Package: CroPlotR (via r-universe)

June 2, 2026

Type Package

Title A Package to Analyze Crop Model Simulations Outputs with Plots and Statistics

Version 1.0.1

Date 2026-02-26

Description CroplotR aims at the standardization of the process of analyzing the outputs of crop models using plots and statistics. Users can generate plots presented in dynamic mode with time on the x axis and any simulated and/or observed variable(s) on the y axis or in scatter mode with simulation on the y axis and observations on the x axis. Users can differentiate simulations and observations depending on the situation, model version, or any grouping variable. Thirty two corresponding statistics can be computed in the same manner to assess the quality of the predictions of a model.

License file LICENSE

URL https://github.com/SticsRPacks/CroPlotR, https://doi.org/10.5281/zenodo.4442330

BugReports https://github.com/SticsRPacks/CroPlotR/issues

Depends R (>= 4.2.0)

Imports cli, dplyr, ggh4x (>= 0.3.1), ggplot2, ggrepel, gridExtra, lifecycle, plyr, reshape2, rlang, tibble, tidyselect, utils

Suggests patchwork, spelling, SticsRFiles (>= 1.1.3), testthat, vdiffr

Config/testthat/edition 3

Encoding UTF-8

Language en-US

LazyData true

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

Config/pak/sysreqs libicu-dev

Repository https://sticsrpacks.r-universe.dev

Date/Publication 2026-04-03 12:22:09 UTC

RemoteUrl https://github.com/SticsRPacks/CroPlotR

RemoteRef HEAD

RemoteSha ba93557c6a9de1b4c4f1e04b608c65af8d1491a6

Contents

`[.cropr_simulation` `[` *method for* `cropr_simulation`

Description

This method ensure keeping the `cropr_simulation` attribute when subsetting a `cropr_simulation` list.

Usage

```
## S3 method for class 'cropr_simulation'
x[...]
```

Arguments

`x`	A `cropr_simulation` list
`...`	An index

Value

A subset of a `cropr_simulation`, keeping its attribute

Examples

```
## Not run:
library(SticsRFiles)
sim <- SticsRFiles::get_sim(workspace = "inst/extdata/stics_example_1")
sim[[1]] # returns a `cropr_simulation` list

## End(Not run)
```

`bind_rows` *Bind simulation list into dataframe*

Description

Bind simulations list with different situations into a single dataframe

Usage

```
bind_rows(..., .id = NULL)
```

Arguments

`...`	Simulation outputs in Cropr format, *i.e.* a named list of `data.frame` for each situation.
`.id`	Name of the column in the new dataframe that identifies the origin of each row. If `...` is a simulation output, it is set to "situation" by default.

Details

If `...` is not of class `cropr_simulation`, it uses the regular function from `dplyr`. See *e.g.* `SticsRFiles::get_sim()` for an example output format.

Value

A single data.frame or tibble binding the rows of all data.Frames or tibbles included in sim

Note

You can perform the same for observations with the following: `bind_rows(obs, .id = "situation")`.

See Also

split_df2sim

Examples

```
## Not run:
# Importing an example with three situations with observation:
workspace <- system.file(file.path("extdata", "stics_example_1"),
  package = "CroPlotR"
)
situations <- SticsRFiles::get_usms_list(
  file =
    file.path(workspace, "usms.xml")
)
sim <- SticsRFiles::get_sim(workspace = workspace, usm = situations)

bind_rows(sim)

## End(Not run)
```

`detect_mixture` *Detects if a situation is a mixture*

Description

This function checks if the situation is a mixture based on the presence of a column named "Dominance" and the uniqueness of its values.

Usage

```
detect_mixture(sim_situation)
```

Arguments

`sim_situation`	A data frame containing the simulated data for one situation.

Value

A logical value indicating if the situation is a mixture.

Examples

```
## Not run:
sim_data <- data.frame(
  Dominance = c("Principal", "Principal", "Associated", "Associated")
)
CroPlotR:::detect_mixture(sim_data)
# Output: TRUE

sim_data <- data.frame(Dominance = c("Single Crop", "Single Crop"))
CroPlotR:::detect_mixture(sim_data)
# Output: FALSE
```

```
sim_data <- data.frame(lai = c(1, 1.2))
CroPlotR:::detect_mixture(sim_data)
# Output: FALSE

## End(Not run)
```

`extract_plot` *Extract plot(s) from ggplot*

Description

Extract a plot corresponding to one or several variables from a ggplot.

Usage

```
extract_plot(
  plot,
  var = NULL,
  situation = NULL,
  force = FALSE,
  verbose = TRUE,
  situations = lifecycle::deprecated()
)
```

Arguments

`plot`	The output of plot_situations.
`var`	Vector of variable names for which plots have to be extracted. Optional, all variables considered by default.
`situation`	A list of situations names to extract from a list of ggplots
`force`	Continue if the plot is not possible ? E.g. no observations for scatter plots. If `TRUE`, return `NULL`, else return an error (default).
`verbose`	Logical value for displaying information while running.
`situations`	**[Deprecated]** `situations` is no longer supported, use `situation` instead.

Value

A (printed) list of ggplot objects, each element being a plot for a situation

`format_cropr` *Format simulations and observations from CropR format to a format usable by CroplotR*

Description

Format simulations (and observations if any) for plotting. This function can be used as a template to include other models in CroPlotR.

Usage

```
format_cropr(
  sim,
  obs = NULL,
  obs_sd = NULL,
  type = c("dynamic", "scatter"),
  select_dyn = c("sim", "common", "obs", "all"),
  select_scat = c("sim", "res"),
  successive = NULL,
  reference_var = NULL,
  variable = NULL,
  verbose = TRUE
)
```

Arguments

`sim`	A simulation data.frame
`obs`	An observation data.frame
`obs_sd`	A data.frame with observation standard deviations
`type`	The type of plot required, either "dynamic" or "scatter"
`select_dyn`	Which data to plot when `type= "dynamic"`? See details.
`select_scat`	Which data to plot when `type= "scatter"`? See details.
`successive`	A list of lists containing the situations to be represented as a contiguous sequence when type = "dynamic" (dates should be contiguous) when `type = "dynamic"` (implies that the situations are correctly ordered).
`reference_var`	Variable selected on x-axis when type is scatter and select_scat is res. It is possible to select between observation and simulation of the reference variable.
`variable`	A character vector indicating the variables to be kept in the formatted data frame. If NULL (default), all variables are kept.
`verbose`	Logical value for displaying information while running.

Details

The `select_dyn` argument can be:

- "sim" (the default): all variables with simulations outputs, and observations when there are some
- "common": variables with simulations outputs and observations in common (used when `type="scatter"`)
- "obs": all variables with observations, and simulations outputs when there are some
- "all": all variables with any observations or simulations outputs

The `select_scat` argument can be:

- "sim" (the default): plots observations in X and simulations in Y.
- "res": plots observations in X and residuals(observations-simulations)in Y.

Value

A pre-formatted `data.frame` or `NULL` if the formatting is not possible (e.g. type="scatter" but no common variables in obs and sim).

Examples

```
## Not run:
# remotes::install_github("Stics RPacks/SticsRPacks")
workspace <- system.file(file.path("extdata", "stics_example_1"),
  package = "CroPlotR"
)
situation <- SticsRFiles::get_usms_list(
  file =
    file.path(workspace, "usms.xml")
)[1]
sim <- SticsRFiles::get_sim(workspace = workspace, usm = situation)
obs <- SticsRFiles::get_obs(workspace = workspace, usm = situation)
formated_df <- format_cropr(
  sim$`IC_Wheat_Pea_2005-2006_N0`,
  obs$`IC_Wheat_Pea_2005-2006_N0`
)
options(max.print = 100)
formated_df

## End(Not run)
```

plot.cropr_simulation *Plot situations by group of simulation*

Description

simulation outputs for one or several situations with or without observations, eventually grouped by a model version (or any group actually)

Usage

```
## S3 method for class 'cropr_simulation'
plot(
  ...,
  obs = NULL,
  obs_sd = NULL,
  type = c("dynamic", "scatter"),
  select_dyn = c("sim", "common", "obs", "all"),
  select_scat = c("sim", "res"),
  var = NULL,
  title = NULL,
  all_situations = TRUE,
  overlap = NULL,
  successive = NULL,
  shape_sit = c("none", "txt", "symbol", "group"),
  situation_group = NULL,
  reference_var = NULL,
  force = FALSE,
  verbose = TRUE
)

autoplot.cropr_simulation(
  ...,
  obs = NULL,
  obs_sd = NULL,
  type = c("dynamic", "scatter"),
  select_dyn = c("sim", "common", "obs", "all"),
  select_scat = c("sim", "res"),
  var = NULL,
  title = NULL,
  all_situations = TRUE,
  overlap = NULL,
  successive = NULL,
  shape_sit = c("none", "txt", "symbol", "group"),
  situation_group = NULL,
  reference_var = NULL,
  force = FALSE,
  verbose = TRUE
```

)

Arguments

...	Simulation outputs (each element= model version), each being a named list of data.frame for each situation. See examples.
obs	A list (each element= situation) of observations data.frames (named by situation)
obs_sd	A list (each element= situation) of standard deviations of observations data.frames (named by situation)
type	The type of plot requested, either "dynamic" (date in X, variable in Y) or scatter (simulated VS observed)
select_dyn	Which data to plot when type= "dynamic"? See details.
select_scat	Which data to plot when type= "scatter"? See details.
var	Vector of variable names for which plots have to be created. Optional, all variables considered by default.
title	A vector of plot titles, named by situation. Use the situation name if NULL, recycled if length one.
all_situations	Boolean (default = TRUE). If TRUE, plot all situations on the same graph.
overlap	A list of lists containing the variables to represent on the same graph when type = "dynamic".
successive	A list of lists containing the situations to be represented as a contiguous sequence. Dates of variables must be continuous within a sequence of situations. when type = "dynamic" (implies that the situations are correctly ordered).
shape_sit	Shape to differentiate between situations when all_situations= TRUE. See details.
situation_group	A list of lists of situations to gather when shape_sit= "group".
reference_var	Variable selected on x-axis when type= "scatter" and select_scat= "res". It is possible to select between observation and simulation of the reference variable. (examples : reference_var = "lai_n_obs", reference_var = "mafruit_sim")
force	Continue if the plot is not possible ? E.g. no observations for scatter plots. If TRUE, return NULL, else return an error (default).
verbose	Boolean. Print information during execution.

Details

The select_dyn argument can be:

- "sim" (the default): all variables with simulations outputs, and o bservations when there are some
- "common": variables with simulations outputs and observations in common
- "obs": all variables with observations, and simulations outputs when there are some
- "all": all variables with any observations or simulations outputs

The select_scat argument can be:

- "sim" (the default): plots observations in X and simulations in Y.
- "res": plots observations in X and residuals (observations-simulations) in Y.

The shape_sit argument can be:

- "none" (the default): Same shape for all situations.
- "txt": Writes the name of the situation above each point.
- "symbol": One shape for each situation.
- "group": One shape for each group of situations described in situation_group.

Value

A (printed) list of ggplot objects, each element being a plot for a situation

Note

The error bar will be equal to 2*obs_sd on each side of the point to have 95% confidence.

The plots titles are given by their situation name.

Examples

```
## Not run:
# Importing an example with three situations with observation:
workspace <- system.file(file.path("extdata", "stics_example_1"),
  package = "CroPlotR"
)
situations <- SticsRFiles::get_usms_list(
  file =
    file.path(workspace, "usms.xml")
)
sim <- SticsRFiles::get_sim(workspace = workspace, usm = situations)
obs <- SticsRFiles::get_obs(workspace = workspace, usm = situations)

plot(sim, obs = obs)

## End(Not run)
```

---

plot.statistics *Plot statistics*

Description

Plot statistics

Usage

```
## S3 method for class 'statistics'
plot(
  x,
  xvar = c("group", "situation"),
  type = c("bar", "radar"),
  group_bar = c("rows", "stack", "dodge"),
  crit_radar = NULL,
  title = NULL,
  force = FALSE,
  verbose = TRUE,
  ...
)
```

Arguments

x	The output of summary.cropr_simulation()
xvar	The variable to use in x, either the group or the situation (the other is used for colouring)
type	The type of plot requested, either "bar" (bar plot) or "radar" (radar chart)
group_bar	Way to display the different statistical criteria when type= "bar". See details.
crit_radar	Statistical criterion chosen to be displayed on the radar chart.
title	The plot title
force	Continue if the plot is not possible ? E.g. no observations for scatter plots. If TRUE, return NULL, else return an error (default).
verbose	Boolean. Print information during execution.
...	Other arguments to pass (for backward compatibility only)

Details

The group_bar argument can be:

- "rows" (the default): One line of graphs per statistical criterion
- "stack": Bars of each statistical criterion stacked
- "dodge": Bars of each statistical criterion side by side

Value

Return a ggplot object with statistics

Examples

```
# Importing an example with three situations with observation:
workspace <- system.file(file.path("extdata", "stics_example_1"),
  package = "CroPlotR"
)
```

```
situations <- SticsRFiles::get_usms_list(
  file =
    file.path(workspace, "usms.xml")
)
sim <- SticsRFiles::get_sim(workspace = workspace, usm = situations)
obs <- SticsRFiles::get_obs(workspace = workspace, usm = situations)

# R2 and nRMSE stats for the simulation:
stats <- summary(sim, obs = obs, stats = c("R2", "nRMSE"))
plot(stats)

# Change the group name:
stats <- summary("stics v9.0" = sim, obs = obs, stats = c("R2", "nRMSE"))
plot(stats)

# R2 and nRMSE stats for two groups of simulations:
summary(sim1 = sim, sim2 = sim, obs = obs, stats = c("R2", "nRMSE"))
```

`predictor_assessment` *Model quality assessment*

Description

Provide several metrics to assess the quality of the predictions of a model (see note) against observations.

Usage

```
n_obs(obs)

mean_obs(obs, na.rm = TRUE)

mean_sim(sim, na.rm = TRUE)

sd_obs(obs, na.rm = TRUE)

sd_sim(sim, na.rm = TRUE)

CV_obs(obs, na.rm = TRUE)

CV_sim(sim, na.rm = TRUE)

r_means(sim, obs, na.rm = TRUE)

R2(sim, obs, na.action = stats::na.omit)

SS_res(sim, obs, na.rm = TRUE)
```

```
Inter(sim, obs, na.action = stats::na.omit)

Slope(sim, obs, na.action = stats::na.omit)

RMSE(sim, obs, na.rm = TRUE)

RMSEs(sim, obs, na.rm = TRUE)

RMSEu(sim, obs, na.rm = TRUE)

nRMSE(sim, obs, na.rm = TRUE)

rRMSE(sim, obs, na.rm = TRUE)

rRMSEs(sim, obs, na.rm = TRUE)

rRMSEu(sim, obs, na.rm = TRUE)

pMSEs(sim, obs, na.rm = TRUE)

pMSEu(sim, obs, na.rm = TRUE)

Bias2(sim, obs, na.rm = TRUE)

SDSD(sim, obs, na.rm = TRUE)

LCS(sim, obs, na.rm = TRUE)

rbias2(sim, obs, na.rm = TRUE)

rSDSD(sim, obs, na.rm = TRUE)

rLCS(sim, obs, na.rm = TRUE)

MAE(sim, obs, na.rm = TRUE)

ABS(sim, obs, na.rm = TRUE)

MSE(sim, obs, na.rm = TRUE)

EF(sim, obs, na.rm = TRUE)

NSE(sim, obs, na.rm = TRUE)

Bias(sim, obs, na.rm = TRUE)

MAPE(sim, obs, na.rm = TRUE)
```

```
FVU(sim, obs, na.rm = TRUE)

RME(sim, obs, na.rm = TRUE)

tSTUD(sim, obs, na.rm = TRUE)

tLimit(sim, obs, risk = 0.05, na.rm = TRUE)

Decision(sim, obs, risk = 0.05, na.rm = TRUE)
```

Arguments

obs	Observed values
na.rm	Boolean. Remove NA values if TRUE (default)
sim	Simulated values
na.action	A function which indicates what should happen when the data contain NAs.
risk	Risk of the statistical test

Details

The statistics for model quality can differ between sources. Here is a short description of each statistic and its equation (see html version for LATEX):

- n_obs(): Number of observations.
- mean_obs(): Mean of observed values
- mean_sim(): Mean of simulated values
- sd_obs(): Standard deviation of observed values
- sd_sim(): standard deviation of simulated values
- CV_obs(): Coefficient of variation of observed values
- CV_sim(): Coefficient of variation of simulated values
- r_means(): Ratio between mean simulated values and mean observed values (%), computed as :

$$r\_means = \frac{100 * \frac{\sum_1^n (\hat{y_i})}{n}}{\frac{\sum_1^n (y_i)}{n}}$$

- R2(): coefficient of determination, computed using stats::lm() on obs~sim.
- SS_res(): residual sum of squares (see notes).
- Inter(): Intercept of regression line, computed using stats::lm() on sim~obs.
- Slope(): Slope of regression line, computed using stats::lm() on sim~obs.
- RMSE(): Root Mean Squared Error, computed as

$$RMSE = \sqrt{\frac{\sum_1^n (\hat{y_i} - y_i)^2}{n}}$$

- `RMSEs()`: Systematic Root Mean Squared Error, computed as

$$RMSEs = \sqrt{\frac{\sum_1^n (\sim y_i - y_i)^2}{n}}$$

- `RMSEu()`: Unsystematic Root Mean Squared Error, computed as

$$RMSEu = \sqrt{\frac{\sum_1^n (\sim y_i - \hat{y_i})^2}{n}}$$

- `NSE()`: Nash-Sutcliffe Efficiency, alias of EF, provided for user convenience.
- `nRMSE()`: Normalized Root Mean Squared Error, also denoted as CV(RMSE), and computed as:

$$nRMSE = \frac{RMSE}{\bar{y}} \cdot 100$$

- `rRMSE()`: Relative Root Mean Squared Error, computed as:

$$rRMSE = \frac{RMSE}{\bar{y}}$$

- `rRMSEs()`: Relative Systematic Root Mean Squared Error, computed as

$$rRMSEs = \frac{RMSEs}{\bar{y}}$$

- `rRMSEu()`: Relative Unsystematic Root Mean Squared Error, computed as:

$$rRMSEu = \frac{RMSEu}{\bar{y}}$$

- `pMSEs()`: Proportion of Systematic Mean Squared Error in Mean Square Error, computed as:

$$pMSEs = \frac{MSEs}{MSE}$$

- `pMSEu()`: Proportion of Unsystematic Mean Squared Error in MEan Square Error, computed as:

$$pMSEu = \frac{MSEu}{MSE}$$

- `Bias2()`: Bias squared (1st term of Kobayashi and Salam (2000) MSE decomposition):

$$Bias2 = Bias^2$$

- `SDSD()`: Difference between sd_obs and sd_sim squared (2nd term of Kobayashi and Salam (2000) MSE decomposition), computed as:

$$SDSD = (sd\_obs - sd\_sim)^2$$

- `LCS()`: Correlation between observed and simulated values (3rd term of Kobayashi and Salam (2000) MSE decomposition), computed as:

$$LCS = 2 * sd\_obs * sd\_sim * (1 - r)$$

- `rbias2()`: Relative bias squared, computed as:

$$rbias2 = \frac{Bias^2}{\bar{y}^2}$$

- `rSDSD()`: Relative difference between sd_obs and sd_sim squared, computed as:

$$rSDSD = \frac{SDSD}{\bar{y}^2}$$

- `rLCS()`: Relative correlation between observed and simulated values, computed as:

$$rLCS = \frac{LCS}{\bar{y}^2}$$

- `MAE()`: Mean Absolute Error, computed as:

$$MAE = \frac{\sum_1^n(|\hat{y}_i - y_i|)}{n}$$

- `ABS()`: Mean Absolute Bias, which is an alias of `MAE()`
- `FVU()`: Fraction of variance unexplained, computed as:

$$FVU = \frac{SS_{res}}{SS_{tot}}$$

- `MSE()`: Mean squared Error, computed as:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

- `EF()`: Model efficiency, also called Nash-Sutcliffe efficiency (NSE). This statistic is related to the FVU as $EF = 1 - FVU$. It is also related to the R^2 because they share the same equation, except SStot is applied relative to the identity function (*i.e.* 1:1 line) instead of the regression line. It is computed as:

$$EF = 1 - \frac{SS_{res}}{SS_{tot}}$$

- `Bias()`: Modelling bias, simply computed as:

$$Bias = \frac{\sum_1^n(\hat{y}_i - y_i)}{n}$$

- `MAPE()`: Mean Absolute Percent Error, computed as:

$$MAPE = \frac{\sum_1^n(\frac{|\hat{y}_i - y_i|}{y_i})}{n}$$

- `RME()`: Relative mean error, computed as:

$$RME = \frac{\sum_1^n(\frac{\hat{y}_i - y_i}{y_i})}{n}$$

- `tSTUD()`: T student test of the mean difference, computed as:

$$tSTUD = \frac{Bias}{\sqrt{(\frac{var(M)}{n_obs})}}$$

- `tLimit()`: T student threshold, computed using `qt()`:

$$tLimit = qt(1 - \frac{\alpha}{2}, df = length(obs) - 1)$$

- `Decision()`: Decision of the t student test of the mean difference (can bias be considered statistically not different from 0 at alpha level 0.05, i.e. 5% probability of erroneously rejecting this hypothesis?), computed as:

$$Decision = abs(tSTUD) < tLimit$$

Value

A statistic depending on the function used.

Note

SS_{res} is the residual sum of squares and SS_{tot} the total sum of squares. They are computed as:

$$SS_{res} = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

$$SS_{tot} = \sum_{i=1}^{n} (y_i - \bar{y})^2$$

Also, it should be noted that y_i refers to the observed values and $\hat{y}_i$ to the predicted values, $\bar{y}$ to the mean value of observations and $\sim y_i$ to values predicted by linear regression.

Examples

```
## Not run:
sim <- rnorm(n = 5, mean = 1, sd = 1)
obs <- rnorm(n = 5, mean = 1, sd = 1)
RMSE(sim, obs)

## End(Not run)
```

save_plot_pdf *Save CroPlotR plots*

Description

Save the plots to a pdf file

Usage

```
save_plot_pdf(
  plot,
  out_dir,
  file_name = "Graphs",
  title = "Plots",
  file_per_var = FALSE,
  stats = NULL,
  force = FALSE,
  verbose = TRUE,
  path = lifecycle::deprecated(),
  filename = lifecycle::deprecated(),
  main = lifecycle::deprecated()
)
```

Arguments

plot	A list of ggplots : output of plot()
out_dir	The path to the directory where to save the plots
file_name	Name of the pdf file
title	Main title of the pdf document
file_per_var	If TRUE, produces one file per variable instead of one with all variables inside
stats	Output of statistics_situationswith all_situations = FALSE. Needed if file_per_var is TRUE. It is used to classify situations according to the descending RMSE.
force	Continue if the plot is not possible ? If TRUE, return NULL, else return an error (default).
verbose	Logical value for displaying information while running
path	**[Deprecated]** path is no longer supported, use out_dir instead.
filename	**[Deprecated]** filename is no longer supported, use file_name instead.
main	**[Deprecated]** title is no longer supported, use out_dir instead.

Value

Save the plots in a pdf file in the folder specified by the path

save_plot_png *Save CroPlotR plot*

Description

Save the plots to disk

Usage

```
save_plot_png(
  plot,
  out_dir,
  suffix = "",
  width = 17,
  height = 12,
  units = "cm",
  dpi = 200,
  scale = 1.2,
  device = NULL,
  path = lifecycle::deprecated()
)
```

Arguments

plot	A list of ggplots : output of `plot()`
out_dir	The path to the directory where to save the plots
suffix	A suffix to append to the file name
width	The plot width
height	The plot heigth
units	The units for plot width and heigth in `units ("in", "cm", or "mm")`
dpi	The plot resolution.
scale	The scaling factor.
device	Device to use. Can either be a device function (e.g. png()), or one of "eps", "ps", "tex" (pictex), "pdf", "jpeg", "tiff", "png", "bmp", "svg" or "wmf" (windows only).
path	**[Deprecated]** `path` is no longer supported, use `out_dir` instead.

Details

The function uses `ggplot2::ggsave()` under the hood.

Value

Save the plots to `path`, named by the situation name, and returns the plots invisibly for piping.

`split_df2sim` *Split data.frame into Cropr format*

Description

Split a row-binded data.frame (or tibble) into a Cropr format simulation list.

Usage

```
split_df2sim(df, add_cropr_attr = TRUE)
```

Arguments

`df`	A single data.frame or tibble containing simulation results (as created by `bind_rows_sim`). MUST include `Date`and `situation` columns.
`add_cropr_attr`	A logical to indicate if the cropr_simulation attribute must be added to the resulting variable Set FALSE if you apply the function to observed data, TRUE otherwise (optional, default value = TRUE).

Value

A named list of `data.frame` for each situation, having the attribute cropr_simulation.

See Also

bind_rows

Examples

```
## Not run:
# Importing an example with three situations with observation:
workspace <- system.file(file.path("extdata", "stics_example_1"),
  package = "CroPlotR"
)
situations <- SticsRFiles::get_usms_list(
  file =
    file.path(workspace, "usms.xml")
)
sim <- SticsRFiles::get_sim(workspace = workspace, usm = situations)

df <- bind_rows(sim)
split_df2sim(df)

## End(Not run)
```

summary.cropr_simulation *Summary statistics of simulations*

Description

Summary statistics for one or several situations with observations, eventually grouped by a model version (or any group actually)

Usage

```
## S3 method for class 'cropr_simulation'
summary(
  ...,
  obs,
  stats = "all",
  all_situations = TRUE,
  verbose = TRUE,
  stat = lifecycle::deprecated()
)
```

Arguments

`...`	Simulation outputs (each element= model version), each being a named list of `data.frame` for each situation. See examples.
`obs`	A list (each element= situation) of observations `data.frames` (named by situation)
`stats`	A character vector of required statistics, "all" for all, or any of `predictor_assessment()`.
`all_situations`	Boolean (default = TRUE). If `TRUE`, computes statistics for all situations.
`verbose`	Logical value for displaying information while running
`stat`	**[Deprecated]** `stat` is no longer supported, use `stats` instead.

Value

A list of statistics `data.frames` named by situation

See Also

All the functions used to compute the statistics: `predictor_assessment()`.

Examples

```
## Not run:
# Importing an example with three situations with observation:
workspace <- system.file(file.path("extdata", "stics_example_1"),
  package = "CroPlotR"
```

```
)
situations <- SticsRFiles::get_usms_list(
  file =
    file.path(workspace, "usms.xml")
)
sim <- SticsRFiles::get_sim(workspace = workspace, usm = situations)
obs <- SticsRFiles::get_obs(workspace = workspace, usm = situations)

# All stats for the simulation:
summary(sim, obs = obs)

# All stats for two groups of simulations:
summary(sim1 = sim, sim2 = sim, obs = obs)

# Only R2 and nRMSE for one group:
summary(sim, obs = obs, stats = c("R2", "nRMSE"))

## End(Not run)
```

Index