<code>correlation_threshold</code>	A value (default 0.9) to indicate the correlation threshold. Predictors with a correlation higher than this value will be removed from the model, using <code>recipes::step_corr()</code>
<code>centre</code>	A <code>logical</code> to indicate whether the predictors should be transformed so that their mean will be 0, using <code>recipes::step_center()</code> . Binary columns will be skipped.

scale	A logical to indicate whether the predictors should be transformed so that their standard deviation will be 1, using recipes::step_scale() . Binary columns will be skipped.
engine	R package or function name to be used for the model, will be passed on to parsnip::set_engine()
mode	Type of predicted value. One of "classification", "regression", or "unknown". If "unknown", the mode will be inferred by the underlying parsnip model where possible. Explicitly setting mode is recommended to avoid ambiguity.
trees	An integer for the number of trees contained in the ensemble.
...	Additional arguments passed to the underlying parsnip model specification (e.g. trees, mtry, penalty). For tune_parameters() , these must be dials parameter objects such as dials::trees() or dials::mtry() . For predict() , these arguments are forwarded to parsnip::predict.model_fit() . Also see <i>Model Functions</i> .
quiet	A logical to silence console output. For the tune_parameters() function, these must be dials package calls, such as dials::trees() (see Examples). For predict() , these must be arguments passed on to parsnip::predict.model_fit()
tree_depth	An integer for maximum depth of the tree.
penalty	A non-negative number representing the total amount of regularization (specific engines only).
epochs	An integer for the number of training iterations.
neighbors	A single integer for the number of neighbors to consider (often called k). For kknn , a value of 5 is used if neighbors is not specified.
weight_func	A <i>single</i> character for the type of kernel function used to weight distances between samples. Valid choices are: "rectangular", "triangular", "epanechnikov", "biweight", "triweight", "cos", "inv", "gaussian", "rank", or "optimal".
object, data	outcome of machine learning model
new_data	A rectangular data object, such as a data frame.
type	A single character value or NULL. Possible values are "numeric", "class", "prob", "conf_int", "pred_int", "quantile", "time", "hazard", "survival", or "raw". When NULL, predict() will choose an appropriate value based on the model's mode.
only_prediction	a logical to indicate whether predictions must be returned as vector , otherwise returns a data.frame
only_certainty	a logical to indicate whether certainties must be returned as vector , otherwise returns a data.frame
correct_mistakes	a logical to indicate whether missing variables and missing values should be added to new_data

impute_algorithm	the algorithm to use in <code>impute()</code> if <code>correct_mistakes = TRUE</code> . Can be "mice" (default) for the Multivariate Imputations by Chained Equations (MICE) algorithm , or "single-point" for a trained median.
add_values	a logical to indicate whether values must be printed in the tiles
cols	columns to use for correlation plot, defaults to everything()
only_params_in_model	a logical to indicate whether only parameters in the model should be tuned
levels	An integer for the number of values of each parameter to use to make the regular grid. levels can be a single integer or a vector of integers that is the same length as the number of parameters in ... levels can be a named integer vector, with names that match the id values of parameters.
k	The number of partitions of the data set
plot_type	the plot type, can be "roc" (default), "gain", "lift" or "pr". These functions rely on yardstick::roc_curve() , yardstick::gain_curve() , yardstick::lift_curve() and yardstick::pr_curve() to construct the curves.

Details

To predict **regression** (numeric values), the function `ml_logistic_regression()` cannot be used.

To predict **classifications** (character values), the function `ml_linear_regression()` cannot be used.

The `apply_model_to()` function prioritises successful prediction over strict validation, as `correct_mistakes` defaults to `TRUE`.

The workflow of the `ml_*()` functions is approximately:



The `predict()` function can be used to fit a model on a new data set. Its wrapper `apply_model_to()` works in the same way, but can also detect and fix missing variables, missing data points, and data type differences between the trained data and the input data.

Use `feature_importances()` to get the importance of all features/variables. Use `autoplot()` afterwards to plot the results. These two functions are combined in `feature_importance_plot()`. Feature importance values are model-specific heuristics and are not comparable across model types.

Use The `tree_plot()` to plot the decision tree. For XGBoost models, `xgboost::xgb.plot.tree()` will be used. For all other tree models, `rpart.plot::rpart.plot()` will be used.

Use `correlation_plot()` to plot the correlation between all variables, even characters. If the input is a certstats ML model, the training data of the model will be plotted.

Use the `get_model_variables()` function to return a zero-row `data.frame` with the variables that were used for training, even before the recipe steps.

Use the `get_variable_weights()` function to determine the (rough) estimated weights of each variable in the model. This is not as reliable as retrieving coefficients, but it does work for any model. The weights are determined by running the model over all the highest and lowest values of each variable in the trained data. The function returns a data set with 1 row, of which the values sum up to 1.

Use the `tune_parameters()` function to analyse tune parameters of any `ml_*()` function. Without any parameters manually defined, it will try to tune all parameters of the underlying ML model. The tuning will be based on a **K-fold cross-validation**, of which the number of partitions can be set with `k`. The number of levels will be used to split the range of the parameters. For example, a range of 1-10 with `levels = 2` will lead to `[1, 10]`, while `levels = 5` will lead to `[1, 3, 5, 7, 9]`. The resulting `data.frame` will be sorted from best to worst. These results can also be plotted using `autoplot()`.

The `check_testing_predictions()` function combines the data used for testing from the original data with its predictions, so the original data can be reviewed per prediction.

Use `autoplot()` on a model to plot the receiver operating characteristic (ROC) curve, the gain curve, the lift curve, or the precision-recall (PR) curve. For the ROC curve, the (overall) area under the curve (AUC) will be printed as subtitle.

Value

A trained machine learning model of class `certstats_ml`, extending a `parsnip::model_fit` object.

The model object contains additional attributes with training data, preprocessing steps, predictions, and performance metrics.

Attributes

The `ml_*()` functions return a model object with the following **attributes**:

- `properties`: a **list** containing model metadata, including the ML function used, engine, mode, training/testing sizes, and model-specific parameters (e.g. `trees`, `tree_depth`)
- `recipe`: a prepped `recipes::recipe()` used for training and testing
- `data_original`: the original input data (after removal of invalid strata)
- `data_structure`: a zero-row `data.frame` containing the trained variables

- `data_means`: column means of numeric training variables
- `data_training`: processed training data after recipe baking
- `data_testing`: processed testing data after recipe baking
- `rows_training`: integer indices of training rows in `data_original`
- `rows_testing`: integer indices of testing rows in `data_original`
- `predictions`: predictions on the testing data
- `metrics`: performance metrics returned by `yardstick::metrics()`
- `correlation_threshold`: numeric correlation threshold used in preprocessing
- `centre`: logical indicating whether centring was applied
- `scale`: logical indicating whether scaling was applied

Model Functions

These functions wrap parsnip model specifications. Arguments set in `...` will be passed on to these parsnip functions.

- `ml_decision_trees`: `parsnip::decision_tree()`
- `ml_linear_regression`: `parsnip::linear_reg()`
- `ml_logistic_regression`: `parsnip::logistic_reg()`
- `ml_neural_network`: `parsnip::mlp()`
- `ml_nearest_neighbour`: `parsnip::nearest_neighbor()`
- `ml_random_forest`: `parsnip::rand_forest()`
- `ml_xg_boost`: `parsnip::boost_tree()`

Examples

```
# 'esbl_tests' is an included data set, see ?esbl_tests
print(esbl_tests, n = 5)

esbl_tests |> correlation_plot(add_values = FALSE) # red will be removed from model

# predict ESBL test outcome based on MICs using 2 different models
model1 <- esbl_tests |> ml_xg_boost(esbl, where(is.double))
model2 <- esbl_tests |> ml_decision_trees(esbl, where(is.double))

# Assessing A Model -----

model1 |> get_metrics()
model2 |> get_metrics()

model1 |> confusion_matrix()

# a correlation plot of a model shows the training data
model1 |> correlation_plot(add_values = FALSE)
```

```

model1 |> feature_importances()
model1 |> feature_importances() |> autoplot()
model2 |> feature_importance_plot()

# decision trees can also have a tree plot
model2 |> tree_plot()

# Applying A Model -----

# simply use base R `predict()` to apply a model:
model1 |> predict(esbl_tests)

# but apply_model_to() contains more info and can apply corrections:
model1 |> apply_model_to(esbl_tests)
# and to format the result, e.g. for an API:
model1 |> apply_model_to(esbl_tests) |> dplyr::slice(1:10) |> format()

# put in only parts of new data:
model1 |> apply_model_to(esbl_tests[, 1:15])
esbl_tests2 <- esbl_tests
esbl_tests2[2, "CIP"] <- NA
esbl_tests2[5, "AMC"] <- NA
# with XGBoost, nothing will be changed (it can correct for missings):
model1 |> apply_model_to(esbl_tests2)
# with random forest (or others), missings will be imputed:
model2 |> apply_model_to(esbl_tests2)

# Tuning A Model -----

# tune the parameters of a model (will take some time)
tuning <- model2 |>
  tune_parameters(k = 5, levels = 3)
autoplot(tuning)

# tuning analysis by specifying (some) parameters
iris |>
  ml_xg_boost(Species) |>
  tune_parameters(mtry = dials::mtry(range = c(1, 3)),
    trees = dials::trees())

# Practical Example #1 -----

# this is what iris data set looks like:
head(iris)
# create a model to predict the species:
iris_model <- iris |> ml_xg_boost(Species)
iris_model_rf <- iris |> ml_random_forest(Species)
# is it a bit reliable?
get_metrics(iris_model)

```

```

# now try to predict species from an arbitrary data set:
to_predict <- data.frame(Sepal.Length = 5,
                        Sepal.Width = 3,
                        Petal.Length = 1.5,
                        Petal.Width = 0.5)

to_predict

# should be 'setosa' in the 'predicted' column with high certainty:
iris_model |> apply_model_to(to_predict)

# API formatting:
iris_model |> apply_model_to(to_predict) |> format()
iris_model |> apply_model_to(to_predict, only_prediction = TRUE)
iris_model |> apply_model_to(to_predict, only_certainty = TRUE)

# which variables are generally important (only trained variables)?
iris_model |> feature_importances()

# how would the model do without the 'Sepal.Length' column?
to_predict <- to_predict[, c("Sepal.Width", "Petal.Width", "Petal.Length")]
to_predict
iris_model |> apply_model_to(to_predict)

# now compare that with a random forest model that requires imputation:
iris_model_rf |> apply_model_to(to_predict)

# the certainty is very different.

# Practical Example #2 -----

# this example shows plotting methods for a model

# train model to predict genus based on MICs:
genus <- esbl_tests |> ml_xg_boost(genus, everything())
genus |> get_metrics()
genus |> confusion_matrix()
genus |> feature_importance_plot()
genus |> autoplot()
genus |> autoplot(plot_type = "gain")
genus |> autoplot(plot_type = "pr")

```

Description

These functions call their original base R namesake, but with a global settable `na.rm` argument.

Usage

```
any(..., na.rm = getOption("na.rm", FALSE))

all(..., na.rm = getOption("na.rm", FALSE))

mean(x, ..., na.rm = getOption("na.rm", FALSE))

sum(..., na.rm = getOption("na.rm", FALSE))

prod(..., na.rm = getOption("na.rm", FALSE))

min(..., na.rm = getOption("na.rm", FALSE))

max(..., na.rm = getOption("na.rm", FALSE))

pmin(..., na.rm = getOption("na.rm", FALSE))

pmax(..., na.rm = getOption("na.rm", FALSE))

range(..., na.rm = getOption("na.rm", FALSE))

sd(x, na.rm = getOption("na.rm", FALSE))

var(x, ..., na.rm = getOption("na.rm", FALSE))

median(x, na.rm = getOption("na.rm", FALSE), ...)

fivenum(x, na.rm = getOption("na.rm", FALSE))

quantile(
  x,
  ...,
  na.rm = getOption("na.rm", FALSE),
  type = getOption("quantile.type", 7)
)

IQR(
  x,
  ...,
  na.rm = getOption("na.rm", FALSE),
  type = getOption("quantile.type", 7)
)
```

Arguments

...	zero or more logical vectors. Other objects of zero length are ignored, and the rest are coerced to logical ignoring any class.
na.rm	logical. If true NA values are removed before the result is computed.

x	an R object. Currently there are methods for numeric/logical vectors and date , date-time and time interval objects. Complex vectors are allowed for trim = 0, only.
type	an integer between 1 and 9 selecting one of the nine quantile algorithms detailed below to be used.

Default values of na.rm

This 'certestats' package supports a global default setting for na.rm in many mathematical functions. This can be set with `options(na.rm = TRUE)` or `options(na.rm = FALSE)`.

For [normality\(\)](#), [quantile\(\)](#) and [IQR\(\)](#), this also applies to the type argument. The default, type = 7, is the default of base R. Use type = 6 to comply with SPSS.

Source

```
base::any()
base::all()
base::mean()
base::sum()
base::prod()
base::min()
base::max()
base::pmin()
base::pmax()
base::range()
stats::sd()
stats::var()
stats::median()
stats::fivenum()
stats::quantile()
stats::IQR()
```

moving_average

Moving Average

Description

Functions to calculate a moving average. These are useful to get rid of (strong) peakiness.

Usage

```

moving_average(x, w, side = "centre", na.rm = getOption("na.rm", FALSE))

moving_sum(x, w, side = "centre", na.rm = getOption("na.rm", FALSE))

moving_Q1(x, w, side = "centre", na.rm = getOption("na.rm", FALSE))

moving_Q3(x, w, side = "centre", na.rm = getOption("na.rm", FALSE))

moving_fn(x, w, fun, side = "centre", ...)

```

Arguments

x	vector of values
w	window length; total number of observations to include. This should preferably be an odd number, so that the same number of values to the left and right of x are included.
side	default is "centre", can also be "left" or "right". This can be used to take a moving average (or sum, or ...) of e.g. the last 7 days.
na.rm	a logical to indicate whether empty must be removed from x
fun	function to apply
...	arguments passed on to fun

Details

Each function can be used over a moving period with [moving_fn\(\)](#). For example, for a moving median: `moving_fn(x, 7, fun = median)`. Or a moving maximum: `moving_fn(x, 5, fun = max)`.

The moving average is determined by averaging $\text{floor}(w / 2)$ values before and after each element of x and all elements in between.

Default values of na.rm

This 'certestats' package supports a global default setting for na.rm in many mathematical functions. This can be set with `options(na.rm = TRUE)` or `options(na.rm = FALSE)`.

For [normality\(\)](#), [quantile\(\)](#) and [IQR\(\)](#), this also applies to the type argument. The default, type = 7, is the default of base R. Use type = 6 to comply with SPSS.

normality

Normality Analysis

Description

Check normality of a vector of values.

Usage

```
normality(
  x,
  na.rm = getOption("na.rm", FALSE),
  type = getOption("quantile.type", 7)
)
```

Arguments

x	vector of values
na.rm	Remove empty values
type	an integer between 1 and 9 selecting one of the nine quantile algorithms detailed below to be used.

Default values of na.rm

This 'certestats' package supports a global default setting for na.rm in many mathematical functions. This can be set with `options(na.rm = TRUE)` or `options(na.rm = FALSE)`.

For `normality()`, `quantile()` and `IQR()`, this also applies to the type argument. The default, type = 7, is the default of base R. Use type = 6 to comply with SPSS.

Examples

```
x <- runif(1000)
normality(x)

x <- rnorm(1000)
normality(x)

x <- rexp(1000, rate = 3)
normality(x)
```

qc_rules

Quality Control (QC) Rules

Description

These rules are used for quality control (QC). Default values are set for **Nelson's QC rules**, but they also support Westgard, AIAG, Montgomery and Healthcare QC rules.

Usage

```
qc_rule1(x, m = mean(x), s = sd(x))

qc_rule2(x, m = mean(x), threshold = 9)

qc_rule3(x, threshold = 6)
```

```

qc_rule4(x, m = mean(x), threshold = 14, direction_mean = FALSE)

qc_rule5(x, m = mean(x), s = sd(x), threshold = 3)

qc_rule6(x, m = mean(x), s = sd(x), threshold = 5)

qc_rule7(x, m = mean(x), s = sd(x), threshold = 15)

qc_rule8(x, m = mean(x), s = sd(x), threshold = 8)

qc_rule_text(rule, threshold)

qc_test(x, m = mean(x), s = sd(x), guideline = "Nelson")

```

Arguments

x	vector with values
m	mean
s	standard deviation
threshold	minimal number of sequential values before rule is triggered (defaults to Nelson's)
direction_mean	a logical to indicate whether n observations in a row must be tested for alternating in direction of the mean
rule	number of the rule
guideline	guideline of QC rules set, must be "Nelson", "Westgard", "AIAG", "Montgomery", or "Healthcare"

Rules list

Rule	Rule explanation:	Nelson	Westgard	AIAG	Montg.	HC
#1	One point is more than 3 standard deviations from the mean	1	1	1	1	1
#2	n (or more) points in a row are on the same side of the mean	9	9	7	8	8
#3	n (or more) points in a row are continually incr. or decr.	6	-	6	6	6
#4	n (or more) points in a row alternate in direction, incr. then decr.	14	-	14	14	-
#5	$n - 1$ out of n points in a row are >2 sd from the mean	3	3	3	3	3
#6	$n - 1$ out of n points in a row are >1 sd from the mean	5	5	5	5	-
#7	$\geq n$ points in a row are within 1 sd of the mean	15	-	15	15	15
#8	$\geq n$ points in a row outside 1 sd of the mean, in both directions	8	-	8	8	-

Montg.: Montgomery, *HC*: Healthcare

Source

Nelson LS. **The Shewhart Control Chart—Tests for Special Causes**. Journal of Quality Technology. Informa UK Limited; 1984 Oct;16(4):237–9. doi:10.1080/00224065.1984.11978921.

Examples

```
x <- qc_test(rnorm(250))
x

# turn into data.frame, e.g. for export
head(as.data.frame(x))

## Not run:

library(plot2)
plot2(x, subtitle = "Workflow 'example123'")

## End(Not run)
```

regression

*Fast Regression Models***Description**

Functions to do fast regression modelling. The functions return a [tibble::tibble](#) with statistics. Use [plot\(\)](#) for an extensive model visualisation.

Usage

```
regression(x, ...)
```

Default S3 method:

```
regression(x, y = NULL, type = "lm", family = stats::gaussian, ...)
```

S3 method for class 'data.frame'

```
regression(x, var1, var2 = NULL, type = "lm", family = stats::gaussian, ...)
```

S3 method for class 'certestats_reg'

```
plot(x, ...)
```

S3 method for class 'certestats_reg'

```
autoplot(object, ...)
```

Arguments

x	vector of values, or a data.frame
...	arguments for lm() or glm()
y	vector of values, optional
type	type of function to use, can be "lm" or "glm"
family	only used for glm()
var1, var2	column to use of x, the var2 argument is optional
object	data to plot

Examples

```
runif(10) |> regression()

data.frame(x = 1:50, y = runif(50)) |>
  regression(x, y)

mrsa_from_blood_years <- c(0, 1, 0, 0, 2, 0, 1, 3, 1, 2, 3, 1, 2)
mrsa_from_blood_years |> plot()

mrsa_from_blood_years |> regression()

mrsa_from_blood_years |> regression() |> plot()
```

remove_outliers	<i>Remove Outliers</i>
-----------------	------------------------

Description

This simple function uses `boxplot.stats()` to determine outliers and removes them from the vector.

Usage

```
remove_outliers(x, coef = 1.5)
```

Arguments

x	a vector of values
coef	a multiple of the IQR that is allowed at maximum to keep values within the accepted range

Examples

```
remove_outliers(c(1,2,1,2,1,2,8))

remove_outliers(c(1,2,1,2,1,2))
```

row_function	<i>Apply Function per Row</i>
--------------	-------------------------------

Description

This can be used to e.g. add a maximum of certain rows.

Usage

```
row_function(fn, ..., data = NULL)
```

Arguments

fn	function to apply, such as <code>max()</code>
...	tidyverse selector helpers, passed on to <code>select()</code>
data	data set, will be determined with <code>pick()</code> if left blank

Examples

```
if (require("dplyr")) {
  iris |>
    mutate(max = row_function(max, where(is.numeric)),
           sepal_mean = row_function(mean, starts_with("Sepal"))) |>
    head()
}
```

weighted_mean	<i>Weighted Mean</i>
---------------	----------------------

Description

Functions to calculate a weighted mean or any other weighted metric.

Usage

```
weighted_mean(x, w, na.rm = getOption("na.rm", FALSE))

weighted_median(x, w, na.rm = getOption("na.rm", FALSE))

weighted_Q1(x, w, na.rm = getOption("na.rm", FALSE))

weighted_Q3(x, w, na.rm = getOption("na.rm", FALSE))

weighted_fn(x, w, fun, ...)
```

Arguments

x	vector of values
w	weights, length 1 or length of x
na.rm	a logical to indicate whether empty must be removed from x
fun	function to apply
...	arguments passed on to fun

Default values of na.rm

This 'certestats' package supports a global default setting for na.rm in many mathematical functions. This can be set with `options(na.rm = TRUE)` or `options(na.rm = FALSE)`.

For [normality\(\)](#), [quantile\(\)](#) and [IQR\(\)](#), this also applies to the type argument. The default, type = 7, is the default of base R. Use type = 6 to comply with SPSS.

Examples

```
x <- c(1:10)
y <- c(1:10)

mean(x)
weighted_mean(x, y)

# essentially equal to:
mean(rep(x, y))

x <- c(0:1000)
y <- c(0:1000)
mean(x)
weighted_mean(x, y)
weighted_median(x, y)
weighted_Q1(x, y)
weighted_Q3(x, y)
weighted_fn(x, y, quantile, c(0.01, 0.99))
```

Index

- * **datasets**
 - esbl_tests, 17
- all (math_functions), 30
- AMR::get_episode(), 12
- any (math_functions), 30
- apply_model_to (machine_learning), 20
- apply_model_to(), 26, 27
- as.binary, 2
- attribute, 19
- attributes, 27
- autoplot(), 12, 27
- autoplot.certstats_feature_importances
 - (machine_learning), 20
- autoplot.certstats_ml
 - (machine_learning), 20
- autoplot.certstats_reg (regression), 36
- autoplot.certstats_tuning
 - (machine_learning), 20
- autoplot.farrington_clusters
 - (detect_farrington), 9
- base::all(), 32
- base::any(), 32
- base::max(), 32
- base::mean(), 14, 32
- base::min(), 32
- base::pmax(), 32
- base::pmin(), 32
- base::prod(), 32
- base::range(), 32
- base::sum(), 32
- boxplot.stats(), 37
- centre_mean (distribution_metrics), 14
- centre_mean(), 16
- check_testing_predictions
 - (machine_learning), 20
- check_testing_predictions(), 27
- ci (distribution_metrics), 14
- ci(), 16
- confusion_matrix, 3
- confusion_matrix.certstats_ml
 - (machine_learning), 20
- correlation_plot (machine_learning), 20
- correlation_plot(), 27
- cqv (distribution_metrics), 14
- cqv(), 16
- cv (distribution_metrics), 14
- cv(), 16
- data.frame, 11, 17, 19, 25, 27
- date, 32
- date-time, 32
- deciles (distribution_metrics), 14
- deciles(), 16
- detect_biomarker_changes, 4
- detect_biomarker_changes(), 9
- detect_disease_clusters, 7
- detect_disease_clusters(), 6–8, 12, 13
- detect_farrington, 9
- detect_farrington(), 12
- different_means, 14
- distribution_metrics, 14
- dplyr::select(), 24
- esbl_tests, 17
- everything(), 18, 24, 26
- ewma (distribution_metrics), 14
- ewma(), 16
- factor, 24
- feature_importance_plot
 - (machine_learning), 20
- feature_importance_plot(), 27
- feature_importances (machine_learning), 20
- feature_importances(), 27
- fivenum (math_functions), 30
- format(), 6, 7, 12

- gain_plot (machine_learning), 20
- get_accuracy (machine_learning), 20
- get_coefficients (machine_learning), 20
- get_episode(), 8
- get_kappa (machine_learning), 20
- get_metrics (machine_learning), 20
- get_mice (impute), 18
- get_mice(), 19
- get_model_variables (machine_learning), 20
- get_model_variables(), 27
- get_original_data (machine_learning), 20
- get_recipe (machine_learning), 20
- get_roc_data (machine_learning), 20
- get_rows_testing (machine_learning), 20
- get_rows_training (machine_learning), 20
- get_specification (machine_learning), 20
- get_variable_weights
 - (machine_learning), 20
- get_variable_weights(), 27
- glm(), 36
- has_cluster_after
 - (detect_disease_clusters), 7
- has_cluster_before
 - (detect_disease_clusters), 7
- has_clusters (detect_disease_clusters), 7
- has_clusters(), 7
- has_farrington_cluster_after
 - (detect_farrington), 9
- has_farrington_cluster_before
 - (detect_farrington), 9
- has_farrington_clusters
 - (detect_farrington), 9
- has_farrington_clusters(), 12
- has_ongoing_cluster
 - (detect_disease_clusters), 7
- has_ongoing_cluster(), 9
- has_ongoing_farrington_cluster
 - (detect_farrington), 9
- impute, 18
- impute(), 26
- IQR (math_functions), 30
- IQR(), 16, 32–34, 39
- is.binary (as.binary), 2
- is_imputed (impute), 18
- is_imputed(), 19
- K-fold cross-validation, 27
- list, 27
- lm(), 36
- logical, 2, 3, 8, 9, 11, 12, 15, 24–26, 33, 39
- machine_learning, 20
- mae (distribution_metrics), 14
- mae(), 16
- mape (distribution_metrics), 14
- mape(), 16
- math_functions, 30
- max (math_functions), 30
- max(), 38
- mean (math_functions), 30
- mean_geometric (different_means), 14
- mean_harmonic (different_means), 14
- median (math_functions), 30
- mice::mice(), 19
- midhinge (distribution_metrics), 14
- midhinge(), 16
- min (math_functions), 30
- ml_decision_trees (machine_learning), 20
- ml_linear_regression
 - (machine_learning), 20
- ml_linear_regression(), 26
- ml_logistic_regression
 - (machine_learning), 20
- ml_logistic_regression(), 26
- ml_nearest_neighbour
 - (machine_learning), 20
- ml_neural_network (machine_learning), 20
- ml_random_forest (machine_learning), 20
- ml_xg_boost (machine_learning), 20
- moving_average, 32
- moving_average(), 8
- moving_fn (moving_average), 32
- moving_fn(), 33
- moving_Q1 (moving_average), 32
- moving_Q3 (moving_average), 32
- moving_sum (moving_average), 32
- mse (distribution_metrics), 14
- mse(), 16
- Multivariate Imputations by Chained Equations (MICE), 19
- Multivariate Imputations by Chained Equations (MICE) algorithm, 26
- n_clusters (detect_disease_clusters), 7

n_farrington_clusters
 (detect_farrington), 9
 normalise (distribution_metrics), 14
 normalise(), 16
 normality, 33
 normality(), 16, 32–34, 39
 normalize (distribution_metrics), 14
 numeric, 2

 parsnip::boost_tree(), 28
 parsnip::decision_tree(), 28
 parsnip::linear_reg(), 28
 parsnip::logistic_reg(), 28
 parsnip::mlp(), 28
 parsnip::model_fit, 27
 parsnip::nearest_neighbor(), 28
 parsnip::predict.model_fit(), 25
 parsnip::rand_forest(), 28
 parsnip::set_engine(), 25
 percentiles (distribution_metrics), 14
 percentiles(), 16
 pick(), 38
 plot(), 12, 36
 plot.certtestats_reg (regression), 36
 pmax (math_functions), 30
 pmin (math_functions), 30
 predict(), 25, 27
 predict.certtestats_ml
 (machine_learning), 20
 prod (math_functions), 30

 qc_rule1 (qc_rules), 34
 qc_rule2 (qc_rules), 34
 qc_rule3 (qc_rules), 34
 qc_rule4 (qc_rules), 34
 qc_rule5 (qc_rules), 34
 qc_rule6 (qc_rules), 34
 qc_rule7 (qc_rules), 34
 qc_rule8 (qc_rules), 34
 qc_rule_text (qc_rules), 34
 qc_rules, 34
 qc_test (qc_rules), 34
 quantile (math_functions), 30
 quantile(), 16, 32–34, 39
 quasiquotation, 3

 range (math_functions), 30
 recipes::recipe(), 24, 27
 recipes::step_center(), 24
 recipes::step_corr(), 24
 recipes::step_rm(), 24
 recipes::step_scale(), 25
 regression, 36
 remove_outliers, 37
 rmse (distribution_metrics), 14
 rmse(), 16
 roc_plot (machine_learning), 20
 row_function, 38
 rpart.plot::rpart.plot(), 27
 rr_ewma (distribution_metrics), 14
 rr_ewma(), 16
 rsample::initial_split(), 24

 scale_sd (distribution_metrics), 14
 scale_sd(), 16
 sd (math_functions), 30
 se (distribution_metrics), 14
 se(), 16
 select(), 38
 stats::fivenum(), 32
 stats::IQR(), 32
 stats::median(), 32
 stats::quantile(), 32
 stats::sd(), 32
 stats::var(), 32
 sum (math_functions), 30
 sum_of_squares (distribution_metrics),
 14
 sum_of_squares(), 16
 surveillance::earsC(), 12, 13
 surveillance::farringtonFlexible(),
 11–13

 tibble::tibble, 17, 36
 time interval, 32
 tree_plot (machine_learning), 20
 tree_plot(), 27
 try_binary (as.binary), 2
 try_binary(), 2
 tune_parameters (machine_learning), 20
 tune_parameters(), 25, 27

 var (math_functions), 30
 vector, 25

 weighted_fn (weighted_mean), 38
 weighted_mean, 38
 weighted_median (weighted_mean), 38

weighted_Q1 (weighted_mean), [38](#)
weighted_Q3 (weighted_mean), [38](#)

xgboost::xgb.plot.tree(), [27](#)

yardstick::gain_curve(), [26](#)
yardstick::lift_curve(), [26](#)
yardstick::metrics(), [28](#)
yardstick::pr_curve(), [26](#)
yardstick::roc_curve(), [26](#)

z_score (distribution_metrics), [14](#)
z_score(), [16](#)