Package ‘waywiser’

April 16, 2025

Type Package

Title Ergonomic Methods for Assessing Spatial Models

Version 0.6.3

Description Assessing predictive models of spatial data can be challenging, both because these models are typically built for extrapolating outside the original region represented by training data and due to potential spatially structured errors, with ``hot spots'' of higher than expected error clustered geographically due to spatial structure in the underlying data. Methods are provided for assessing models fit to spatial data, including approaches for measuring the spatial structure of model errors, assessing model predictions at multiple spatial scales, and evaluating where predictions can be made safely. Methods are particularly useful for models fit using the 'tidymodels' framework. Methods include Moran's I ('Moran' (1950) <doi:10.2307/2332142>), Geary's C ('Geary' (1954) <doi:10.2307/2986645>), Getis-Ord's G ('Ord' and 'Getis' (1995) <doi:10.1111/j.1538-4632.1995.tb00912.x>), agreement coefficients from 'Ji' and Gallo (2006) (<doi:10.14358/PERS.72.7.823>), agreement metrics from 'Willmott' (1981) (<doi:10.1080/02723646.1981.10642213>) and 'Willmott' 'et' 'al'. (2012) (<doi:10.1002/joc.2419>), an implementation of the area of applicability methodology from 'Meyer' and 'Pebesma' (2021) (<doi:10.1111/2041-210X.13650>), and an implementation of multi-scale assessment as described in 'Riemann' 'et' 'al'. (2010) (<doi:10.1016/j.rse.2010.05.010>).

License MIT + file LICENSE

URL `https://github.com/ropensci/waywiser`,
`https://docs.ropensci.org/waywiser/`

BugReports `https://github.com/ropensci/waywiser/issues`

Depends R (>= 4.0)

Imports dplyr (>= 1.1.0), fields, FNN, glue, hardhat, Matrix, purrr, rlang (>= 1.1.0), sf (>= 1.0-0), spdep (>= 1.1-9), stats, tibble, tidyselect, vctrs, yardstick (>= 1.2.0)

Suggests applicable, caret, CAST, covr, exactextractr, ggplot2, knitr, modeldata, recipes, rmarkdown, rsample, spatialsample, terra, testthat (>= 3.0.0), tidymodels, tidyr, tigris, units, vip, whisker, withr

VignetteBuilder knitr

Config/Needs/website kableExtra

Config/testthat/edition 3

Config/testthat/parallel true

Encoding UTF-8

Language en-US

LazyData true

RoxygenNote 7.3.2

NeedsCompilation no

Author Michael Mahoney [aut, cre] (<https://orcid.org/0000-0003-2402-304X>), Lucas Johnson [ctb] (<https://orcid.org/0000-0002-7953-0260>), Virgilio Gómez-Rubio [rev] (Virgilio reviewed the package (v. 0.2.0.9000) for rOpenSci, see <https://github.com/ropensci/software-review/issues/571>), Jakub Nowosad [rev] (Jakub reviewed the package (v. 0.2.0.9000) for rOpenSci, see <https://github.com/ropensci/software-review/issues/571>), Posit Software, PBC [cph, fnd]

Maintainer Michael Mahoney <mike.mahoney.218@gmail.com>

Repository CRAN

Date/Publication 2025-04-16 00:10:02 UTC

Contents

`guerry` *Guerry "Moral Statistics" (1830s)*

Description

This data and description are taken from the geodaData R package. Classic social science foundational study by Andre-Michel Guerry on crime, suicide, literacy and other "moral statistics" in 1830s France. Data from the R package Guerry (Michael Friendly and Stephane Dray).

Usage

```
guerry
```

Format

An sf data frame with 85 rows, 23 variables, and a geometry column:

dept Department ID: Standard numbers for the departments

Region Region of France ('N'='North', 'S'='South', 'E'='East', 'W'='West', 'C'='Central'). Corsica is coded as NA.

Dprtmnt Department name: Departments are named according to usage in 1830, but without accents. A factor with levels *Ain Aisne Allier ... Vosges Yonne*

Crm_prs Population per Crime against persons.

Crm_prp Population per Crime against property.

Litercy Percent of military conscripts who can read and write.

Donatns Donations to the poor.

Infants Population per illegitimate birth.

Suicids Population per suicide.

Maincty Size of principal city ('1:Sm', '2:Med', '3:Lg'), used as a surrogate for population density. Large refers to the top 10, small to the bottom 10; all the rest are classed Medium.

Wealth Per capita tax on personal property. A ranked index based on taxes on personal and movable property per inhabitant.

Commerc Commerce and Industry, measured by the rank of the number of patents / population.

Clergy Distribution of clergy, measured by the rank of the number of Catholic priests in active service population.

Crim_prn Crimes against parents, measured by the rank of the ratio of crimes against parents to all crimes – Average for the years 1825-1830.

Infntcd Infanticides per capita. A ranked ratio of number of infanticides to population – Average for the years 1825-1830.

Dntn_cl Donations to the clergy. A ranked ratio of the number of bequests and donations inter vivios to population – Average for the years 1815-1824.

Lottery Per capita wager on Royal Lottery. Ranked ratio of the proceeds bet on the royal lottery to population — Average for the years 1822-1826.

Desertn Military desertion, ratio of number of young soldiers accused of desertion to the force of the military contingent, minus the deficit produced by the insufficiency of available billets – Average of the years 1825-1827.

Instrct Instruction. Ranks recorded from Guerry's map of Instruction. Note: this is inversely related to Literacy

Prsttts Number of prostitutes registered in Paris from 1816 to 1834, classified by the department of their birth

Distanc Distance to Paris (km). Distance of each department centroid to the centroid of the Seine (Paris)

Area Area (1000 km^2).

Pop1831 Population in 1831, in 1000s

Details

Sf object, units in m. EPSG 27572: NTF (Paris) / Lambert zone II.

Source

- Angeville, A. (1836). Essai sur la Statistique de la Population française Paris: F. Doufour.
- Guerry, A.-M. (1833). Essai sur la statistique morale de la France Paris: Crochard. English translation: Hugh P. Whitt and Victor W. Reinking, Lewiston, N.Y. : Edwin Mellen Press, 2002.
- Parent-Duchatelet, A. (1836). De la prostitution dans la ville de Paris, 3rd ed, 1857, p. 32, 36

https://geodacenter.github.io/data-and-lab/Guerry/

Examples

```
if (requireNamespace("sf", quietly = TRUE)) {
  library(sf)
  data(guerry)

  plot(guerry["Donatns"])
}
```

`ny_trees` *Number of trees and aboveground biomass for Forest Inventory and Analysis plots in New York State*

Description

The original data is derived from the Forest Inventory and Analysis program, implemented by the US Department of Agriculture's Forest Service.

Usage

```
ny_trees
```

Format

An sf object using EPSG 5070: NAD83 / Conus Albers (in meters), with 5,303 rows and 5 columns:

yr The year measurements were taken.

plot A unique identifier signifying the plot measurements were taken at.

n_trees The number of trees present on a plot.

agb The total aboveground biomass at the plot location, in pounds.

geometry The centroid of the plot location.

`predict.ww_area_of_applicability` *Predict from a* `ww_area_of_applicability`

Description

Predict from a `ww_area_of_applicability`

Usage

```
## S3 method for class 'ww_area_of_applicability'
predict(object, new_data, ...)
```

Arguments

`object`	A `ww_area_of_applicability` object.
`new_data`	A data frame or matrix of new samples.
`...`	Not used.

Details

The function computes the distance indices of the new data and whether or not they are "inside" the area of applicability.

Value

A tibble of predictions, with two columns: `di`, numeric, contains the "dissimilarity index" of each point in `new_data`, while `aoa`, logical, contains whether a row is inside (`TRUE`) or outside (`FALSE`) the area of applicability.

Note that this function is often called using `terra::predict()`, in which case `aoa` will be converted to numeric implicitly; `1` values correspond to cells "inside" the area of applicability and `0` corresponds to cells "outside" the AOA.

The number of rows in the tibble is guaranteed to be the same as the number of rows in `new_data`. Rows with `NA` predictor values will have `NA` `di` and `aoa` values.

See Also

Other area of applicability functions: `ww_area_of_applicability()`

Examples

```
library(vip)
train <- gen_friedman(1000, seed = 101) # ?vip::gen_friedman
test <- train[701:1000, ]
train <- train[1:700, ]
pp <- stats::ppr(y ~ ., data = train, nterms = 11)
metric_name <- ifelse(
  packageVersion("vip") > package_version("0.3.2"),
  "rsq",
  "rsquared"
)

importance <- vip::vi_permute(
  pp,
  target = "y",
  metric = metric_name,
  pred_wrapper = predict,
  train = train
)

aoa <- ww_area_of_applicability(y ~ ., train, test, importance = importance)
predict(aoa, test)
```

worldclim_simulation *Simulated data based on WorldClim Bioclimatic variables*

Description

This data is adapted from the CAST vignette `vignette("cast02-AOA-tutorial", package = "CAST")`. The original data is derived from the Worldclim global climate variables.

Usage

```
worldclim_simulation
```

Format

An sf object with 10,000 rows and 6 columns:

bio2 Mean Diurnal Range (Mean of monthly (max temp - min temp))

bio10 Mean Temperature of Warmest Quarter

bio13 Precipitation of Wettest Month

bio19 Precipitation of Coldest Quarter

geometry The location of the sampled point.

response A virtual species distribution, generated using the `generateSpFromPCA()` function from the `virtualspecies` package.

Source

https://www.worldclim.org

ww_agreement_coefficient *Agreement coefficients and related methods*

Description

These functions calculate the agreement coefficient and mean product difference (MPD), as well as their systematic and unsystematic components, from Ji and Gallo (2006). Agreement coefficients provides a useful measurement of agreement between two data sets which is bounded, symmetrical, and can be decomposed into systematic and unsystematic components; however, it assumes a linear relationship between the two data sets and treats both "truth" and "estimate" as being of equal quality, and as such may not be a useful metric in all scenarios.

Usage

```
ww_agreement_coefficient(data, ...)

## S3 method for class 'data.frame'
ww_agreement_coefficient(data, truth, estimate, na_rm = TRUE, ...)

ww_agreement_coefficient_vec(truth, estimate, na_rm = TRUE, ...)

ww_systematic_agreement_coefficient(data, ...)

## S3 method for class 'data.frame'
ww_systematic_agreement_coefficient(data, truth, estimate, na_rm = TRUE, ...)

ww_systematic_agreement_coefficient_vec(truth, estimate, na_rm = TRUE, ...)

ww_unsystematic_agreement_coefficient(data, ...)

## S3 method for class 'data.frame'
ww_unsystematic_agreement_coefficient(data, truth, estimate, na_rm = TRUE, ...)

ww_unsystematic_agreement_coefficient_vec(truth, estimate, na_rm = TRUE, ...)

ww_unsystematic_mpd(data, ...)

## S3 method for class 'data.frame'
ww_unsystematic_mpd(data, truth, estimate, na_rm = TRUE, ...)

ww_unsystematic_mpd_vec(truth, estimate, na_rm = TRUE, ...)

ww_systematic_mpd(data, ...)

## S3 method for class 'data.frame'
ww_systematic_mpd(data, truth, estimate, na_rm = TRUE, ...)

ww_systematic_mpd_vec(truth, estimate, na_rm = TRUE, ...)

ww_unsystematic_rmpd(data, ...)

## S3 method for class 'data.frame'
ww_unsystematic_rmpd(data, truth, estimate, na_rm = TRUE, ...)

ww_unsystematic_rmpd_vec(truth, estimate, na_rm = TRUE, ...)

ww_systematic_rmpd(data, ...)

## S3 method for class 'data.frame'
ww_systematic_rmpd(data, truth, estimate, na_rm = TRUE, ...)
```

```
ww_systematic_rmpd_vec(truth, estimate, na_rm = TRUE, ...)
```

Arguments

data	A data.frame containing the columns specified by the truth and estimate arguments.
...	Not currently used.
truth	The column identifier for the true results (that is numeric). This should be an unquoted column name although this argument is passed by expression and supports quasiquotation (you can unquote column names). For _vec() functions, a numeric vector.
estimate	The column identifier for the predicted results (that is also numeric). As with truth this can be specified different ways but the primary method is to use an unquoted variable name. For _vec() functions, a numeric vector.
na_rm	A logical value indicating whether NA values should be stripped before the computation proceeds.

Details

Agreement coefficient values range from 0 to 1, with 1 indicating perfect agreement. truth and estimate must be the same length. This function is not explicitly spatial and as such can be applied to data with any number of dimensions and any coordinate reference system.

Value

A tibble with columns .metric, .estimator, and .estimate and 1 row of values. For grouped data frames, the number of rows returned will be the same as the number of groups. For _vec() functions, a single value (or NA).

References

Ji, L. and Gallo, K. 2006. "An Agreement Coefficient for Image Comparison." Photogrammetric Engineering & Remote Sensing 72(7), pp 823–833, doi: 10.14358/PERS.72.7.823.

See Also

Other agreement metrics: ww_willmott_d()

Other yardstick metrics: ww_global_geary_c(), ww_global_moran_i(), ww_local_geary_c(), ww_local_getis_ord_g(), ww_local_moran_i(), ww_willmott_d()

Examples

```
# Calculated values match Ji and Gallo 2006:
x <- c(6, 8, 9, 10, 11, 14)
y <- c(2, 3, 5, 5, 6, 8)

ww_agreement_coefficient_vec(x, y)
ww_systematic_agreement_coefficient_vec(x, y)
ww_unsystematic_agreement_coefficient_vec(x, y)
```

```
ww_systematic_mpd_vec(x, y)
ww_unsystematic_mpd_vec(x, y)
ww_systematic_rmpd_vec(x, y)
ww_unsystematic_rmpd_vec(x, y)

example_df <- data.frame(x = x, y = y)
ww_agreement_coefficient(example_df, x, y)
ww_systematic_agreement_coefficient(example_df, x, y)
ww_unsystematic_agreement_coefficient(example_df, x, y)
ww_systematic_mpd(example_df, x, y)
ww_unsystematic_mpd(example_df, x, y)
ww_systematic_rmpd(example_df, x, y)
ww_unsystematic_rmpd(example_df, x, y)
```

---

ww_area_of_applicability *Find the area of applicability*

Description

This function calculates the "area of applicability" of a model, as introduced by Meyer and Pebesma (2021). While the initial paper introducing this method focused on spatial models, there is nothing inherently spatial about the method; it can be used with any type of data (and, because it does not care about the spatial arrangement of your data, can be used with 2D or 3D spatial data, and with geographic or projected CRS).

Usage

```
ww_area_of_applicability(x, ...)

## S3 method for class 'data.frame'
ww_area_of_applicability(x, testing = NULL, importance, ..., na_rm = FALSE)

## S3 method for class 'matrix'
ww_area_of_applicability(x, testing = NULL, importance, ..., na_rm = FALSE)

## S3 method for class 'formula'
ww_area_of_applicability(
  x,
  data,
  testing = NULL,
  importance,
  ...,
  na_rm = FALSE
)

## S3 method for class 'recipe'
```

```
ww_area_of_applicability(
  x,
  data,
  testing = NULL,
  importance,
  ...,
  na_rm = FALSE
)

## S3 method for class 'rset'
ww_area_of_applicability(x, y = NULL, importance, ..., na_rm = FALSE)
```

Arguments

`x`	Either a data frame, matrix, formula (specifying predictor terms on the right-hand side), recipe (from `recipes::recipe()`, or `rset` object, produced by resampling functions from rsample or spatialsample. If `x` is a recipe, it should be the same one used to pre-process the data used in your model. If the recipe used to build the area of applicability doesn't match the one used to build the model, the returned area of applicability won't be correct.
`...`	Not currently used.
`testing`	A data frame or matrix containing the data used to validate your model. This should be the same data as used to calculate all model accuracy metrics. If this argument is `NULL`, then this function will use the training data (from `x` or `data`) to calculate within-sample distances. This may result in the area of applicability threshold being set too high, with the result that too many points are classed as "inside" the area of applicability.
`importance`	Either: • A data.frame with two columns: `term`, containing the names of each variable in the training and testing data, and `estimate`, containing the (raw or scaled) feature importance for each variable. • An object of class `vi` with at least two columns, `Variable` and `Importance`. All variables in the training data (`x` or `data`, depending on the context) must have a matching importance estimate, and all terms with importance estimates must be in the training data.
`na_rm`	A logical of length 1, indicating whether observations (in both training and testing) with `NA` values in predictors should be removed. Only predictor variables are considered, and this value has no impact on predictions (where `NA` values produce `NA` predictions). If `na_rm = FALSE` and `NA` values are found, this function returns an error.
`data`	The data frame representing your "training" data, when using the formula or recipe methods.
`y`	Optional: a recipe (from `recipes::recipe()`) or formula. If `y` is a recipe, it should be the same one used to pre-process the data used in your model. If the recipe used to build the area of applicability doesn't match the one used to build the model, the returned area of applicability won't be correct.

Details

Predictions made on points "inside" the area of applicability should be as accurate as predictions made on the data provided to `testing`. That means that generally `testing` should be your final hold-out set so that predictions on points inside the area of applicability are accurately described by your reported model metrics. When passing an `rset` object to `x`, predictions made on points "inside" the area of applicability instead should be as accurate as predictions made on the assessment sets during cross-validation.

This method assumes your model was fit using dummy variables in the place of any non-numeric predictor, and that you have one importance score per dummy variable. Having non-numeric predictors will cause this function to fail.

Value

A `ww_area_of_applicability` object, which can be used with `predict()` to calculate the distance of new data to the original training data, and determine if new data is within a model's area of applicability.

Differences from CAST

This implementation differs from Meyer and Pebesma (2021) (and therefore from CAST) when using cross-validated data in order to minimize data leakage. Namely, in order to calculate the dissimilarity index DI_k, CAST:

1. Rescales all data used for cross validation at once, lumping assessment folds in with analysis data.
2. Calculates a single $\bar{d}$ as the mean distance between all points in the rescaled data set, including between points in the same assessment fold.
3. For each point k that's used in an assessment fold, calculates d_k as the minimum distance between k and any point in its corresponding analysis fold.
4. Calculates DI_k by dividing d_k by $\bar{d}$ (which was partially calculated as the distance between k and the rest of the rescaled data).

Because assessment data is used to calculate constants for rescaling analysis data and $\bar{d}$, the assessment data may appear too "similar" to the analysis data when calculating DI_k. As such, waywiser treats each fold in an `rset` independently:

1. Each analysis set is rescaled independently.
2. Separate $\bar{d}$ are calculated for each fold, as the mean distance between all points in the analysis set for that fold.
3. Identically to CAST, d_k is the minimum distance between a point k in the assessment fold and any point in the corresponding analysis fold.
4. DI_k is then found by dividing d_k by $\bar{d}$, which was calculated independently from k.

Predictions are made using the full training data set, rescaled once (in the same way as CAST), and the mean $\bar{d}$ across folds, under the assumption that the "final" model in use will be retrained using the entire data set.

In practice, this means waywiser produces very slightly higher $\bar{d}$ values than CAST and a slightly higher area of applicability threshold than CAST when using `rset` objects.

References

H. Meyer and E. Pebesma. 2021. "Predicting into unknown space? Estimating the area of applicability of spatial prediction models," Methods in Ecology and Evolution 12(9), pp 1620 - 1633, doi: 10.1111/2041-210X.13650.

See Also

Other area of applicability functions: `predict.ww_area_of_applicability()`

Examples

```
train <- vip::gen_friedman(1000, seed = 101) # ?vip::gen_friedman
test <- train[701:1000, ]
train <- train[1:700, ]
pp <- stats::ppr(y ~ ., data = train, nterms = 11)
metric_name <- ifelse(
  packageVersion("vip") > package_version("0.3.2"),
  "rsq",
  "rsquared"
)

importance <- vip::vi_permute(
  pp,
  target = "y",
  metric = metric_name,
  pred_wrapper = predict,
  train = train
)

aoa <- ww_area_of_applicability(y ~ ., train, test, importance = importance)
predict(aoa, test)

# Equivalent methods for calculating AOA:
ww_area_of_applicability(train[2:11], test[2:11], importance)
ww_area_of_applicability(
  as.matrix(train[2:11]),
  as.matrix(test[2:11]),
  importance
)
```

`ww_build_neighbors` *Make 'nb' objects from sf objects*

Description

These functions can be used for geographic or projected coordinate reference systems and expect 2D data.

Usage

```
ww_build_neighbors(data, nb = NULL, ..., call = rlang::caller_env())
```

Arguments

`data`	An sf object (of class "sf" or "sfc").
`nb`	An object of class "nb" (in which case it will be returned unchanged), or a function to create an object of class "nb" from `data` and `...`, or `NULL`. See details.
`...`	Arguments passed to the neighbor-creating function.
`call`	The execution environment of a currently running function, e.g. `call = caller_env()`. The corresponding function call is retrieved and mentioned in error messages as the source of the error. You only need to supply `call` when throwing a condition from a helper function which wouldn't be relevant to mention in the message. Can also be `NULL` or a defused function call to respectively not display any call or hard-code a code to display. For more information about error calls, see Including function calls in error messages.

Details

When `nb = NULL`, the method used to create neighbors from `data` is dependent on what geometry type `data` is:

- If `nb = NULL` and `data` is a point geometry (classes "sfc_POINT" or "sfc_MULTIPOINT") the "nb" object will be created using `ww_make_point_neighbors()`.
- If `nb = NULL` and `data` is a polygon geometry (classes "sfc_POLYGON" or "sfc_MULTIPOLYGON") the "nb" object will be created using `ww_make_polygon_neighbors()`.
- If `nb = NULL` and `data` is any other geometry type, the "nb" object will be created using the centroids of the data as points, with a warning.

Value

An object of class "nb".

Examples

```
ww_build_neighbors(guerry)
```

ww_build_weights — *Build "listw" objects of spatial weights*

Description

These functions can be used for geographic or projected coordinate reference systems and expect 2D data.

Usage

```
ww_build_weights(x, wt = NULL, include_self = FALSE, ...)
```

Arguments

`x`	Either an sf object or a "nb" neighbors list object. If an sf object, will be converted into a neighbors list via `ww_build_neighbors()`.
`wt`	Either a "listw" object (which will be returned unchanged), a function for creating a "listw" object from `x`, or `NULL`, in which case weights will be constructed via `spdep::nb2listw()`.
`include_self`	Include each region itself in its own list of neighbors?
`...`	Arguments passed to the weight constructing function.

Value

A `listw` object.

Examples

```
ww_build_weights(guerry)
```

ww_global_geary_c — *Global Geary's C statistic*

Description

Calculate the global Geary's C statistic for model residuals. `ww_global_geary_c()` returns the statistic itself, while `ww_global_geary_pvalue()` returns the associated p value. These functions are meant to help assess model predictions, for instance by identifying if there are clusters of higher residuals than expected. For statistical testing and inference applications, use `spdep::geary.test()` instead.

Usage

```
ww_global_geary_c(data, ...)

ww_global_geary_c_vec(truth, estimate, wt, na_rm = FALSE, ...)

ww_global_geary_pvalue(data, ...)

ww_global_geary_pvalue_vec(truth, estimate, wt = NULL, na_rm = FALSE, ...)
```

Arguments

`data`	A `data.frame` containing the columns specified by the `truth` and `estimate` arguments.
`...`	Additional arguments passed to `spdep::geary()` (for `ww_global_geary_c()`) or `spdep::geary.test()` (for `ww_global_geary_pvalue()`).
`truth`	The column identifier for the true results (that is `numeric`). This should be an unquoted column name although this argument is passed by expression and supports quasiquotation (you can unquote column names). For `_vec()` functions, a `numeric` vector.
`estimate`	The column identifier for the predicted results (that is also `numeric`). As with `truth` this can be specified different ways but the primary method is to use an unquoted variable name. For `_vec()` functions, a `numeric` vector.
`wt`	A `listw` object, for instance as created with `ww_build_weights()`. For data.frame input, may also be a function that takes `data` and returns a `listw` object.
`na_rm`	A `logical` value indicating whether `NA` values should be stripped before the computation proceeds.

Details

These functions can be used for geographic or projected coordinate reference systems and expect 2D data.

Value

A tibble with columns .metric, .estimator, and .estimate and 1 row of values. For grouped data frames, the number of rows returned will be the same as the number of groups. For `_vec()` functions, a single value (or NA).

References

Geary, R. C. (1954). "The Contiguity Ratio and Statistical Mapping". The Incorporated Statistician. 5 (3): 115–145. doi:10.2307/2986645.

Cliff, A. D., Ord, J. K. 1981 Spatial processes, Pion, p. 17.

See Also

Other autocorrelation metrics: `ww_global_moran_i()`, `ww_local_geary_c()`, `ww_local_getis_ord_g()`, `ww_local_moran_i()`

Other yardstick metrics: `ww_agreement_coefficient()`, `ww_global_moran_i()`, `ww_local_geary_c()`, `ww_local_getis_ord_g()`, `ww_local_moran_i()`, `ww_willmott_d()`

Examples

```
guerry_model <- guerry
guerry_lm <- lm(Crm_prs ~ Litercy, guerry_model)
guerry_model$predictions <- predict(guerry_lm, guerry_model)

ww_global_geary_c(guerry_model, Crm_prs, predictions)
ww_global_geary_pvalue(guerry_model, Crm_prs, predictions)

wt <- ww_build_weights(guerry_model)

ww_global_geary_c_vec(
  guerry_model$Crm_prs,
  guerry_model$predictions,
  wt = wt
)
ww_global_geary_pvalue_vec(
  guerry_model$Crm_prs,
  guerry_model$predictions,
  wt = wt
)
```

`ww_global_moran_i` *Global Moran's I statistic*

Description

Calculate the global Moran's I statistic for model residuals. `ww_global_moran_i()` returns the statistic itself, while `ww_global_moran_pvalue()` returns the associated p value. These functions are meant to help assess model predictions, for instance by identifying if there are clusters of higher residuals than expected. For statistical testing and inference applications, use `spdep::moran.test()` instead.

Usage

```
ww_global_moran_i(data, ...)

ww_global_moran_i_vec(truth, estimate, wt = NULL, na_rm = FALSE, ...)

ww_global_moran_pvalue(data, ...)

ww_global_moran_pvalue_vec(truth, estimate, wt = NULL, na_rm = FALSE, ...)
```

Arguments

data	A data.frame containing the columns specified by the truth and estimate arguments.
...	Additional arguments passed to spdep::moran() (for ww_global_moran_i()) or spdep::moran.test() (for ww_global_moran_pvalue()).
truth	The column identifier for the true results (that is numeric). This should be an unquoted column name although this argument is passed by expression and supports quasiquotation (you can unquote column names). For _vec() functions, a numeric vector.
estimate	The column identifier for the predicted results (that is also numeric). As with truth this can be specified different ways but the primary method is to use an unquoted variable name. For _vec() functions, a numeric vector.
wt	A listw object, for instance as created with ww_build_weights(). For data.frame input, may also be a function that takes data and returns a listw object.
na_rm	A logical value indicating whether NA values should be stripped before the computation proceeds.

Details

These functions can be used for geographic or projected coordinate reference systems and expect 2D data.

Value

A tibble with columns .metric, .estimator, and .estimate and 1 row of values. For grouped data frames, the number of rows returned will be the same as the number of groups. For _vec() functions, a single value (or NA).

References

Moran, P.A.P. (1950). "Notes on Continuous Stochastic Phenomena." Biometrika, 37(1/2), pp 17. doi: 10.2307/2332142

Cliff, A. D., Ord, J. K. 1981 Spatial processes, Pion, p. 17.

See Also

Other autocorrelation metrics: ww_global_geary_c(), ww_local_geary_c(), ww_local_getis_ord_g(), ww_local_moran_i()

Other yardstick metrics: ww_agreement_coefficient(), ww_global_geary_c(), ww_local_geary_c(), ww_local_getis_ord_g(), ww_local_moran_i(), ww_willmott_d()

Examples

```
guerry_model <- guerry
guerry_lm <- lm(Crm_prs ~ Litercy, guerry_model)
guerry_model$predictions <- predict(guerry_lm, guerry_model)
```

```
ww_global_moran_i(guerry_model, Crm_prs, predictions)
ww_global_moran_pvalue(guerry_model, Crm_prs, predictions)

wt <- ww_build_weights(guerry_model)

ww_global_moran_i_vec(
  guerry_model$Crm_prs,
  guerry_model$predictions,
  wt = wt
)
ww_global_moran_pvalue_vec(
  guerry_model$Crm_prs,
  guerry_model$predictions,
  wt = wt
)
```

`ww_local_geary_c` *Local Geary's C statistic*

Description

Calculate the local Geary's C statistic for model residuals. `ww_local_geary_c()` returns the statistic itself, while `ww_local_geary_pvalue()` returns the associated p value. These functions are meant to help assess model predictions, for instance by identifying clusters of higher residuals than expected. For statistical testing and inference applications, use `spdep::localC_perm()` instead.

Usage

```
ww_local_geary_c(data, ...)

ww_local_geary_c_vec(truth, estimate, wt, na_rm = FALSE, ...)

ww_local_geary_pvalue(data, ...)

ww_local_geary_pvalue_vec(truth, estimate, wt = NULL, na_rm = FALSE, ...)
```

Arguments

`data`	A `data.frame` containing the columns specified by the `truth` and `estimate` arguments.
`...`	Additional arguments passed to `spdep::localC()` (for `ww_local_geary_c()`) or `spdep::localC_perm()` (for `ww_local_geary_pvalue()`).
`truth`	The column identifier for the true results (that is `numeric`). This should be an unquoted column name although this argument is passed by expression and supports quasiquotation (you can unquote column names). For `_vec()` functions, a `numeric` vector.

estimate — The column identifier for the predicted results (that is also numeric). As with truth this can be specified different ways but the primary method is to use an unquoted variable name. For _vec() functions, a numeric vector.

wt — A listw object, for instance as created with ww_build_weights(). For data.frame input, may also be a function that takes data and returns a listw object.

na_rm — A logical value indicating whether NA values should be stripped before the computation proceeds.

Details

These functions can be used for geographic or projected coordinate reference systems and expect 2D data.

Value

A tibble with columns .metric, .estimator, and .estimate and nrow(data) rows of values. For _vec() functions, a numeric vector of length(truth) (or NA).

References

Anselin, L. 1995. Local indicators of spatial association, Geographical Analysis, 27, pp 93–115. doi: 10.1111/j.1538-4632.1995.tb00338.x.

Anselin, L. 2019. A Local Indicator of Multivariate Spatial Association: Extending Geary's C. Geographical Analysis, 51, pp 133-150. doi: 10.1111/gean.12164

See Also

Other autocorrelation metrics: ww_global_geary_c(), ww_global_moran_i(), ww_local_getis_ord_g(), ww_local_moran_i()

Other yardstick metrics: ww_agreement_coefficient(), ww_global_geary_c(), ww_global_moran_i(), ww_local_getis_ord_g(), ww_local_moran_i(), ww_willmott_d()

Examples

```
guerry_model <- guerry
guerry_lm <- lm(Crm_prs ~ Litercy, guerry_model)
guerry_model$predictions <- predict(guerry_lm, guerry_model)

ww_local_geary_c(guerry_model, Crm_prs, predictions)
ww_local_geary_pvalue(guerry_model, Crm_prs, predictions)

wt <- ww_build_weights(guerry_model)

ww_local_geary_c_vec(
  guerry_model$Crm_prs,
  guerry_model$predictions,
  wt = wt
)
ww_local_geary_pvalue_vec(
  guerry_model$Crm_prs,
```

```
  guerry_model$predictions,
  wt = wt
)
```

---

ww_local_getis_ord_g *Local Getis-Ord G and G* statistic*

Description

Calculate the local Getis-Ord G and G* statistic for model residuals. `ww_local_getis_ord_g()` returns the statistic itself, while `ww_local_getis_ord_pvalue()` returns the associated p value. These functions are meant to help assess model predictions, for instance by identifying clusters of higher residuals than expected. For statistical testing and inference applications, use `spdep::localG_perm()` instead.

Usage

```
ww_local_getis_ord_g(data, ...)

ww_local_getis_ord_g_vec(truth, estimate, wt, na_rm = FALSE, ...)

ww_local_getis_ord_g_pvalue(data, ...)

ww_local_getis_ord_g_pvalue_vec(truth, estimate, wt, na_rm = FALSE, ...)
```

Arguments

`data`	A `data.frame` containing the columns specified by the `truth` and `estimate` arguments.
`...`	Additional arguments passed to `spdep::localG()` (for `ww_local_getis_ord_g()`) or `spdep::localG_perm()` (for `ww_local_getis_ord_pvalue()`).
`truth`	The column identifier for the true results (that is `numeric`). This should be an unquoted column name although this argument is passed by expression and supports quasiquotation (you can unquote column names). For `_vec()` functions, a `numeric` vector.
`estimate`	The column identifier for the predicted results (that is also `numeric`). As with `truth` this can be specified different ways but the primary method is to use an unquoted variable name. For `_vec()` functions, a `numeric` vector.
`wt`	A `listw` object, for instance as created with `ww_build_weights()`. For data.frame input, may also be a function that takes `data` and returns a `listw` object.
`na_rm`	A `logical` value indicating whether `NA` values should be stripped before the computation proceeds.

Details

These functions can be used for geographic or projected coordinate reference systems and expect 2D data.

Value

A tibble with columns .metric, .estimator, and .estimate and `nrow(data)` rows of values. For `_vec()` functions, a numeric vector of `length(truth)` (or NA).

References

Ord, J. K. and Getis, A. 1995. Local spatial autocorrelation statistics: distributional issues and an application. Geographical Analysis, 27, 286–306. doi: 10.1111/j.1538-4632.1995.tb00912.x

See Also

Other autocorrelation metrics: `ww_global_geary_c()`, `ww_global_moran_i()`, `ww_local_geary_c()`, `ww_local_moran_i()`

Other yardstick metrics: `ww_agreement_coefficient()`, `ww_global_geary_c()`, `ww_global_moran_i()`, `ww_local_geary_c()`, `ww_local_moran_i()`, `ww_willmott_d()`

Examples

```
guerry_model <- guerry
guerry_lm <- lm(Crm_prs ~ Litercy, guerry_model)
guerry_model$predictions <- predict(guerry_lm, guerry_model)

ww_local_getis_ord_g(guerry_model, Crm_prs, predictions)
ww_local_getis_ord_g_pvalue(guerry_model, Crm_prs, predictions)

wt <- ww_build_weights(guerry_model)

ww_local_getis_ord_g_vec(
  guerry_model$Crm_prs,
  guerry_model$predictions,
  wt = wt
)
ww_local_getis_ord_g_pvalue_vec(
  guerry_model$Crm_prs,
  guerry_model$predictions,
  wt = wt
)
```

ww_local_moran_i — *Local Moran's I statistic*

Description

Calculate the local Moran's I statistic for model residuals. `ww_local_moran_i()` returns the statistic itself, while `ww_local_moran_pvalue()` returns the associated p value. These functions are meant to help assess model predictions, for instance by identifying clusters of higher residuals than expected. For statistical testing and inference applications, use `spdep::localmoran_perm()` instead.

Usage

```
ww_local_moran_i(data, ...)

ww_local_moran_i_vec(truth, estimate, wt, na_rm = FALSE, ...)

ww_local_moran_pvalue(data, ...)

ww_local_moran_pvalue_vec(truth, estimate, wt = NULL, na_rm = FALSE, ...)
```

Arguments

`data`	A `data.frame` containing the columns specified by the `truth` and `estimate` arguments.
`...`	Additional arguments passed to `spdep::localmoran()`.
`truth`	The column identifier for the true results (that is `numeric`). This should be an unquoted column name although this argument is passed by expression and supports quasiquotation (you can unquote column names). For `_vec()` functions, a `numeric` vector.
`estimate`	The column identifier for the predicted results (that is also `numeric`). As with `truth` this can be specified different ways but the primary method is to use an unquoted variable name. For `_vec()` functions, a `numeric` vector.
`wt`	A `listw` object, for instance as created with `ww_build_weights()`. For data.frame input, may also be a function that takes `data` and returns a `listw` object.
`na_rm`	A `logical` value indicating whether `NA` values should be stripped before the computation proceeds.

Details

These functions can be used for geographic or projected coordinate reference systems and expect 2D data.

Value

A tibble with columns .metric, .estimator, and .estimate and `nrow(data)` rows of values. For `_vec()` functions, a numeric vector of `length(truth)` (or NA).

References

Anselin, L. 1995. Local indicators of spatial association, Geographical Analysis, 27, pp 93–115. doi: 10.1111/j.1538-4632.1995.tb00338.x.

Sokal, R. R, Oden, N. L. and Thomson, B. A. 1998. Local Spatial Autocorrelation in a Biological Model. Geographical Analysis, 30, pp 331–354. doi: 10.1111/j.1538-4632.1998.tb00406.x

See Also

Other autocorrelation metrics: `ww_global_geary_c()`, `ww_global_moran_i()`, `ww_local_geary_c()`, `ww_local_getis_ord_g()`

Other yardstick metrics: `ww_agreement_coefficient()`, `ww_global_geary_c()`, `ww_global_moran_i()`, `ww_local_geary_c()`, `ww_local_getis_ord_g()`, `ww_willmott_d()`

Examples

```
guerry_model <- guerry
guerry_lm <- lm(Crm_prs ~ Litercy, guerry_model)
guerry_model$predictions <- predict(guerry_lm, guerry_model)

ww_local_moran_i(guerry_model, Crm_prs, predictions)
ww_local_moran_pvalue(guerry_model, Crm_prs, predictions)

wt <- ww_build_weights(guerry_model)

ww_local_moran_i_vec(
  guerry_model$Crm_prs,
  guerry_model$predictions,
  wt = wt
)
ww_local_moran_pvalue_vec(
  guerry_model$Crm_prs,
  guerry_model$predictions,
  wt = wt
)
```

`ww_make_point_neighbors`

Make 'nb' objects from point geometries

Description

This function uses `spdep::knearneigh()` and `spdep::knn2nb()` to create a "nb" neighbors list.

Usage

```
ww_make_point_neighbors(data, k = 1, sym = FALSE, ...)
```

Arguments

`data`	An `sfc_POINT` or `sfc_MULTIPOINT` object.
`k`	How many nearest neighbors to use in `spdep::knearneigh()`.
`sym`	Force the output neighbors list (from `spdep::knn2nb()`) to symmetry.
`...`	Other arguments passed to `spdep::knearneigh()`.

Details

These functions can be used for geographic or projected coordinate reference systems and expect 2D data.

Value

An object of class "nb"

Examples

```
ww_make_point_neighbors(ny_trees)
```

`ww_make_polygon_neighbors`

Make 'nb' objects from polygon geometries

Description

This function is an extremely thin wrapper around `spdep::poly2nb()`, renamed to use the waywiser "ww" prefix.

Usage

```
ww_make_polygon_neighbors(data, ...)
```

Arguments

`data`	An `sfc_POLYGON` or `sfc_MULTIPOLYGON` object.
`...`	Additional arguments passed to `spdep::poly2nb()`.

Details

These functions can be used for geographic or projected coordinate reference systems and expect 2D data.

Value

An object of class "nb"

Examples

```
ww_make_polygon_neighbors(guerry)
```

ww_multi_scale   *Evaluate metrics at multiple scales of aggregation*

Description

Evaluate metrics at multiple scales of aggregation

Usage

```
ww_multi_scale(
  data = NULL,
  truth,
  estimate,
  metrics = list(yardstick::rmse, yardstick::mae),
  grids = NULL,
  ...,
  na_rm = TRUE,
  aggregation_function = "mean",
  autoexpand_grid = TRUE,
  progress = TRUE
)
```

Arguments

data	Either: a point geometry `sf` object containing the columns specified by the `truth` and `estimate` arguments; a `SpatRaster` from the `terra` package containing layers specified by the `truth` and `estimate` arguments; or `NULL` if `truth` and `estimate` are `SpatRaster` objects.
truth, estimate	If `data` is an `sf` object, the names (optionally unquoted) for the columns in `data` containing the true and predicted values, respectively. If `data` is a `SpatRaster` object, either (quoted) layer names or indices which will select the true and predicted layers, respectively, via `terra::subset()` If `data` is `NULL`, `SpatRaster` objects with a single layer containing the true and predicted values, respectively.
metrics	Either a `yardstick::metric_set()` object, or a list of functions which will be used to construct a `yardstick::metric_set()` object specifying the performance metrics to evaluate at each scale.
grids	Optionally, a list of pre-computed `sf` or `sfc` objects specifying polygon boundaries to use for assessments.
...	Arguments passed to `sf::st_make_grid()`. **You almost certainly should provide these arguments as lists.** For instance, passing `n = list(c(1, 2))` will create a single 1x2 grid; passing `n = c(1, 2)` will create a 1x1 grid *and* a 2x2 grid.

`na_rm` Boolean: Should polygons with NA values be removed before calculating metrics? Note that this does *not* impact how values are aggregated to polygons: if you want to remove NA values before aggregating, provide a function to `aggregation_function` which will remove NA values.

`aggregation_function`
The function to use to aggregate predictions and true values at various scales, by default `mean()`. For the `sf` method, you can pass any function which takes a single vector and returns a scalar. For raster methods, any function accepted by `exactextractr::exact_extract()` (note that built-in function names must be quoted). Note that this function does *not* pay attention to the value of `na_rm`; any NA handling you want to do during aggregation should be handled by this function directly.

`autoexpand_grid`
Boolean: if `data` is in geographic coordinates and `grids` aren't provided, the grids generated by `sf::st_make_grid()` may not contain all observations. If `TRUE`, this function will automatically expand generated grids by a tiny factor to attempt to capture all observations.

`progress` Boolean: if `data` is `NULL`, should aggregation via `exactextractr::exact_extract()` show a progress bar? Separate progress bars will be shown for each time `truth` and `estimate` are aggregated.

Value

A tibble with six columns: `.metric`, with the name of the metric that the row describes; `.estimator`, with the name of the estimator used, `.estimate`, with the output of the metric function; `.grid_args`, with the arguments passed to `sf::st_make_grid()` via `...` (if any), `.grid`, containing the grids used to aggregate predictions, as well as the aggregated values of `truth` and `estimate` as well as the count of non-NA values for each, and `.notes`, which (if `data` is an `sf` object) will indicate any observations which were not used in a given assessment.

Raster inputs

If `data` is `NULL`, then `truth` and `estimate` should both be `SpatRaster` objects, as created via `terra::rast()`. These rasters will then be aggregated to each grid using `exactextractr::exact_extract()`. If `data` is a `SpatRaster` object, then `truth` and `estimate` should be indices to select the appropriate layers of the raster via `terra::subset()`.

Grids are calculated using the bounding box of `truth`, under the assumption that you may have extrapolated into regions which do not have matching "true" values. This function does not check that `truth` and `estimate` overlap at all, or that they are at all contained within the grid.

Creating grid blocks

The grid blocks can be controlled by passing arguments to `sf::st_make_grid()` via `...`. Some particularly useful arguments include:

- `cellsize`: Target cellsize, expressed as the "diameter" (shortest straight-line distance between opposing sides; two times the apothem) of each block, in map units.
- `n`: The number of grid blocks in the x and y direction (columns, rows).

- square: A logical value indicating whether to create square (TRUE) or hexagonal (FALSE) cells.

If both cellsize and n are provided, then the number of blocks requested by n of sizes specified by cellsize will be returned, likely not lining up with the bounding box of data. If only cellsize is provided, this function will return as many blocks of size cellsize as fit inside the bounding box of data. If only n is provided, then cellsize will be automatically adjusted to create the requested number of cells.

Grids are created by mapping over each argument passed via ... simultaneously, in a similar manner to mapply() or purrr::pmap(). This means that, for example, passing n = list(c(1, 2)) will create a single 1x2 grid, while passing n = c(1, 2) will create a 1x1 grid *and* a 2x2 grid. It also means that arguments will be recycled using R's standard vector recycling rules, so that passing n = c(1, 2) and square = FALSE will create two separate grids of hexagons.

This function can be used for geographic or projected coordinate reference systems and expects 2D data.

References

Riemann, R., Wilson, B. T., Lister, A., and Parks, S. (2010). "An effective assessment protocol for continuous geospatial datasets of forest characteristics using USFS Forest Inventory and Analysis (FIA) data." Remote Sensing of Environment 114(10), pp 2337-2352, doi: 10.1016/j.rse.2010.05.010 .

Examples

```
data(ames, package = "modeldata")
ames_sf <- sf::st_as_sf(ames, coords = c("Longitude", "Latitude"), crs = 4326)
ames_model <- lm(Sale_Price ~ Lot_Area, data = ames_sf)
ames_sf$predictions <- predict(ames_model, ames_sf)

ww_multi_scale(
  ames_sf,
  Sale_Price,
  predictions,
  n = list(
    c(10, 10),
    c(1, 1)
  ),
  square = FALSE
)

# or, mostly equivalently
# (there will be a slight difference due to `autoexpand_grid = TRUE`)
grids <- list(
  sf::st_make_grid(ames_sf, n = c(10, 10), square = FALSE),
  sf::st_make_grid(ames_sf, n = c(1, 1), square = FALSE)
)
ww_multi_scale(ames_sf, Sale_Price, predictions, grids = grids)
```

ww_willmott_d *Willmott's d and related values*

Description

These functions calculate Willmott's d value, a proposed replacement for R2 which better differentiates between types and magnitudes of possible covariations. Additional functions calculate systematic and unsystematic components of MSE and RMSE; the sum of the systematic and unsystematic components of MSE equal total MSE (though the same is not true for RMSE).

Usage

```
ww_willmott_d(data, ...)

## S3 method for class 'data.frame'
ww_willmott_d(data, truth, estimate, na_rm = TRUE, ...)

ww_willmott_d_vec(truth, estimate, na_rm = TRUE, ...)

ww_willmott_d1(data, ...)

## S3 method for class 'data.frame'
ww_willmott_d1(data, truth, estimate, na_rm = TRUE, ...)

ww_willmott_d1_vec(truth, estimate, na_rm = TRUE, ...)

ww_willmott_dr(data, ...)

## S3 method for class 'data.frame'
ww_willmott_dr(data, truth, estimate, na_rm = TRUE, ...)

ww_willmott_dr_vec(truth, estimate, na_rm = TRUE, ...)

ww_systematic_mse(data, ...)

## S3 method for class 'data.frame'
ww_systematic_mse(data, truth, estimate, na_rm = TRUE, ...)

ww_systematic_mse_vec(truth, estimate, na_rm = TRUE, ...)

ww_unsystematic_mse(data, ...)

## S3 method for class 'data.frame'
ww_unsystematic_mse(data, truth, estimate, na_rm = TRUE, ...)

ww_unsystematic_mse_vec(truth, estimate, na_rm = TRUE, ...)
```

```
ww_systematic_rmse(data, ...)

## S3 method for class 'data.frame'
ww_systematic_rmse(data, truth, estimate, na_rm = TRUE, ...)

ww_systematic_rmse_vec(truth, estimate, na_rm = TRUE, ...)

ww_unsystematic_rmse(data, ...)

## S3 method for class 'data.frame'
ww_unsystematic_rmse(data, truth, estimate, na_rm = TRUE, ...)

ww_unsystematic_rmse_vec(truth, estimate, na_rm = TRUE, ...)
```

Arguments

`data`	A `data.frame` containing the columns specified by the `truth` and `estimate` arguments.
`...`	Not currently used.
`truth`	The column identifier for the true results (that is `numeric`). This should be an unquoted column name although this argument is passed by expression and supports quasiquotation (you can unquote column names). For `_vec()` functions, a `numeric` vector.
`estimate`	The column identifier for the predicted results (that is also `numeric`). As with `truth` this can be specified different ways but the primary method is to use an unquoted variable name. For `_vec()` functions, a `numeric` vector.
`na_rm`	A `logical` value indicating whether `NA` values should be stripped before the computation proceeds.

Details

Values of d and d1 range from 0 to 1, with 1 indicating perfect agreement. Values of dr range from -1 to 1, with 1 similarly indicating perfect agreement. Values of RMSE are in the same units as `truth` and `estimate`, while values of MSE are in squared units. `truth` and `estimate` must be the same length. This function is not explicitly spatial and as such can be applied to data with any number of dimensions and any coordinate reference system.

Value

A tibble with columns .metric, .estimator, and .estimate and 1 row of values. For grouped data frames, the number of rows returned will be the same as the number of groups. For `_vec()` functions, a single value (or NA).

References

Willmott, C. J. 1981. "On the Validation of Models". Physical Geography 2(2), pp 184-194, doi: 10.1080/02723646.1981.10642213.

Willmott, C. J. 1982. "Some Comments on the Evaluation of Model Performance". Bulletin of the American Meteorological Society 63(11), pp 1309-1313, doi: 10.1175/1520-0477(1982)063<1309:SCOTEO>2.0.CO;2.

Willmott C. J., Ackleson S. G., Davis R. E., Feddema J. J., Klink K. M., Legates D. R., O'Donnell J., Rowe C. M. 1985. "Statistics for the evaluation of model performance." Journal of Geophysical Research 90(C5): 8995–9005, doi: 10.1029/jc090ic05p08995

Willmott, C. J., Robeson, S. M., and Matsuura, K. "A refined index of model performance". International Journal of Climatology 32, pp 2088-2094, doi: 10.1002/joc.2419.

See Also

Other agreement metrics: `ww_agreement_coefficient()`

Other yardstick metrics: `ww_agreement_coefficient()`, `ww_global_geary_c()`, `ww_global_moran_i()`, `ww_local_geary_c()`, `ww_local_getis_ord_g()`, `ww_local_moran_i()`

Examples

```
x <- c(6, 8, 9, 10, 11, 14)
y <- c(2, 3, 5, 5, 6, 8)

ww_willmott_d_vec(x, y)
ww_willmott_d1_vec(x, y)
ww_willmott_dr_vec(x, y)
ww_systematic_mse_vec(x, y)
ww_unsystematic_mse_vec(x, y)
ww_systematic_rmse_vec(x, y)
ww_unsystematic_rmse_vec(x, y)

example_df <- data.frame(x = x, y = y)
ww_willmott_d(example_df, x, y)
ww_willmott_d1(example_df, x, y)
ww_willmott_dr(example_df, x, y)
ww_systematic_mse(example_df, x, y)
ww_unsystematic_mse(example_df, x, y)
ww_systematic_rmse(example_df, x, y)
ww_unsystematic_rmse(example_df, x, y)
```

Index