

# Package: spatialRF (via r-universe)

November 6, 2024

**Title** Easy Spatial Modeling with Random Forest

**Version** 1.1.4

**URL** <https://blasbenito.github.io/spatialRF/>

**BugReports** <https://github.com/BlasBenito/spatialRF/issues/>

**Description** Automatic generation and selection of spatial predictors for spatial regression with Random Forest. Spatial predictors are surrogates of variables driving the spatial structure of a response variable. The package offers two methods to generate spatial predictors from a distance matrix among training cases: 1) Moran's Eigenvector Maps (MEMs; Dray, Legendre, and Peres-Neto 2006 <[DOI:10.1016/j.ecolmodel.2006.02.015](https://doi.org/10.1016/j.ecolmodel.2006.02.015)>): computed as the eigenvectors of a weighted matrix of distances; 2) RFsp (Hengl et al. <[DOI:10.7717/peerj.5518](https://doi.org/10.7717/peerj.5518)>): columns of the distance matrix used as spatial predictors. Spatial predictors help minimize the spatial autocorrelation of the model residuals and facilitate an honest assessment of the importance scores of the non-spatial predictors. Additionally, functions to reduce multicollinearity, identify relevant variable interactions, tune random forest hyperparameters, assess model transferability via spatial cross-validation, and explore model results via partial dependence curves and interaction surfaces are included in the package. The modelling functions are built around the highly efficient 'ranger' package (Wright and Ziegler 2017 <[DOI:10.18637/jss.v077.i01](https://doi.org/10.18637/jss.v077.i01)>).

**License** GPL-3

**Depends** R (>= 2.10)

**Imports** dplyr, ggplot2, magrittr, stats, tibble, utils, foreach,  
doParallel, ranger, rlang, tidyr, tidyselect, huxtable,  
patchwork, viridis

**Suggests** testthat, spelling

**Encoding** UTF-8

**LazyData** true

**Roxygen** list(markdown = TRUE, old\_usage = FALSE)

**RoxygenNote** 7.2.1

**Language** en-US

**Config/pak/sysreqs** texlive libicu-dev libxml2-dev

**Repository** <https://blasbenito.r-universe.dev>

**RemoteUrl** <https://github.com/blasbenito/spatialrf>

**RemoteRef** HEAD

**RemoteSha** c004558423b9ea39ce2c0ce68636c641c95d8334

## Contents

auc . . . . .	3
auto_cor . . . . .	4
auto_vif . . . . .	6
beowulf_cluster . . . . .	7
case_weights . . . . .	9
default_distance_thresholds . . . . .	10
distance_matrix . . . . .	10
double_center_distance_matrix . . . . .	11
filter_spatial_predictors . . . . .	12
get_evaluation . . . . .	13
get_importance . . . . .	14
get_importance_local . . . . .	15
get_moran . . . . .	16
get_performance . . . . .	17
get_predictions . . . . .	18
get_residuals . . . . .	19
get_response_curves . . . . .	20
get_spatial_predictors . . . . .	21
is_binary . . . . .	22
make_spatial_fold . . . . .	23
make_spatial_folds . . . . .	25
mem . . . . .	27
mem_multithreshold . . . . .	28
moran . . . . .	29
moran_multithreshold . . . . .	30
objects_size . . . . .	32
optimization_function . . . . .	33
pca . . . . .	34
pca_multithreshold . . . . .	35
plant_richness_df . . . . .	36
plot_evaluation . . . . .	37
plot_importance . . . . .	39
plot_moran . . . . .	40
plot_optimization . . . . .	42

plot_residuals_diagnostics . . . . .	43
plot_response_curves . . . . .	45
plot_response_surface . . . . .	46
plot_training_df . . . . .	48
plot_training_df_moran . . . . .	49
plot_tuning . . . . .	51
prepare_importance_spatial . . . . .	52
print.rf . . . . .	53
print_evaluation . . . . .	54
print_importance . . . . .	55
print_moran . . . . .	56
print_performance . . . . .	57
rank_spatial_predictors . . . . .	58
rescale_vector . . . . .	61
residuals_diagnostics . . . . .	62
residuals_test . . . . .	63
rf . . . . .	63
rf_compare . . . . .	66
rf_evaluate . . . . .	69
rf_importance . . . . .	72
rf_repeat . . . . .	75
rf_spatial . . . . .	78
rf_tuning . . . . .	83
root_mean_squared_error . . . . .	85
select_spatial_predictors_recursive . . . . .	87
select_spatial_predictors_sequential . . . . .	90
standard_error . . . . .	93
statistical_mode . . . . .	93
the_feature_engineer . . . . .	94
thinning . . . . .	97
thinning_til_n . . . . .	98
vif . . . . .	99
weights_from_distance_matrix . . . . .	100

**Index****102**

auc

*Area under the ROC curve***Description**

Computes the area under the ROC curve in models with binary responses.

**Usage**

```
auc(o, p)
```

**Arguments**

- o                    Numeric vector with observations, must have the same length as p.
- p                    Numeric vector with predictions, must have the same length as o.

**Value**

Numeric, AUC value.

**Examples**

```
if(interactive()){
  out <- auc(
    o = c(0, 0, 1, 1),
    p = c(0.1, 0.6, 0.4, 0.8)
  )
}
```

---

 auto\_cor

---

*Multicollinearity reduction via Pearson correlation*


---

**Description**

Computes the correlation matrix among a set of predictors, orders the correlation matrix according to a user-defined preference order, and removes variables one by one, taking into account the preference order, until the remaining ones are below a given Pearson correlation threshold.

**Warning:** variables in preference.order not in colnames(x), and non-numeric columns are removed silently from x and preference.order. The same happens with rows having NA values (`na.omit()` is applied). The function issues a warning if zero-variance columns are found.

**Usage**

```
auto_cor(
  x = NULL,
  preference.order = NULL,
  cor.threshold = 0.5,
  verbose = TRUE
)
```

**Arguments**

- x                    A data frame with predictors, or the result of `auto_vif()` Default: NULL.
- preference.order    Character vector indicating the user's order of preference to keep variables. Doesn't need to contain If not provided, variables in x are prioritised by their column order. Default: NULL.

cor.threshold	Numeric between 0 and 1, with recommended values between 0.5 and 0.9. Maximum Pearson correlation between any pair of the selected variables. Default: 0.50
verbose	Logical. if TRUE, describes the function operations to the user. Default:: TRUE

### Details

Can be chained together with [auto\\_vif\(\)](#) through pipes, see the examples below.

### Value

List with three slots:

- cor: correlation matrix of the selected variables.
- selected.variables: character vector with the names of the selected variables.
- selected.variables.df: data frame with the selected variables.

### See Also

[auto\\_vif\(\)](#)

### Examples

```
if(interactive()){  
  
  #load data  
  data(plant_richness_df)  
  
  #on a data frame  
  out <- auto_cor(x = plant_richness_df[, 5:21])  
  
  #getting the correlation matrix  
  out$cor  
  
  #getting the names of the selected variables  
  out$selected.variables  
  
  #getting the data frame of selected variables  
  out$selected.variables.df  
  
  #on the result of auto_vif  
  out <- auto_vif(x = plant_richness_df[, 5:21])  
  out <- auto_cor(x = out)  
  
  #with pipes  
  out <- plant_richness_df[, 5:21] %>%  
  auto_vif() %>%  
  auto_cor()  
  
}
```

---

 auto\_vif

*Multicollinearity reduction via Variance Inflation Factor*


---

## Description

Selects predictors that are not linear combinations of other predictors by using computing their variance inflation factors (VIF). Allows the user to define an order of preference for the selection of predictors. **Warning:** variables in `preference.order` not in `colnames(x)`, and non-numeric columns are removed silently from `x` and `preference.order`. The same happens with rows having NA values (`na.omit()` is applied). The function issues a warning if zero-variance columns are found.

## Usage

```
auto_vif(
  x = NULL,
  preference.order = NULL,
  vif.threshold = 5,
  verbose = TRUE
)
```

## Arguments

<code>x</code>	A data frame with predictors or the result of <code>auto_cor()</code> . Default: NULL.
<code>preference.order</code>	a character vector with columns names of <code>x</code> ordered by the user preference, Default: NULL.
<code>vif.threshold</code>	Numeric between 2.5 and 10 defining the selection threshold for the VIF analysis. Higher numbers result in a more relaxed variable selection. Default: 5.
<code>verbose</code>	Logical. if TRUE, describes the function operations to the user. Default:: TRUE

## Details

This function has two modes of operation:

- 1. When the argument `preference.order` is NULL, the function removes on each iteration the variable with the highest VIF until all VIF values are lower than `vif.threshold`.
- 2. When `preference.order` is provided, the variables are selected by giving them priority according to their order in `preference.order`. If there are variables not in `preference.order`, these are selected as in option 1. Once both groups of variables have been processed, all variables are put together and selected by giving priority to the ones in `preference.order`. This method preserves the variables desired by the user as much as possible.

Can be chained together with `auto_cor()` through pipes, see the examples below.

**Value**

List with three slots:

- `vif`: data frame with the names of the selected variables and their respective VIF scores.
- `selected.variables`: character vector with the names of the selected variables.
- `selected.variables.df`: data frame with the selected variables.

**See Also**

[auto\\_cor\(\)](#)

**Examples**

```
if(interactive()){

#loading data
data(plant_richness_df)

#on a data frame
out <- auto_vif(x = plant_richness_df[, 5:21])

#getting out the vif data frame
out$vif

#getting the names of the selected variables
out$selected.variables

#getting the data frame of selected variables
out$selected.variables.df

#on the result of auto_cor
out <- auto_cor(x = plant_richness_df[, 5:21])
out <- auto_vif(x = out)

#with pipes
out <- plant_richness_df[, 5:21] %>%
  auto_cor() %>%
  auto_vif()

}
```

---

beowulf\_cluster

*Defines a beowulf cluster*

---

**Description**

Defines a Beowulf cluster from the IPs of the machines in the cluster, the number of cores of each machine, and the user name. The returned cluster has to be registered with `doParallel::registerDoParallel()`.

**Usage**

```
beowulf_cluster(
  cluster.ips = NULL,
  cluster.cores = NULL,
  cluster.user = Sys.info()[["user"]],
  cluster.port = "11000",
  outfile = NULL
)
```

**Arguments**

<code>cluster.ips</code>	Character vector with the IPs of the machines in the cluster. The first machine will be considered the main node of the cluster, and will generally be the machine on which the R code is being executed. Default: <code>NULL</code> .
<code>cluster.cores</code>	Numeric integer vector, number of cores on each machine. Default: <code>NULL</code> .
<code>cluster.user</code>	Character string, name of the user (should be the same throughout machines), Defaults to the current system user.
<code>cluster.port</code>	Character, port used by the machines in the cluster to communicate. The firewall in all computers must allow traffic from and to such port. Default: <code>"11000"</code>
<code>outfile</code>	Where to direct the messages provided by the workers. When working on a local computer, <code>""</code> prints the worker's messages in the console. A text file path will append worker's messages on the given file. Default: <code>/dev/null</code> on Linux and <code>nul:</code> on windows.

**Value**

A list ready to be used as input for the `spec` argument of the function [makeCluster](#).

**Examples**

```
if(interactive()){
  beowulf.cluster <- beowulf_cluster(
    cluster.ips = c(
      "10.42.0.1",
      "10.42.0.34",
      "10.42.0.104"
    ),
    cluster.cores = c(7, 4, 4),
    cluster.user = "blas",
    cluster.port = "11000"
  )

  doParallel::registerDoParallel(cl = beowulf.cluster)

  #PARALLELIZED foreach LOOP HERE

  parallel::stopCluster(cl = beowulf.cluster)
}
```



```
}
```

---

case_weights	<i>Generates case weights for binary data</i>
--------------	---

---

### Description

When the data is binary, setting the `manager` argument `case.weights` helps to minimize the issues produced by class imbalance. This function takes a binary response variable and returns a vector of weights populated with the values  $1/\#zeros$  and  $1/\#ones$ . It is used internally by the function `rf()`.

### Usage

```
case_weights(data = NULL, dependent.variable.name = NULL)
```

### Arguments

<code>data</code>	Data frame with a response variable and a set of predictors. Default: NULL
<code>dependent.variable.name</code>	Character string with the name of the response variable. Must be in the column names of data. Default: NULL

### Value

A vector with a length equal to `nrow(data)` with the respective weights of the cases.

### Examples

```
if(interactive()){  
  
  data <- data.frame(  
    response = c(0, 0, 0, 1, 1)  
  )  
  
  case_weights(  
    data = data,  
    dependent.variable.name = "response"  
  )  
  
}
```

---

`default_distance_thresholds`*Default distance thresholds to generate spatial predictors*

---

**Description**

Generates four distance thresholds, from 0 to  $\max(\text{distance.matrix})/2$ .

**Usage**

```
default_distance_thresholds(distance.matrix = NULL)
```

**Arguments**

`distance.matrix`  
Distance matrix. Default: NULL.

**Value**

A numeric vector with distance thresholds.

**Examples**

```
if(interactive()){  
  
  #loading example distance matrix  
  data(distance_matrix)  
  
  #computing set of default distance thresholds  
  default_distance_thresholds(distance_matrix)  
  
}
```

---

`distance_matrix`*Matrix of distances among ecoregion edges.*

---

**Description**

Distance matrix (in km) among the edges of the American ecoregions described in the [plant\\_richness\\_df](#) dataset.

**Usage**

```
data(distance_matrix)
```

**Format**

A numeric matrix with 227 rows and columns.

**See Also**

[plant\\_richness\\_df](#)

---

double\_center\_distance\_matrix

*Double centers a distance matrix*

---

**Description**

Generates a double-centered matrix (row and column means are zero) from the weights of a distance matrix (see [weights\\_from\\_distance\\_matrix\(\)](#)) and a distance threshold. This is a required step before the computation of Moran's Eigenvector Maps.

**Usage**

```
double_center_distance_matrix (  
  distance.matrix = NULL,  
  distance.threshold = 0  
)
```

**Arguments**

distance.matrix

Distance matrix. Default: NULL.

distance.threshold

Numeric, positive, in the range of values of x. Distances below this value in the distance matrix are set to 0. Default: 0.

**Value**

A double-centered matrix of the same dimensions as x.

**See Also**

[weights\\_from\\_distance\\_matrix\(\)](#), [mem\(\)](#), [mem\\_multithreshold\(\)](#)

**Examples**

```
if(interactive()){  
  
  #loading the distance matrix  
  data(distance_matrix)  
  
  x <- double_center_distance_matrix(  

```

```

    distance.matrix = distance_matrix
  )
x
}

```

---

filter\_spatial\_predictors

*Removes redundant spatial predictors*

---

### Description

Removes spatial predictors that are pair-wise correlated with other spatial predictors (which happens when there are several close distance thresholds), and spatial predictors correlated with non-spatial predictors.

### Usage

```

filter_spatial_predictors(
  data = NULL,
  predictor.variable.names = NULL,
  spatial.predictors.df = NULL,
  cor.threshold = 0.5
)

```

### Arguments

`data` Data frame with a response variable and a set of predictors. Default: NULL

`predictor.variable.names` Character vector with the names of the predictive variables. Every element of this vector must be in the column names of `data`. Default: NULL

`spatial.predictors.df` Data frame of spatial predictors.

`cor.threshold` Numeric between 0 and 1, maximum Pearson correlation between any pair of the selected variables. Default: 0.50

### Value

A data frame with non-redundant spatial predictors.

### Examples

```

if(interactive()){
  #loading data
  data("distance_matrix")
  data("plant_richness_df")
}

```

```

#computing Moran's Eigenvector Maps
spatial.predictors.df <- mem_multithreshold(
  distance_matrix = distance_matrix,
  distance.thresholds = c(0, 1000)
)

#filtering spatial predictors
spatial.predictors.df <- filter_spatial_predictors(
  data = plant_richness_df,
  predictor.variable.names = colnames(plant_richness_df)[5:21],
  spatial.predictors.df = spatial.predictors.df,
  cor.threshold = 0.50
)

}

```

---

get_evaluation	<i>Gets performance data frame from a cross-validated model</i>
----------------	---

---

### Description

Returns performance metrics produced by [rf\\_evaluate\(\)](#).

### Usage

```
get_evaluation(model)
```

### Arguments

model            A model fitted with [rf\\_evaluate\(\)](#).

### Value

A data frame with evaluation scores. The following columns are shown:

- model: Identifies the given model. The values are "Full", (original model introduced into [rf\\_evaluate\(\)](#)), "Training" (model trained on an independent training spatial fold), and "Testing" (predictive performance of the training model on an independent testing spatial fold). The performance values of the "Testing" model represent the model performance on unseen data, and hence its ability to generalize.
- metric: Four values representing different evaluation metrics, "rmse", "nrmse", "r.squared", and "pseudo.r.squared".
- mean, sd, min, and max: Average, standard deviation, minimum, and maximum of each metric across the evaluation (cross-validation) iterations.

### See Also

[rf\\_evaluate\(\)](#), [plot\\_evaluation\(\)](#), [print\\_evaluation\(\)](#)

**Examples**

```

if(interactive()){

#loading data
data(plant_richness_df)
data(distance_matrix)

#fitting a random forest model
rf.model <- rf(
  data = plant_richness_df,
  dependent.variable.name = "richness_species_vascular",
  predictor.variable.names = colnames(plant_richness_df)[5:21],
  distance.matrix = distance_matrix,
  distance.thresholds = 0,
  n.cores = 1,
  verbose = FALSE
)

#evaluating the model with spatial cross-validation
rf.model <- rf_evaluate(
  model = rf.model,
  xy = plant_richness_df[, c("x", "y")],
  n.cores = 1,
  verbose = FALSE
)

#getting evaluation results from the model
x <- get_evaluation(rf.model)
x

}

```

---

get\_importance

*Gets the global importance data frame from a model*


---

**Description**

Gets variable importance scores from `rf()`, `rf_repeat()`, and `rf_spatial()` models.

**Usage**

```
get_importance(model)
```

**Arguments**

`model` A model fitted with `rf()`, `rf_repeat()`, or `rf_spatial()`. Default: NULL

**Value**

A data frame with variable names and importance scores.

**See Also**

[rf\(\)](#), [rf\\_repeat\(\)](#), [rf\\_spatial\(\)](#), [plot\\_importance\(\)](#), [print\\_importance\(\)](#).

**Examples**

```
if(interactive()){  
  
  data(plant_richness_df)  
  data(distance_matrix)  
  
  rf.model <- rf(  
    data = plant_richness_df,  
    dependent.variable.name = "richness_species_vascular",  
    predictor.variable.names = colnames(plant_richness_df)[5:21],  
    distance.matrix = distance_matrix,  
    distance.thresholds = 0,  
    n.cores = 1,  
    verbose = FALSE  
  )  
  
  x <- get_importance(rf.model)  
  x  
  
}
```

---

get\_importance\_local    *Gets the local importance data frame from a model*

---

**Description**

Gets local importance scores from [rf\(\)](#), [rf\\_repeat\(\)](#), and [rf\\_spatial\(\)](#) models.

**Usage**

```
get_importance_local(model)
```

**Arguments**

model                    A model fitted with [rf\(\)](#), [rf\\_repeat\(\)](#), or [rf\\_spatial\(\)](#). Default: NULL

**Value**

A data frame with variable names and local importance scores.

**See Also**

[rf\(\)](#), [rf\\_repeat\(\)](#), [rf\\_spatial\(\)](#), [plot\\_importance\(\)](#), [print\\_importance\(\)](#).

## Examples

```
if(interactive()){

#loading example data
data(plant_richness_df)
data(distance_matrix)

#fitting a random forest model
rf.model <- rf(
  data = plant_richness_df,
  dependent.variable.name = "richness_species_vascular",
  predictor.variable.names = colnames(plant_richness_df)[5:21],
  distance.matrix = distance_matrix,
  distance.thresholds = 0,
  n.cores = 1,
  verbose = FALSE
)

#getting importance scores
x <- get_importance_local(rf.model)
x

}
```

---

get\_moran

*Gets Moran's I test of model residuals*

---

## Description

Returns the Moran's I test on the residuals of a model produced by [rf\(\)](#), [rf\\_repeat\(\)](#), or [rf\\_spatial\(\)](#).

## Usage

```
get_moran(model)
```

## Arguments

model            A model fitted with [rf\(\)](#), [rf\\_repeat\(\)](#), or [rf\\_spatial\(\)](#)

## Value

A data frame with Moran's I test results produced by [moran\\_multithreshold\(\)](#).

## See Also

[moran\(\)](#), [moran\\_multithreshold\(\)](#), [plot\\_moran\(\)](#), [print\\_moran\(\)](#).



## Examples

```
if(interactive()){

  #loading example data
  data(plant_richness_df)
  data(distance_matrix)

  #fitting a random forest model
  rf.model <- rf(
    data = plant_richness_df,
    dependent.variable.name = "richness_species_vascular",
    predictor.variable.names = colnames(plant_richness_df)[5:21],
    distance.matrix = distance_matrix,
    distance.thresholds = c(0, 1000, 2000),
    n.cores = 1,
    verbose = FALSE
  )

  #getting Moran's I of the residuals
  x <- get_moran(rf.model)

}
```

---

get\_performance

*Gets out-of-bag performance scores from a model*

---

## Description

Returns the performance slot of an `rf()`, `rf_repeat()`, or `rf_spatial()` model computed on the out-of-bag data.

## Usage

```
get_performance(model)
```

## Arguments

model            Model fitted with `rf()`, `rf_repeat()`, or `rf_spatial()`.

## Value

A data frame with four columns:

- `metric` Name of the performance metric.
- `median` Value of the performance metric. Truly a median only if the model is fitted with `rf_repeat()`.
- `median_absolute_deviation` median absolute deviation (MAD), only if the model is fitted with `rf_repeat()`, and NA otherwise.

**See Also**

[print\\_performance\(\)](#)

**Examples**

```
if(interactive()){

  #loading example data
  data(plant_richness_df)
  data(distance.matrix)

  #fitting random forest model
  rf.model <- rf(
    data = plant_richness_df,
    dependent.variable.name = "richness_species_vascular",
    predictor.variable.names = colnames(plant_richness_df)[5:21],
    distance.matrix = distance_matrix,
    distance.thresholds = 0,
    n.cores = 1,
    verbose = FALSE
  )

  #getting model performance
  x <- get_performance(rf.model)
  x

}
```

---

get\_predictions

*Gets model predictions*

---

**Description**

Returns model predictions from a model fitted with [rf\(\)](#), [rf\\_repeat\(\)](#), or [rf\\_spatial\(\)](#).

**Usage**

```
get_predictions(model)
```

**Arguments**

model            A model produced by [rf\(\)](#), [rf\\_repeat\(\)](#), or [rf\\_spatial\(\)](#).

**Value**

A vector with predictions, or median of the predictions across repetitions if the model was fitted with [rf\\_repeat\(\)](#).

**Examples**

```
if(interactive()){

#loading example data
data(plant_richness_df)

#fitting a random forest model
rf.model <- rf(
  data = plant_richness_df,
  dependent.variable.name = "richness_species_vascular",
  predictor.variable.names = colnames(plant_richness_df)[5:21],
  n.cores = 1,
  verbose = FALSE
)

#get vector of predictions
x <- get_predictions(rf.model)
x

}
```

---

get\_residuals

*Gets model residuals*

---

**Description**

Returns the residuals of models fitted with `rf()`, `rf_repeat()`, or `rf_spatial()`.

**Usage**

```
get_residuals(model)
```

**Arguments**

model            A model fitted with `rf()`, `rf_repeat()`, or `rf_spatial()`.

**Value**

A vector with model residuals, or the median of model residuals across repetitions if the model was fitted with `rf_repeat()`.

**Examples**

```
if(interactive()){

#load example data
data(plant_richness_df)

#fit random forest model
```

```

rf.model <- rf(
  data = plant_richness_df,
  dependent.variable.name = "richness_species_vascular",
  predictor.variable.names = colnames(plant_richness_df)[5:21],
  n.cores = 1,
  verbose = FALSE
)

#getting vector with residuals
x <- get_residuals(rf.model)
x

}

```

---

get\_response\_curves     *Gets data to allow custom plotting of response curves*

---

### Description

Generates and returns the data required to plot the response curves of a model fitted with `rf()`, `rf_repeat()`, or `rf_spatial()`.

### Usage

```

get_response_curves(
  model = NULL,
  variables = NULL,
  quantiles = c(0.1, 0.5, 0.9),
  grid.resolution = 200,
  verbose = TRUE
)

```

### Arguments

model	A model fitted with <code>rf()</code> , <code>rf_repeat()</code> , or <code>rf_spatial()</code> .
variables	Character vector, names of predictors to plot. If NULL, the most important variables (importance higher than the median) in model are selected. Default: NULL.
quantiles	Numeric vector with values between 0 and 1, argument probs of <a href="#">quantile</a> . Quantiles to set the other variables to. Default: <code>c(0.1, 0.5, 0.9)</code>
grid.resolution	Integer between 20 and 500. Resolution of the plotted curve Default: 100
verbose	Logical, if TRUE the plot is printed. Default: TRUE

### Details

All variables that are not plotted in a particular response curve are set to the values of their respective quantiles, and the response curve for each one of these quantiles is shown in the plot.

**Value**

A data frame with the following columns:

- response: Predicted values of the response, obtained with `stats::predict()`.
- predictor: Values of the given predictor.
- quantile: Grouping column, values of the quantiles at which the other predictors are set to generate the response curve.
- model: Model number, only relevant if the model was produced with `rf_repeat()`.
- predictor.name: Grouping variable, name of the predictor.
- response.name: Grouping variable, name of the response variable.

**See Also**

[plot\\_response\\_curves\(\)](#)

**Examples**

```
if(interactive()){  
  
  #loading example data  
  data(plant_richness_df)  
  
  #fitting random forest model  
  out <- rf(  
    data = plant_richness_df,  
    dependent.variable.name = "richness_species_vascular",  
    predictor.variable.names = colnames(plant_richness_df)[5:21],  
    n.cores = 1,  
    verbose = FALSE  
  )  
  
  #getting data frame with response curves  
  p <- get_response_curves(out)  
  head(p)  
  
}
```

---

get\_spatial\_predictors

*Gets the spatial predictors of a spatial model*

---

**Description**

Returns spatial predictors from a model fitted with `rf_spatial()` in order of importance.

**Usage**

```
get_spatial_predictors(model)
```

**Arguments**

model                    A model fitted with `rf_spatial()`.

**Value**

A data frame with the spatial predictors included in the model.

**Examples**

```
if(interactive()){

  #loading example data
  data(distance_matrix)
  data(plant_richness_df)

  #fittind spatial model
  model <- rf_spatial(
    data = plant_richness_df,
    dependent.variable.name = "richness_species_vascular",
    predictor.variable.names = colnames(plant_richness_df)[5:21],
    distance.matrix = distance_matrix,
    distance.thresholds = c(0, 1000),
    n.cores = 1,
    method = "mem.moran.sequential"
  )

  #getting data frame with the selected spatial predictors
  spatial.predictors <- get_spatial_predictors(model)
  head(spatial.predictors)

}
```

---

is\_binary

*Checks if dependent variable is binary with values 1 and 0*

---

**Description**

Returns TRUE if `dependent.variable.name` is a binary variable with the values 1 and 0.

**Usage**

```
is_binary(data = NULL, dependent.variable.name = NULL)
```

**Arguments**

data                    Data frame with a response variable and a set of predictors. Default: NULL

dependent.variable.name  
 Character string with the name of the response variable. Must be in the column names of data. Default: NULL

**Value**

Logical.

**Examples**

```
if(interactive()){  
  
  #dummy data frame  
  data <- data.frame(  
    response = c(0, 0, 0, 1, 1)  
  )  
  
  #checking if response is binary  
  is_binary(  
    data = data,  
    dependent.variable.name = "response"  
  )  
}
```

---

make\_spatial\_fold      *Makes one training and one testing spatial folds*

---

**Description**

Used internally by [make\\_spatial\\_folds\(\)](#) and [rf\\_evaluate\(\)](#). Uses the coordinates of a point `xy.i` to generate two spatially independent data folds from the data frame `xy`. It does so by growing a rectangular buffer from `xy.i` until a number of records defined by `training.fraction` is inside the buffer. The indices of these records are then stored as "training" in the output list. The indices of the remaining records outside of the buffer are stored as "testing". These training and testing records can be then used to evaluate a model on independent data via cross-validation.

**Usage**

```
make_spatial_fold(  
  data = NULL,  
  dependent.variable.name = NULL,  
  xy.i = NULL,  
  xy = NULL,  
  distance.step.x = NULL,  
  distance.step.y = NULL,  
  training.fraction = 0.8  
)
```

**Arguments**

<code>data</code>	Data frame with a response variable and a set of predictors. Default: NULL
<code>dependent.variable.name</code>	Character string with the name of the response variable. Must be in the column names of data. Default: NULL
<code>xy.i</code>	One row data frame with at least three columns: "x" (longitude), "y" (latitude), and "id" (integer, id of the record). Can be a row of xy. Default: NULL.
<code>xy</code>	A data frame with at least three columns: "x" (longitude), "y" (latitude), and "id" (integer, index of the record). Default: NULL.
<code>distance.step.x</code>	Numeric, distance step used during the growth in the x axis of the buffers defining the training folds. Default: NULL (1/1000th the range of the x coordinates).
<code>distance.step.y</code>	Numeric, distance step used during the growth in the y axis of the buffers defining the training folds. Default: NULL (1/1000th the range of the y coordinates).
<code>training.fraction</code>	Numeric, fraction of the data to be included in the training fold, Default: 0.8.

**Value**

A list with two slots named `training` and `testing` with the former having the indices of the training records selected from `xy`, and the latter having the indices of the testing records.

**See Also**

[make\\_spatial\\_folds\(\)](#), [rf\\_evaluate\(\)](#)

**Examples**

```
if(interactive()){

  #loading example data
  data(plant_richness_df)

  #getting case coordinates
  xy <- plant_richness_df[, 1:3]
  colnames(xy) <- c("id", "x", "y")

  #building a spatial fold centered in the first pair of coordinates
  out <- make_spatial_fold(
    xy.i = xy[1, ],
    xy = xy,
    training.fraction = 0.6
  )

  #indices of the training and testing folds
  out$training
  out$testing
}
```



```

#plotting the data
plot(xy[, c("x", "y")], type = "n", xlab = "", ylab = "")
#plots training points
points(xy[out$training, c("x", "y")], col = "red4", pch = 15)
#plots testing points
points(xy[out$testing, c("x", "y")], col = "blue4", pch = 15)
#plots xy.i
points(xy[1, c("x", "y")], col = "black", pch = 15, cex = 2)

}

```

---

make\_spatial\_folds      *Makes training and testing spatial folds*

---

### Description

Applies `make_spatial_fold()` to every record in a data frame `xy.selected` to generate as many spatially independent folds over the dataset `xy` as rows are in `xy.selected`.

### Usage

```

make_spatial_folds(
  data = NULL,
  dependent.variable.name = NULL,
  xy.selected = NULL,
  xy = NULL,
  distance.step.x = NULL,
  distance.step.y = NULL,
  training.fraction = 0.75,
  n.cores = parallel::detectCores() - 1,
  cluster = NULL
)

```

### Arguments

<code>data</code>	Data frame with a response variable and a set of predictors. Default: NULL
<code>dependent.variable.name</code>	Character string with the name of the response variable. Must be in the column names of <code>data</code> . Default: NULL
<code>xy.selected</code>	Data frame with at least three columns: "x" (longitude), "y" (latitude), and "id" (integer, id of the record). Usually a subset of <code>xy</code> . Usually the result of applying <code>thinning()</code> or <code>thinning_til_n()</code> to 'xy' Default: NULL.
<code>xy</code>	data frame with at least three columns: "x" (longitude), "y" (latitude), and "id" (integer, index of the record). Default: NULL.
<code>distance.step.x</code>	Numeric, distance step used during the growth in the x axis of the buffers defining the training folds. Default: NULL (1/1000th the range of the x coordinates).

distance.step.y	Numeric, distance step used during the growth in the y axis of the buffers defining the training folds. Default: NULL (1/1000th the range of the y coordinates).
training.fraction	numeric, fraction of the data to be included in the growing buffer as training data, Default: 0.75
n.cores	Integer, number of cores to use for parallel execution. Creates a socket cluster with <code>parallel::makeCluster()</code> , runs operations in parallel with <code>foreach</code> and <code>%dopar%</code> , and stops the cluster with <code>parallel::clusterStop()</code> when the job is done. Default: <code>parallel::detectCores() - 1</code>
cluster	A cluster definition generated with <code>parallel::makeCluster()</code> . If provided, overrides <code>n.cores</code> . When <code>cluster = NULL</code> (default value), and <code>model</code> is provided, the cluster in <code>model</code> , if any, is used instead. If this cluster is NULL, then the function uses <code>n.cores</code> instead. The function does not stop a provided cluster, so it should be stopped with <code>parallel::stopCluster()</code> afterwards. The cluster definition is stored in the output list under the name "cluster" so it can be passed to other functions via the <code>model</code> argument, or using the <code>%&gt;%</code> pipe. Default: NULL

**Value**

A list with as many slots as rows are in `xy.selected`. Each slot has two slots named `training` and `testing`, with the former having the indices of the training records selected from `xy`, and the latter having the indices of the testing records.

**See Also**

[make\\_spatial\\_fold\(\)](#), [rf\\_evaluate\(\)](#)

**Examples**

```
if(interactive()){
  #loading example data
  data(plant_richness_df)

  #getting case coordinates
  xy <- plant_richness_df[, 1:3]
  colnames(xy) <- c("id", "x", "y")

  #thinning til 20 cases
  xy.selected <- thinning_til_n(
    xy = xy,
    n = 20
  )

  #making spatial folds centered on these 20 cases
  out <- make_spatial_folds(
    xy.selected = xy.selected,
    xy = xy,
```

```

distance.step = 0.05, #degrees
training.fraction = 0.6,
n.cores = 1
)

#plotting training and testing folds
plot(xy[ c("x", "y")], type = "n", xlab = "", ylab = "")
#plots training points
points(xy[out[[10]]$training, c("x", "y")], col = "red4", pch = 15)
#plots testing points
points(xy[out[[10]]$testing, c("x", "y")], col = "blue4", pch = 15)
#plots xy.i
points(xy[10, c("x", "y")], col = "black", pch = 15, cex = 2)
}

```

mem

*Moran's Eigenvector Maps of a distance matrix***Description**

Computes the positive Moran's Eigenvector Maps of a distance matrix.

**Usage**

```

mem(
  distance.matrix = NULL,
  distance.threshold = 0,
  colnames.prefix = "mem"
)

```

**Arguments**

`distance.matrix`  
Distance matrix. Default: NULL.

`distance.threshold`  
Numeric vector with distance thresholds defining different neighborhood extents within the distance matrix, Default: 0

`colnames.prefix`  
Character, name prefix for the output columns. Default: "mem"

**Details**

Takes the distance matrix `x`, double-centers it with `double_center_distance_matrix()`, applies `eigen`, and returns eigenvectors with positive normalized eigenvalues (a.k.a Moran's Eigenvector Maps, or MEMs). These MEMs are later used as spatial predictors by `rf_spatial()`.

**Value**

A data frame with positive Moran's Eigenvector Maps.

**See Also**

[mem\\_multithreshold\(\)](#), [rf\\_spatial\(\)](#)

**Examples**

```
if(interactive()){  
  
  #loading example distance matrix  
  data(distance_matrix)  
  
  #Moran's Eigenvector Maps of the distance matrix  
  mem <- mem(x = distance_matrix)  
  
}
```

---

mem\_multithreshold      *Moran's Eigenvector Maps for different distance thresholds*

---

**Description**

Computes Moran's Eigenvector Maps of a distance matrix (using [mem\(\)](#)) over different distance thresholds.

**Usage**

```
mem_multithreshold(  
  distance.matrix = NULL,  
  distance.thresholds = NULL,  
  max.spatial.predictors = NULL  
)
```

**Arguments**

`distance.matrix`  
Distance matrix. Default: NULL.

`distance.thresholds`  
Numeric vector with distance thresholds defining neighborhood in the distance matrix, Default: NULL.

`max.spatial.predictors`  
Maximum number of spatial predictors to generate. Only useful to save memory when the distance matrix `x` is very large. Default: NULL.

**Details**

The function takes the distance matrix `x`, computes its weights at difference distance thresholds, double-centers the resulting weight matrices with [double\\_center\\_distance\\_matrix\(\)](#), applies [eigen](#) to each double-centered matrix, and returns eigenvectors with positive normalized eigenvalues for different distance thresholds.

**Value**

A data frame with as many rows as the distance matrix `x` containing positive Moran's Eigenvector Maps. The data frame columns are named "spatial\_predictor\_DISTANCE\_COLUMN", where DISTANCE is the given distance threshold, and COLUMN is the column index of the given spatial predictor.

**Examples**

```
if(interactive()){
  #loading example data
  data(distance_matrix)

  #computing Moran's eigenvector maps for 0, 1000, and 2000 km
  mem.df <- mem_multithreshold(
    distance.matrix = distance_matrix,
    distance.thresholds = c(0, 1000, 2000)
  )
  head(mem.df)
}
```

---

 moran

*Moran's I test*


---

**Description**

Computes the spatial correlation coefficient (Moran's I) of a vector given a distance matrix, and a distance threshold used to define "neighborhood".

**Usage**

```
moran(
  x = NULL,
  distance.matrix = NULL,
  distance.threshold = NULL,
  verbose = TRUE
)
```

**Arguments**

<code>x</code>	Numeric vector, generally model residuals, Default: NULL
<code>distance.matrix</code>	Distance matrix among cases in <code>x</code> . The number of rows of this matrix must be equal to the length of <code>x</code> . Default: NULL
<code>distance.threshold</code>	numeric value in the range of values available in <code>distance.matrix</code> . Distances below such threshold are set to 0. Default: NULL (which defaults to 0).
<code>verbose</code>	Logical, if TRUE, prints a Moran's I plot. Default: TRUE

## Details

Inspired in the Moran.I() function of the [ape](#) package.

## Value

A list with three named slots:

- test: Data frame with observed and expected Moran's I values, p-value, and interpretation.
- plot: Moran's plot of the vector x against the spatial lags of x.
- plot.df: Data used in the Moran's plot.

## See Also

[moran\\_multithreshold\(\)](#)

## Examples

```
if(interactive()){  
  
  #loading example data  
  data(distance_matrix)  
  data(plant_richness)  
  
  #Moran's I of the response variable  
  out <- moran(  
    x = plant_richness$richness_species_vascular,  
    distance.matrix = distance_matrix  
  )  
  out  
  
}
```

---

`moran_multithreshold` *Moran's I test on a numeric vector for different neighborhoods*

---

## Description

Applies [moran\(\)](#) to different distance thresholds at the same time.

## Usage

```
moran_multithreshold(  
  x = NULL,  
  distance.matrix = NULL,  
  distance.thresholds = NULL,  
  verbose = TRUE  
)
```

**Arguments**

<code>x</code>	Numeric vector, generally model residuals, Default: NULL
<code>distance.matrix</code>	Distance matrix among cases in <code>x</code> . The number of rows of this matrix must be equal to the length of <code>x</code> . Default: NULL
<code>distance.thresholds</code>	Numeric vector, distances below each value are set to 0 on separated copies of the distance matrix for the computation of Moran's I at different neighborhood distances. If NULL, it defaults to <code>seq(0, max(distance.matrix)/4, length.out = 2)</code> . Default: NULL
<code>verbose</code>	Logical, if TRUE, plots Moran's I values for each distance threshold. Default: TRUE

**Details**

Using different distance thresholds helps to take into account the uncertainty about what "neighborhood" means in ecological systems (1000km in geological time means little, but 100m might be quite a long distance for a tree to disperse seeds over), and allows to explore spatial autocorrelation of model residuals for several minimum-distance criteria at once.

**Value**

A named list with the slots:

- `df`: Data frame with the results of [moran](#) per distance threshold.
- `plot`: A plot of Moran's I across distance thresholds.
- `max.moran`: Maximum value of Moran's I across thresholds.
- `max.moran.distance.threshold`: Distance threshold with the maximum Moran's I value.

**See Also**

[moran\(\)](#)

**Examples**

```
if(interactive()){
  #loading example data
  data(distance_matrix)
  data(plant_richness)

  #computing Moran's I for the response variable at several reference distances
  out <- moran_multithreshold(
    x = plant_richness$richness_species_vascular,
    distance.matrix = distance_matrix,
    distance.thresholds = c(0, 100, 1000, 10000),
    plot = TRUE
  )
  out
}
```

```
}
```

---

objects_size	<i>Shows size of objects in the R environment</i>
--------------	---

---

### Description

Shows the size of the objects currently in the R environment. Helps to locate large objects cluttering the R environment and/or causing memory problems during the execution of large workflows.

### Usage

```
objects_size(n = 10)
```

### Arguments

n                      Number of objects to show, Default: 10

### Value

A data frame with the row names indicating the object name, the field 'Type' indicating the object type, 'Size' indicating the object size, and the columns 'Length/Rows' and 'Columns' indicating the object dimensions if applicable.

### Examples

```
if(interactive()){  
  
  #creating dummy objects  
  x <- matrix(runif(100), 10, 10)  
  y <- matrix(runif(10000), 100, 100)  
  
  #reading their in-memory size  
  objects_size()  
  
}
```



---

optimization\_function *Optimization equation to select spatial predictors*

---

### Description

Optimizes the selection of spatial predictors using two different methods: "moran.i", and "p.value".

### Usage

```
optimization_function(
  x = NULL,
  weight.r.squared = NULL,
  weight.penalization.n.predictors = NULL,
  optimization.method = "moran.i"
)
```

### Arguments

**x** Optimization data frame generated internally by [select\\_spatial\\_predictors\\_sequential\(\)](#) or [select\\_spatial\\_predictors\\_recursive\(\)](#). Default: NULL

**weight.r.squared** Numeric between 0 and 1, weight of R-squared in the optimization process. Default: NULL

**weight.penalization.n.predictors** Numeric between 0 and 1, weight of the penalization on the number of added spatial predictors. Default: NULL

**optimization.method** Character, one of "moran.i", and "p.value". Default: "moran.i"

### Details

The method "moran.i" tries to maximize  $1 - \text{Moran's I}$  while taking into account the R-squared of the model and a penalization on the number of introduced spatial predictors through the expression  $(1 - \text{Moran's I}) + w1 * \text{r.squared} - w2 * \text{penalization}$

The method "p.value" uses a binary version of the p-values of Moran's I (1 if  $\geq 0.05$ , 0 otherwise), and uses the expression

$\max(1 - \text{Moran's I}, \text{binary p-value}) + w1 * \text{r.squared} - w2 * \text{penalization}$

The "moran.i" method generally selects more spatial predictors than the "p.value" method.

### Value

A numeric vector with the optimization criteria.

### See Also

[select\\_spatial\\_predictors\\_recursive\(\)](#), [select\\_spatial\\_predictors\\_sequential\(\)](#)

**Description**

Extracts all factors of a principal component analysis of a matrix or data frame. Just a convenient wrapper for [prcomp](#).

**Usage**

```
pca(  
  x = NULL,  
  colnames.prefix = "pca_factor"  
)
```

**Arguments**

`x` numeric matrix or data frame, Default: NULL  
`colnames.prefix` character, name prefix for the output columns, Default: 'pca\_factor'

**Details**

Columns in `x` with zero variance are removed before computing the PCA.

**Value**

A data frame with the PCA factors of `x`.

**See Also**

[pca\\_multithreshold\(\)](#)

**Examples**

```
if(interactive()){  
  
  #load example distance matrix  
  data(distance_matrix)  
  
  #PCA of the distance matrix  
  out <- pca(x = distance_matrix)  
  out  
  
}
```

---

pca\_multithreshold      *PCA of a distance matrix over distance thresholds*

---

## Description

Computes PCA factors of a distance matrix over different distance thresholds to generate spatial predictors for a model fitted with [rf\\_spatial\(\)](#).

## Usage

```
pca_multithreshold(  
  distance.matrix = NULL,  
  distance.thresholds = NULL,  
  max.spatial.predictors = NULL  
)
```

## Arguments

`distance.matrix`  
Distance matrix. Default: NULL

`distance.thresholds`  
Numeric vector with distance thresholds defining neighborhood in the distance matrix, Default: 0

`max.spatial.predictors`  
Integer, maximum number of spatial predictors to generate. Only useful when the distance matrix `x` is very large. Default: NULL

## Details

The distance matrix is converted into weights with [weights\\_from\\_distance\\_matrix\(\)](#) before computing the PCA. This produces more meaningful spatial predictors than using the distance matrix as is.

## Value

A data frame with the PCA factors of the thresholded matrix. The data frame columns are named "spatial\_predictor\_DISTANCE\_COLUMN", where DISTANCE is the given distance threshold, and COLUMN is the column index of the given predictor.

## See Also

[pca\(\)](#)

**Examples**

```
if(interactive()){  
  
  #loading example distance matrix  
  load(distance_matrix)  
  
  #PCA factors of the distance matrix for two reference distances  
  x <- pca_multithreshold(  
    distance.matrix = distance_matrix,  
    distance.thresholds = c(0, 1000)  
  )  
  head(x)  
  
}
```

---

plant_richness_df	<i>Plant richness and predictors of American ecoregions</i>
-------------------	---

---

**Description**

Richness of vascular plants of the American ecoregions as defined in [Ecoregions 2017](#).

**Usage**

```
data(plant_richness_df)
```

**Format**

A data frame with 227 rows and 22 columns:

- `ecoregion_id`: Id of the ecoregion).
- `x`: Longitude in degrees (WGS84).
- `y`: Latitude in degrees (WGS84).
- `richness_species_vascular`: Number of vascular species found in the ecoregion. Response variable.
- `bias_area_km2`: Area of the ecoregion in squared kilometers.
- `bias_species_per_record`: Number of species divided by the number of spatial GBIF records available in the ecoregion as a measure of sampling bias.
- `climate_aridity_index_average`: Average of the ecoregion.
- `climate_hypervolume`: Volume of the climatic envelope of the ecoregion, computed with the [hypervolume](#) package.
- `climate_velocity_lgm_average`: Average climate velocity of the ecoregion since the Last Glacial Maximum.
- `neighbors_count`: Number of immediate neighbors of the ecoregion as a measure of connectivity/isolation.

- neighbors\_percent\_shared\_edge: Percentage of shared edge with the neighbors as a measure of connectivity/isolation.
- human\_population\_density: Population density of the ecoregion.
- topography\_elevation\_average: Average elevation of the ecoregion.
- landcover\_herbs\_percent\_average: Average cover percentage of herbs extracted from MODIS Vegetation Continuous Fields.
- fragmentation\_cohesion: Geographic fragmentation index of the ecoregion as computed with the R package [landscapemetrics](#).
- fragmentation\_division: Another fragmentation index.
- neighbors\_area: Total area of the ecoregions's immediate neighbors.
- human\_population: Human population in the ecoregion.
- human\_footprint\_average: Average human footprint in the ecoregion.
- climate\_bio1\_average: Average mean annual temperature.
- climate\_bio15\_minimum: Average precipitation seasonality.

### See Also

[distance\\_matrix](#)

---

plot\_evaluation

*Plots the results of a spatial cross-validation*

---

### Description

Plots the results of an spatial cross-validation performed with [rf\\_evaluate\(\)](#).

### Usage

```
plot_evaluation(  
  model,  
  fill.color = viridis::viridis(  
    3,  
    option = "F",  
    alpha = 0.8,  
    direction = -1  
  ),  
  line.color = "gray30",  
  verbose = TRUE,  
  notch = TRUE  
)
```

**Arguments**

model	A model resulting from <code>rf_evaluate()</code> .
fill.color	Character vector with three hexadecimal codes (e.g. "#440154FF" "#21908CFF" "#FDE725FF"), or function generating a palette (e.g. <code>viridis::viridis(3)</code> ). Default: <code>viridis::viridis(3, option = "F", alpha = 0.8, direction = -1)</code>
line.color	Character string, color of the line produced by <code>ggplot2::geom_smooth()</code> . Default: "gray30"
verbose	Logical, if TRUE the plot is printed. Default: TRUE
notch	Logical, if TRUE, boxplot notches are plotted. Default: TRUE

**Value**

A `ggplot`.

**See Also**

[rf\\_evaluate\(\)](#), [get\\_evaluation\(\)](#), [print\\_evaluation\(\)](#).

**Examples**

```
if(interactive()){

#loading example data
data(plant_richness_df)
data(distance_matrix)

#fitting a random forest model
rf.model <- rf(
  data = plant_richness_df,
  dependent.variable.name = "richness_species_vascular",
  predictor.variable.names = colnames(plant_richness_df)[5:21],
  distance.matrix = distance_matrix,
  distance.thresholds = 0,
  n.cores = 1,
  verbose = FALSE
)

#evaluating the model with spatial cross-validation
rf.model <- rf_evaluate(
  model = rf.model,
  xy = plant_richness_df[, c("x", "y")],
  n.cores = 1
)

#plotting the evaluation results
plot_evaluation(rf.model)

}
```

---

plot_importance	<i>Plots the variable importance of a model</i>
-----------------	---

---

### Description

Plots variable importance scores of `rf()`, `rf_repeat()`, and `rf_spatial()` models. Distributions of importance scores produced with `rf_repeat()` are plotted using `ggplot2::geom_violin`, which shows the median of the density estimate rather than the actual median of the data. However, the violin plots are ordered from top to bottom by the real median of the data to make small differences in median importance easier to spot. This function does not plot the result of `rf_importance()` yet, but you can find it under `model$importance$cv.per.variable.plot`.

### Usage

```
plot_importance(  
  model,  
  fill.color = viridis::viridis(  
    100,  
    option = "F",  
    direction = -1,  
    alpha = 1,  
    end = 0.9  
  ),  
  line.color = "white",  
  verbose = TRUE  
)
```

### Arguments

<code>model</code>	A model fitted with <code>rf()</code> , <code>rf_repeat()</code> , or <code>rf_spatial()</code> , or a data frame with variable importance scores (only for internal use within the package functions).
<code>fill.color</code>	Character vector with hexadecimal codes (e.g. "#440154FF" "#21908CFF" "#FDE725FF"), or function generating a palette (e.g. <code>viridis::viridis(100)</code> ). Default: <code>viridis::viridis(100, option = "F", direction = -1, alpha = 0.8, end = 0.9)</code>
<code>line.color</code>	Character string, color of the line produced by <code>ggplot2::geom_smooth()</code> . Default: "white"
<code>verbose</code>	Logical, if TRUE, the plot is printed. Default: TRUE

### Value

A `ggplot`.

### See Also

[print\\_importance\(\)](#), [get\\_importance\(\)](#)

## Examples

```
if(interactive()){

#loading example data
data(plant_richness_df)
data(distance_matrix)

#fitting a random forest model
rf.model <- rf(
  data = plant_richness_df,
  dependent.variable.name = "richness_species_vascular",
  predictor.variable.names = colnames(plant_richness_df)[5:21],
  distance.matrix = distance_matrix,
  distance.thresholds = 0,
  n.cores = 1,
  verbose = FALSE
)

#plotting variable importance scores
plot_importance(model = rf.model)

}
```

---

plot\_moran

*Plots a Moran's I test of model residuals*

---

## Description

Plots the results of spatial autocorrelation tests for a variety of functions within the package. The x axis represents the Moran's I estimate, the y axis contains the values of the distance thresholds, the dot sizes represent the p-values of the Moran's I estimate, and the red dashed line represents the theoretical null value of the Moran's I estimate.

## Usage

```
plot_moran(
  model,
  point.color = viridis::viridis(
    100,
    option = "F",
    direction = -1
  ),
  line.color = "gray30",
  option = 1,
  ncol = 1,
  verbose = TRUE
)
```



**Arguments**

model	A model fitted with <code>rf()</code> , <code>rf_repeat()</code> , or <code>rf_spatial()</code> , or a data frame generated by <code>moran()</code> . Default: NULL
point.color	Colors of the plotted points. Can be a single color name (e.g. "red4"), a character vector with hexadecimal codes (e.g. "#440154FF" "#21908CFF" "#FDE725FF"), or function generating a palette (e.g. <code>viridis::viridis(100)</code> ). Default: <code>viridis::viridis(100, option = "F")</code>
line.color	Character string, color of the line produced by <code>ggplot2::geom_smooth()</code> . Default: "gray30"
option	Integer, type of plot. If 1 (default) a line plot with Moran's I and p-values across distance thresholds is returned. If 2, scatterplots of residuals versus lagged residuals per distance threshold and their corresponding slopes are returned. In models fitted with <code>rf_repeat()</code> , the residuals and lags of the residuals are computed from the median residuals across repetitions. Option 2 is disabled if x is a data frame generated by <code>moran()</code> .
ncol	Number of columns of the plot. Only relevant when <code>option = 2</code> . Argument <code>ncol</code> of <code>wrap_plots</code> .
verbose	Logical, if TRUE, the resulting plot is printed, Default: TRUE

**Value**

A ggplot.

**See Also**

`moran()`, `moran_multithreshold()`

**Examples**

```
if(interactive()){

  #loading example data
  data(plant_richness_df)
  data(distance.matrix)

  #fitting a random forest model
  rf.model <- rf(
    data = plant_richness_df,
    dependent.variable.name = "richness_species_vascular",
    predictor.variable.names = colnames(plant_richness_df)[5:21],
    distance.matrix = distance_matrix,
    distance.thresholds = c(0, 1000, 2000),
    n.cores = 1,
    verbose = FALSE
  )

  #Incremental/multiscale Moran's I
  plot_moran(rf.model)
```

```
#Moran's scatterplot
plot_moran(rf.model, option = 2)

}
```

---

plot\_optimization      *Optimization plot of a selection of spatial predictors*

---

### Description

Plots optimization data frames produced by [select\\_spatial\\_predictors\\_sequential\(\)](#) and [select\\_spatial\\_predictors\\_recursive\(\)](#).

### Usage

```
plot_optimization(
  model,
  point.color = viridis::viridis(
    100,
    option = "F",
    direction = -1
  ),
  verbose = TRUE
)
```

### Arguments

model	A model produced by <a href="#">rf_spatial()</a> , or an optimization data frame produced by <a href="#">select_spatial_predictors_sequential()</a> or <a href="#">select_spatial_predictors_recursive()</a> .
point.color	Colors of the plotted points. Can be a single color name (e.g. "red4"), a character vector with hexadecimal codes (e.g. "#440154FF" "#21908CFF" "#FDE725FF"), or function generating a palette (e.g. <code>viridis::viridis(100)</code> ). Default: <code>viridis::viridis(100, option = "F", direction = -1)</code>
verbose	Logical, if TRUE the plot is printed. Default: TRUE

### Details

If the method used to fit a model with [rf\\_spatial\(\)](#) is "hengl", the function returns nothing, as this method does not require optimization.

### Value

A ggplot.

**Examples**

```
if(interactive()){

  #loading example data
  data(distance_matrix)
  data(plant_richness_df)

  #names of the response and predictors
  dependent.variable.name <- "richness_species_vascular"
  predictor.variable.names <- colnames(plant_richness_df)[5:21]

  #spatial model
  model <- rf_spatial(
    data = plant_richness_df,
    dependent.variable.name = dependent.variable.name,
    predictor.variable.names = predictor.variable.names,
    distance.matrix = distance_matrix,
    distance.thresholds = 0,
    method = "mem.moran.sequential",
    n.cores = 1,
    seed = 1
  )

  #plotting selection of spatial predictors
  plot_optimization(model = model)

}
```

---

plot\_residuals\_diagnostics

*Plot residuals diagnostics*

---

**Description**

Plots normality and autocorrelation tests of model residuals.

**Usage**

```
plot_residuals_diagnostics(
  model,
  point.color = viridis::viridis(100, option = "F"),
  line.color = "gray10",
  fill.color = viridis::viridis(4, option = "F", alpha = 0.95)[2],
  option = 1,
  ncol = 1,
  verbose = TRUE
)
```

**Arguments**

model	A model produced by <code>rf()</code> , <code>rf_repeat()</code> , or <code>rf_spatial()</code> .
point.color	Colors of the plotted points. Can be a single color name (e.g. "red4"), a character vector with hexadecimal codes (e.g. "#440154FF" "#21908CFF" "#FDE725FF"), or function generating a palette (e.g. <code>viridis::viridis(100)</code> ). Default: <code>viridis::viridis(100, option = "F")</code>
line.color	Character string, color of the line produced by <code>ggplot2::geom_smooth()</code> . Default: "gray30"
fill.color	Character string, fill color of the bars produced by <code>ggplot2::geom_histogram()</code> . Default: <code>viridis::viridis(4, option = "F", alpha = 0.95)[2]</code>
option	(argument of <code>plot_moran()</code> ) Integer, type of plot. If 1 (default) a line plot with Moran's I and p-values across distance thresholds is returned. If 2, scatterplots of residuals versus lagged residuals per distance threshold and their corresponding slopes are returned. In models fitted with <code>rf_repeat()</code> , the residuals and lags of the residuals are computed from the median residuals across repetitions. Option 2 is disabled if <code>x</code> is a data frame generated by <code>moran()</code> .
ncol	(argument of <code>plot_moran()</code> ) Number of columns of the Moran's I plot if option = 2.
verbose	Logical, if TRUE, the resulting plot is printed, Default: TRUE

**Value**

A patchwork object.

**Examples**

```
if(interactive()){

  #load example data
  data(plant_richness_df)
  data(distance_matrix)

  #fit a random forest model
  x <- rf(
    data = plant_richness_df,
    dependent.variable.name = "richness_species_vascular",
    predictor.variable.names = colnames(plant_richness_df)[5:21],
    distance.matrix = distance_matrix,
    n.cores = 1
  )

  #residuals diagnostics
  plot_residuals_diagnostics(x)

}
```

---

plot\_response\_curves *Plots the response curves of a model.*

---

### Description

Plots the response curves of models fitted with `rf()`, `rf_repeat()`, or `rf_spatial()`.

### Usage

```
plot_response_curves(
  model = NULL,
  variables = NULL,
  quantiles = c(0.1, 0.5, 0.9),
  grid.resolution = 200,
  line.color = viridis::viridis(length(quantiles), option = "F", end = 0.9),
  ncol = 2,
  show.data = FALSE,
  verbose = TRUE
)
```

### Arguments

model	A model fitted with <code>rf()</code> , <code>rf_repeat()</code> , or <code>rf_spatial()</code> .
variables	Character vector, names of predictors to plot. If NULL, the most important variables (importance higher than the median) in <code>x</code> are selected. Default: NULL.
quantiles	Numeric vector with values between 0 and 1, argument probs of <a href="#">quantile</a> . Quantiles to set the other variables to. Default: <code>c(0.1, 0.5, 0.9)</code>
grid.resolution	Integer between 20 and 500. Resolution of the plotted curve Default: 100
line.color	Character vector with colors, or function to generate colors for the lines representing quantiles. Must have the same number of colors as quantiles are defined. Default: <code>viridis::viridis(length(quantiles), option = "F", end = 0.9)</code>
ncol	Integer, argument of <a href="#">wrap_plots</a> . Defaults to the rounded squared root of the number of plots. Default: 2
show.data	Logical, if TRUE, the observed data is plotted along with the response curves. Default: FALSE
verbose	Logical, if TRUE the plot is printed. Default: TRUE

### Details

All variables that are not plotted in a particular response curve are set to the values of their respective quantiles, and the response curve for each one of these quantiles is shown in the plot. When the input model was fitted with `rf_repeat()` with `keep.models = TRUE`, then the plot shows the median of all model runs, and each model run separately as a thinner line. The output list can be plotted all at once with `patchwork::wrap_plots(p)` or `cowplot::plot_grid(plotlist = p)`, or one by one by extracting each plot from the list.

**Value**

A list with slots named after the selected variables, with one ggplot each.

**See Also**

[plot\\_response\\_surface\(\)](#)

**Examples**

```
if(interactive()){  
  
  #loading example data  
  data(plant_richness_df)  
  
  #fitting a random forest model  
  m <- rf(  
    data = plant_richness_df,  
    dependent.variable.name = "richness_species_vascular",  
    predictor.variable.names = colnames(plant_richness_df)[5:21],  
    n.cores = 1,  
    verbose = FALSE  
  )  
  
  #response curves of most important predictors  
  plot_response_curves(model = m)  
  
}
```

---

`plot_response_surface` *Plots the response surfaces of a random forest model*

---

**Description**

Plots response surfaces for any given pair of predictors in a [rf\(\)](#), [rf\\_repeat\(\)](#), or [rf\\_spatial\(\)](#) model.

**Usage**

```
plot_response_surface(  
  model = NULL,  
  a = NULL,  
  b = NULL,  
  quantiles = 0.5,  
  grid.resolution = 100,  
  point.size.range = c(0.5, 2.5),  
  point.alpha = 1,  
  fill.color = viridis::viridis(100, option = "F", direction = -1, alpha = 0.9),  
  point.color = "gray30",  
  verbose = TRUE  
)
```

**Arguments**

model	A model fitted with <code>rf()</code> , <code>rf_repeat()</code> , or <code>rf_spatial()</code> . Default NULL
a	Character string, name of a model predictor. If NULL, the most important variable in model is selected. Default: NULL
b	Character string, name of a model predictor. If NULL, the second most important variable in model is selected. Default: NULL
quantiles	Numeric vector between 0 and 1. Argument probs of the function <a href="#">quantile</a> . Quantiles to set the other variables to. Default: 0.5
grid.resolution	Integer between 20 and 500. Resolution of the plotted surface Default: 100
point.size.range	Numeric vector of length 2 with the range of point sizes used by <a href="#">geom_point</a> . Using <code>c(-1, -1)</code> removes the points. Default: <code>c(0.5, 2.5)</code>
point.alpha	Numeric between 0 and 1, transparency of the points. Setting it to 0 removes all points. Default: 1.
fill.color	Character vector with hexadecimal codes (e.g. "#440154FF" "#21908CFF" "#FDE725FF"), or function generating a palette (e.g. <code>viridis::viridis(100)</code> ). Default: <code>viridis::viridis(100, option = "F", direction = -1, alpha = 0.9)</code>
point.color	Character vector with a color name (e.g. "red4"). Default: gray30
verbose	Logical, if TRUE the plot is printed. Default: TRUE

**Details**

All variables that are not a or b in a response curve are set to the values of their respective quantiles to plot the response surfaces. The output list can be plotted all at once with `patchwork::wrap_plots(p)` or `cowplot::plot_grid(plotlist = p)`, or one by one by extracting each plot from the list.

**Value**

A list with slots named after the selected quantiles, each one with a ggplot.

**See Also**

[plot\\_response\\_curves\(\)](#)

**Examples**

```
if(interactive()){

#load example data
data(plant_richness_df)

#fit random forest model
out <- rf(
  data = plant_richness_df,
  dependent.variable.name = "richness_species_vascular",
  predictor.variable.names = colnames(plant_richness_df)[5:21],
```

```

    n.cores = 1,
    verbose = FALSE
  )

  #plot interactions between most important predictors
  plot_response_surfaces(x = out)

}

```

---

plot\_training\_df      *Scatterplots of a training data frame*

---

### Description

Plots the dependent variable against each predictor.

### Usage

```

plot_training_df(
  data = NULL,
  dependent.variable.name = NULL,
  predictor.variable.names = NULL,
  ncol = 4,
  method = "loess",
  point.color = viridis::viridis(100, option = "F"),
  line.color = "gray30"
)

```

### Arguments

data	Data frame with a response variable and a set of predictors. Default: NULL
dependent.variable.name	Character string with the name of the response variable. Must be in the column names of data. If the dependent variable is binary with values 1 and 0, the argument case.weights of ranger is populated by the function <a href="#">case_weights()</a> . Default: NULL
predictor.variable.names	Character vector with the names of the predictive variables. Every element of this vector must be in the column names of data. Optionally, the result of <a href="#">auto_cor()</a> or <a href="#">auto_vif()</a> Default: NULL
ncol	Number of columns of the plot. Argument ncol of <a href="#">wrap_plots</a> .
method	Method for <a href="#">geom_smooth</a> , one of: "lm", "glm", "gam", "loess", or a function, for example <code>mgcv::gam</code> Default: 'loess'
point.color	Colors of the plotted points. Can be a single color name (e.g. "red4"), a character vector with hexadecimal codes (e.g. "#440154FF" "#21908CFF" "#FDE725FF"), or function generating a palette (e.g. <code>viridis::viridis(100)</code> ). Default: <code>viridis::viridis(100, option = "F")</code>



line.color      Character string, color of the line produced by ggplot2::geom\_smooth(). Default: "gray30"

## Value

A `wrap_plots` object.

## Examples

```
if(interactive()){  
  
  #load example data  
  data(plant_richness_df)  
  
  #scatterplot of the training data  
  plot_training_data(  
    data = plant_richness_df,  
    dependent.variable.name = "richness_species_vascular",  
    predictor.variable.names = colnames(plant_richness_df)[5:21]  
  )  
}
```

---

plot\_training\_df\_moran

*Moran's I plots of a training data frame*

---

## Description

Plots the the Moran's I test of the response and the predictors in a training data frame.

## Usage

```
plot_training_df_moran(  
  data = NULL,  
  dependent.variable.name = NULL,  
  predictor.variable.names = NULL,  
  distance.matrix = NULL,  
  distance.thresholds = NULL,  
  fill.color = viridis::viridis(100, option = "F", direction = -1),  
  point.color = "gray30"  
)
```

## Arguments

data              Data frame with a response variable and a set of predictors. Default: NULL

<code>dependent.variable.name</code>	Character string with the name of the response variable. Must be in the column names of data. If the dependent variable is binary with values 1 and 0, the argument <code>case.weights</code> of <code>ranger</code> is populated by the function <code>case_weights()</code> . Default: NULL
<code>predictor.variable.names</code>	Character vector with the names of the predictive variables. Every element of this vector must be in the column names of data. Optionally, the result of <code>auto_cor()</code> or <code>auto_vif()</code> Default: NULL
<code>distance.matrix</code>	Squared matrix with the distances among the records in data. The number of rows of <code>distance.matrix</code> and data must be the same. If not provided, the computation of the Moran's I of the residuals is omitted. Default: NULL
<code>distance.thresholds</code>	Numeric vector, distances below each value are set to 0 on separated copies of the distance matrix for the computation of Moran's I at different neighborhood distances. If NULL, it defaults to <code>seq(0, max(distance.matrix)/4, length.out = 2)</code> . Default: NULL
<code>fill.color</code>	Character vector with hexadecimal codes (e.g. "#440154FF" "#21908CFF" "#FDE725FF"), or function generating a palette (e.g. <code>viridis::viridis(100)</code> ). Default: <code>viridis::viridis(100, option = "F", direction = -1)</code>
<code>point.color</code>	Character vector with a color name (e.g. "red4"). Default: gray30

**Value**

A `ggplot2` object.

**Examples**

```
if(interactive()){

  #load example data
  data(plant_richness_df)
  data(distance_matrix)

  #plot Moran's I of training data
  plot_moran_training_data(
    data = plant_richness_df,
    dependent.variable.name = "richness_species_vascular",
    predictor.variable.names = colnames(plant_richness_df)[5:21],
    distance.matrix = distance_matrix,
    distance.thresholds = c(
      0,
      2000,
      4000,
      6000,
      8000
    )
  )
}
```

---

plot\_tuning                      *Plots a tuning object produced by rf\_tuning()*

---

### Description

Plots the tuning of the hyperparameters `num.trees`, `mtry`, and `min.node.size` performed by `rf_tuning()`.

### Usage

```
plot_tuning(  
  model,  
  point.color = viridis::viridis(  
    100,  
    option = "F"  
  ),  
  verbose = TRUE  
)
```

### Arguments

<code>model</code>	A model fitted with <code>rf_tuning()</code> . Default: NULL
<code>point.color</code>	Colors of the plotted points. Can be a single color name (e.g. "red4"), a character vector with hexadecimal codes (e.g. "#440154FF" "#21908CFF" "#FDE725FF"), or function generating a palette (e.g. <code>viridis::viridis(100)</code> ). Default: <code>viridis::viridis(100, option = "F")</code>
<code>verbose</code>	Logical, if TRUE, the plot is printed. Default: TRUE

### Value

A `ggplot`.

### See Also

[rf\\_tuning\(\)](#)

### Examples

```
if(interactive()){  
  
  #load example data  
  data(plant_richness_df)  
  
  #fit random forest model  
  rf.model <- rf(  
    data = plant_richness_df,  
    dependent.variable.name = "richness_species_vascular",  
    predictor.variable.names = colnames(plant_richness_df)[5:21],
```

```
distance.matrix = distance_matrix,
distance.thresholds = 0,
n.cores = 1,
verbose = FALSE
)

#tune random forest model
rf.model <- rf_tuning(
  model = rf.model,
  xy = plant_richness_df[, c("x", "y")],
  n.cores = 1,
  verbose = FALSE
)

#generate tuning plot
plot_tuning(model = rf.model)

}
```

---

prepare\_importance\_spatial

*Prepares variable importance objects for spatial models*

---

## Description

Prepares variable importance data frames and plots for models fitted with `rf_spatial()`.

## Usage

```
prepare_importance_spatial(model)
```

## Arguments

`model` An importance data frame with spatial predictors, or a model fitted with `rf_spatial()`.

## Value

A list with importance data frames in different formats depending on whether the model was fitted with `rf()` or `rf_repeat()`.

## Examples

```
if(interactive()){

  #loading example data
  data(distance_matrix)
  data(plant_richness_df)

  #fittind spatial model
  model <- rf_spatial(
```

```

    data = plant_richness_df,
    dependent.variable.name = "richness_species_vascular",
    predictor.variable.names = colnames(plant_richness_df)[5:21],
    distance.matrix = distance_matrix,
    distance.thresholds = 0,
    n.cores = 1
  )

  #preparing the importance data frame
  importance <- prepare_importance_spatial(model)
  names(importance)

}

```

---

print.rf

*Custom print method for random forest models*


---

### Description

Custom print method for models fitted with `rf()`, `rf_repeat()`, and `rf_spatial()`.

### Usage

```
## S3 method for class 'rf'
print(x, ...)
```

### Arguments

`x` A model fitted with `rf()`, `rf_repeat()`, or `rf_spatial()`.  
`...` Additional arguments for print methods.

### Value

Prints model details to the console.

### See Also

[print\\_evaluation\(\)](#), [print\\_importance\(\)](#), [print\\_moran\(\)](#), [print\\_performance\(\)](#)

### Examples

```

if(interactive()){

  #loading example data
  data("plant_richness_df")
  data("distance_matrix")

  #fitting random forest model
  rf.model <- rf(

```

```
data = plant_richness_df,
dependent.variable.name = "richness_species_vascular",
predictor.variable.names = colnames(plant_richness_df)[5:21],
distance.matrix = distance_matrix,
distance.thresholds = 0,
n.cores = 1
)

#printing model summary
print(rf.model)
}
```

---

print_evaluation	<i>Prints cross-validation results</i>
------------------	--

---

### Description

Prints the results of an spatial cross-validation performed with [rf\\_evaluate\(\)](#).

### Usage

```
print_evaluation(model)
```

### Arguments

model            A model resulting from [rf\\_evaluate\(\)](#).

### Value

A table printed to the standard output.

### See Also

[plot\\_evaluation\(\)](#), [get\\_evaluation\(\)](#)

### Examples

```
if(interactive()){

#loading example data
data(plant_richness_df)
data(distance_matrix)

#fitting random forest model
rf.model <- rf(
  data = plant_richness_df,
  dependent.variable.name = "richness_species_vascular",
  predictor.variable.names = colnames(plant_richness_df)[5:21],
  distance.matrix = distance_matrix,
  distance.thresholds = 0,
```

```
n.cores = 1,  
verbose = FALSE  
)  
  
#evaluation with spatial cross-validation  
rf.model <- rf_evaluate(  
  model = rf.model,  
  xy = plant_richness_df[, c("x", "y")],  
  n.cores = 1  
)  
  
#checking evaluation results  
print_evaluation(rf.model)  
  
}
```

---

print_importance	<i>Prints variable importance</i>
------------------	-----------------------------------

---

### Description

Prints variable importance scores from [rf](#), [rf\\_repeat](#), and [rf\\_spatial](#) models.

### Usage

```
print_importance(  
  model,  
  verbose = TRUE  
)
```

### Arguments

model	A model fitted with <a href="#">rf</a> , <a href="#">rf_repeat</a> , or <a href="#">rf_spatial</a> .
verbose	Logical, if TRUE, variable importance is returned. Default: TRUE

### Value

A table printed to the standard output.

### See Also

[plot\\_importance\(\)](#), [get\\_importance\(\)](#)

## Examples

```
if(interactive()){

#loading example data
data(plant_richness_df)
data(distance.matrix)

#fitting a random forest model
rf.model <- rf(
  data = plant_richness_df,
  dependent.variable.name = "richness_species_vascular",
  predictor.variable.names = colnames(plant_richness_df)[5:21],
  distance.matrix = distance_matrix,
  distance.thresholds = 0,
  n.cores = 1,
  verbose = FALSE
)

#printing variable importance scores
print_importance(model = rf.model)

}
```

---

print\_moran

*Prints results of a Moran's I test*

---

## Description

Prints the results of a Moran's I test on the residuals of a model.

## Usage

```
print_moran(
  model,
  caption = NULL,
  verbose = TRUE
)
```

## Arguments

model	A model fitted with <code>rf()</code> , <code>rf_repeat()</code> , or <code>rf_spatial()</code> .
caption	Character, caption of the output table, Default: NULL
verbose	Logical, if TRUE, the resulting table is printed into the console, Default: TRUE

## Value

Prints a table in the console using the [huxtable](#) package.



**See Also**

[moran\(\)](#), [moran\\_multithreshold\(\)](#), [get\\_moran\(\)](#), [plot\\_moran\(\)](#)

**Examples**

```
if(interactive()){

  #loading example data
  data(plant_richness_df)
  data(distance.matrix)

  #fitting random forest model
  rf.model <- rf(
    data = plant_richness_df,
    dependent.variable.name = "richness_species_vascular",
    predictor.variable.names = colnames(plant_richness_df)[5:21],
    distance.matrix = distance_matrix,
    distance.thresholds = c(0, 1000, 2000),
    n.cores = 1,
    verbose = FALSE
  )

  #printing Moran's I of model's residuals
  print_moran(rf.model)

}
```

---

print\_performance      *print\_performance*

---

**Description**

Prints the performance slot of a model fitted with [rf\(\)](#), [rf\\_repeat\(\)](#), or [rf\\_spatial\(\)](#). For models fitted with [rf\\_repeat\(\)](#) it shows the median and the median absolute deviation of each performance measure.

**Usage**

```
print_performance(model)
```

**Arguments**

model                      Model fitted with [rf\(\)](#), [rf\\_repeat\(\)](#), or [rf\\_spatial\(\)](#).

**Value**

Prints model performance scores to the console.

**See Also**

[print\\_performance\(\)](#), [get\\_performance\(\)](#)

**Examples**

```
if(interactive()){

  #loading example data
  data(plant_richness_df)
  data(distance.matrix)

  #fitting a random forest model
  rf.model <- rf(
    data = plant_richness_df,
    dependent.variable.name = "richness_species_vascular",
    predictor.variable.names = colnames(plant_richness_df)[5:21],
    distance.matrix = distance_matrix,
    distance.thresholds = 0,
    n.cores = 1,
    verbose = FALSE
  )

  #printing performance scores
  print_performance(rf.model)

}
```

---

rank\_spatial\_predictors

*Ranks spatial predictors*

---

**Description**

Ranks spatial predictors generated by [mem\\_multithreshold\(\)](#) or [pca\\_multithreshold\(\)](#) by their effect in reducing the Moran's I of the model residuals (`ranking.method = "effect"`), or by their own Moran's I (`ranking.method = "moran"`).

In the former case, one model of the type  $y \sim \text{predictors} + \text{spatial\_predictor}_X$  is fitted per spatial predictor, and the Moran's I of this model's residuals is compared with the one of the model without spatial predictors ( $y \sim \text{predictors}$ ), to finally rank the spatial predictor from maximum to minimum difference in Moran's I.

In the latter case, the spatial predictors are ordered by their Moran's I alone (this is the faster option).

In both cases, spatial predictors that are redundant with others at a Pearson correlation  $> 0.5$  and spatial predictors with no effect (no reduction of Moran's I or Moran's I of the spatial predictor equal or lower than 0) are removed.

This function has been designed to be used internally by [rf\\_spatial\(\)](#) rather than directly by a user.

**Usage**

```
rank_spatial_predictors(
  data = NULL,
  dependent.variable.name = NULL,
  predictor.variable.names = NULL,
  distance.matrix = NULL,
  distance.thresholds = NULL,
  ranger.arguments = NULL,
  spatial.predictors.df = NULL,
  ranking.method = c("moran", "effect"),
  reference.moran.i = 1,
  verbose = FALSE,
  n.cores = parallel::detectCores() - 1,
  cluster = NULL
)
```

**Arguments**

`data` Data frame with a response variable and a set of predictors. Default: NULL

`dependent.variable.name` Character string with the name of the response variable. Must be in the column names of data. Default: NULL

`predictor.variable.names` Character vector with the names of the predictive variables. Every element of this vector must be in the column names of data. Default: NULL

`distance.matrix` Squared matrix with the distances among the records in data. The number of rows of `distance.matrix` and `data` must be the same. If not provided, the computation of the Moran's I of the residuals is omitted. Default: NULL

`distance.thresholds` Numeric vector with neighborhood distances. All distances in the distance matrix below each value in `distance.thresholds` are set to 0 for the computation of Moran's I. If NULL, it defaults to `seq(0, max(distance.matrix), length.out = 4)`. Default: NULL

`ranger.arguments` List with [ranger](#) arguments. See [rf](#) or [rf\\_repeat](#) for further details.

`spatial.predictors.df` Data frame of spatial predictors.

`ranking.method` Character, method used by to rank spatial predictors. The method "effect" ranks spatial predictors according how much each predictor reduces Moran's I of the model residuals, while the method "moran" ranks them by their own Moran's I. Default: "moran".

`reference.moran.i` Moran's I of the residuals of the model without spatial predictors. Default: 1

`verbose` Logical, ff TRUE, messages and plots generated during the execution of the function are displayed, Default: TRUE

n.cores	Integer, number of cores to use for parallel execution. Creates a socket cluster with <code>parallel::makeCluster()</code> , runs operations in parallel with <code>foreach</code> and <code>%dopar%</code> , and stops the cluster with <code>parallel::clusterStop()</code> when the job is done. Default: <code>parallel::detectCores() - 1</code>
cluster	A cluster definition generated with <code>parallel::makeCluster()</code> . If provided, overrides <code>n.cores</code> . When <code>cluster = NULL</code> (default value), and <code>model</code> is provided, the cluster in <code>model</code> , if any, is used instead. If this cluster is <code>NULL</code> , then the function uses <code>n.cores</code> instead. The function does not stop a provided cluster, so it should be stopped with <code>parallel::stopCluster()</code> afterwards. The cluster definition is stored in the output list under the name "cluster" so it can be passed to other functions via the <code>model</code> argument, or using the <code>%&gt;%</code> pipe. Default: <code>NULL</code>

### Value

A list with four slots:

- `method`: Character, name of the method used to rank the spatial predictors.
- `criteria`: Data frame with two different configurations depending on the ranking method. If `ranking.method = "effect"`, the columns contain the names of the spatial predictors, the `r`-squared of the model, the Moran's I of the model residuals, the difference between the Moran's I of the model including the given spatial predictor, and the Moran's I of the model fitted without spatial predictors, and the interpretation of the Moran's I value. If `ranking.method = "moran"`, only the name of the spatial predictor and its Moran's I are in the output data frame.
- `ranking`: Ordered character vector with the names of the spatial predictors selected.
- `spatial.predictors.df`: data frame with the selected spatial predictors in the order of the ranking.

### Examples

```
if(interactive()){

  #loading distance matrix
  data(distance_matrix)

  #computing Moran's Eigenvector Maps
  mem.df <- mem(
    distance.matrix = distance_matrix[1:50, 1:50],
    distance.threshold = 0
  )

  #ranking by the Moran's I of the spatial predictor
  rank <- rank_spatial_predictors(
    distance.matrix = distance_matrix[1:50, 1:50],
    distance.thresholds = 0,
    spatial.predictors.df = mem.df,
    ranking.method = "moran",
    n.cores = 1
  )
}
```

```
#checking Moran's I of MEMs
rank$criteria

#checking rank of MEMs
rank$ranking
}
```

---

rescale_vector	<i>Rescales a numeric vector into a new range</i>
----------------	---

---

### Description

Rescales a numeric vector to a new range.

### Usage

```
rescale_vector(  
  x = NULL,  
  new.min = 0,  
  new.max = 1,  
  integer = FALSE  
)
```

### Arguments

x	Numeric vector. Default: NULL
new.min	New minimum value. Default: 0
new.max	New maximum value. Default: 1
integer	Logical, if TRUE, coerces the output to integer. Default: FALSE

### Value

A numeric vector of the same length as x, but with its values rescaled between new.min and new.max.

### Examples

```
if(interactive()){  
  
  out <- rescale_vector(  
    x = rnorm(100),  
    new.min = 0,  
    new.max = 100,  
    integer = TRUE  
  )  
  out  
  
}
```

---

residuals\_diagnostics *Normality test of a numeric vector*

---

### Description

Applies a Shapiro-Wilks test to a numeric vector, and plots the qq plot and the histogram.

### Usage

```
residuals_diagnostics(residuals, predictions)
```

### Arguments

`residuals`      Numeric vector, model residuals.  
`predictions`    Numeric vector, model predictions.

### Details

The function `shapiro.test()` has a hard limit of 5000 cases. If the model residuals have more than 5000 cases, then `sample(x = residuals, size = 5000)` is applied to the model residuals before the test.

### Value

A list with four slots:

`w` W statistic returned by `shapiro.test()`.  
`p.value` p-value of the Shapiro test.  
`interpretation` Character vector, one of "x is normal", "x is not normal".  
`plot` A patchwork plot with the qq plot and the histogram of x.

### See Also

[ggplot](#), [aes](#), [geom\\_qq\\_line](#), [ggtheme](#), [labs](#), [geom\\_freqpoly](#), [geom\\_abline](#) [plot\\_annotation](#)

### Examples

```
if(interactive()){  
  residuals_diagnostics(  
    residuals = runif(100),  
    predictions = runif(100)  
  )  
}
```

---

residuals_test	<i>Normality test of a numeric vector</i>
----------------	---

---

**Description**

Applies a Shapiro-Wilks test to a numeric vector, and returns a list with the statistic W, its p-value, and a character string with the interpretation.

**Usage**

```
residuals_test(residuals)
```

**Arguments**

residuals      Numeric vector, model residuals.

**Value**

A list with four slots:

/item w W statistic returned by [shapiro.test\(\)](#). /item p.value p-value of the Shapiro test.  
/item interpretation Character vector, one of "x is normal", "x is not normal". /item plot  
A patchwork plot with the qq plot and the histogram of x.

**See Also**

[ggplot](#), [aes](#), [geom\\_qq\\_line](#), [ggtheme](#), [labs](#), [geom\\_freqpoly](#), [geom\\_abline](#) [plot\\_annotation](#)

**Examples**

```
if(interactive()){  
  residuals_test(residuals = runif(100))  
}
```

---

rf	<i>Random forest models with Moran's I test of the residuals</i>
----	--

---

**Description**

A convenient wrapper for [ranger](#) that completes its output by providing the Moran's I of the residuals for different distance thresholds, the rmse and nrmse (as computed by [root\\_mean\\_squared\\_error\(\)](#)), and variable importance scores based on a scaled version of the data generated by [scale](#).

**Usage**

```
rf(
  data = NULL,
  dependent.variable.name = NULL,
  predictor.variable.names = NULL,
  distance.matrix = NULL,
  distance.thresholds = NULL,
  xy = NULL,
  ranger.arguments = NULL,
  scaled.importance = FALSE,
  seed = 1,
  verbose = TRUE,
  n.cores = parallel::detectCores() - 1,
  cluster = NULL
)
```

**Arguments**

**data** Data frame with a response variable and a set of predictors. Default: NULL

**dependent.variable.name** Character string with the name of the response variable. Must be in the column names of data. If the dependent variable is binary with values 1 and 0, the argument `case.weights` of `ranger` is populated by the function `case_weights()`. Default: NULL

**predictor.variable.names** Character vector with the names of the predictive variables. Every element of this vector must be in the column names of data. Optionally, the result of `auto_cor()` or `auto_vif()`. Default: NULL

**distance.matrix** Squared matrix with the distances among the records in data. The number of rows of `distance.matrix` and `data` must be the same. If not provided, the computation of the Moran's I of the residuals is omitted. Default: NULL

**distance.thresholds** Numeric vector with neighborhood distances. All distances in the distance matrix below each value in `distance.thresholds` are set to 0 for the computation of Moran's I. If NULL, it defaults to `seq(0, max(distance.matrix), length.out = 4)`. Default: NULL

**xy** (optional) Data frame or matrix with two columns containing coordinates and named "x" and "y". It is not used by this function, but it is stored in the slot `ranger.arguments$xy` of the model, so it can be used by `rf_evaluate()` and `rf_tuning()`. Default: NULL

**ranger.arguments** Named list with `ranger` arguments (other arguments of this function can also go here). All `ranger` arguments are set to their default values except for 'importance', that is set to 'permutation' rather than 'none'. The ranger arguments `x`, `y`, and `formula` are disabled. Please, consult the help file of `ranger` if you are not familiar with the arguments of this function.



scaled.importance	Logical, if TRUE, the function scales data with <a href="#">scale</a> and fits a new model to compute scaled variable importance scores. This makes variable importance scores of different models somewhat comparable. Default: FALSE
seed	Integer, random seed to facilitate reproducibility. If set to a given number, the returned model is always the same. Default: 1
verbose	Boolean. If TRUE, messages and plots generated during the execution of the function are displayed. Default: TRUE
n.cores	Integer, number of cores to use. Default: <code>parallel::detectCores() - 1</code>
cluster	A cluster definition generated with <code>parallel::makeCluster()</code> . This function does not use the cluster, but can pass it on to other functions when using the <code>%&gt;%</code> pipe. It will be stored in the slot <code>cluster</code> of the output list. Default: NULL

## Details

Please read the help file of [ranger](#) for further details. Notice that the formula interface of [ranger](#) is supported through `ranger.arguments`, but variable interactions are not allowed (but check [the\\_feature\\_engineer\(\)](#)).

## Value

A ranger model with several extra slots:

- `ranger.arguments`: Stores the values of the arguments used to fit the ranger model.
- `importance`: A list containing a data frame with the predictors ordered by their importance, a ggplot showing the importance values, and local importance scores (difference in accuracy between permuted and non permuted variables for every case, computed on the out-of-bag data).
- `performance`: performance scores: R squared on out-of-bag data, R squared ( $\text{cor}(\text{observed}, \text{predicted})^2$ ), pseudo R squared ( $\text{cor}(\text{observed}, \text{predicted})$ ), RMSE, and normalized RMSE (NRMSE).
- `residuals`: residuals, normality test of the residuals computed with [residuals\\_test\(\)](#), and spatial autocorrelation of the residuals computed with [moran\\_multithreshold\(\)](#).

## Examples

```
if(interactive()){

  #loading example data
  data("plant_richness_df")
  data("distance_matrix")

  #fittind random forest model
  out <- rf(
    data = plant_richness_df,
    dependent.variable.name = "richness_species_vascular",
    predictor.variable.names = colnames(plant_richness_df)[5:21],
    distance.matrix = distance_matrix,
    distance.thresholds = 0,
```

```

    n.cores = 1
  )

class(out)

#data frame with ordered variable importance
out$importance$per.variable

#variable importance plot
out$importance$per.variable.plot

#performance
out$performance

#spatial correlation of the residuals
out$spatial.correlation.residuals$per.distance

#plot of the Moran's I of the residuals for different distance thresholds
out$spatial.correlation.residuals$plot

#predictions for new data as done with ranger models:
predicted <- stats::predict(
  object = out,
  data = plant_richness_df,
  type = "response"
)$predictions

#alternative data input methods
#####

#ranger.arguments can contain ranger arguments and any other rf argument
my.ranger.arguments <- list(
  data = plant_richness_df,
  dependent.variable.name = "richness_species_vascular",
  predictor.variable.names = colnames(plant_richness_df)[8:21],
  distance.matrix = distance_matrix,
  distance.thresholds = c(0, 1000)
)

#fitting model with these ranger arguments
out <- rf(
  ranger.arguments = my.ranger.arguments,
  n.cores = 1
)
}

```

## Description

Uses `rf_evaluate()` to compare the performance of several models on independent spatial folds via spatial cross-validation.

## Usage

```
rf_compare(
  models = NULL,
  xy = NULL,
  repetitions = 30,
  training.fraction = 0.75,
  metrics = c("r.squared", "pseudo.r.squared", "rmse", "nrmse", "auc"),
  distance.step = NULL,
  distance.step.x = NULL,
  distance.step.y = NULL,
  fill.color = viridis::viridis(100, option = "F", direction = -1, alpha = 0.8),
  line.color = "gray30",
  seed = 1,
  verbose = TRUE,
  n.cores = parallel::detectCores() - 1,
  cluster = NULL
)
```

## Arguments

<code>models</code>	Named list with models resulting from <code>rf()</code> , <code>rf_spatial()</code> , <code>rf_tuning()</code> , or <code>rf_evaluate()</code> . Example: <code>models = list(a = model.a, b = model.b)</code> . Default: <code>NULL</code>
<code>xy</code>	Data frame or matrix with two columns containing coordinates and named "x" and "y". Default: <code>NULL</code>
<code>repetitions</code>	Integer, number of spatial folds to use during cross-validation. Must be lower than the total number of rows available in the model's data. Default: <code>30</code>
<code>training.fraction</code>	Proportion between 0.5 and 0.9 indicating the proportion of records to be used as training set during spatial cross-validation. Default: <code>0.75</code>
<code>metrics</code>	Character vector, names of the performance metrics selected. The possible values are: "r.squared" ( $\text{cor}(\text{obs}, \text{pred})^2$ ), "pseudo.r.squared" ( $\text{cor}(\text{obs}, \text{pred})$ ), "rmse" ( $\sqrt{\text{sum}((\text{obs} - \text{pred})^2) / \text{length}(\text{obs})}$ ), "nrmse" ( $\text{rmse} / (\text{quantile}(\text{obs}, 0.75) - \text{quantile}(\text{obs}, 0.25))$ ). Default: <code>c("r.squared", "pseudo.r.squared", "rmse", "nrmse")</code>
<code>distance.step</code>	Numeric, argument <code>distance.step</code> of <code>thinning_til_n()</code> . distance step used during the selection of the centers of the training folds. These fold centers are selected by thinning the data until a number of folds equal or lower than <code>repetitions</code> is reached. Its default value is 1/1000th the maximum distance within records in <code>xy</code> . Reduce it if the number of training folds is lower than expected.

<code>distance.step.x</code>	Numeric, argument <code>distance.step.x</code> of <code>make_spatial_folds()</code> . Distance step used during the growth in the x axis of the buffers defining the training folds. Default: NULL (1/1000th the range of the x coordinates).
<code>distance.step.y</code>	Numeric, argument <code>distance.step.y</code> of <code>make_spatial_folds()</code> . Distance step used during the growth in the y axis of the buffers defining the training folds. Default: NULL (1/1000th the range of the y coordinates).
<code>fill.color</code>	Character vector with hexadecimal codes (e.g. "#440154FF" "#21908CFF" "#FDE725FF"), or function generating a palette (e.g. <code>viridis::viridis(100)</code> ). Default: <code>viridis::viridis(100, option = "F", direction = -1)</code>
<code>line.color</code>	Character string, color of the line produced by <code>ggplot2::geom_smooth()</code> . Default: "gray30"
<code>seed</code>	Integer, random seed to facilitate reproducibility. If set to a given number, the results of the function are always the same. Default: 1.
<code>verbose</code>	Logical. If TRUE, messages and plots generated during the execution of the function are displayed, Default: TRUE
<code>n.cores</code>	Integer, number of cores to use for parallel execution. Creates a socket cluster with <code>parallel::makeCluster()</code> , runs operations in parallel with <code>foreach</code> and <code>%dopar%</code> , and stops the cluster with <code>parallel::clusterStop()</code> when the job is done. Default: <code>parallel::detectCores() - 1</code>
<code>cluster</code>	A cluster definition generated with <code>parallel::makeCluster()</code> . If provided, overrides <code>n.cores</code> . When <code>cluster = NULL</code> (default value), and <code>model</code> is provided, the cluster in <code>model</code> , if any, is used instead. If this cluster is NULL, then the function uses <code>n.cores</code> instead. The function does not stop a provided cluster, so it should be stopped with <code>parallel::stopCluster()</code> afterwards. The cluster definition is stored in the output list under the name "cluster" so it can be passed to other functions via the <code>model</code> argument, or using the <code>%&gt;%</code> pipe. Default: NULL

**Value**

A list with three slots:

- `comparison.df`: Data frame with one performance value per spatial fold, metric, and model.
- `spatial.folds`: List with the indices of the training and testing records for each evaluation repetition.
- `plot`: Violin-plot of `comparison.df`.

**See Also**

[rf\\_evaluate\(\)](#)

**Examples**

```
if(interactive()){
```

```

#loading example data
data(distance_matrix)
data(plant_richness_df)

#fitting random forest model
rf.model <- rf(
  data = plant_richness_df,
  dependent.variable.name = "richness_species_vascular",
  predictor.variable.names = colnames(plant_richness_df)[5:21],
  distance.matrix = distance_matrix,
  distance.thresholds = 0,
  n.cores = 1
)

#fitting a spatial model with Moran's Eigenvector Maps
rf.spatial <- rf_spatial(
  model = rf.model,
  n.cores = 1
)

#comparing the spatial and non spatial models
comparison <- rf_compare(
  models = list(
    `Non spatial` = rf.model,
    Spatial = rf.spatial
  ),
  xy = plant_richness_df[, c("x", "y")],
  metrics = c("r.squared", "rmse"),
  n.cores = 1
)
}

```

---

rf\_evaluate

*Evaluates random forest models with spatial cross-validation*


---

### Description

Evaluates the performance of random forest on unseen data over independent spatial folds.

### Usage

```

rf_evaluate(
  model = NULL,
  xy = NULL,
  repetitions = 30,
  training.fraction = 0.75,
  metrics = c("r.squared", "pseudo.r.squared", "rmse", "nrmse", "auc"),
  distance.step = NULL,
  distance.step.x = NULL,

```

```

distance.step.y = NULL,
grow.testing.folds = FALSE,
seed = 1,
verbose = TRUE,
n.cores = parallel::detectCores() - 1,
cluster = NULL
)

```

## Arguments

model	Model fitted with <code>rf()</code> , <code>rf_repeat()</code> , or <code>rf_spatial()</code> .
xy	Data frame or matrix with two columns containing coordinates and named "x" and "y". If NULL, the function will throw an error. Default: NULL
repetitions	Integer, number of spatial folds to use during cross-validation. Must be lower than the total number of rows available in the model's data. Default: 30
training.fraction	Proportion between 0.5 and 0.9 indicating the proportion of records to be used as training set during spatial cross-validation. Default: 0.75
metrics	Character vector, names of the performance metrics selected. The possible values are: "r.squared" ( $\text{cor}(\text{obs}, \text{pred})^2$ ), "pseudo.r.squared" ( $\text{cor}(\text{obs}, \text{pred})$ ), "rmse" ( $\sqrt{\text{sum}((\text{obs} - \text{pred})^2) / \text{length}(\text{obs})}$ ), "nrmse" ( $\text{rmse} / (\text{quantile}(\text{obs}, 0.75) - \text{quantile}(\text{obs}, 0.25))$ ), and "auc" (only for binary responses with values 1 and 0). Default: <code>c("r.squared", "pseudo.r.squared", "rmse", "nrmse")</code>
distance.step	Numeric, argument <code>distance.step</code> of <code>thinning_til_n()</code> . distance step used during the selection of the centers of the training folds. These fold centers are selected by thinning the data until a number of folds equal or lower than repetitions is reached. Its default value is 1/1000th the maximum distance within records in xy. Reduce it if the number of training folds is lower than expected.
distance.step.x	Numeric, argument <code>distance.step.x</code> of <code>make_spatial_folds()</code> . Distance step used during the growth in the x axis of the buffers defining the training folds. Default: NULL (1/1000th the range of the x coordinates).
distance.step.y	Numeric, argument <code>distance.step.y</code> of <code>make_spatial_folds()</code> . Distance step used during the growth in the y axis of the buffers defining the training folds. Default: NULL (1/1000th the range of the y coordinates).
grow.testing.folds	Logic. By default, this function grows contiguous training folds to keep the spatial structure of the data as intact as possible. However, when setting <code>grow.testing.folds = TRUE</code> , the argument <code>training.fraction</code> is set to $1 - \text{training.fraction}$ , and the training and testing folds are switched. This option might be useful when the training data has a spatial structure that does not match well with the default behavior of the function. Default: FALSE
seed	Integer, random seed to facilitate reproducibility. If set to a given number, the results of the function are always the same. Default: 1.

verbose	Logical. If TRUE, messages and plots generated during the execution of the function are displayed, Default: TRUE
n.cores	Integer, number of cores to use for parallel execution. Creates a socket cluster with <code>parallel::makeCluster()</code> , runs operations in parallel with <code>foreach</code> and <code>%dopar%</code> , and stops the cluster with <code>parallel::clusterStop()</code> when the job is done. Default: <code>parallel::detectCores() - 1</code>
cluster	A cluster definition generated with <code>parallel::makeCluster()</code> . If provided, overrides <code>n.cores</code> . When <code>cluster = NULL</code> (default value), and <code>model</code> is provided, the cluster in <code>model</code> , if any, is used instead. If this cluster is NULL, then the function uses <code>n.cores</code> instead. The function does not stop a provided cluster, so it should be stopped with <code>parallel::stopCluster()</code> afterwards. The cluster definition is stored in the output list under the name "cluster" so it can be passed to other functions via the <code>model</code> argument, or using the <code>%&gt;%</code> pipe. Default: NULL

## Details

The evaluation algorithm works as follows: the number of repetitions and the input dataset (stored in `model$ranger.arguments$data`) are used as inputs for the function `thinning_til_n()`, that applies `thinning()` to the input data until as many cases as repetitions are left, and as separated as possible. Each of these remaining records will be used as a "fold center". From that point, the fold grows, until a number of points equal (or close) to `training.fraction` is reached. The indices of the records within the grown spatial fold are stored as "training" in the output list, and the remaining ones as "testing". Then, for each spatial fold, a "training model" is fitted using the cases corresponding with the training indices, and predicted over the cases corresponding with the testing indices. The model predictions on the "unseen" data are compared with the observations, and the performance measures (R squared, pseudo R squared, RMSE and NRMSE) computed.

## Value

A model of the class "rf\_evaluate" with a new slot named "evaluation", that is a list with the following slots:

- `training.fraction`: Value of the argument `training.fraction`.
- `spatial.folds`: Result of applying `make_spatial_folds()` on the data coordinates. It is a list with as many slots as repetitions are indicated by the user. Each slot has two slots named "training" and "testing", each one having the indices of the cases used on the training and testing models.
- `per.fold`: Data frame with the evaluation results per spatial fold (or repetition). It contains the ID of each fold, it's central coordinates, the number of training and testing cases, and the training and testing performance measures: R squared, pseudo R squared (`cor(observed, predicted)`), `rmse`, and `normalized rmse`.
- `per.model`: Same data as above, but organized per fold and model ("Training", "Testing", and "Full").
- `aggregated`: Same data, but aggregated by model and performance measure.

**Examples**

```

if(interactive()){

#loading example data
data(plant_richness_df)
data(distance_matrix)

#fitting random forest model
rf.model <- rf(
  data = plant_richness_df,
  dependent.variable.name = "richness_species_vascular",
  predictor.variable.names = colnames(plant_richness_df)[5:21],
  distance.matrix = distance_matrix,
  distance.thresholds = 0,
  n.cores = 1,
  verbose = FALSE
)

#evaluation with spatial cross-validation
rf.model <- rf_evaluate(
  model = rf.model,
  xy = plant_richness_df[, c("x", "y")],
  n.cores = 1
)

#checking evaluation results
plot_evaluation(rf.model)
print_evaluation(rf.model)
x <- get_evaluation(rf.model)

}

```

---

rf\_importance

*Contribution of each predictor to model transferability*


---

**Description**

Evaluates the contribution of the predictors to model transferability via spatial cross-validation. The function returns the median increase or decrease in a given evaluation metric (R2, pseudo R2, RMSE, nRMSE, or AUC) when a variable is introduced in a model, by comparing and evaluating via spatial cross-validation models with and without the given variable. This function was devised to provide importance scores that would be less sensitive to spatial autocorrelation than those computed internally by random forest on the out-of-bag data. This function is experimental.

**Usage**

```

rf_importance(
  model = NULL,
  xy = NULL,

```



```

repetitions = 30,
training.fraction = 0.75,
metric = c("r.squared", "pseudo.r.squared", "rmse", "nrmse", "auc"),
distance.step = NULL,
distance.step.x = NULL,
distance.step.y = NULL,
fill.color = viridis::viridis(100, option = "F", direction = -1, alpha = 1, end = 0.9),
line.color = "white",
seed = 1,
verbose = TRUE,
n.cores = parallel::detectCores() - 1,
cluster = NULL
)

```

### Arguments

model	Model fitted with <code>rf()</code> and/or <code>rf_spatial()</code> . The function doesn't work with models fitted with <code>rf_repeat()</code> . Default: NULL
xy	Data frame or matrix with two columns containing coordinates and named "x" and "y". If NULL, the function will throw an error. Default: NULL
repetitions	Integer, number of spatial folds to use during cross-validation. Must be lower than the total number of rows available in the model's data. Default: 30
training.fraction	Proportion between 0.5 and 0.9 indicating the proportion of records to be used as training set during spatial cross-validation. Default: 0.75
metric	Character, names of the performance metric to use. The possible values are: "r.squared" ( $\text{cor}(\text{obs}, \text{pred})^2$ ), "pseudo.r.squared" ( $\text{cor}(\text{obs}, \text{pred})$ ), "rmse" ( $\sqrt{\text{sum}((\text{obs} - \text{pred})^2)/\text{length}(\text{obs})}$ ), "nrmse" ( $\text{rmse}/(\text{quantile}(\text{obs}, 0.75) - \text{quantile}(\text{obs}, 0.25))$ ), and "auc" (only for binary responses with values 1 and 0). Default: "r.squared"
distance.step	Numeric, argument <code>distance.step</code> of <code>thinning_til_n()</code> . distance step used during the selection of the centers of the training folds. These fold centers are selected by thinning the data until a number of folds equal or lower than <code>repetitions</code> is reached. Its default value is 1/1000th the maximum distance within records in <code>xy</code> . Reduce it if the number of training folds is lower than expected.
distance.step.x	Numeric, argument <code>distance.step.x</code> of <code>make_spatial_folds()</code> . Distance step used during the growth in the x axis of the buffers defining the training folds. Default: NULL (1/1000th the range of the x coordinates).
distance.step.y	Numeric, argument <code>distance.step.y</code> of <code>make_spatial_folds()</code> . Distance step used during the growth in the y axis of the buffers defining the training folds. Default: NULL (1/1000th the range of the y coordinates).
fill.color	Character vector with hexadecimal codes (e.g. "#440154FF" "#21908CFF" "#FDE725FF"), or function generating a palette (e.g. <code>viridis::viridis(100)</code> ). Default: <code>viridis::viridis(100, option = "F", direction = -1, alpha = 0.8, end = 0.9)</code>

<code>line.color</code>	Character string, color of the line produced by <code>ggplot2::geom_smooth()</code> . Default: "white"
<code>seed</code>	Integer, random seed to facilitate reproducibility. If set to a given number, the results of the function are always the same. Default: 1.
<code>verbose</code>	Logical. If TRUE, messages and plots generated during the execution of the function are displayed, Default: TRUE
<code>n.cores</code>	Integer, number of cores to use for parallel execution. Creates a socket cluster with <code>parallel::makeCluster()</code> , runs operations in parallel with <code>foreach</code> and <code>%dopar%</code> , and stops the cluster with <code>parallel::clusterStop()</code> when the job is done. Default: <code>parallel::detectCores() - 1</code>
<code>cluster</code>	A cluster definition generated with <code>parallel::makeCluster()</code> . If provided, overrides <code>n.cores</code> . When <code>cluster = NULL</code> (default value), and <code>model</code> is provided, the cluster in <code>model</code> , if any, is used instead. If this cluster is NULL, then the function uses <code>n.cores</code> instead. The function does not stop a provided cluster, so it should be stopped with <code>parallel::stopCluster()</code> afterwards. The cluster definition is stored in the output list under the name "cluster" so it can be passed to other functions via the <code>model</code> argument, or using the <code>%&gt;%</code> pipe. Default: NULL

### Value

The input model with new data in its "importance" slot. The new importance scores are included in the data frame `model$importance$per.variable`, under the column names "importance.cv" (median contribution to transferability over spatial cross-validation repetitions), "importance.cv.mad" (median absolute deviation of the performance scores over spatial cross-validation repetitions), "importance.cv.percent" ("importance.cv" expressed as a percent, taking the full model's performance as baseline), and "importance.cv.mad" (median absolute deviation of "importance.cv"). The plot is stored as "cv.per.variable.plot".

### Examples

```
if(interactive()){

#loading example data
data(plant_richness_df)
data(distance_matrix)
xy <- plant_richness_df[, c("x", "y")]

#fitting random forest model
rf.model <- rf(
  data = plant_richness_df,
  dependent.variable.name = "richness_species_vascular",
  predictor.variable.names = colnames(plant_richness_df)[5:21],
  distance.matrix = distance_matrix,
  distance.thresholds = 0,
  xy = xy,
  n.cores = 1,
  verbose = FALSE
)
```

```
#computing predictor contribution to model transferability
rf.model <- rf_importance(rf.model)

}
```

---

rf\_repeat

*Fits several random forest models on the same data*

---

## Description

Fits several random forest models on the same data in order to capture the effect of the algorithm's stochasticity on the variable importance scores, predictions, residuals, and performance measures. The function relies on the median to aggregate performance and importance values across repetitions. It is recommended to use it after a model is fitted (`rf()` or `rf_spatial()`), tuned (`rf_tuning()`), and/or evaluated (`rf_evaluate()`). This function is designed to be used after fitting a model with `rf()` or `rf_spatial()`, tuning it with `rf_tuning()` and evaluating it with `rf_evaluate()`.

## Usage

```
rf_repeat(
  model = NULL,
  data = NULL,
  dependent.variable.name = NULL,
  predictor.variable.names = NULL,
  distance.matrix = NULL,
  distance.thresholds = NULL,
  xy = NULL,
  ranger.arguments = NULL,
  scaled.importance = FALSE,
  repetitions = 10,
  keep.models = TRUE,
  seed = 1,
  verbose = TRUE,
  n.cores = parallel::detectCores() - 1,
  cluster = NULL
)
```

## Arguments

model	A model fitted with <code>rf()</code> . If provided, the data and ranger arguments are taken directly from the model definition (stored in <code>model\$ranger.arguments</code> ). Default: NULL
data	Data frame with a response variable and a set of predictors. Default: NULL

<code>dependent.variable.name</code>	Character string with the name of the response variable. Must be in the column names of data. If the dependent variable is binary with values 1 and 0, the argument <code>case.weights</code> of <code>ranger</code> is populated by the function <code>case_weights()</code> . Default: NULL
<code>predictor.variable.names</code>	Character vector with the names of the predictive variables. Every element of this vector must be in the column names of data. Default: NULL
<code>distance.matrix</code>	Squared matrix with the distances among the records in data. The number of rows of <code>distance.matrix</code> and <code>data</code> must be the same. If not provided, the computation of the Moran's I of the residuals is omitted. Default: NULL
<code>distance.thresholds</code>	Numeric vector with neighborhood distances. All distances in the distance matrix below each value in <code>distance.thresholds</code> are set to 0 for the computation of Moran's I. If NULL, it defaults to <code>seq(0, max(distance.matrix), length.out = 4)</code> . Default: NULL
<code>xy</code>	(optional) Data frame or matrix with two columns containing coordinates and named "x" and "y". It is not used by this function, but it is stored in the slot <code>ranger.arguments\$xy</code> of the model, so it can be used by <code>rf_evaluate()</code> and <code>rf_tuning()</code> . Default: NULL
<code>ranger.arguments</code>	Named list with <code>ranger</code> arguments (other arguments of this function can also go here). All <code>ranger</code> arguments are set to their default values except for 'importance', that is set to 'permutation' rather than 'none'. Please, consult the help file of <code>ranger</code> if you are not familiar with the arguments of this function.
<code>scaled.importance</code>	Logical. If TRUE, and 'importance = "permutation"', the function scales 'data' with <code>scale</code> and fits a new model to compute scaled variable importance scores. Default: FALSE
<code>repetitions</code>	Integer, number of random forest models to fit. Default: 10
<code>keep.models</code>	Logical, if TRUE, the fitted models are returned in the <code>models</code> slot. Set to FALSE if the accumulation of models is creating issues with the RAM memory available. Default: TRUE.
<code>seed</code>	Integer, random seed to facilitate reproducibility. If set to a given number, the results of the function are always the same. Default: 1.
<code>verbose</code>	Logical, if TRUE, messages and plots generated during the execution of the function are displayed, Default: TRUE
<code>n.cores</code>	Integer, number of cores to use for parallel execution. Creates a socket cluster with <code>parallel::makeCluster()</code> , runs operations in parallel with <code>foreach</code> and <code>%dopar%</code> , and stops the cluster with <code>parallel::clusterStop()</code> when the job is done. Default: <code>parallel::detectCores() - 1</code>
<code>cluster</code>	A cluster definition generated with <code>parallel::makeCluster()</code> . If provided, overrides <code>n.cores</code> . When <code>cluster = NULL</code> (default value), and <code>model</code> is provided, the cluster in <code>model</code> , if any, is used instead. If this cluster is NULL, then the function uses <code>n.cores</code> instead. The function does not stop a provided cluster, so it should be stopped with <code>parallel::stopCluster()</code> afterwards. The

cluster definition is stored in the output list under the name "cluster" so it can be passed to other functions via the `model` argument, or using the `%>%` pipe.  
Default: NULL

## Value

A ranger model with several new slots:

- `ranger.arguments`: Stores the values of the arguments used to fit the ranger model.
- `importance`: A list containing a data frame with the predictors ordered by their importance, a `ggplot` showing the importance values, and local importance scores.
- `performance`: out-of-bag performance scores: R squared, pseudo R squared, RMSE, and normalized RMSE (NRMSE).
- `pseudo.r.squared`: computed as the correlation between the observations and the predictions.
- `residuals`: residuals, normality test of the residuals computed with `residuals_test()`, and spatial autocorrelation of the residuals computed with `moran_multithreshold()`.

## Examples

```
if(interactive()){

  #loading example data
  data(plant_richness_df)
  data(distance_matrix)

  #fitting 5 random forest models
  out <- rf_repeat(
    data = plant_richness_df,
    dependent.variable.name = "richness_species_vascular",
    predictor.variable.names = colnames(plant_richness_df)[5:21],
    distance.matrix = distance_matrix,
    distance.thresholds = 0,
    repetitions = 5,
    n.cores = 1
  )

  #data frame with ordered variable importance
  out$importance$per.variable

  #per repetition
  out$importance$per.repetition

  #variable importance plot
  out$importance$per.repetition.plot

  #performance
  out$performance

  #spatial correlation of the residuals for different distance thresholds
```

```

out$spatial.correlation.residuals$per.distance

#plot of the Moran's I of the residuals for different distance thresholds
out$spatial.correlation.residuals$plot

#using a model as an input for rf_repeat()
rf.model <- rf(
  data = plant_richness_df,
  dependent.variable.name = "richness_species_vascular",
  predictor.variable.names = colnames(plant_richness_df)[8:21],
  distance.matrix = distance_matrix,
  distance.thresholds = 0,
  n.cores = 1
)

#repeating the model 5 times
rf.repeat <- rf_repeat(
  model = rf.model,
  n.cores = 1
)

rf.repeat$performance
rf.repeat$importance$per.repetition.plot

}

```

---

rf\_spatial

*Fits spatial random forest models*


---

## Description

Fits spatial random forest models using different methods to generate, rank, and select spatial predictors acting as proxies of spatial processes not considered by the non-spatial predictors. The end goal is providing the model with information about the spatial structure of the data to minimize the spatial correlation (Moran's I) of the model residuals and generate honest variable importance scores.

## Usage

```

rf_spatial(
  model = NULL,
  data = NULL,
  dependent.variable.name = NULL,
  predictor.variable.names = NULL,
  distance.matrix = NULL,
  distance.thresholds = NULL,
  xy = NULL,
  ranger.arguments = NULL,

```

```

scaled.importance = TRUE,
method = c("mem.moran.sequential", "mem.effect.sequential", "mem.effect.recursive",
  "hengl", "hengl.moran.sequential", "hengl.effect.sequential",
  "hengl.effect.recursive", "pca.moran.sequential", "pca.effect.sequential",
  "pca.effect.recursive"),
max.spatial.predictors = NULL,
weight.r.squared = NULL,
weight.penalization.n.predictors = NULL,
seed = 1,
verbose = TRUE,
n.cores = parallel::detectCores() - 1,
cluster = NULL
)

```

## Arguments

model	A model fitted with <code>rf()</code> . If used, the arguments <code>data</code> , <code>dependent.variable.name</code> , <code>predictor.variable.names</code> , <code>distance.matrix</code> , <code>distance.thresholds</code> , <code>ranger.arguments</code> , and <code>scaled.importance</code> are taken directly from the model definition. Default: NULL
data	Data frame with a response variable and a set of predictors. Default: NULL
dependent.variable.name	Character string with the name of the response variable. Must be in the column names of data. If the dependent variable is binary with values 1 and 0, the argument <code>case.weights</code> of <code>ranger</code> is populated by the function <code>case_weights()</code> . Default: NULL
predictor.variable.names	Character vector with the names of the predictive variables. Every element of this vector must be in the column names of data. Default: NULL
distance.matrix	Squared matrix with the distances among the records in data. The number of rows of <code>distance.matrix</code> and <code>data</code> must be the same. If not provided, the computation of the Moran's I of the residuals is omitted. Default: NULL
distance.thresholds	Numeric vector with distances in the same units as <code>distance.matrix</code> . Distances below each distance threshold are set to 0 on separated copies of the distance matrix to compute Moran's I at different neighborhood distances. If NULL, it defaults to <code>seq(0, max(distance.matrix)/2, length.out = 4)</code> (defined by <code>default_distance_thresholds()</code> ). Default: NULL
xy	(optional) Data frame or matrix with two columns containing coordinates and named "x" and "y". It is not used by this function, but it is stored in the slot <code>ranger.arguments\$xy</code> of the model, so it can be used by <code>rf_evaluate()</code> and <code>rf_tuning()</code> . Default: NULL
ranger.arguments	Named list with <code>ranger</code> arguments (other arguments of this function can also go here). All <code>ranger</code> arguments are set to their default values except for 'importance', that is set to 'permutation' rather than 'none'. Please, consult the help file of <code>ranger</code> if you are not familiar with the arguments of this function.

scaled.importance	Logical. If TRUE, and 'importance = "permutation', the function scales 'data' with <a href="#">scale</a> and fits a new model to compute scaled variable importance scores. Default: TRUE
method	Character, method to build, rank, and select spatial predictors. One of: <ul style="list-style-type: none"> <li>• "hengl"</li> <li>• "hengl.moran.sequential" (experimental)</li> <li>• "hengl.effect.sequential" (experimental)</li> <li>• "hengl.effect.recursive" (experimental)</li> <li>• "pca.moran.sequential" (experimental)</li> <li>• "pca.effect.sequential" (experimental)</li> <li>• "pca.effect.recursive" (experimental)</li> <li>• "mem.moran.sequential"</li> <li>• "mem.effect.sequential"</li> <li>• "mem.effect.recursive"</li> </ul>
max.spatial.predictors	Integer, maximum number of spatial predictors to generate. Useful when memory problems arise due to a large number of spatial predictors, Default: NULL
weight.r.squared	Numeric between 0 and 1, weight of R-squared in the selection of spatial components. See <a href="#">Details</a> , Default: NULL
weight.penalization.n.predictors	Numeric between 0 and 1, weight of the penalization for adding an increasing number of spatial predictors during selection. Default: NULL
seed	Integer, random seed to facilitate reproducibility. Default: 1.
verbose	Logical. If TRUE, messages and plots generated during the execution of the function are displayed, Default: TRUE
n.cores	Integer, number of cores to use for parallel execution. Creates a socket cluster with <code>parallel::makeCluster()</code> , runs operations in parallel with <code>foreach</code> and <code>%dopar%</code> , and stops the cluster with <code>parallel::clusterStop()</code> when the job is done. Default: <code>parallel::detectCores() - 1</code>
cluster	A cluster definition generated with <code>parallel::makeCluster()</code> . If provided, overrides <code>n.cores</code> . When <code>cluster = NULL</code> (default value), and <code>model</code> is provided, the cluster in <code>model</code> , if any, is used instead. If this cluster is NULL, then the function uses <code>n.cores</code> instead. The function does not stop a provided cluster, so it should be stopped with <code>parallel::stopCluster()</code> afterwards. The cluster definition is stored in the output list under the name "cluster" so it can be passed to other functions via the <code>model</code> argument, or using the <code>%&gt;%</code> pipe. Default: NULL

## Details

The function uses three different methods to generate spatial predictors ("hengl", "pca", and "mem"), two methods to rank them in order to define in what order they are introduced in the model ("effect" and "moran), and two methods to select the spatial predictors that minimize the spatial correlation



of the model residuals ("sequential" and "recursive"). All method names but "heng1" (that uses the complete distance matrix as predictors in the spatial model) are named by combining a method to generate the spatial predictors, a method to rank them, and a method to select them, separated by a point. Examples are "mem.moran.sequential" or "mem.effect.recursive". All combinations are not possible, since the ranking method "moran" cannot be used with the selection method "recursive" (because the logics behind them are very different, see below). Methods to generate spatial predictors:

- "heng1": named after the method RFsp presented in the paper "Random forest as a generic framework for predictive modeling of spatial and spatio-temporal variables", by Hengl et al. (2018), where the authors propose to use the distance matrix among records as predictors in spatial random forest models (RFsp method). In this function, all methods starting with "heng1" use either the complete distance matrix, or select columns of the distance matrix as spatial predictors.
- "mem": Generates Moran's Eigenvector Maps, that is, the eigenvectors of the double-centered weights of the distance matrix. The method is described in "Spatial modelling: a comprehensive framework for principal coordinate analysis of neighbour matrices (PCNM)", by Dray et al. (2006), and "Statistical methods for temporal and space-time analysis of community composition data", by Legendre and Gauthier (2014).
- "pca": Computes spatial predictors from the principal component analysis of a weighted distance matrix (see [weights\\_from\\_distance\\_matrix\(\)](#)). This is an experimental method, use with caution.

Methods to rank spatial predictors (see [rank\\_spatial\\_predictors\(\)](#)):

- "moran": Computes the Moran's I of each spatial predictor, selects the ones with positive values, and ranks them from higher to lower Moran's I.
- "effect": If a given non-spatial random forest model is defined as  $y = p1 + \dots + pn$ , being  $p1 + \dots + pn$  the set of predictors, for every spatial predictor generated (spX) a spatial model  $y = p1 + \dots + pn + spX$  is fitted, and the Moran's I of its residuals is computed. The spatial predictors are then ranked by how much they help to reduce spatial autocorrelation between the non-spatial and the spatial model.

Methods to select spatial predictors:

- "sequential" (see [select\\_spatial\\_predictors\\_sequential\(\)](#)): The spatial predictors are added one by one in the order they were ranked, and once all spatial predictors are introduced, the best first n predictors are selected. This method is similar to the one employed in the MEM methodology (Moran's Eigenvector Maps) described in the paper "Spatial modelling: a comprehensive framework for principal coordinate analysis of neighbour matrices (PCNM)", by Dray et al. (2006), and "Statistical methods for temporal and space-time analysis of community composition data", by Legendre and Gauthier (2014). This method generally introduces tens of predictors into the model, but usually offers good results.
- "recursive" (see [select\\_spatial\\_predictors\\_recursive\(\)](#)): This method tries to find the smallest combination of spatial predictors that reduce the spatial correlation of the model's residuals the most. The algorithm goes as follows: 1. The first ranked spatial predictor is introduced into the model; 2. the remaining predictors are ranked again using the "effect" method, using the model in 1. as reference. The first spatial predictor in the resulting ranking is then introduced into the model, and the steps 1. and 2. are repeated until spatial predictors

stop having an effect in reducing the Moran's I of the model residuals. This method takes longer to compute, but generates smaller sets of spatial predictors. This is an experimental method, use with caution.

Once ranking procedure is completed, an algorithm is used to select the minimal subset of spatial predictors that reduce the most the Moran's I of the residuals: for each new spatial predictor introduced in the model, the Moran's I of the residuals, its p-value, a binary version of the p-value (0 if < 0.05 and 1 if >= 0.05), the R-squared of the model, and a penalization linear with the number of spatial predictors introduced (computed as  $(1 / \text{total spatial predictors}) * \text{introduced spatial predictors}$ ) are rescaled between 0 and 1. Then, the optimization criteria is computed as  $\max(1 - \text{Moran's I}, \text{p-value binary}) + (\text{weight} * \text{R-squared})$ . The predictors from the first one to the one with the highest optimization criteria are then selected as the best ones in reducing the spatial correlation of the model residuals, and used along with data to fit the final spatial model.

## Value

A ranger model with several new slots:

- `ranger.arguments`: Values of the arguments used to fit the ranger model.
- `importance`: A list containing the vector of variable importance as originally returned by `ranger` (scaled or not depending on the value of `scaled.importance`), a data frame with the predictors ordered by their importance, and a `ggplot` showing the importance values.
- `performance`: With the out-of-bag R squared, pseudo R squared, RMSE and NRMSE of the model.
- `residuals`: residuals, normality test of the residuals computed with `residuals_test()`, and spatial autocorrelation of the residuals computed with `moran_multithreshold()`.
- `spatial`: A list with four slots:
  - `method`: Character, method used to generate, rank, and select spatial predictors.
  - `names`: Character vector with the names of the selected spatial predictors. Not returned if the method is "heng1".
  - `optimization`: Criteria used to select the spatial predictors. Not returned if the method is "heng1".
  - `plot`: Plot of the criteria used to select the spatial predictors. Not returned if the method is "heng1".

## Examples

```
if(interactive()){

  #loading example data
  data(distance_matrix)
  data(plant_richness_df)

  #names of the response and predictors
  dependent.variable.name <- "richness_species_vascular"
  predictor.variable.names <- colnames(plant_richness_df)[5:21]

  #heng1
  model <- rf_spatial(
```

```

    data = plant_richness_df,
    dependent.variable.name = dependent.variable.name,
    predictor.variable.names = predictor.variable.names,
    distance.matrix = distance_matrix,
    distance.thresholds = 0,
    method = "heng1",
    n.cores = 1
  )

#mem.moran.sequential
model <- rf_spatial(
  data = plant_richness_df,
  dependent.variable.name = dependent.variable.name,
  predictor.variable.names = predictor.variable.names,
  distance.matrix = distance_matrix,
  distance.thresholds = 0,
  method = "mem.moran.sequential",
  n.cores = 1
)

#fitting an rf_spatial model from an rf model
rf.model <- rf(
  data = plant_richness_df,
  dependent.variable.name = "richness_species_vascular",
  predictor.variable.names = colnames(plant_richness_df)[5:21],
  distance.matrix = distance_matrix,
  distance.thresholds = 0,
  n.cores = 1,
  verbose = FALSE
)
rf.model$spatial.correlation.residuals$plot

#spatial version of the rf model
rf.spatial <- rf_spatial(model = rf.model)
rf.spatial$spatial.correlation.residuals$plot

}

```

**Description**

Finds the optimal set of random forest hyperparameters `num.trees`, `mtry`, and `min.node.size` via grid search by maximizing the model's R squared, or AUC, if the response variable is binomial, via spatial cross-validation performed with `rf_evaluate()`.

**Usage**

```
rf_tuning(
  model = NULL,
  num.trees = NULL,
  mtry = NULL,
  min.node.size = NULL,
  xy = NULL,
  repetitions = 30,
  training.fraction = 0.75,
  seed = 1,
  verbose = TRUE,
  n.cores = parallel::detectCores() - 1,
  cluster = NULL
)
```

**Arguments**

model	A model fitted with <code>rf()</code> . If provided, the training data is taken directly from the model definition (stored in <code>model\$ranger.arguments</code> ). Default: <code>NULL</code>
num.trees	Numeric integer vector with the number of trees to fit on each model repetition. Default: <code>c(500, 1000, 2000)</code> .
mtry	Numeric integer vector, number of predictors to randomly select from the complete pool of predictors on each tree split. Default: <code>floor(seq(1, length(predictor.variable.names), length.out = 4))</code>
min.node.size	Numeric integer, minimal number of cases in a terminal node. Default: <code>c(5, 10, 20, 40)</code>
xy	Data frame or matrix with two columns containing coordinates and named "x" and "y". If <code>NULL</code> , the function will throw an error. Default: <code>NULL</code>
repetitions	Integer, number of independent spatial folds to use during the cross-validation. Default: <code>30</code> .
training.fraction	Proportion between 0.2 and 0.9 indicating the number of records to be used in model training. Default: <code>0.75</code>
seed	Integer, random seed to facilitate reproducibility. If set to a given number, the results of the function are always the same. Default: <code>1</code> .
verbose	Logical. If <code>TRUE</code> , messages and plots generated during the execution of the function are displayed, Default: <code>TRUE</code>
n.cores	Integer, number of cores to use for parallel execution. Creates a socket cluster with <code>parallel::makeCluster()</code> , runs operations in parallel with <code>foreach</code> and <code>%dopar%</code> , and stops the cluster with <code>parallel::clusterStop()</code> when the job is done. Default: <code>parallel::detectCores() - 1</code>
cluster	A cluster definition generated with <code>parallel::makeCluster()</code> . If provided, overrides <code>n.cores</code> . When <code>cluster = NULL</code> (default value), and <code>model</code> is provided, the cluster in <code>model</code> , if any, is used instead. If this cluster is <code>NULL</code> , then the function uses <code>n.cores</code> instead. The function does not stop a provided cluster, so it should be stopped with <code>parallel::stopCluster()</code> afterwards. The

cluster definition is stored in the output list under the name "cluster" so it can be passed to other functions via the `model` argument, or using the `%>%` pipe. Default: NULL

**Value**

A model with a new slot named `tuning`, with a data frame with the results of the tuning analysis.

**See Also**

[rf\\_evaluate\(\)](#)

**Examples**

```
if(interactive()){  
  
  #loading example data  
  data(plant_richness_df)  
  data(distance_matrix)  
  
  #fitting model to tune  
  out <- rf(  
    data = plant_richness_df,  
    dependent.variable.name = "richness_species_vascular",  
    predictor.variable.names = colnames(plant_richness_df)[5:21],  
    distance.matrix = distance_matrix,  
    distance.thresholds = 0,  
    n.cores = 1  
  )  
  
  #model tuning  
  tuning <- rf_tuning(  
    model = out,  
    num.trees = c(100, 500),  
    mtry = c(2, 8),  
    min.node.size = c(5, 10),  
    xy = plant_richness_df[, c("x", "y")],  
    n.cores = 1  
  )  
  
}
```

---

root\_mean\_squared\_error

*RMSE and normalized RMSE*

---

**Description**

Computes the rmse or normalized rmse (nrmse) between two numeric vectors of the same length representing observations and model predictions.

**Usage**

```
root_mean_squared_error(  
  o,  
  p,  
  normalization = c("rmse", "all", "mean", "sd", "maxmin", "iq")  
)
```

**Arguments**

**o** Numeric vector with observations, must have the same length as **p**.

**p** Numeric vector with predictions, must have the same length as **o**.

**normalization** character, normalization method, Default: "rmse" (see Details).

**Details**

The normalization methods go as follows:

- "rmse": RMSE with no normalization.
- "mean": RMSE divided by the mean of the observations (rmse/mean(o)).
- "sd": RMSE divided by the standard deviation of the observations (rmse/sd(o)).
- "maxmin": RMSE divided by the range of the observations (rmse/(max(o) - min(o))).
- "iq": RMSE divided by the interquartile range of the observations (rmse/(quantile(o, 0.75) - quantile(o, 0.25)))

**Value**

Named numeric vector with either one or 5 values, as selected by the user.

**Examples**

```
if(interactive()){  
  
  root_mean_squared_error(  
    o = runif(10),  
    p = runif(10)  
  )  
  
}
```

---

`select_spatial_predictors_recursive`*Finds optimal combinations of spatial predictors*

---

## Description

Selects spatial predictors following these steps:

1. Gets the spatial predictors ranked by `rank_spatial_predictors()` and fits a model of the form  $y \sim \text{predictors} + \text{best\_spatial\_predictor\_1}$ . The Moran's I of the residuals of this model is used as reference value for the next step.
2. The remaining spatial predictors are introduced again into `rank_spatial_predictors()`, and the spatial predictor with the highest ranking is introduced in a new model of the form  $y \sim \text{predictors} + \text{best\_spatial\_predictor\_1} + \text{best\_spatial\_predictor\_2}$ .
3. Steps 1 and 2 are repeated until the Moran's I doesn't improve for a number of repetitions equal to the 20 percent of the total number of spatial predictors introduced in the function.

This method allows to select the smallest set of spatial predictors that have the largest joint effect in reducing the spatial correlation of the model residuals, while maintaining the model's R-squared as high as possible. As a consequence of running `rank_spatial_predictors()` on each iteration, this method includes in the final model less spatial predictors than the sequential method implemented in `select_spatial_predictors_sequential()` would do, while minimizing spatial correlation and maximizing the R squared of the model as much as possible.

## Usage

```
select_spatial_predictors_recursive(  
  data = NULL,  
  dependent.variable.name = NULL,  
  predictor.variable.names = NULL,  
  distance.matrix = NULL,  
  distance.thresholds = NULL,  
  ranger.arguments = NULL,  
  spatial.predictors.df = NULL,  
  spatial.predictors.ranking = NULL,  
  weight.r.squared = 0.25,  
  weight.penalization.n.predictors = 0,  
  n.cores = parallel::detectCores() - 1,  
  cluster = NULL  
)
```

## Arguments

<code>data</code>	Data frame with a response variable and a set of predictors. Default: NULL
<code>dependent.variable.name</code>	Character string with the name of the response variable. Must be in the column names of data. Default: NULL

<code>predictor.variable.names</code>	Character vector with the names of the predictive variables. Every element of this vector must be in the column names of data. Default: NULL
<code>distance.matrix</code>	Squared matrix with the distances among the records in data. The number of rows of <code>distance.matrix</code> and data must be the same. If not provided, the computation of the Moran's I of the residuals is omitted. Default: NULL
<code>distance.thresholds</code>	Numeric vector with neighborhood distances. All distances in the distance matrix below each value in <code>distance.thresholds</code> are set to 0 for the computation of Moran's I. If NULL, it defaults to <code>seq(0, max(distance.matrix), length.out = 4)</code> . Default: NULL
<code>ranger.arguments</code>	Named list with <code>ranger</code> arguments (other arguments of this function can also go here). All <code>ranger</code> arguments are set to their default values except for 'importance', that is set to 'permutation' rather than 'none'. Please, consult the help file of <code>ranger</code> if you are not familiar with the arguments of this function.
<code>spatial.predictors.df</code>	Data frame of spatial predictors.
<code>spatial.predictors.ranking</code>	Ranking of predictors returned by <code>rank_spatial_predictors()</code> .
<code>weight.r.squared</code>	Numeric between 0 and 1, weight of R-squared in the optimization index. Default: 0.25
<code>weight.penalization.n.predictors</code>	Numeric between 0 and 1, weight of the penalization for the number of spatial predictors added in the optimization index. Default: 0
<code>n.cores</code>	Integer, number of cores to use. Default: <code>parallel::detectCores() - 1</code>
<code>cluster</code>	A cluster definition generated by <code>parallel::makeCluster()</code> . Default: NULL

## Details

The algorithm works as follows. If the function `rank_spatial_predictors()` returns 10 ranked spatial predictors (sp1 to sp10, being sp7 the best one), `select_spatial_predictors_recursive()` is going to first fit the model  $y \sim \text{predictors} + \text{sp7}$ . Then, the spatial predictors sp2 to sp9 are again ranked with `rank_spatial_predictors()` using the model  $y \sim \text{predictors} + \text{sp7}$  as reference (at this stage, some of the spatial predictors might be dropped due to lack of effect). When the new ranking of spatial predictors is ready (let's say they are sp5, sp3, and sp4), the best one (sp5) is included in the model  $y \sim \text{predictors} + \text{sp7} + \text{sp5}$ , and the remaining ones go again to `rank_spatial_predictors()` to repeat the process until spatial predictors are depleted.

## Value

A list with two slots: `optimization`, a data frame with the index of the spatial predictor added on each iteration, the spatial correlation of the model residuals, and the R-squared of the model, and `best.spatial.predictors`, that is a character vector with the names of the spatial predictors that minimize the Moran's I of the residuals and maximize the R-squared of the model.



**Examples**

```
if(interactive()){

#loading example data
data(distance_matrix)
data(plant_richness_df)

#response and predictor names
dependent.variable.name = "richness_species_vascular"
predictor.variable.names = colnames(plant_richness_df)[5:21]

#non-spatial model
model <- rf(
  data = plant_richness_df,
  dependent.variable.name = dependent.variable.name,
  predictor.variable.names = predictor.variable.names,
  distance.matrix = distance_matrix,
  distance.thresholds = 0,
  n.cores = 1
)

#preparing spatial predictors
spatial.predictors <- mem_multithreshold(
  distance.matrix = distance_matrix,
  distance.thresholds = 0
)

#ranking spatial predictors
spatial.predictors.ranking <- rank_spatial_predictors(
  data = plant_richness_df,
  dependent.variable.name = dependent.variable.name,
  predictor.variable.names = predictor.variable.names,
  spatial.predictors.df = spatial.predictors,
  ranking.method = "moran",
  reference.moran.i = model$spatial.correlation.residuals$max.moran,
  distance.matrix = distance_matrix,
  distance.thresholds = 0,
  n.cores = 1
)

#selecting the best subset of predictors
selection <- select_spatial_predictors_recursive(
  data = plant_richness_df,
  dependent.variable.name = dependent.variable.name,
  predictor.variable.names = predictor.variable.names,
  distance.matrix = distance_matrix,
  distance.thresholds = 0,
  spatial.predictors.df = spatial.predictors,
  spatial.predictors.ranking = spatial.predictors.ranking,
  n.cores = 1
)
```

```

selection$optimization
selection$best.spatial.predictors
plot_optimization(selection$optimization)
}

```

---

```
select_spatial_predictors_sequential
```

*Sequential introduction of spatial predictors into a model*

---

### Description

Selects spatial predictors by adding them sequentially into a model while monitoring the Moran's I of the model residuals and the model's R-squared. Once all the available spatial predictors have been added to the model, the function identifies the first  $n$  predictors that minimize the spatial correlation of the residuals and maximize R-squared, and returns the names of the selected spatial predictors and a data frame with the selection criteria.

### Usage

```

select_spatial_predictors_sequential(
  data = NULL,
  dependent.variable.name = NULL,
  predictor.variable.names = NULL,
  distance.matrix = NULL,
  distance.thresholds = NULL,
  ranger.arguments = NULL,
  spatial.predictors.df = NULL,
  spatial.predictors.ranking = NULL,
  weight.r.squared = 0.75,
  weight.penalization.n.predictors = 0.25,
  verbose = FALSE,
  n.cores = parallel::detectCores() - 1,
  cluster = NULL
)

```

### Arguments

`data` Data frame with a response variable and a set of predictors. Default: NULL

`dependent.variable.name` Character string with the name of the response variable. Must be in the column names of data. Default: NULL

`predictor.variable.names` Character vector with the names of the predictive variables. Every element of this vector must be in the column names of data. Default: NULL

<code>distance.matrix</code>	Squared matrix with the distances among the records in data. The number of rows of <code>distance.matrix</code> and data must be the same. If not provided, the computation of the Moran's I of the residuals is omitted. Default: NULL
<code>distance.thresholds</code>	Numeric vector with neighborhood distances. All distances in the distance matrix below each value in <code>distance.thresholds</code> are set to 0 for the computation of Moran's I. If NULL, it defaults to <code>seq(0, max(distance.matrix), length.out = 4)</code> . Default: NULL
<code>ranger.arguments</code>	Named list with <a href="#">ranger</a> arguments (other arguments of this function can also go here). All <a href="#">ranger</a> arguments are set to their default values except for 'importance', that is set to 'permutation' rather than 'none'. Please, consult the help file of <a href="#">ranger</a> if you are not familiar with the arguments of this function.
<code>spatial.predictors.df</code>	Data frame of spatial predictors.
<code>spatial.predictors.ranking</code>	Ranking of the spatial predictors returned by <a href="#">rank_spatial_predictors()</a> .
<code>weight.r.squared</code>	Numeric between 0 and 1, weight of R-squared in the optimization index. Default: 0.75
<code>weight.penalization.n.predictors</code>	Numeric between 0 and 1, weight of the penalization for the number of spatial predictors added in the optimization index. Default: 0.25
<code>verbose</code>	Logical, ff TRUE, messages and plots generated during the execution of the function are displayed, Default: FALSE
<code>n.cores</code>	Integer, number of cores to use. Default: <code>parallel::detectCores() - 1</code>
<code>cluster</code>	A cluster definition generated by <code>parallel::makeCluster()</code> . Default: NULL

## Details

The algorithm works as follows: If the function [rank\\_spatial\\_predictors](#) returns 10 spatial predictors (sp1 to sp10, ordered from best to worst), [select\\_spatial\\_predictors\\_sequential](#) is going to fit the models  $y \sim \text{predictors} + \text{sp1}$ ,  $y \sim \text{predictors} + \text{sp1} + \text{sp2}$ , until all spatial predictors are used in  $y \sim \text{predictors} + \text{sp1} \dots \text{sp10}$ . The model with lower Moran's I of the residuals and higher R-squared (computed on the out-of-bag data) is selected, and its spatial predictors returned.

## Value

A list with two slots: `optimization`, a data frame with the index of the spatial predictor added on each iteration, the spatial correlation of the model residuals, and the R-squared of the model, and `best.spatial.predictors`, that is a character vector with the names of the spatial predictors that minimize the Moran's I of the residuals and maximize the R-squared of the model.

## Examples

```
if(interactive()){
```

```
#loading example data
data(distance_matrix)
data(plant_richness_df)

#common arguments
dependent.variable.name = "richness_species_vascular"
predictor.variable.names = colnames(plant_richness_df)[5:21]

#non-spatial model
model <- rf(
  data = plant_richness_df,
  dependent.variable.name = dependent.variable.name,
  predictor.variable.names = predictor.variable.names,
  distance.matrix = distance_matrix,
  distance.thresholds = 0,
  n.cores = 1
)

#preparing spatial predictors
spatial.predictors <- mem_multithreshold(
  distance.matrix = distance_matrix,
  distance.thresholds = 0
)

#ranking spatial predictors by their Moran's I (faster option)
spatial.predictors.ranking <- rank_spatial_predictors(
  ranking.method = "moran",
  spatial.predictors.df = spatial.predictors,
  reference.moran.i = model$spatial.correlation.residuals$max.moran,
  distance.matrix = distance_matrix,
  distance.thresholds = 0,
  n.cores = 1
)

#selecting the best subset of predictors
selection <- select_spatial_predictors_sequential(
  data = plant_richness_df,
  dependent.variable.name = dependent.variable.name,
  predictor.variable.names = predictor.variable.names,
  distance.matrix = distance_matrix,
  distance.thresholds = 0,
  spatial.predictors.df = spatial.predictors,
  spatial.predictors.ranking = spatial.predictors.ranking,
  n.cores = 1
)

selection$optimization
selection$best.spatial.predictors
plot_optimization(selection$optimization)
}
```

---

standard_error	<i>Standard error of the mean of a numeric vector</i>
----------------	---

---

**Description**

Computes the standard error of the mean of a numeric vector as `round(sqrt(var(x)/length(x)), 3)`

**Usage**

```
standard_error(x)
```

**Arguments**

x                    A numeric vector.

**Details**

The function removes NA values before computing the standard error, and rounds the result to 3 decimal places.

**Value**

A numeric value.

**Examples**

```
if(interactive()){  
  standard_error(runif(10))  
}
```

---

statistical_mode	<i>Statistical mode of a vector</i>
------------------	-------------------------------------

---

**Description**

Computes the mode of a numeric or character vector

**Usage**

```
statistical_mode(x)
```

**Arguments**

x                    Numeric or character vector.

**Value**

Statistical mode of x.

**Examples**

```
if(interactive()){
  statistical_mode(c(10, 9, 10, 8))
}
```

---

the\_feature\_engineer *Suggest variable interactions and composite features for random forest models*

---

**Description**

Suggests candidate variable interactions and composite features able to improve predictive accuracy over data not used to train the model via spatial cross-validation with `rf_evaluate()`. For a pair of predictors a and b, interactions are built via multiplication ( $a * b$ ), while composite features are built by extracting the first factor of a principal component analysis performed with `pca()`, after rescaling a and b between 1 and 100. Interactions and composite features are named `a . x . b` and `a . pca . b` respectively.

Candidate variables a and b are selected from those predictors in `predictor.variable.names` with a variable importance above `importance.threshold` (set by default to the median of the importance scores).

For each interaction and composite feature, a model including all the predictors plus the interaction or composite feature is fitted, and its R squared (or AUC if the response is binary) computed via spatial cross-validation (see `rf_evaluate()`) is compared with the R squared of the model without interactions or composite features.

From all the potential interactions screened, only those with a positive increase in R squared (or AUC when the response is binomial) of the model, a variable importance above the median, and a maximum correlation among themselves and with the predictors in `predictor.variable.names` not higher than `cor.threshold` (set to 0.5 by default) are selected. Such a restrictive set of rules ensures that the selected interactions can be used right away for modeling purposes without increasing model complexity unnecessarily. However, the suggested variable interactions might not make sense from a domain expertise standpoint, so please, examine them with care.

The function returns the criteria used to select the interactions, and the data required to use these interactions a model.

**Usage**

```
the_feature_engineer(
  data = NULL,
  dependent.variable.name = NULL,
  predictor.variable.names = NULL,
```

```

xy = NULL,
ranger.arguments = NULL,
repetitions = 30,
training.fraction = 0.75,
importance.threshold = 0.75,
cor.threshold = 0.75,
point.color = viridis::viridis(100, option = "F", alpha = 0.8),
seed = NULL,
verbose = TRUE,
n.cores = parallel::detectCores() - 1,
cluster = NULL
)

```

## Arguments

<code>data</code>	Data frame with a response variable and a set of predictors. Default: NULL
<code>dependent.variable.name</code>	Character string with the name of the response variable. Must be in the column names of data. If the dependent variable is binary with values 1 and 0, the argument <code>case.weights</code> of <code>ranger</code> is populated by the function <code>case_weights()</code> . Default: NULL
<code>predictor.variable.names</code>	Character vector with the names of the predictive variables, or object of class "variable_selection" produced by <code>auto_vif()</code> and/or <code>auto_cor()</code> . Every element of this vector must be in the column names of data. Default: NULL
<code>xy</code>	Data frame or matrix with two columns containing coordinates and named "x" and "y". If not provided, the comparison between models with and without variable interactions is not done.
<code>ranger.arguments</code>	Named list with <code>ranger</code> arguments (other arguments of this function can also go here). All <code>ranger</code> arguments are set to their default values except for 'importance', that is set to 'permutation' rather than 'none'. Please, consult the help file of <code>ranger</code> if you are not familiar with the arguments of this function.
<code>repetitions</code>	Integer, number of spatial folds to use during cross-validation. Must be lower than the total number of rows available in the model's data. Default: 30
<code>training.fraction</code>	Proportion between 0.5 and 0.9 indicating the proportion of records to be used as training set during spatial cross-validation. Default: 0.75
<code>importance.threshold</code>	Numeric between 0 and 1, quantile of variable importance scores over which to select individual predictors to explore interactions among them. Larger values reduce the number of potential interactions explored. Default: 0.75
<code>cor.threshold</code>	Numeric, maximum Pearson correlation between any pair of the selected interactions, and between any interaction and the predictors in <code>predictor.variable.names</code> . Default: 0.75
<code>point.color</code>	Colors of the plotted points. Can be a single color name (e.g. "red4"), a character vector with hexadecimal codes (e.g. "#440154FF" "#21908CFF" "#FDE725FF"),

	or function generating a palette (e.g. <code>viridis::viridis(100)</code> ). Default: <code>viridis::viridis(100, option = "F", alpha = 0.8)</code>
seed	Integer, random seed to facilitate reproducibility. If set to a given number, the results of the function are always the same. Default: NULL
verbose	Logical. If TRUE, messages and plots generated during the execution of the function are displayed. Default: TRUE
n.cores	Integer, number of cores to use for parallel execution. Creates a socket cluster with <code>parallel::makeCluster()</code> , runs operations in parallel with <code>foreach</code> and <code>%dopar%</code> , and stops the cluster with <code>parallel::clusterStop()</code> when the job is done. Default: <code>parallel::detectCores() - 1</code>
cluster	A cluster definition generated with <code>parallel::makeCluster()</code> . If provided, overrides <code>n.cores</code> . When <code>cluster = NULL</code> (default value), and <code>model</code> is provided, the cluster in <code>model</code> , if any, is used instead. If this cluster is NULL, then the function uses <code>n.cores</code> instead. The function does not stop a provided cluster, so it should be stopped with <code>parallel::stopCluster()</code> afterwards. The cluster definition is stored in the output list under the name "cluster" so it can be passed to other functions via the <code>model</code> argument, or using the <code>%&gt;%</code> pipe. Default: NULL

## Value

A list with seven slots:

- `screening`: Data frame with selection scores of all the interactions considered.
- `selected`: Data frame with selection scores of the selected interactions.
- `df`: Data frame with the computed interactions.
- `plot`: List of plots of the selected interactions versus the response variable. The output list can be plotted all at once with `patchwork::wrap_plots(p)` or `cowplot::plot_grid(plotlist = p)`, or one by one by extracting each plot from the list.
- `data`: Data frame with the response variable, the predictors, and the selected interactions, ready to be used as `data` argument in the package functions.
- `dependent.variable.name`: Character, name of the response.
- `predictor.variable.names`: Character vector with the names of the predictors and the selected interactions.

## Examples

```
if(interactive()){

  #load example data
  data(plant_richness_df)

  new.features <- the_feature_engineer(
    data = plant_richness_df,
    dependent.variable.name = "richness_species_vascular",
    predictor.variable.names = colnames(plant_richness_df)[5:21],
    n.cores = 1,
```



```
      verbose = TRUE
    )

    new.features$screening
    new.features$selected
    new.features$columns

  }
```

---

**thinning***Applies thinning to pairs of coordinates*

---

### Description

Resamples a set of points with x and y coordinates to impose a minimum distance among nearby points.

### Usage

```
thinning(xy, minimum.distance = NULL)
```

### Arguments

<code>xy</code>	A data frame with columns named "x" and "y" representing geographic coordinates.
<code>minimum.distance</code>	Numeric, minimum distance to be set between nearby points, in the same units as the coordinates of <code>xy</code> .

### Details

Generally used to remove redundant points that could produce pseudo-replication, and to limit sampling bias by disaggregating clusters of points.

### Value

A data frame with the same columns as `xy` with points separated by the defined minimum distance.

### See Also

[thinning\\_til\\_n\(\)](#)

**Examples**

```

if(interactive()){

  #load example data
  data(plant_richness_df)

  #thinning to points separated by 5 degrees
  plant_richness.thin <- thinning(
    x = plant_richness_df,
    minimum.distance = 5 #points separated by at least 5 degrees
  )

  plant_richness.thin

}

```

---

thinning\_til\_n

*Applies thinning to pairs of coordinates until reaching a given n*


---

**Description**

Resamples a set of points with x and y coordinates by increasing the distance step by step until a given sample size is obtained.

**Usage**

```

thinning_til_n(
  xy,
  n = 30,
  distance.step = NULL
)

```

**Arguments**

xy	A data frame with columns named "x" and "y" representing geographic coordinates. Default: NULL
n	Integer, number of samples to obtain. Must be lower than nrow(xy). Default: 30
distance.step	Numeric, distance step used during the thinning iterations. If NULL, the one percent of the maximum distance among points in xy is used. Default: NULL

**Value**

A data frame with the same columns as xy with a row number close to n.

**See Also**

[thinning\(\)](#)

## Examples

```
if(interactive()){  
  
  #loading example data  
  data(plant_richness_df)  
  
  #thinning to ~20 records  
  plant_richness.thin <- thinning_til_n(  
    x = plant_richness_df,  
    n = 20  
  )  
  
  plant_richness.thin  
  
}
```

---

vif

*Variance Inflation Factor of a data frame*

---

## Description

Computes the variance inflation factor (VIF) of the columns in a data frame. **Warning:** variables in preference.order not in colnames(x), and non-numeric columns are removed silently from x and preference.order. The same happens with rows having NA values (`na.omit()` is applied). The function issues a warning if zero-variance columns are found.

## Usage

```
vif(x)
```

## Arguments

x                      Data frame with numeric columns, typically containing a set of model predictors.

## Value

A data frame with two columns having the name of the variables in 'x' and their respective VIF values.

## See Also

[auto\\_vif\(\)](#), [auto\\_cor\(\)](#)

**Examples**

```
if(interactive()){  
  data(plant_richness_df)  
  vif(plant_richness_df[, 5:21])  
}
```

---

weights\_from\_distance\_matrix

*Transforms a distance matrix into a matrix of weights*

---

**Description**

Transforms a distance matrix into weights ( $1/\text{distance.matrix}$ ) normalized by the row sums. Used to compute Moran's I values and Moran's Eigenvector Maps. Allows to apply a threshold to the distance matrix before computing the weights.

**Usage**

```
weights_from_distance_matrix(  
  distance.matrix = NULL,  
  distance.threshold = 0  
)
```

**Arguments**

`distance.matrix`  
Distance matrix. Default: NULL.

`distance.threshold`  
Numeric, positive, in the range of values of `distance.matrix`. Distances below this value in the distance matrix are set to 0., Default: 0.

**Value**

A weighted distance matrix.

**Examples**

```
if(interactive()){  
  
  #loading example distance matrix  
  data(distance_matrix)  
  
  #computing matrix of weights  
  distance.matrix.weights <- weights_from_distance_matrix(  
    distance.matrix = distance_matrix
```

*weights\_from\_distance\_matrix*

101

```
)  
distance.matrix.weights  
}
```

# Index

- \* **datasets**
  - distance\_matrix, 10
  - plant\_richness\_df, 36
- aes, 62, 63
- auc, 3
- auto\_cor, 4
- auto\_cor(), 6, 7, 48, 50, 64, 95, 99
- auto\_vif, 6
- auto\_vif(), 4, 5, 48, 50, 64, 95, 99
  
- beowulf\_cluster, 7
  
- case\_weights, 9
- case\_weights(), 48, 50, 64, 76, 79, 95
  
- default\_distance\_thresholds, 10
- default\_distance\_thresholds(), 79
- distance\_matrix, 10, 37
- double\_center\_distance\_matrix, 11
- double\_center\_distance\_matrix(), 27, 28
  
- eigen, 27, 28
  
- filter\_spatial\_predictors, 12
  
- geom\_abline, 62, 63
- geom\_freqpoly, 62, 63
- geom\_point, 47
- geom\_qq\_line, 62, 63
- geom\_smooth, 48
- get\_evaluation, 13
- get\_evaluation(), 38, 54
- get\_importance, 14
- get\_importance(), 39, 55
- get\_importance\_local, 15
- get\_moran, 16
- get\_moran(), 57
- get\_performance, 17
- get\_performance(), 58
- get\_predictions, 18
  
- get\_residuals, 19
- get\_response\_curves, 20
- get\_spatial\_predictors, 21
- ggplot, 62, 63
- ggtheme, 62, 63
  
- huxtable, 56
  
- is\_binary, 22
  
- labs, 62, 63
  
- make\_spatial\_fold, 23
- make\_spatial\_fold(), 25, 26
- make\_spatial\_folds, 25
- make\_spatial\_folds(), 23, 24, 68, 70, 71, 73
- makeCluster, 8
- mem, 27
- mem(), 11, 28
- mem\_multithreshold, 28
- mem\_multithreshold(), 11, 28, 58
- moran, 29, 31
- moran(), 16, 30, 31, 41, 44, 57
- moran\_multithreshold, 30
- moran\_multithreshold(), 16, 30, 41, 57, 65, 77, 82
  
- na.omit(), 4, 6, 99
  
- objects\_size, 32
- optimization\_function, 33
  
- pca, 34
- pca(), 35, 94
- pca\_multithreshold, 35
- pca\_multithreshold(), 34, 58
- plant\_richness\_df, 10, 11, 36
- plot\_annotation, 62, 63
- plot\_evaluation, 37
- plot\_evaluation(), 13, 54
- plot\_importance, 39

- plot\_importance(), 15, 55
- plot\_moran, 40
- plot\_moran(), 16, 44, 57
- plot\_optimization, 42
- plot\_residuals\_diagnostics, 43
- plot\_response\_curves, 45
- plot\_response\_curves(), 21, 47
- plot\_response\_surface, 46
- plot\_response\_surface(), 46
- plot\_training\_df, 48
- plot\_training\_df\_moran, 49
- plot\_tuning, 51
- prcomp, 34
- prepare\_importance\_spatial, 52
- print.rf, 53
- print\_evaluation, 54
- print\_evaluation(), 13, 38, 53
- print\_importance, 55
- print\_importance(), 15, 39, 53
- print\_moran, 56
- print\_moran(), 16, 53
- print\_performance, 57
- print\_performance(), 18, 53, 58
  
- quantile, 20, 45, 47
  
- ranger, 59, 63–65, 76, 79, 88, 91, 95
- rank\_spatial\_predictors, 58, 91
- rank\_spatial\_predictors(), 81, 87, 88, 91
- rescale\_vector, 61
- residuals\_diagnostics, 62
- residuals\_test, 63
- residuals\_test(), 65, 77, 82
- rf, 55, 59, 63
- rf(), 9, 14–20, 39, 41, 44–47, 52, 53, 56, 57, 67, 70, 73, 75, 79, 84
- rf\_compare, 66
- rf\_evaluate, 69
- rf\_evaluate(), 13, 23, 24, 26, 37, 38, 54, 64, 67, 68, 75, 76, 79, 83, 85, 94
- rf\_importance, 72
- rf\_importance(), 39
- rf\_repeat, 55, 59, 75
- rf\_repeat(), 14–21, 39, 41, 44–47, 52, 53, 56, 57, 70, 73
- rf\_spatial, 55, 78
- rf\_spatial(), 14–22, 27, 28, 35, 39, 41, 42, 44–47, 52, 53, 56–58, 67, 70, 73, 75
- rf\_tuning, 83
  
- rf\_tuning(), 51, 64, 67, 75, 76, 79
- root\_mean\_squared\_error, 85
- root\_mean\_squared\_error(), 63
  
- scale, 63, 65, 76, 80
- select\_spatial\_predictors\_recursive, 87
- select\_spatial\_predictors\_recursive(), 33, 42, 81, 88
- select\_spatial\_predictors\_sequential, 90, 91
- select\_spatial\_predictors\_sequential(), 33, 42, 81, 87
- shapiro.test(), 62, 63
- standard\_error, 93
- statistical\_mode, 93
  
- the\_feature\_engineer, 94
- the\_feature\_engineer(), 65
- thinning, 97
- thinning(), 25, 71, 98
- thinning\_til\_n, 98
- thinning\_til\_n(), 25, 67, 70, 71, 73, 97
  
- vif, 99
  
- weights\_from\_distance\_matrix, 100
- weights\_from\_distance\_matrix(), 11, 35, 81
- wrap\_plots, 41, 45, 48, 49