Package 'SeaVal'

July 21, 2025

Title Validation of Seasonal Weather Forecasts

Version 1.2.0

Description Provides tools for processing and evaluating seasonal weather forecasts, with an emphasis on tercile forecasts. We follow the World Meteorological Organization's ``Guidance on Verification of Operational Seasonal Climate Forecasts", S.J.Mason (2018, ISBN: 978-92-63-11220-0, URL: <https://library.wmo.int/idurl/4/56227>). The development was supported by the European Union's Horizon 2020 research and innovation programme under grant agreement no. 869730 (CONFER). A comprehensive online tutorial is available at <https://seasonalforecastingengine.github.io/SeaValDoc/>.

URL https://seasonalforecastingengine.github.io/SeaValDoc/, https://github.com/SeasonalForecastingEngine/SeaVal

License GPL (>= 3)

Encoding UTF-8

Depends R (>= 2.10), data.table, ggplot2

Imports ggnewscale, ggplotify, lifecycle, maps, ncdf4, patchwork, RColorBrewer, scales, stringr

LazyData true

RoxygenNote 7.2.3

Collate 'auxiliary_functions.R' 'chirps.R' 'data.R' 'diagrams.R' 'ICPAC_temp.R' 'ncdf_to_dt.R' 'plotting.R' 'quantiles.R' 'scores.R' 'seasonal.R' 'SeaVal-package.R' 'spatial_grids.R' 'upscaling.R' 'utils.R'

NeedsCompilation no

Author Claudio Heinrich-Mertsching [aut, cre, cph] (ORCID: <https://orcid.org/0000-0003-3581-6416>), Celine Cunen [ctb], Michael Scheuerer [ctb]

Maintainer Claudio Heinrich-Mertsching <claudio.heinrich@hotmail.de>

Repository CRAN

Date/Publication 2024-06-14 15:20:05 UTC

Contents

add_climatology . . . 4
add_country . . . 4
add_country_names . . . 5
add_tercile_cat . . . 6
add_tercile_probs . . . 7
are_all_elements_within_eps . . . 7
by_cols_ens_fc_score . . . 8
by_cols_terc_fc_score . . . 8
by_cols_terc_fc_score_sp . . . 9
checks_ens_fc_score . . . 9
checks_terc_fc_score . . . 9
chirps_dir . . . 10
chirps_monthly . . . 10
chirps_ver_map_quantiles . . . 11
climatology_ens_forecast . . . 12
climatology_threshold_exceedence . . . 12
combine . . . 13
complete_regular_grid . . . 14
convert_monthly_to_seasonal . . . 14
CPA . . . 16
create_diagram_by_level . . . 17
CRPS . . . 18
CRPSS . . . 19
crps_aux . . . 20
crps_aux_esc . . . 20
data_dir . . . 21
delete_redundant_files . . . 21
dimvars . . . 22
disc_score_dt . . . 22
DISS . . . 23
download_chirps_monthly . . . 24
download_chirps_monthly_high . . . 25
download_chirps_monthly_low . . . 26
download_chirps_prelim_aux . . . 27
dt_to_netcdf . . . 28
EA_country_names . . . 29
ecmwf_monthly . . . 29
EIR . . . 30
fc_cols . . . 31
get_mask . . . 32
get_quantiles . . . 32
get_terciles . . . 33
ggplot_dt . . . 34
GHA_extent . . . 36
gha_plot . . . 37
grid_info . . . 37

HS . . . 38
HSS . . . 39
IGS . . . 40
IGSS . . . 41
indicator_times_value_aux . . . 42
load_chirps . . . 42
lt_cols . . . 43
MB . . . 44
MBS . . . 45
modify_dt_map_plotting . . . 46
MSD_to_YM . . . 46
MSE . . . 47
MSES . . . 48
netcdf_to_dt . . . 49
obs_cols . . . 50
obs_dimvars . . . 51
PCC . . . 51
profit_graph . . . 52
REL . . . 53
rel_diag . . . 55
rel_diag_vec . . . 56
RES . . . 57
restrict_to_country . . . 58
restrict_to_GHA . . . 59
ROCS . . . 59
ROC_curve . . . 60
roc_curve_vec . . . 61
roc_score_vec . . . 62
round_probs . . . 63
RPS . . . 63
RPSS . . . 64
run_dimension_check_ens_fc_score . . . 65
run_dimension_check_terc_forecast . . . 65
season_strings_to_int . . . 66
set_spatial_grid . . . 66
space_dimvars . . . 68
SRC . . . 69
tc_cols . . . 70
tendency_diag . . . 70
tercile_plot . . . 71
tfc_from_efc . . . 72
tfc_gha_plot . . . 73
tfc_plot . . . 74
time_dimvars . . . 76
upscale_chirps . . . 76
upscale_regular_lon_lat . . . 77
ver_map . . . 79
ver_map_chirps . . . 80

Index **81**

`add_climatology` *Add climatology to a data table*

Description

The climatology is the average over years (and members for ensemble forecases), taken separately for each month, season, and coordinate. By default, the average is taken over all years in the data table, but you can change this using the years-argument. By default, climatologies (averages) are calculated for each column that is not recognized as dimension variable and does not contain characters.

Usage

```
add_climatology(dt, data_cols = NULL, years = NULL, by = dimvars(dt))
```

Arguments

`dt`	the data table.
`data_cols`	For which columns do you want to derive the climatology? The default i
`years`	The average over which years should be considered as climatology. The default is all years in dt.
`by`	column names to group by.

Value

The provided data table with an extra climatology column

Examples

```
dt = add_climatology(chirps_monthly)
```

`add_country` *Same as add_country_names*

Description

This is a synonyme for `add_country_names`. Following a more intuitive naming convention, that is more in-line with `add_climatology` and `add_tercile_cat`.

Usage

```
add_country(dt, regions = EA_country_names())
```

Arguments

dt	the data table.
regions	Character vector of country names for which shapefiles are loaded.

Value

The provided data table with an extra column with country names

Examples

```
dt = add_country(chirps_monthly)
```

---

add_country_names *Add country names to a data table with lon/lat coordinates*

Description

Takes a data table with lon/lat coordinates and adds a column 'country' to it, containing the name of the country, the coordinate belongs to.

Usage

```
add_country_names(dt, regions = EA_country_names())
```

Arguments

dt	the data table.
regions	Character vector of country names for which shapefiles are loaded. By default, countries in East Africa are loaded, see EA_country_names. If you set regions = '.', the entire world is loaded, but this makes the function slower.

Value

The provided data table with an extra column with country names

Examples

```
dt = add_country_names(chirps_monthly)
```

`add_tercile_cat` *Add a tercile-category column to a data table*

Description

Given a data table with multiple years of data, this function derives the tercile category per year. It first derives terciles for the data and then returns, for each row, a -1 if the data falls into the lowest tercile, 0 if it falls between 1st and second tercile, and +1 if it falls above the third tercile. Allows grouping by levels (e.g. months and location-coordinates): Tercile categories are derived separately for each level.

Usage

```
add_tercile_cat(
  dt,
  datacol = NULL,
  years = NULL,
  by = setdiff(dimvars(dt), c("year", "member"))
)
```

Arguments

`dt`	the data table.
`datacol`	Name of the column where the data is stored. If NULL, the function guesses.
`years`	Optional, if provided only these years are used for establishing climatology terciles.
`by`	names of columns to group by.

Value

The provided data table with an extra column tercile_cat

Examples

```
dt = add_tercile_cat(chirps_monthly)
```

add_tercile_probs *Add tercile probabilities to ensemble forecasts*

Description

Adds columns 'below', 'normal' and 'above', containing predicted tercile probabilities, to a data table with ensemble forecasts. The predicted probability is always the fraction of members ending up in the respective tercile. The data table should either already have a column 'tercile_cat' (added by `add_tercile_cat`), or `add_tercile_cat` will be run first.

Usage

```
add_tercile_probs(dt, f = NULL, by = setdiff(dimvars(dt), "member"), ...)
```

Arguments

`dt`	the data table.
`f`	name of the column containing the forecast.
`by`	names of columns to group by
`...`	passed on to `add_tercile_cat`.

Value

The provided data table, with added columns 'above', 'normal', and 'below'

Examples

```
dt = add_tercile_probs(ecmwf_monthly)
```

are_all_elements_within_eps *Check if all elements of x are within tolerance eps of any element in y*

Description

Auxiliary function, used for checking whether spatial grids are regular, with allowing for rounding errors.

Usage

```
are_all_elements_within_eps(x, y, eps)
```

Arguments

x	A numeric vector, sorted in increasing order.
y	A numeric vector, sorted in increasing order.
eps	The tolerance within which we consider two values to be equal.

Value

A boolean value, TRUE if all x are in y within tolerance eps, FALSE otherwise.

---

by_cols_ens_fc_score *Auxiliary function*

Description

returns the default column names to group by when calculating scores of ensemble forecasts.

Usage

```
by_cols_ens_fc_score(dt = NULL)
```

Arguments

dt	optional. You can provide a data table, then the function returns the names of grouping variables in this data table.

Value

A vector of characters with the column names.

---

by_cols_terc_fc_score *Auxiliary function*

Description

returns the default column names to group by when calculating scores for tercile forecasts.

Usage

```
by_cols_terc_fc_score(dt = NULL)
```

Arguments

dt	optional. You can provide a data table, then the function returns the names of grouping variables in this data table.

Value

A vector of characters with the column names.

by_cols_terc_fc_score_sp *Auxiliary function*

Description

Gets column names to group by when calculating scores for tercile forecasts. Some tercile forecasts, such as ROC score or SRC (slope of reliability curve) require many data points and should therefore be pooled in space. This auxiliary function returns the default column names to group by for these scores. The suffix _sp stands for spatial pooling.

Usage

```
by_cols_terc_fc_score_sp(dt = NULL)
```

Arguments

dt	optional. You can provide a data table, then the function returns the names of grouping variables in this data table.

Value

A vector of characters with the column names.

checks_ens_fc_score *Auxiliary function for scores of ensemble forecasts.*

Description

Checks whether the data table contains columns with names that are not allowed, or whether it is missing columns that are required.

Usage

```
checks_ens_fc_score()
```

checks_terc_fc_score *Auxiliary function for scores for tercile forecasts.*

Description

Checks whether the data table contains columns with names that are not allowed, or whether it is missing columns that are required.

Usage

```
checks_terc_fc_score()
```

chirps_dir *CHIRPS directory*

Description

Auxiliary function to access/set the directory for loading and saving CHIRPS data.

Usage

```
chirps_dir(dir = file.path(data_dir(), "CHIRPS"))
```

Arguments

dir	The directory

Value

The directory path.

Examples

```
if(interactive()){chirps_dir()}
```

chirps_monthly *Monthly mean precipitation*

Description

This dataset contains observed monthly mean precipitation for the greater horn of Africa, for November - December 1991-2020. The unit of precipitation is mm/day. It also contains the tercile category, where -1 means below normal rainfall (lowest tercile for this location and month), 0 is normal and 1 is above normal.The data source is CHIRPS-blended, upscaled to a half-degree grid.

Usage

```
data(chirps_monthly)
```

Format

An object of class `data.table` (inherits from `data.frame`) with 209040 rows and 6 columns.

Source

http://iridl.ldeo.columbia.edu/SOURCES/.UCSB/.CHIRPS/.v2p0/.monthly/.global/.precipitation/

`chirps_ver_map_quantiles` *Calculates and saves the quantiles of CHIRPS data required for verification maps.*

Description

Calculates and saves the quantiles of CHIRPS data required for verification maps.

Usage

```
chirps_ver_map_quantiles(
  clim_period = 1991:2020,
  version = "UCSB",
  resolution = "low",
  CHIRPS_dir = chirps_dir(),
  seasons = TRUE
)
```

Arguments

`clim_period`	which years should be considered for the quantiles.
`version`	which version of CHIRPS, 'UCSB' or 'ICPAC'? Can be a vector with both.
`resolution`	If this is set to 'high', the quantiles are also calculated for high-resolution CHIRPS data. This is not nicely implemented right now and will take a lot of memory and time.
`CHIRPS_dir`	directory the CHIRPS data is stored in.
`seasons`	Are we plotting for seasonal or monthly forecasts?

Value

data table with quantiles.

Examples

```
## Not run: chirps_ver_map_quantiles()
```

climatology_ens_forecast

Returns a leave-one-year-out climatology-based ensemble forecast

Description

for a given year, the ensemble forecast simply consists of the observations in all other years. This is essentially an auxiliary function for computing skill scores relative to climatology.

Usage

```
climatology_ens_forecast(obs_dt, by = setdiff(dimvars(obs_dt), "year"))
```

Arguments

obs_dt	Data table containing observations, must contain a column 'year'.
by	character vector containing the column names of the grouping variables, e.g. `c('month','lon','lat')`.

Value

Long data table with the typical ensemble-forecast looks, i.e. containing a column 'member'.

Examples

```
dt = climatology_ens_forecast(chirps_monthly)
```

climatology_threshold_exceedence

Get climatological prediction for exceedence probabilities.

Description

The climatological prediction for exceedence probabilities is the fraction of observed years where the observation exceeded the threshold. It's calculated from leave-one-year-out climatology.

Usage

```
climatology_threshold_exceedence(
  obs_dt,
  o = "prec",
  by = setdiff(dimvars(obs_dt), "year"),
  thresholds = c(200, 300, 350, 400)
)
```

Arguments

obs_dt	Data table containing observations.
o	column name of the observation. Mostly observed precipitation in mm.
by	By which columns should be grouped?
thresholds	vector of thresholds for which the exceedence probabilities should be derived.

Value

Data table with the climatological probabilities of exceedence for the provided thresholds.

Examples

```
dt = climatology_threshold_exceedence(chirps_monthly)
```

---

combine *Combine two data tables*

Description

Function for combining two data tables, e.g. with predictions and observations. This is a user-friendly wrapper for merge. It guesses the columns to merge by (the dimension variables contained in both data tables) and adds some warnings when merges are attempted that are likely not correctly specified by the user.

Usage

```
combine(dt1, dt2, ...)
```

Arguments

dt1	first data table
dt2	second data table
...	passed on to data.table::merge

Value

The merged data table

Examples

```
# merge ECMWF-forecasts and CHIRPS observations:
dt = ecmwf_monthly[month == 11]
setnames(dt,'prec','forecast') # forecasts and observations both have a column 'prec'
dt_new = combine(dt,chirps_monthly)
```

complete_regular_grid *Expand Regular Spatial Grid*

Description

First checks whether the spatial coordinates in a data table are part of a *regular grid*. If they are, the function returns the smallest *regular complete grid* including all coordinates. See `set_spatial_grid` for more information.

Usage

```
complete_regular_grid(dt)
```

Arguments

`dt`	A data table object containing the spatial grid with coordinates.

Value

A data table with the completed spatial grid. Has the grid-attribute.

Examples

```
dt = data.table(lon = c(1, 2, 3), lat = c(1, 2, 3))
completed_grid = complete_regular_grid(dt)
print(completed_grid)
```

convert_monthly_to_seasonal *Convert a data table from monthly to seasonal format*

Description

Converts monthly to seasonal data. The function default values are set for precipitation. In particular, default behavior is to *sum* values over all months contained in a season. If you want to average instead (for temperature, for example), you can change the aggregation function `FUN`.

Usage

```
convert_monthly_to_seasonal(
  dt,
  vars = NULL,
  by = NULL,
  FUN = sum,
  ...,
  seasons = c("MAM", "JJAS", "OND"),
  only_complete_seasons = TRUE
)
```

Arguments

dt	A data table containing the values for conversion.
vars	Character vector of names of the columns containing the values for conversion. Default is to try converting everything that is not contained in by and that is not recognized as tercile category (see tc_cols()) or tercile forecast ('below', 'normal', 'above').
by	Character vector of column names to group by. Separate values are derived for each unique combination of values in by. Defaults to all dimvars() that are not 'month', which is usually what you want.
FUN	function for aggregation.
...	arguments passed to FUN, for example na.rm.
seasons	Vector of character strings specifying seasons. See details. Defaults to c('MAM', 'JJAS', 'OND'), which are the seasons considered in the COFs for the Greater Horn of Africa.
only_complete_seasons	Logical. If TRUE, only values are kept for which we have data for all months. For example, no OND values would be derived if the data for December is missing.

Details

Note that it is impossible to derive seasonal tercile categories from monthly ones (and similar for seasonal tercile forecasts). For getting these, you should convert to seasonal *before* deriving the tercile categories or forecasts, e.g. by using add_tercile_cat() or tfc_from_efc().

Seasons are provided as sequences of capitalized initial characters of the months they contain, e.g. 'MAM' for March-April-May. They can have any length from 1 to 12 months and are allowed to overlap and wrap over the end of the year (for example, you can provide seasons = c('OND', 'NDJ') to derive values for October-December and November-January). If a season includes months from 2 years, it gets assigned the year of its starting month. For example, season = 'NDJ' and year = 2000 refers to values for the season November 2000 to January 2001.

Factor- or Character-valued columns cannot be aggregated to seasonal values. If vars contains columns that are factor- or character-valued, it checks whether they take a unique value for each grouping level provided in by. If yes, they are kept, else they are discarded. A typical case where this is useful is when your data table contains country names (see add_country()). The grouping levels usually include 'lon', 'lat', so there is only one country name per grouping level and the name is kept.

Examples

```
# returns empty data table, because the example data does not contain data for a full season:
dt = convert_monthly_to_seasonal(chirps_monthly)

# remove terc_cat first to avoid the warning,
# and set season to the months we actually have data for:
dt2 = convert_monthly_to_seasonal(copy(chirps_monthly)[,terc_cat := NULL], seasons = c('ND'))
print(dt2)

# season OND, get monthly averages rather than sums, and force the function to derive values
# even though we do not have October-data:
dt3 = convert_monthly_to_seasonal(chirps_monthly,
                                  seasons = c('OND'),
                                  FUN = mean,
                                  only_complete_seasons = FALSE)
print(dt3)
```

CPA *Coefficients of Predictive Ability*

Description

Function for calculating coefficients of predictive ability (CPAs) of ensemble mean forecasts stored in long data tables:#' Can also handle point forecasts. Warning: This metric always needs several years of data since the ranks on which it is based are calculated across multi-year samples.

Usage

```
CPA(
  dt,
  f,
  o = "obs",
  by = by_cols_ens_fc_score(dt),
  pool = "year",
  mem = "member",
  dim.check = TRUE
)
```

Arguments

dt	Data table containing the predictions.
f	column name of the prediction.
o	column name of the observations.
by	column names of grouping variables, all of which need to be columns in dt. A separate CPA is computed for each value of the grouping variables. Default is to group by all instances of month, season, lon, lat, system and lead_time that are columns in dt.

pool	column name(s) for the variable(s) along which is averaged. Needs to contain 'year' per warning above.
mem	Number of column containing the number of the ensemble member.
dim.check	Logical. If True, a simple test whether the dimensions match up is conducted: The data table should only have one row for each level of c(by,pool,mem)

Value

A data table with the scores

Examples

```
dt = data.table(fc = 1:4,obs = c(4,4,7,7),member = c(1,2,1,2),year = c(1999,1999,2000,2000))
CPA(dt,f = 'fc')
```

`create_diagram_by_level` *Auxiliary function to simplify grouping for diagrams*

Description

Only works for functions that return a single plot if by == NULL. This is not the case for some functions plotting results for all three categories, e.g. reliability diagrams or ROC curves.

Usage

```
create_diagram_by_level(FUN, by, dt, ...)
```

Arguments

FUN	The name of the function creating the diagram
by	Column names in dt to group by
dt	data table (cannot be part of ..., because a sub-data-table is passed to FUN)
...	arguments passed to FUN

CRPS *Continuous Ranked Probability Score*

Description

Taking CRPSs of ensemble forecasts stored in long data tables:

Usage

```
CRPS(
  dt,
  f,
  o = "obs",
  by = by_cols_ens_fc_score(),
  pool = "year",
  mem = "member",
  dim.check = TRUE,
  ens_size_correction = FALSE
)
```

Arguments

dt	Data table containing predictions and observations.
f	column name of the forecasts. May not be called 'f'
o	column name of the observations.
by	column names of grouping variables, all of which need to be columns in dt. Default is to group by all instances of month, season, lon, lat, system and lead_time that are columns in dt.
pool	column name(s) for the variable(s) over which is averaged. Typically just 'year'.
mem	Name of the column identifying the ensemble member.
dim.check	Logical. If True, a simple test whether the dimensions match up is conducted: The data table should only have one row for each level of c(by,pool,mem)
ens_size_correction	logical. If TRUE, the CRPS is corrected for sample size (see Ferro et al. 2008: 'On the effect of ensemble size on the discrete and continuous ranked probability scores'). This is slower, but you should do it if you compare ensembles of different size.

Value

A data table with the scores

Examples

```
dt = data.table(fc = 1:4,obs = c(4,4,7,7),member = c(1,2,1,2),year = c(1999,1999,2000,2000))
CRPS(dt,f = 'fc')
```

CRPSS *Continuous Ranked Probability Skill Score*

Description

Function for taking CRPS skill scores of ensemble forecasts stored in long data tables. The skill score needs a climatological forecast as reference. This is so far always based on the leave-one-year-out climatology.

Usage

```
CRPSS(dt, f, o = "obs", by = by_cols_ens_fc_score(), pool = c("year"), ...)
```

Arguments

dt	Data table containing predictions and observations.
f	column name of the prediction.
o	column name of the observations.
by	column names of grouping variables, all of which need to be columns in dt. A separate CRPS is computed for each value of the grouping variables. Default is to group by all instances of month, season, lon, lat, system and lead_time that are columns in dt.
pool	column name(s) for the variable(s) along which is averaged. Needs to contain 'year' since the reference climatology forecast is leave-one-year-out.
...	passed on to CRPS_ens_fc, in particular mem and dim.check

Value

A data table with the scores

Examples

```
dt = data.table(fc = 1:4,obs = c(4,4,7,7),member = c(1,2,1,2),year = c(1999,1999,2000,2000))
CRPSS(dt,f = 'fc')
```

`crps_aux` *Auxiliary function for calculating crps.*

Description

Mostly copy-paste from `scoringRules:::crps_edf`. Adjusted to the data table format, where the observation is a vector of the same length as the ensemble forecast, but is just repeated (which is why only `y[1]`) is used.

Usage

```
crps_aux(y, dat)
```

Arguments

`y`	vector of length m with m identical entries, the observation
`dat`	vector of length m containing the m ensemble forecasts

`crps_aux_esc` *Auxiliary function for calculating crps with ensemble size correction by Ferro et al. 2008.*

Description

Mostly copy-paste from `scoringRules::crps_edf`. Adjusted to the data table format, where the observation is a vector of the same length as the ensemble forecast, but is just repeated (which is why only `y[1]`) is used.

Usage

```
crps_aux_esc(y, dat)
```

Arguments

`y`	vector of length m with m identical entries, the observation
`dat`	vector of length m containing the m ensemble forecasts

`data_dir` *Auxiliary function to access and change the directory used to load and save data.*

Description

The package allows to download and organize CHIRPS data. This function specifies the directory where the data is stored. The first time this function is called, it asks the user to configure the directory.

Usage

```
data_dir(set_dir = FALSE)
```

Arguments

`set_dir`	logical. Set this to TRUE if you have to redefine your data directory.

Value

The current data directory as string.

Examples

```
if(interactive()){
data_dir()
}
```

`delete_redundant_files` *Auxiliary function cleaning out the directories, called at the end of the CHIRPS download.*

Description

Auxiliary function cleaning out the directories, called at the end of the CHIRPS download.

Usage

```
delete_redundant_files(dir)
```

Arguments

`dir`	the directory of the high dimensional CHIRPS data.

dimvars *Get dimension variables*

Description

The function returns all names currently considered dimension variables. Following the logic of netcdfs, data tables usually have columns specifying coordinates (or dimvars) and other columns containing data for these dimvars. Dimension variables can be spatial or temporal coordinates, or the lead time of a forecast or the member in an ensemble forecast, etc...

Usage

```
dimvars(dt = NULL)
```

Arguments

dt	Optional data table. If a data table is provided only the dimvars of the data table are returned.

Value

A vector of characters with the column names considered dimvars.

Examples

```
dimvars()
```

disc_score_dt *Generalized Discrimination score*

Description

Calculate the Generalized discrimination score from a data.table with data belonging to a single group (as defined by the by variable in the DISS function), for example a single location and month. Formula (5a) from Mason&2018 is used in the calculation. Mostly auxiliary function for the DISS function.

Usage

```
disc_score_dt(year, obs, pB, pN, pA)
```

Arguments

year	a vector of pool variables, typically year.
obs	a vector of observations the observation column, needs to contain -1 if it falls into the first category, 0 for the second and 1 for the third category.
pB	a vector of probabilities for the first category.
pN	a vector of probabilities for the second category.
pA	a vector of probabilities for the third category.

Value

A data table with the scores

Examples

```
disc_score_dt(year = 1999:2001,
              obs = c(-1,0,0),
              pB = c(0.5,0.3,0),
              pN = c(0.3,0.3,0.7),
              pA = c(0.2,0.4,0.3))
```

DISS *Generalized discrimination score*

Description

A generalisation of the ROC score for more than two categories. This score is not proper, but can be used to assess the discrimination of a tercile forecast.

Usage

```
DISS(
  dt,
  f = c("below", "normal", "above"),
  o = tc_cols(dt),
  by = by_cols_terc_fc_score_sp(),
  pool = "year",
  dim.check = TRUE
)
```

Arguments

dt	Data table containing the predictions.
f	column names of the prediction.

o	column name of the observations (either in obs_dt, or in dt if obs_dt = NULL). The observation column needs to contain -1 if it falls into the first category (corresponding to `fcs[1]`), 0 for the second and 1 for the third category.
by	column names of grouping variables, all of which need to be columns in dt. Default is to group by all instances of month, season, lon, lat, system and lead_time that are columns in dt.
pool	column name(s) for the variable(s) along which is averaged, typically just 'year'.
dim.check	Logical. If TRUE, the function tests whether the data table contains only one row per coordinate-level, as should be the case.

Value

A data table with the scores

Examples

```
dt = data.table(below = c(0.5,0.3,0),
                normal = c(0.3,0.3,0.7),
                above = c(0.2,0.4,0.3),
                tc_cat = c(-1,0,0),
                year = 1:3)
print(dt)
DISS(dt)
```

---

download_chirps_monthly

Download monthly CHIRPS-data

Description

Download CHIRPS monthly data for the GHA-region and save it as netcdfs. The data is downloaded either from the IRI data library or from ICPAC (depending on `version`), because these data library allows to subset before downloading, unlike the original source at UCSB. As of Feb 2022, the entire CHIRPS-monthly data for the GHA-region is roughly 800MB on disk. The original spatial resolution of CHIRPS is 0.05 degree lon/lat. However, for many applications a coarser resolution is perfectly fine. The function therefore offers the option to also create and save a coarser, upscaled version of the CHIRPS data that allows much faster data processing. Alternatively you can also ONLY save the upscaled version to save disk space (roughly 8MB on disk).

Usage

```
download_chirps_monthly(
  resolution = "both",
  update = TRUE,
  version = "UCSB",
  years = NULL,
  months = NULL,
```

```
  extent = GHA_extent(),
  timeout_limit = 300,
 upscale_grid = data.table(expand.grid(lon = seq(extent[1], extent[2], 0.5), lat =
   seq(extent[3], extent[4], 0.5)))
)
```

Arguments

resolution	Shall the data be upscaled? Takes one of three arguments: • **'both'** (the default) downloads and saves the data on full resolution and additionally derives an upscaled version. Both will be available later. • **'high'** downloads and saves on original resolution, but does not upscale. • **'low'** (for saving disk space) downloads the original resolution, upscales immediately and only saves the upscaled version.
update	Logical, if TRUE, previously created files are skipped.
version	Should be 'UCSB' (for University of California Santa Barbara, the original source of CHIRPS) or 'ICPAC' (for downloading the ICPAC version CHIRPS blended)
years, months	Which years and months do you want to load? NULL loads everything there is.
extent	vector of length four (xmin,xmax,ymin,ymax), restricting the spatial area.
timeout_limit	how many seconds (per file, i.e. per yearmonth) before the download is aborted?
upscale_grid	The coarse grid to which the data is upscaled (only used when resolution is either 'both' or 'high'). Only change this if you know what you are doing.

Value

Nothing.

Examples

```
if(interactive()){
download_chirps_monthly(years = 2020, months = 1)
}
```

---

download_chirps_monthly_high

Auxiliary function called by download_chirps_monthly

Description

Auxiliary function called by download_chirps_monthly

Usage

```
download_chirps_monthly_high(
  update,
  version,
  years,
  months,
  extent,
  timeout_limit,
  save_dir = file.path(chirps_dir(), version)
)
```

Arguments

update, version, years, months, extent, timeout_limit	see download_chirps_monthly.
save_dir	directory where the chirps data is stored.

`download_chirps_monthly_low` *Auxiliary function called by download_chirps_monthly*

Description

Auxiliary function called by download_chirps_monthly

Usage

```
download_chirps_monthly_low(
  update,
  version,
  years,
  months,
  extent,
  timeout_limit,
  upscale_grid,
  root_dir = file.path(chirps_dir(), version)
)
```

Arguments

update, version, years, months, extent, timeout_limit	see download_chirps_monthly.
upscale_grid	To which grid shall we upscale? Needs a data table with lon/lat columns
root_dir	directory where the high-dimensional chirps data would be stored. The upscaled data is then stored in root_dir/upscaled/.

download_chirps_prelim_aux *Auxiliary function for downloading the preliminary CHIRPS monthly data*

Description

This data becomes available earlier, but it has to be downloaded from UCSB. The function checks whether the non-preliminary version exists and only downloads otherwise. Annoyingly, the grid of UCBS and IRIDL are shifted against each other. Therefore this function also interpolates the UCSB data to the IRIDL grid, which makes it a bit slower. In particular, everything will crash if you have never downloaded a non-preliminary file and try to download a preliminary one.

Usage

```
download_chirps_prelim_aux(
  years,
  months,
  extent,
  timeout_limit = 300,
  nonprelim_dir = file.path(chirps_dir(), "monthly"),
  save_dir = file.path(nonprelim_dir, "prelim")
)
```

Arguments

years	years for which you want to download
months	months for which you want to download
extent	Spatial window for downloading
timeout_limit	How many seconds before download is aborted.
nonprelim_dir	Directory where the non-preliminary CHIRPS data is stored.
save_dir	Directory where the function stores the preliminary data.

Value

nothing

Examples

```
if(interactive()){
download_chirps_prelim_aux(years = 2023, months = 10)
}
```

dt_to_netcdf *Write a netcdf from a long data table*

Description

This function writes a netcdf from a long data table, the usual data format in SeaVal. If not specified, it guesses (based on column names) which columns contain dimension variables and which contain variables. The function currently does not support writing netcdfs with multiple variables that have different sets of dimension variables!

It allows to store character columns in netcdfs (essentially labelling them as integers and storing a legend). This legend is automatically interpreted when the netcdf is read with `netcdf_to_dt()`, but is also humanly readable.

Usage

```
dt_to_netcdf(
  dt,
  nc_out,
  vars = NULL,
  units = NULL,
  dim_vars = dimvars(dt),
  dim_var_units = NULL,
  check = interactive(),
  description = NULL
)
```

Arguments

dt	a data.table
nc_out	File name (including path) of the netcdf to write.
vars	names of columns in dt containing variables. If this is NULL, the function guesses and asks for confirmation.
units	character vector containing the units for vars (in the same order). If this is NULL (default), the user is prompted for input.
dim_vars	names of columns in dt containing dimension variables. If this is NULL, the function guesses and asks for confirmation.
dim_var_units	character vector containing the units for dim_vars (in the same order). If this is NULL (default), the user is prompted for input (except for lon/lat).
check	If check is TRUE, the function asks the user whether an existing file should be overwritten, and whether the correct dimvars have been guessed.
description	For adding a global attribute 'Description' as a string.

Value

none.

Examples

```
example_dt = data.table(lon = 1:3, month = 4:6, prec = 7:9)
file_name = tempfile()
dt_to_netcdf(dt = example_dt, nc_out = file_name,
             vars = "prec", units = "mm",
             dim_vars = c("lon","month"), dim_var_units = c("degree longitude","month"))
```

EA_country_names *Get names of countries in east Africa*

Description

This is an auxiliary function used in add_country_names, so only these names are recognized by default.

Usage

```
EA_country_names()
```

Value

A character-vector of country names.

Examples

```
EA_country_names()
```

---

ecmwf_monthly *Monthly mean precipitation forecast example dataset*

Description

This is a small example dataset containing hindcasts of monthly mean precipitation for illustration purposes. The forecasts are contained for the entire GHA-region, for November and December 2018-2020. The forecasts are issued by the ECMWF SEAS 5 model and initialized in August. The unit of precipitation is mm/day. Only the first 3 ensemble members are provided. The dataset also contains tercile probability forecasts, which are derived from the full 51 member ensemble. The probability for a tercile for a given year, month and location is always computed as the fraction of ensemble members falling into that tercile, computed from all ensemble predictions for the month and location under consideration. This dataset was generated using Copernicus Climate Change Service information (2020).

Usage

```
data(ecmwf_monthly)
```

Format

An object of class `data.table` (inherits from `data.frame`) with 37224 rows and 9 columns.

Source

https://cds.climate.copernicus.eu

EIR *Effective Interest Rate*

Description

This score is suitable for tercile category forecasts. Using log2 for now (?). According to Mason, the averaging here should be over many years at a single locations and for discrete time-periods (so Mason prefers to take the average after averaging over different locations, but I keep it like this for now).

Usage

```
EIR(
  dt,
  f = c("below", "normal", "above"),
  o = tc_cols(dt),
  by = by_cols_terc_fc_score(),
  pool = "year",
  dim.check = TRUE
)
```

Arguments

`dt`	Data table containing the predictions.
`f`	column names of the prediction.
`o`	column name of the observations (either in `obs_dt`, or in dt if `obs_dt = NULL`). The observation column needs to contain -1 if it falls into the first category (corresponding to `fcs[1]`), 0 for the second and 1 for the third category.
`by`	column names of grouping variables, all of which need to be columns in dt. Default is to group by all instances of month, season, lon, lat, system and lead_time that are columns in dt.
`pool`	column name(s) for the variable(s) along which is averaged, typically just 'year'.
`dim.check`	Logical. If TRUE, the function tests whether the data table contains only one row per coordinate-level, as should be the case.

Value

A data table with the scores

Examples

```
dt = data.table(below = c(0.5,0.3,0),
                normal = c(0.3,0.3,0.7),
                above = c(0.2,0.4,0.3),
                tc_cat = c(-1,0,0),
                lon = 1:3)
print(dt)
EIR(dt)
```

---

fc_cols *Forecast column names*

Description

returns the columns names that are recognized as (ensemble-) forecast values

Usage

```
fc_cols(dt = NULL)
```

Arguments

dt	optional data table. If provided, the function guesses which column contains the forecast values. Else it returns all recognized forecast column names.

Value

Character vector with column names.

Examples

```
fc_cols()
```

get_mask *Function to create a mask of dry regions from CHIRPS*

Description

A gridpoint is masked for a given season (either 'MAM', 'JJAS' or 'OND'), if, on average, less than 10% of the annual total of rainfall occur during the season. This function loads CHIRPS data, and derives this mask as a data table of lon, lat coordinates, only containing the coordinates that shouldn't be masked. You can apply the mask to an existing data table using dt = combine(dt,mask).

Usage

```
get_mask(
  season,
  clim_years = 1990:2020,
  version = "UCSB",
  resolution = "low",
  us = (resolution == "low")
)
```

Arguments

season	For which season do you want to calculate the mask? Needs to be either 'MAM', 'JJAS' or 'OND'.
clim_years	Numeric vector of years. Which years should be used to establish the mask?
version, resolution, us	Passed to load_chirps. Which CHIRPS version do you want to use and on what resolution?

Examples

```
if(interactive()) get_mask('MAM')
```

get_quantiles *Calculate quantiles from a data table*

Description

The quantiles are saved in/returned as a list with the following elements:

- dt - A data table with quantiles for each level of by (not the same as the input-dt).
- quantiles - the vector of quantiles that were used.
- group - a data table containing the levels the quantiles are grouped over, e.g. all years the quantiles are calculated over.

- data_col_name - the name of data_col, see below, so that you know what the quantiles actually were computed from.
- description - the description string, if provided.

Usage

```
get_quantiles(
  dt,
  data_col = setdiff(names(dt), dimvars(dt))[1],
  qqs = c(10, 20, 33, 67, 80, 90),
  by = setdiff(dimvars(dt), c("year", "member")),
  description = NULL,
  save_file = NULL
)
```

Arguments

dt	Data table containing the data.
data_col	The name of the column in dt containing the data for which the quantiles are derived. By default the first column that is not a dimension variable is selected.
qqs	Vector of quantiles. If one of them is larger 1 they are interpreted as percent. Default is the quantiles used in the verification maps.
by	Column names in dt. Levels by which the quantiles are calculated
description	Optional description string.
save_file	Optional name of save file.

Value

Nothing if save_file is provided. Otherwise the list described above

Examples

```
get_quantiles(chirps_monthly)
```

---

get_terciles *get terciles from a data table*

Description

This function wraps `get_quantiles` with the fixed quantiles 0.33 and 0.67.

Usage

```
get_terciles(...)
```

Arguments

...	passed on to get_quantiles.

Value

See get_quantiles.

Examples

```
# takes a few seconds:
get_terciles(chirps_monthly)
```

---

ggplot_dt *plotting function for spatial data*

Description

Plots spatial data from a data.table. The data table needs to contain columns named 'lon' and 'lat'. The grid needs to be regular. If spatial data is contained for several levels (e.g. mutliple times or multiple ensemble members), only the data for the first level is plotted. By default, the first column that is not recognized as a dimension variable is plotted, see data_col. For the most common data-columns, reasonable color scales are selected automatically.

Usage

```
ggplot_dt(
  dt,
  data_col = NULL,
  mn = NULL,
  discrete_cs = FALSE,
  rr = NULL,
  low = NULL,
  mid = NULL,
  high = NULL,
  name = data_col,
  midpoint = NULL,
  breaks = NULL,
  na.value = "gray75",
  oob = NULL,
  guide = guide_colorbar(barwidth = 0.5, barheight = 10),
  ...,
  binwidth = NULL,
  bin_midpoint = midpoint,
  add_map = TRUE,
  extent = NULL,
```

```
  expand.x = c(0, 0),
  expand.y = c(0, 0),
  dimension_check = TRUE
)
```

Arguments

dt	Data table containing the data for plotting.
data_col	The name of the column in dt containing the data for plotting. If nothing is provided (the default), the first column that is not a dimension variable or 'member' is selected.
mn	optional plot title
discrete_cs	Logical. Should the color scale be discretized?
rr, low, mid, high, name, breaks, na.value, oob, guide, ...	Arguments for the color scale, passed to scale_fill_gradient2 or scale_fill_steps2 (depending on whether discrete_cs == TRUE). rr replaces limits (specifying the range of the color scale) for consistency with the older plotting functions from the PostProcessing package. na.value specifies the color of missing values. oob specifies the treatment of out-of-boundary values, i.e. values beyond the limits. The ... argument can in particular be used to customize the legend/colorbar using the function guide_colorbar, see https://ggplot2.tidyverse.org/reference/guide_colou Moreover, when discrete_cs == TRUE you can pass the arguments n.breaks,breaks to customize the scale. If you use n.breaks you might also want to set nice.breaks = FALSE, see ?scale_fill_steps2.
midpoint	midpoint of the color scale, passed to scale_fill_gradient2 or scale_fill_steps2 (depending on whether discrete_cs == TRUE). If you set it to NULL (the default), the midpoint is set to the center of the data range (or the center of rr, if provided), such that the entire color scale is used. Note that this default differs from the default behavior of scale_fill_gradient2 or scale_fill_steps2. Specifying the midpoint can often be a convenient way to force a color scale with only two colors (for example, by setting it to the minimum or maximum of your data).
binwidth, bin_midpoint	only used when discrete_cs == TRUE. Normally, the breaks for the discrete colorscale are specified by n.breaks (which is not reliable, since they're adjusted to be 'nice'), or by specifying the breaks explicitly (which is often tedious). This gives you a third option, namely specifying how far the breaks should be apart, and specifying the centerpoint for one of the bins (default is midpoint, or the center of rr if midpoint is not provided). For example, if your color scale shows percentages and you'd like 4 categories, ranging from white to red, this is easiest achieved by binwidth = 25, midpoint = 12.5.
add_map	logical, defaults to TRUE, mostly for internal use. Set to FALSE to remove borders (e.g. if you want to add them yourself from a shapefile).
extent	An optional four-element vector in the order xmin,xmax,ymin,ymax for specifying the spatial extent of the plot. Default is to fit the extent to the data.

`expand.x`, `expand.y`
: vectors with two entries to be added to xlims/ylims of the plot. E.g. expand.x = c(-0.5,0.5) expands the plot by half a longitude both on the right and left hand side

`dimension_check`
: Logical. By default the function checks that there are not multiple values per coordinate (and subsets to the first level if there are several, e.g. to the first year and month (by appearance in `dt`) if `dt` contains data for several years and months).

Value

a ggplot object.

Author(s)

Claudio Heinrich

Examples

```
ex_dt = chirps_monthly[lat <0 & month == 12 & year == 2020]
pp = ggplot_dt(ex_dt)
if(interactive()) plot(pp)
```

GHA_extent *GHA-bounding-box*

Description

Returns a lon/lat bounding box for the greater horn of Africa region. Format is c(xmin,xmax,ymin,ymax), as for raster::extent

Usage

```
GHA_extent()
```

Value

A numeric vectorof length 4.

Examples

```
GHA_extent()
```

gha_plot *Plotting function with different map for Greater Horn of Africa*

Description

This essentially wraps ggplot_dt, but uses a different map for borders. The map is part of the package and is the one currently used during GHACOFs at ICPAC.

Usage

```
gha_plot(..., expand.x = c(-0.5, 0.5), expand.y = c(-0.5, 2))

ggplot_dt_shf(...)

ggplot_dt_gha_map(...)
```

Arguments

..., expand.x, expand.y	passed to ggplot_dt

Examples

```
dt = chirps_monthly[lon %between% c(30,40) & lat < 0 & month == 11 & year == 2020]
pp = gha_plot(dt)
if(interactive()) plot(pp)
```

grid_info *Retrieve spatial grid information from a data table*

Description

This function prints out spatial grid information from a data table. If the grid-attribute does not exist set_spatial_grid is called first.

Usage

```
grid_info(dt)
```

Arguments

dt	A data table

Value

This function does not return a value; instead, it prints a message to the console with the grid information.

Examples

```
dt = data.table(lon = runif(10), lat = runif(10))
grid_info(dt)
```

HS *Hit score*

Description

This score is suitable for tercile category forecasts. This score is the frequency at which the highest probability category actually happens. The function also provides the frequency at which the second-highest probability category, and lowest probability category, actually happens.

Usage

```
HS(
  dt,
  f = c("below", "normal", "above"),
  o = tc_cols(dt),
  by = by_cols_terc_fc_score(),
  pool = "year",
  dim.check = TRUE
)
```

Arguments

dt	Data table containing the predictions.
f	column names of the prediction.
o	column name of the observations (either in obs_dt, or in dt if obs_dt = NULL). The observation column needs to contain -1 if it falls into the first category (corresponding to fcs[1]), 0 for the second and 1 for the third category.
by	column names of grouping variables, all of which need to be columns in dt. Default is to group by all instances of month, season, lon, lat, system and lead_time that are columns in dt.
pool	column name(s) for the variable(s) along which is averaged, typically just 'year'.
dim.check	Logical. If TRUE, the function tests whether the data table contains only one row per coordinate-level, as should be the case.

Value

A data table with the scores

Examples

```
dt = data.table(below = c(0.5,0.3,0),
                normal = c(0.3,0.3,0.7),
                above = c(0.2,0.4,0.3),
                tc_cat = c(-1,0,0),
                lon = 1:3)
print(dt)
HS(dt)
```

HSS *Hit Skill Score*

Description

This score is suitable for tercile category forecasts. The skill score is the difference between the hit scores for the categories with the highest and lowest probabilities.

Usage

```
HSS(
  dt,
  f = c("below", "normal", "above"),
  o = tc_cols(dt),
  by = by_cols_terc_fc_score(),
  pool = "year",
  dim.check = TRUE
)
```

Arguments

dt	Data table containing the predictions.
f	column names of the prediction.
o	column name of the observations (either in obs_dt, or in dt if obs_dt = NULL). The observation column needs to contain -1 if it falls into the first category (corresponding to fcs[1]), 0 for the second and 1 for the third category.
by	column names of grouping variables, all of which need to be columns in dt. Default is to group by all instances of month, season, lon, lat, system and lead_time that are columns in dt.
pool	column name(s) for the variable(s) along which is averaged, typically just 'year'.
dim.check	Logical. If TRUE, the function tests whether the data table contains only one row per coordinate-level, as should be the case.

Value

A data table with the scores

Examples

```
dt = data.table(below = c(0.5,0.3,0),
                normal = c(0.3,0.3,0.7),
                above = c(0.2,0.4,0.3),
                tc_cat = c(-1,0,0),
                year = 1999:2001)
print(dt)
HSS(dt)
```

IGS *Ignorance Score*

Description

This score is suitable for tercile category forecasts. Using log2 for now (?).

Usage

```
IGS(
  dt,
  f = c("below", "normal", "above"),
  o = tc_cols(dt),
  by = by_cols_terc_fc_score(),
  pool = "year",
  dim.check = TRUE
)
```

Arguments

dt	Data table containing the predictions.
f	column names of the prediction.
o	column name of the observations (either in obs_dt, or in dt if obs_dt = NULL). The observation column needs to contain -1 if it falls into the first category (corresponding to fcs[1]), 0 for the second and 1 for the third category.
by	column names of grouping variables, all of which need to be columns in dt. Default is to group by all instances of month, season, lon, lat, system and lead_time that are columns in dt.
pool	column name(s) for the variable(s) along which is averaged, typically just 'year'.
dim.check	Logical. If TRUE, the function tests whether the data table contains only one row per coordinate-level, as should be the case.

Value

A data table with the scores

Examples

```
dt = data.table(below = c(0.5,0.3,0),
                normal = c(0.3,0.3,0.7),
                above = c(0.2,0.4,0.3),
                tc_cat = c(-1,0,0),
                lon = 1:3)
print(dt)
IGS(dt)
```

IGSS *Ignorance Skill score*

Description

This score is suitable for tercile category forecasts. Using log2 for now (?). This is the "usual" skill score (not the effective interest rate).

Usage

```
IGSS(
  dt,
  f = c("below", "normal", "above"),
  o = tc_cols(dt),
  by = by_cols_terc_fc_score(),
  pool = "year",
  dim.check = TRUE
)
```

Arguments

dt	Data table containing the predictions.
f	column names of the prediction.
o	column name of the observations (either in obs_dt, or in dt if obs_dt = NULL). The observation column needs to contain -1 if it falls into the first category (corresponding to fcs[1]), 0 for the second and 1 for the third category.
by	column names of grouping variables, all of which need to be columns in dt. Default is to group by all instances of month, season, lon, lat, system and lead_time that are columns in dt.
pool	column name(s) for the variable(s) along which is averaged, typically just 'year'.
dim.check	Logical. If TRUE, the function tests whether the data table contains only one row per coordinate-level, as should be the case.

Value

A data table with the scores

Examples

```
dt = data.table(below = c(0.5,0.3,0),
                normal = c(0.3,0.3,0.7),
                above = c(0.2,0.4,0.3),
                tc_cat = c(-1,0,0),
                lon = 1:3)
print(dt)
IGSS(dt)
```

`indicator_times_value_aux` *Auxiliary function for multiplying two numbers such that 0 x infty is 0. Needed for the ignorance score: 0log(0) should be 0.*

Description

Auxiliary function for multiplying two numbers such that 0 x infty is 0. Needed for the ignorance score: 0log(0) should be 0.

Usage

```
indicator_times_value_aux(indicator, value)
```

Arguments

`indicator`	logical input vector
`value`	numeric input vector

Value

indicator x value with 0*infty = 0

`load_chirps` *Function for loading CHIRPS (monthly) data.*

Description

The data has to be previously downloaded, see `download_chirps_monthly`. The resulting data table contains precip in unit mm/day.

Usage

```
load_chirps(
  years = NULL,
  months = NULL,
  version = "UCSB",
  resolution = "low",
  us = (resolution == "low"),
  load_prelim = TRUE
)
```

Arguments

years, months	Optional subset of years and months you want to load. The default is to load everything that has been downloaded locally. You can update your local CHIRPS download by calling download_chirps_monthly
version	Either 'UCSB' to load the original version from UCSB or 'ICPAC' to load CHIRPS blended (both need to be downloaded first).
resolution	Either 'low' for loading the coarser upscaled version, or 'high' for loading the data on full resolution
us	logical. If TRUE, the upscaled version is loaded. Takes precedence over resolution.
load_prelim	logical. Should preliminary data be loaded? Note that the preliminary data is always from UCSB, not from ICPAC.

Value

the derived data table

Examples

```
if(interactive()){
load_chirps()
}
```

---

lt_cols *Data table column names that are recognized as leadtime*

Description

Data table column names that are recognized as leadtime

Usage

```
lt_cols()
```

Examples

```
lt_cols()
```

MB *Multicategory Brier score*

Description

This score is suitable for tercile category forecasts.

Usage

```
MB(
  dt,
  f = c("below", "normal", "above"),
  o = tc_cols(dt),
  by = by_cols_terc_fc_score(),
  pool = "year",
  dim.check = TRUE
)
```

Arguments

dt	Data table containing the predictions.
f	column names of the prediction.
o	column name of the observations (either in obs_dt, or in dt if obs_dt = NULL). The observation column needs to contain -1 if it falls into the first category (corresponding to fcs[1]), 0 for the second and 1 for the third category.
by	column names of grouping variables, all of which need to be columns in dt. Default is to group by all instances of month, season, lon, lat, system and lead_time that are columns in dt.
pool	column name(s) for the variable(s) along which is averaged, typically just 'year'.
dim.check	Logical. If TRUE, the function tests whether the data table contains only one row per coordinate-level, as should be the case.

Value

A data table with the scores

Examples

```
dt = data.table(below = c(0.5,0.3,0),
                normal = c(0.3,0.3,0.7),
                above = c(0.2,0.4,0.3),
                tc_cat = c(-1,0,0),
                lon = 1:3)
print(dt)
MB(dt)
```

MBS *Multicategory Brier Skill score*

Description

This score is suitable for tercile category forecasts.

Usage

```
MBS(
  dt,
  f = c("below", "normal", "above"),
  o = tc_cols(dt),
  by = by_cols_terc_fc_score(),
  pool = "year",
  dim.check = TRUE
)
```

Arguments

dt	Data table containing the predictions.
f	column names of the prediction.
o	column name of the observations (either in obs_dt, or in dt if obs_dt = NULL). The observation column needs to contain -1 if it falls into the first category (corresponding to fcs[1]), 0 for the second and 1 for the third category.
by	column names of grouping variables, all of which need to be columns in dt. Default is to group by all instances of month, season, lon, lat, system and lead_time that are columns in dt.
pool	column name(s) for the variable(s) along which is averaged, typically just 'year'.
dim.check	Logical. If TRUE, the function tests whether the data table contains only one row per coordinate-level, as should be the case.

Value

A data table with the scores

Examples

```
dt = data.table(below = c(0.5,0.3,0),
                normal = c(0.3,0.3,0.7),
                above = c(0.2,0.4,0.3),
                tc_cat = c(-1,0,0),
                lon = 1:3)
print(dt)
MBS(dt)
```

`modify_dt_map_plotting` *Auxiliary function for checking dimensions for map-plotting*

Description

Auxiliary function for checking dimensions for map-plotting

Usage

```
modify_dt_map_plotting(dt, data_col)
```

Arguments

`dt`	Data table containing the data for plotting
`data_col`	Name of column containing the data for plotting

`MSD_to_YM` *Converts time given as 'months since date' (MSD) into years and months (YM)*

Description

Converts time given as 'months since date' (MSD) into years and months (YM)

Usage

```
MSD_to_YM(dt, timecol = "time", origin = "1981-01-01")
```

Arguments

`dt`	a data table.
`timecol`	name of the column containing the time.
`origin`	The time column contains time in the format month since which date?

Value

data table with two new columns 'month' and 'year', the timecol is deleted.

Examples

```
dt = MSD_to_YM(data.table(time = 0:12))
```

MSE *Mean Square Error of ensemble forecasts.*

Description

Derives the MSE of ensemble forecasts stored in long data tables. Can also handle point forecast.

Usage

```
MSE(
  dt,
  f,
  o = "obs",
  by = by_cols_ens_fc_score(),
  pool = "year",
  mem = "member",
  dim.check = TRUE
)
```

Arguments

dt	Data table containing the predictions.
f	column name of the prediction.
o	column name of the observations.
by	column names of grouping variables, all of which need to be columns in dt. Default is to group by all instances of month, season, lon, lat, system and lead_time that are columns in dt.
pool	column name(s) for the variable(s) along which is averaged, typically just 'year'.
mem	Name of the column identifying the ensemble member. Only used if check_dimension is TRUE. Is NULL for a point forecast.
dim.check	Logical. If True, a simple test whether the dimensions match up is conducted: The data table should only have one row for each level of c(by,pool,mem)

Value

A data table with the scores

Examples

```
dt = data.table(fc = 1:4,obs = c(4,4,7,7),member = c(1,2,1,2),year = c(1999,1999,2000,2000))
MSE(dt,f = 'fc')
```

MSES *Mean Square Error Skill score*

Description

Function for taking MSE skill scores of ensemble forecasts stored in long data tables. Can also handle point forecasts. The skill score needs a climatological forecast as reference. This is so far always based on the leave-one-year-out climatology.

Usage

```
MSES(dt, f, o = "obs", by = by_cols_ens_fc_score(), pool = c("year"), ...)
```

Arguments

dt	Data table containing the predictions.
f	column name of the prediction.
o	column name of the observations.
by	column names of grouping variables, all of which need to be columns in dt. A separate MSE is computed for each value of the grouping variables. Default is to group by all instances of month, season, lon, lat, system and lead_time that are columns in dt.
pool	column name(s) for the variable(s) along which is averaged. Needs to contain 'year' since the reference climatology forecast is leave-one-year-out.
...	passed on to MSE

Value

A data table with the scores

Examples

```
dt = data.table(fc = 1:4,obs = c(4,4,7,7),member = c(1,2,1,2),year = c(1999,1999,2000,2000))
MSES(dt,f = 'fc')
```

netcdf_to_dt *function for converting netcdfs to long data tables.*

Description

The function converts netcdfs into long data.tables. Be aware that the data table can be much larger in memory, especially if you have many dimension variables.

Usage

```
netcdf_to_dt(
  nc,
  vars = NULL,
  verbose = 2,
  trymerge = TRUE,
  subset_list = NULL,
  keep_nas = FALSE
)
```

Arguments

nc	Either a character string with the name of the .nc file (including path), or an object of type ncdf4.
vars	Which variables should be read from the netcdf? Either a character vector of variable names, or an integer vector of variable indices. If this is NULL, all variables are read.
verbose	Either 0, 1 or 2. How much information should be printed? The default (2) is to print the entire netcdf information (as output by `ncdf4::nc_open`), 1 just prints the units for all variables, 0 (or any other input) prints nothing.
trymerge	logical. If TRUE, a single data table containing all variables is returned, else a list of data tables, one for each variable. The latter is much more memory efficient if you have multiple variables depending on different dimensions.
subset_list	A named list for reading only subsets of the data. Currently only 'rectangle subsetting' is provided, i.e. you can provide two limit values for each dimension and everything between will be read. The names of the pages of subset_list must correspond to the names of dimension variables in the netcdf, and each page should contain a (two-element-)range vector. For example, subsetting a global dataset to just East Africa could look like this: subset_list = list(latitude = c(-15,25),longitude = c(20,55)). Non-rectangular subsetting during reading a netcdf seems to be difficult, see ncvar_get. Every dimension variable not named in subset_list is read entirely.
keep_nas	Should missing values be kept? If FALSE (the default), missing values are not included in the returned data table. If this is set to TRUE, the data table is constructed from the full data-cube (meaning its number of rows is the product of the length of the dimension variables, even if many coordinates have missing

data). This makes the returned data table potentially much larger and is almost never an advantage. It is only allowed, because it can make complex bookkeeping tasks easier (specifically upscaling many CHIRPS-netcdfs with the same coordinates while saving the upscaling weights in a matrix).

Value

A data table if `trymerge == TRUE` or else a list of data tables.

Examples

```
# filename of example-netcdf file:
fn = system.file("extdata", "example.nc", package="SeaVal")

dt = netcdf_to_dt(fn)
print(dt)
```

---

obs_cols *Observation column names*

Description

Note that this function guesses column names for observed precip, not observed tercile category.

Usage

```
obs_cols(dt = NULL)
```

Arguments

`dt`	optional data table. If provided, the function guesses which column contains the observations. Else it returns all recognized observation column names.

Value

Character vector with column names.

Examples

```
obs_cols()
```

obs_dimvars *Auxiliary function returning observation dimvars.*

Description

Observation dimvars are column names in a data table that resemble coordinates for which only one observation may exist.

Usage

```
obs_dimvars(dt = NULL)
```

Arguments

dt	optional. You can provide a data table, then the function returns the names of coordinate columns in this data table.

Value

Character vector with column names.

Examples

```
obs_dimvars
```

PCC *Pearson Correlation Coefficient*

Description

Function for calculating Pearson correlation coefficients (PCCs) of ensemble mean forecasts stored in long data tables. Can also handle point forecasts. This metric always needs several years of data since the means and standard deviations are calculated across time.

Usage

```
PCC(
  dt,
  f,
  o = "obs",
  by = by_cols_ens_fc_score(dt),
  pool = "year",
  mem = "member",
  dim.check = TRUE
)
```

Arguments

dt	Data table containing the predictions.
f	column name of the prediction.
o	column name of the observations.
by	column names of grouping variables, all of which need to be columns in dt. A separate PCC is computed for each value of the grouping variables. Default is to group by all instances of month, season, lon, lat, system and lead_time that are columns in dt.
pool	column name(s) for the variable(s) along which is averaged. Needs to contain 'year' per warning above.
mem	Name of the column identifying the ensemble member. Only used if check_dimension is TRUE. Is NULL for a point forecast.
dim.check	Logical. If True, a simple test whether the dimensions match up is conducted: The data table should only have one row for each level of c(by,pool,mem)

Value

A data table with the scores

Examples

```
dt = data.table(fc = 1:4,obs = c(4,4,7,7),member = c(1,2,1,2),year = c(1999,1999,2000,2000))
PCC(dt,f = 'fc')
```

---

profit_graph *(Accumulative) profit graphs*

Description

These graphs really only make sense if you have 50 or less observations. Typical application would be when you compare seasonal mean forecasts to station data for a single location.

Usage

```
profit_graph(
  dt,
  accumulative = TRUE,
  f = c("below", "normal", "above"),
  o = tc_cols(dt),
  by = NULL,
  pool = setdiff(dimvars(dt), by),
  dim.check = TRUE
)
```

Arguments

dt	Data table containing tercile forecasts
accumulative	Logic. Should the accumulative profit be plotted or the profit per forecast?
f	column names of the prediction columns
o	column name of the observation column
by	column names of grouping variables. Default is NULL.
pool	column names of pooling variables (used for the dimension check). Default is all dimvars.
dim.check	Logical. If TRUE, the function checks whether the columns in by and pool span the entire data table.

Value

A list of gg objects which can be plotted by ggpubr::ggarrange (for example)

Examples

```
dt = data.table(below = c(0.5,0.3,0),
                normal = c(0.3,0.3,0.7),
                above = c(0.2,0.4,0.3),
                tc_cat = c(-1,0,0),
                lon = 1:3)
print(dt)
p1 = profit_graph(dt)
p2 = profit_graph(dt,accumulative = FALSE)

if(interactive()){
plot(p1)
plot(p2)
}
```

REL *Reliability score*

Description

Computes both the reliability component of the Brier score or reliability component of the Ignorance score. Mason claims to prefer the ignorance score version, but this has a very high chance of being NA. Mason writes that the scores are unstable for single locations and that one should pool over many locations. Requires the specification of probability bins. One score for each category (below, normal, above) and also the sum of the scores.

Values close to 0 indicate reliable forecasts. Higher values mean less reliable forecasts.

Usage

```
REL(
  dt,
  bins = c(0.3, 0.35001),
  f = c("below", "normal", "above"),
  o = tc_cols(dt),
  by = by_cols_terc_fc_score(),
  pool = "year",
  dim.check = TRUE
)
```

Arguments

dt	Data table containing the predictions.
bins	probability bins, defaults to ("<30", "30-35",">35") which is given as c(0.30, 0.35001).
f	column names of the prediction.
o	column name of the observations (either in obs_dt, or in dt if obs_dt = NULL). The observation column needs to contain -1 if it falls into the first category (corresponding to `fcs[1]`), 0 for the second and 1 for the third category.
by	column names of grouping variables, all of which need to be columns in dt. Default is to group by all instances of month, season, lon, lat, system and lead_time that are columns in dt.
pool	column name(s) for the variable(s) along which is averaged, typically just 'year'.
dim.check	Logical. If TRUE, the function tests whether the data table contains only one row per coordinate-level, as should be the case.

Value

A data table with the scores

Examples

```
dt = data.table(below = c(0.5,0.3,0),
                normal = c(0.3,0.3,0.7),
                above = c(0.2,0.4,0.3),
                tc_cat = c(-1,0,0),
                year = 1:3)
print(dt)
REL(dt)
```

`rel_diag` *Reliability Diagrams for tercile forecasts*

Description

Creates reliability diagrams from a data table containing tercile forecasts It wraps `rel_diag_vec`, see `?rel_diag_vec` for more details. about the output diagrams. The output format is very much inspired by Figure 5 of Mason&2018. By default, 4 diagrams are drawn, one for each the prediction of above-, normal- and below-values, plus one for all forecasts together. You can provide a 'by' argument to obtain separate reliability diagrams for different values of the by-columns. E.g., when you data table contains a column named 'season', you can set by = 'season'. Then, the function will output a list of 16 diagrams, 4 for each season.

Usage

```
rel_diag(
  dt,
  f = c("below", "normal", "above"),
  o = tc_cols(dt),
  by = NULL,
  pool = setdiff(dimvars(dt), by),
  binwidth = 0.05,
  dim.check = TRUE
)
```

Arguments

`dt`	Data table containing tercile forecasts
`f`	column names of the prediction columns
`o`	column name of the observation column
`by`	column names of grouping variables. Default is to not group.
`pool`	column names of pooling variables (used for the dimension check). Default is all dimvars.
`binwidth`	bin width for discretizing probabilities.
`dim.check`	Logical. If TRUE, the function checks whether the columns in by and pool span the entire data table.

Value

A list of gg objects which can be plotted by ggpubr::ggarrange (for example)

Examples

```
dt = data.table(below = c(0.5,0.3,0),
                normal = c(0.3,0.3,0.7),
                above = c(0.2,0.4,0.3),
                tc_cat = c(-1,0,0),
                lon = 1:3)
print(dt)
pp = rel_diag(dt)
if(interactive()) plot(pp)
```

`rel_diag_vec` *Reliability diagram from vectors of probabilities and observations*

Description

The probabilities have to be rounded beforehand (see `round_probs`), because the diagram draws a point for each level of the probabilities. The diagram includes a histogram indicating the forecast relative frequency for each probability bin. The diagram shows the reliability curve and the diagonal for reference. Moreover, it shows a regression line fitted by weighted linear regression where the forecast relative frequencies are used as weights. A horizontal and vertical line indicate the frequency of observation = TRUE over the entire dataset.

Usage

```
rel_diag_vec(discrete_probs, obs, slope_only = FALSE)
```

Arguments

`discrete_probs`	Vector of (rounded) probabilites.
`obs`	Vector of logical observations.
`slope_only`	logical. If set to TRUE, only the slope of the reliability curve is returned

Value

A gg object.

Examples

```
discrete_probs = seq(0,1,length.out = 5)
obs = c(FALSE,FALSE,TRUE,TRUE,TRUE)
pp = rel_diag_vec(discrete_probs,obs)
if(interactive()) plot(pp)
```

RES *Resolution score*

Description

Computes both the resolution component of the Brier score or resolution component of the Ignorance score. Mason claims to prefer the ignorance score version, but this has a very high chance of being NA (much higher than for the full ignorance score itself, I think we should drop it for that reason). Mason writes that the scores are unstable for single locations and that one should pool over many locations. Requires the specification of probability bins. One score for each category (below, normal, above) and also the sum of the scores. Values close to 0 means low resolution. Higher values mean higher resolution.

Usage

```
RES(
  dt,
  bins = c(0.3, 0.35001),
  f = c("below", "normal", "above"),
  o = tc_cols(dt),
  by = by_cols_terc_fc_score(),
  pool = "year",
  dim.check = TRUE
)
```

Arguments

dt	Data table containing the predictions.
bins	probability bins, defaults to c("<30", "30-35",">35")
f	column names of the prediction.
o	column name of the observations (either in obs_dt, or in dt if obs_dt = NULL). The observation column needs to contain -1 if it falls into the first category (corresponding to `fcs[1]`), 0 for the second and 1 for the third category.
by	column names of grouping variables, all of which need to be columns in dt. Default is to group by all instances of month, season, lon, lat, system and lead_time that are columns in dt.
pool	column name(s) for the variable(s) along which is averaged, typically just 'year'.
dim.check	Logical. If TRUE, the function tests whether the data table contains only one row per coordinate-level, as should be the case.

Value

A data table with the scores

Examples

```
dt = data.table(below = c(0.5,0.3,0),
                normal = c(0.3,0.3,0.7),
                above = c(0.2,0.4,0.3),
                tc_cat = c(-1,0,0),
                year = 1:3)
print(dt)
RES(dt)
```

`restrict_to_country` *restricts data to a specified country*

Description

Restricts a dataset to one or more countries, specified by their names. If you have lon/lat data and don't know which countries these coordinates belong to, see `add_country_names`. Can restrict data to a rectangle around a given country as well (usually looks nicer for plotting).

Usage

```
restrict_to_country(dt, ct, rectangle = FALSE, tol = 1)
```

Arguments

`dt`	the data table.
`ct`	name of the country, or vector containing multiple country names
`rectangle`	logical. If FALSE (default), the data is restricted to the gridcells for which the centerpoint lies within the selected country (e.g. for computing mean scores for a country). If TRUE, the data is kept for a rectangle containing the entire country, therefore also containing gridpoints outside the country. This is the preferred option for plotting data for a specific country.
`tol`	Only used when `rectangle == TRUE`. A tolerance value for widening the plotting window, making things look a bit nicer.

Value

the data table, restricted to the selected country

Examples

```
# example data:
ex_dt = chirps_monthly[lat < 0 & month == 11 & year == 2020]
dt = restrict_to_country(ex_dt,'Kenya')
```

restrict_to_GHA *restricts data to the Greater Horn of Africa*

Description

Wraps `restrict_to_country`, and restricts to the GHA-region usually considered in CONFER, see `EA_country_names`.

Usage

```
restrict_to_GHA(dt, ...)

restrict_to_confer_region(...)
```

Arguments

`dt`	the data table.
`...`	passed on to `restrict_to_country`

Value

the data table, restricted to the selected country

Examples

```
ex_dt = chirps_monthly[lat < 0 & month == 11 & year == 2020]
dt = restrict_to_GHA(ex_dt)
```

ROCS *ROC-score/Area Under Curve(AUC)*

Description

This score is not proper, but can be used to assess the resolution of a tercile forecast. The ROC score requires more datapoints to be robust than e.g. the ignorance or Brier score. Therefore the default is to pool the data in space and only calculate one score per season.

Usage

```
ROCS(
  dt,
  f = c("below", "normal", "above"),
  o = tc_cols(dt),
  by = by_cols_terc_fc_score_sp(dt),
  pool = c("year", space_dimvars(dt)),
  dim.check = TRUE
)
```

Arguments

dt	Data table containing the predictions.
f	column names of the prediction.
o	column name of the observations (either in obs_dt, or in dt if obs_dt = NULL). The observation column needs to contain -1 if it falls into the first category (corresponding to fcs[1]), 0 for the second and 1 for the third category.
by	column names of grouping variables, all of which need to be columns in dt. Default is to group by all instances of month, season, system and lead_time that are columns in dt.
pool	column name(s) for the variable(s) along which is averaged, typically just 'year'.
dim.check	Logical. If TRUE, the function tests whether the data table contains only one row per coordinate-level, as should be the case.

Value

A data table with the scores

Examples

```
dt = data.table(below = c(0.5,0.3,0),
                normal = c(0.3,0.3,0.7),
                above = c(0.2,0.4,0.3),
                tc_cat = c(-1,0,0),
                lon = 1:3)
print(dt)
ROCS(dt)
```

ROC_curve *ROC curve for tercile forecasts*

Description

Creates ROC curves from a data table containing tercile forecasts. It wraps roc_curve_vec. By default, 4 ROC-curves are drawn, one for each the prediction of above-, normal- and below-values, plus one for all forecasts together. You can provide a 'by' argument to obtain separate ROC-curves for different values of the by-columns. E.g., when your data table contains a column named 'season', you can set by = 'season'. Then, the function will output a list of 16 ROC-curvess, 4 for each season.

Usage

```
ROC_curve(
  dt,
  f = c("below", "normal", "above"),
  o = tc_cols(dt),
```

```
  by = NULL,
  pool = setdiff(dimvars(dt), by),
  interpolate = TRUE,
  dim.check = TRUE
)
```

Arguments

dt	Data table containing tercile forecasts
f	column names of the prediction columns
o	column name of the observation column
by	column names of grouping variables. Default is to not group.
pool	column names of pooling variables (used for the dimension check). Default is all dimvars.
interpolate	Logical. If TRUE, the curve connects the dots making up the ROC curve (which looks nicer), if not a step function is drawn (which is closer to the mathematical definition of the ROC curve).
dim.check	Logical. If TRUE, the function checks whether the columns in by and pool span the entire data table.

Value

A list of gg objects which can be plotted by `ggpubr::ggarrange` (for example)

Examples

```
dt = data.table(below = c(0.5,0.3,0),
                normal = c(0.3,0.3,0.7),
                above = c(0.2,0.4,0.3),
                tc_cat = c(-1,0,0),
                lon = 1:3)
print(dt)
pp = ROC_curve(dt)
if(interactive()) plot(pp)
```

---

roc_curve_vec *ROC curves*

Description

Plot the ROC-curve for a vector of probabilities and corresponding observations.

Usage

```
roc_curve_vec(probs, obs, interpolate = TRUE)
```

Arguments

probs	vector with probabilities (between 0 and 1)
obs	vector with categorical observations
interpolate	logical. If TRUE the ROC-curve is interpolated and drawn as a continuous function. Otherwise it is drawn as a step function.

Value

a gg object

Examples

```
probs = seq(0,1,length.out = 5)
obs = c(FALSE,FALSE,TRUE,FALSE,TRUE)
pp = roc_curve_vec(probs,obs)
if(interactive()) plot(pp)
```

---

roc_score_vec *ROC score (AUC)*

Description

Calculates the area under curve (AUC) or ROC-score from a vector of probabilities and corresponding observations. Formula (1a) from Mason&2018 is used in the calculation, corresponding to trapezoidal interpolation. This is mostly an auxiliary function for the ROCS function, but also used in the ROC-diagram function, where the AUC is added to the diagrams.

Usage

```
roc_score_vec(probs, obs)
```

Arguments

probs	vector with probabilities (between 0 and 1)
obs	vector with categorical observations (as TRUE/FALSE)

Value

numeric. The ROC score.

Examples

```
roc_score_vec(probs = c(0.1,0.6,0.3,0.4),
              obs = c(FALSE,TRUE,TRUE,FALSE))
```

round_probs *auxiliary function for rounding probabilities*

Description

takes a vector of probabilities (between 0 and 1) and rounds them to the scale specified by binwidth. This is used for reliability diagrams, where one point is drawn for each bin. 0 is always at the center of the first interval for rounding: E.g. if binwidth = 0.05 (the default), then probabilities up to 0.025 are rounded to 0, probs between 0.025 and 0.075 are rounded to 0.05, etc.

Usage

```
round_probs(probs, binwidth = 0.05)
```

Arguments

probs	vector of probabilities (between 0 and 1, not percent)
binwidth	width of the bins for rounding.

Value

vector with rounded probabilities

Examples

```
round_probs(c(0.001,0.7423))
```

RPS *Ranked Probability score*

Description

This score is suitable for tercile category forecasts.

Usage

```
RPS(
  dt,
  f = c("below", "normal", "above"),
  o = tc_cols(dt),
  by = by_cols_terc_fc_score(),
  pool = "year",
  dim.check = TRUE
)
```

Arguments

dt	Data table containing the predictions.
f	column names of the prediction.
o	column name of the observations (either in obs_dt, or in dt if obs_dt = NULL). The observation column needs to contain -1 if it falls into the first category (corresponding to fcs[1]), 0 for the second and 1 for the third category.
by	column names of grouping variables, all of which need to be columns in dt. Default is to group by all instances of month, season, lon, lat, system and lead_time that are columns in dt.
pool	column name(s) for the variable(s) along which is averaged, typically just 'year'.
dim.check	Logical. If TRUE, the function tests whether the data table contains only one row per coordinate-level, as should be the case.

Value

A data table with the scores

Examples

```
dt = data.table(below = c(0.5,0.3,0),
                normal = c(0.3,0.3,0.7),
                above = c(0.2,0.4,0.3),
                tc_cat = c(-1,0,0),
                year = 1:3)
print(dt)
RPS(dt)
```

RPSS *Ranked Probability skill score*

Description

This score is suitable for tercile category forecasts.

Usage

```
RPSS(
  dt,
  f = c("below", "normal", "above"),
  o = tc_cols(dt),
  by = by_cols_terc_fc_score(),
  pool = "year",
  dim.check = TRUE
)
```

Arguments

dt	Data table containing the predictions.
f	column names of the prediction.
o	column name of the observations (either in obs_dt, or in dt if obs_dt = NULL). The observation column needs to contain -1 if it falls into the first category (corresponding to fcs[1]), 0 for the second and 1 for the third category.
by	column names of grouping variables, all of which need to be columns in dt. Default is to group by all instances of month, season, lon, lat, system and lead_time that are columns in dt.
pool	column name(s) for the variable(s) along which is averaged, typically just 'year'.
dim.check	Logical. If TRUE, the function tests whether the data table contains only one row per coordinate-level, as should be the case.

Value

A data table with the scores

@examples dt = data.table(below = c(0.5,0.3,0), normal = c(0.3,0.3,0.7), above = c(0.2,0.4,0.3), tc_cat = c(-1,0,0), year = 1:3) print(dt) RPSS(dt)

---

run_dimension_check_ens_fc_score

Auxiliary Function

Description

called inside functions that calculate scores for ensemble forecasts. Checks whether the provided data table has the right format.

Usage

```
run_dimension_check_ens_fc_score()
```

---

run_dimension_check_terc_forecast

Auxiliary Function

Description

called inside functions that calculate scores for ensemble forecasts. Checks whether the provided data table has the right format.

Usage

```
run_dimension_check_terc_forecast()
```

season_strings_to_int *Auxiliary function for decoding season-strings*

Description

Auxiliary function for decoding season-strings

Usage

```
season_strings_to_int(seasons = c("MAM", "JJAS", "OND"))
```

Arguments

seasons	A character vector of season strings, see convert_monthly_to_seasonal() for details

set_spatial_grid *Set Spatial Grid Attributes to a Data Table*

Description

This function creates the spatial grid attribute for a data table. If the data table already has such an attribute, missing information is filled in. In particular, the function checks whether a grid is regular, allowing for rounding errors in the grid coordinates, see details below. By default the grid coordinates are rounded to a regular grid if they are very close to being regular. While this sounds dangerous, it is almost always desirable to treat coordinates like that when working with data tables.

Usage

```
set_spatial_grid(
  dt,
  coor_cns = NULL,
  check_regular = TRUE,
  regular_tolerance = 1,
  verbose = FALSE
)
```

Arguments

dt	A data table object.
coor_cns	Optional character vector of length two indicating the names of the spatial coordinates within the data table in order x,y. Default (NULL) makes the function guess based on column names.
check_regular	A logical indicating whether to check for regularity of the grid. This should essentially always be done but can be suppressed for speed. Defaults to TRUE.

regular_tolerance
: Value >= 0 specifying the amount of rounding error we allow for still recognizing a grid as regular. Given in percent of the minimum of dx and dy. Default is 1. Based on this value coordinates are rounded to the smallest after-comma-digit making them regular, as long as this rounding introduces less error than min(dx,dy)*regular_tolerance/100. Set this to NULL if you are absolutely certain that you don't want to round/change the grid. Doing this or decreasing this below 1 is not recommended, see details below.

verbose
: Logical. If TRUE, the grid information is printed out (by a call to grid_info).

Details

The grid attribute is a named list with (some of) the following pages:

- coor_cns: Character vector of length two specifying the names of the data-table-columns containing the spatial grids (in order x,y).
- x,y: Numeric vectors of all unique x- and y-coordinates in increasing order (NAs not included).
- regular: Logical. Is the grid *regular*? See details below.
- dx,dy: Step sizes of the regular grid (only contained if regular = TRUE). By convention we set dx to 9999 if only one x-coordinate is present, likewise for dy.
- complete: Logical. Is the regular grid *complete*? See details below.

We call a grid *regular* if there is a coordinate (x0,y0) and positive values dx, dy, such that each coordinate of the grid can be written as (x0 + n*dx,y0 + m*dy) for integers n,m. Importantly, a regular grid does not need to be "a complete rectangle", we allow for missing coordinates, see details below. We call it a *regular complete grid* if the grid contains these numbers for all integers n, m between some limits n_min and n_max, respectively m_min, m_max.

Checking regularity properly is a difficult problem, because we allow for missing coordinates in the grid and allow for rounding errors. For the treatment of rounding errors it is not recommended to set regular_tolerance to NULL or a very small value (e.g. 0.1 or smaller). In this case, grids that are regular in praxis are frequently not recognized as regular: Take for example the three x-coordinates 1, 1.5001, 2.4999. They are supposed to be rounded to 1 digit after the comma and then the grid is regular with dx = 0.5. However, if regular_tolerance is NULL, the grid will be marked as irregular. Similarly, if regular_tolerance is too small, the function is not allowed to make rounding errors of 0.0001 and the grid will also not be recognized as regular.

When it comes to the issue of missing values in the grid, we are (deliberately) a bit sloppy and only check whether the coordinates are part of a grid with dx being the minimum x-difference between two coordinates, and similar dy. This may not detect regularity, when we have data that is sparse on a regular grid. An example would be the three lon/lat coordinates c(0,0), c(2,0), c(5,0). They clearly lie on the regular integer-lon/lat- grid. However, the grid would show as not regular, because dx is not checked for smaller values than 2. This choice is on purpose, since for most applications grids with many (or mostly) holes should be treated as irregular (e.g. plotting, upscaling, etc.). The most important case of regular but not complete grids is gridded data that is restricted to a certain region, e.g. a country or restricted to land. This is what we think of when we think of a regular incomplete grid, and for such data the check works perfectly.

Note that at the very bottom it is the definition of regularity itself that is a bit tricky: If we allow dx, dy to go all the way down to the machine-delta, then pretty much any set of coordinates represented

in a computer is part of a regular grid. This hints at testing and detecting regularity actually depending on how small you're willing to make your dx,dy. An example in 1 dimension: consider the three 1-dimensional coordinates 0, 1, and m/n, with m and n integers without common divisors and m>n. It is not difficult to see that these coordinates are part of a regular grid and that the largest dx for detecting this is 1/n. This shows that you can have very small coordinate sets that are in theory regular, but their regularity can be arbitrarily hard to detect. An example of a grid that is truely not regular are the three x-coordinates 0,1,a with a irrational.

Value

Nothing, the attributes of dt are set in the parent environment. Moreover, the grid coordinates may be rounded If regular

Examples

```
dt = data.table(lon = 1:4, lat = rep(1:2,each = 2), some_data = runif(4))
print(dt)
attr(dt,'grid')

set_spatial_grid(dt)
attr(dt,'grid')
```

---

space_dimvars *Auxiliary function*

Description

returns all column names indicating a spatial coordinate.

Usage

```
space_dimvars(dt = NULL)
```

Arguments

dt	optional. You can provide a data table, then the function returns the names of spatial coordinate columns in this data table.

Value

Character vector with column names.

Examples

```
space_dimvars()
```

SRC *Compute the slope of the reliability curve*

Description

Values below 1 indicate a lack of resolution or overconfidence, 1 is perfect, above means underconfident. This score requires more datapoints to be robust than e.g. the ignorance or Brier score. Therefore the default is to pool the data in space and only calculate one score per season.

Usage

```
SRC(
  dt,
  f = c("below", "normal", "above"),
  o = tc_cols(dt),
  by = by_cols_terc_fc_score_sp(dt),
  pool = c("year", space_dimvars(dt)),
  dim.check = TRUE
)
```

Arguments

dt	Data table containing the predictions.
f	column names of the prediction.
o	column name of the observations (either in obs_dt, or in dt if obs_dt = NULL). The observation column needs to contain -1 if it falls into the first category (corresponding to `fcs[1]`), 0 for the second and 1 for the third category.
by	column names of grouping variables, all of which need to be columns in dt. Default is to group by all instances of month, season, lon, lat, system and lead_time that are columns in dt.
pool	column name(s) for the variable(s) along which is averaged, typically just 'year'.
dim.check	Logical. If TRUE, the function tests whether the data table contains only one row per coordinate-level, as should be the case.

Value

A data table with the scores

@examples dt = data.table(below = c(0.5,0.3,0), normal = c(0.3,0.3,0.7), above = c(0.2,0.4,0.3), tc_cat = c(-1,0,0), year = 1:3) print(dt) SRC(dt)

tc_cols *Tercile column names*

Description

which column names are interpreted as observed tercile categories

Usage

```
tc_cols(dt = NULL)
```

Arguments

dt	optional data table. If provided, the function guesses which column contains the observations. Else it returns all recognized column names.

Value

Character vector with column names.

Examples

```
tc_cols()
```

tendency_diag *Tendency diagram from a data table containing tercile forecasts.*

Description

Tendency diagram from a data table containing tercile forecasts.

Usage

```
tendency_diag(
  dt,
  f = c("below", "normal", "above"),
  o = tc_cols(dt),
  by = NULL,
  pool = setdiff(dimvars(dt), by),
  dim.check = TRUE
)
```

Arguments

dt	Data table containing tercile forecasts
f	column names of the prediction columns
o	column name of the observation column
by	column names of grouping variables. Default is to not group.
pool	column names of pooling variables (used for the dimension check). Default is all dimvars.
dim.check	Logical. If TRUE, the function checks whether the columns in by and pool span the entire data table.

Value

If by == NULL a gg object, otherwise a list of gg objects that can be plotted by ggpubr::ggarrange (for example)

Examples

```
dt = data.table(below = c(0.5,0.3,0),
                normal = c(0.3,0.3,0.7),
                above = c(0.2,0.4,0.3),
                tc_cat = c(-1,0,0),
                lon = 1:3)
print(dt)
pp = tendency_diag(dt)
if(interactive()) plot(pp)
```

---

tercile_plot *Function for plotting terciles*

Description

Function for plotting terciles

Usage

```
tercile_plot(
  dt,
  data_col = tc_cols(dt),
  mn = NULL,
  low = "orange",
  mid = "cyan",
  high = "green1",
  name = "",
  labels = c("Wetter", "Average", "Drier"),
  na.value = "white",
  extent = NULL,
```

```
  expand.x = c(-0.5, 0.5),
  expand.y = c(-0.5, 2),
  dimension_check = TRUE
)
```

Arguments

dt	data table
data_col	Name of the column containing the observed tercile category
mn	optional title for the plot.
low, mid, high	colors for the three categories
name	optional title for the colorscale
labels	How to label the three categories
na.value	How to color missing values
extent	Optional vector of length 4 specifying the plotting borders in order xmin, xmax, ymin, ymax.
expand.x, expand.y	How far should the plotting borders be extended (beyond the data range)?
dimension_check	Logical. By default the function checks that there are not multiple values per coordinate (and subsets to the first level if there are several, e.g. to the first year and month (by appearance in dt) if dt contains data for several years and months).

Examples

```
dt = chirps_monthly[month == 12 & lat <0 & year == 2018]
p = tercile_plot(dt = dt)
if(interactive()) plot(p)
```

---

tfc_from_efc *Get tercile probability forecast from ensemble forecasts*

Description

The function takes a data table containing ensemble predictions and reduces it to predicted tercile probabilities. The data table should either have a column 'tercile_cat' or it will be generated in the process (by add_tercile_cat). In particular, if you don't know the tercile category of the ensemble predictions, your data table should contain hindcasts as well, such that the tercile categories are calculated correctly. The probability for 'below', for example, is the fraction of ensemble members predicting below normal (for this coordinate).

Usage

```
tfc_from_efc(dt, by = setdiff(dimvars(dt), "member"), keep_cols = NULL, ...)
```

Arguments

dt	The data table.
by	Names of columns to group by.
keep_cols	A vector of column names that you want to keep. Column names in by are kept automatically.
...	passed on to add_tercile_probs.

Value

A new data table with tercile forecasts

Examples

```
test_dt = ecmwf_monthly[lat < 0 & month == 11]
tfc = tfc_from_efc(test_dt)
```

---

tfc_gha_plot *Plotting function with different map for Greater Horn of Africa*

Description

This function wraps tfc_plot(), but uses a different map for borders. The map is part of the package and is the one currently used during GHACOFs at ICPAC.

Usage

```
tfc_gha_plot(
  ...,
  expand.x = c(-0.5, 0.5),
  expand.y = c(-0.5, 2),
  showplot = TRUE
)
```

Arguments

..., expand.x, expand.y, showplot	passed to tfc_plot().

Examples

```
dt = tfc_from_efc(ecmwf_monthly[month == 11 & lat < 0])
pp = tfc_gha_plot(dt[year == 2018], expand.y = c(0.5,0.5))
if(interactive()) plot(pp)
```

`tfc_plot` *plotting function for tercile forecasts*

Description

Plots spatial maps of tercile forecasts. Requires a data table with three columns `"below"`, `"normal"`, `"above"` which sum to 1. For each gridpoint only the highest of the three values is plotted, so there are three colorscales.

Usage

```
tfc_plot(
  dt,
  discrete_cs = TRUE,
  rmax = NULL,
  below = "brown",
  normal = "gold",
  above = "forestgreen",
  na.value = "gray75",
  cs_names = c("below", "normal", "above"),
  oob = NULL,
  guide_barwidth = grid::unit(0.01, units = "npc"),
  guide_barheight = grid::unit(0.15, units = "npc"),
  legend_horizontal = FALSE,
  binwidth = "auto",
  add_map = TRUE,
  extent = NULL,
  expand.x = c(0, 0),
  expand.y = c(0, 0),
  showplot = TRUE,
  dimension_check = TRUE
)
```

Arguments

`dt`	Data table containing the data for plotting.
`discrete_cs`	Logical. Do you want to use discrete color scales (default) or not.
`rmax`	Optional value to fix the range of the colorscale (lower limit is always 0.33).
`below`, `normal`, `above`	Colors to use for the three categories. Default is `'brown'`, `'gold'`, `'forestgreen'`.

na.value — Color to use for missing value. Default is 'gray75'.

cs_names — Character vector of length three giving the legend titles for the below-, normal-, and above category.

oob — Behavior for data above r_max. Passed to ggplot2::scale_fill_continuous() if discrete_cs == FALSE or else to ggplot2::scale_fill_steps().

guide_barwidth, guide_barheight — value to specify the width and height of the color guide. Are flipped if legend_horizontal is TRUE. Use units(...,"npc") to make it work across all output devices.

legend_horizontal — Logical. Set to TRUE to show the legend horizontally underneath the plot.

binwidth — Width of the steps when a discrete colorscale is used.

add_map — logical, defaults to TRUE, mostly for internal use. Set to FALSE to remove borders (e.g. if you want to add them yourself from a shapefile).

extent — An optional four-element vector in the order xmin,xmax,ymin,ymax for specifying the spatial extent of the plot. Default is to fit the extent to the data.

expand.x, expand.y — vectors with two entries to be added to xlims/ylims of the plot. E.g. expand.x = c(-0.5,0.5) expands the plot by half a longitude both on the right and left hand side.

showplot — Logical. Should the plot be displayed at the end?

dimension_check — Logical. By default the function checks that there are not multiple values per coordinate (and subsets to the first level if there are several, e.g. to the first year and month (by appearance in dt) if dt contains data for several years and months).

Value

a ggplot object.

Author(s)

Claudio Heinrich

Examples

```
#dt = tfc_from_efc(ecmwf_monthly[month == 11 & lat < 0])
#pp = tfc_plot(dt[year == 2018])
#if(interactive()) plot(pp)
```

time_dimvars *Auxiliary function*

Description

returns all column names indicating a temporal coordinate.

Usage

```
time_dimvars(dt = NULL)
```

Arguments

dt	optional. You can provide a data table, then the function returns the names of temporal coordinate columns in this data table.

Value

Character vector with column names.

Examples

```
time_dimvars()
```

upscale_chirps *Upscales monthly CHIRPS data to a coarser grid*

Description

this is mostly auxiliary and called from download_chirps_monthly. Uses the function upscale_regular_lon_lat, but derives the weights for upscaling only once for efficiency and avoids simultaneous loading of all CHIRPS data.

Usage

```
upscale_chirps(
  update = TRUE,
  years = NULL,
  months = NULL,
 upscale_grid = data.table(expand.grid(lon = seq(GHA_extent()[1], GHA_extent()[2], 0.5),
    lat = seq(GHA_extent()[3], GHA_extent()[4], 0.5))),
  root_dir = NULL,
  version = "UCSB",
  us_dir = file.path(root_dir, "upscaled")
)
```

Arguments

update	Logical, if TRUE, files that have already been upscaled are skipped
years, months	Which years and months do you want to upscale? NULL upscales everything there is (except if update is TRUE).
upscale_grid	A regular lon/lat grid for upscaling. Defaults to half degrees.
root_dir	directory where the high-resolution file is stored.
version	Version specifier, should be 'UCSB' or 'ICPAC'. The latter only works if you have access to CHIRPS blended.
us_dir	Directory where the low-resolution file will be stored.

Value

Nothing.

Examples

```
if(interactive()){
upscale_chirps()
}
```

---

upscale_regular_lon_lat

Function for matching data between different grids

Description

Upscales data from one regular lon-lat grid to another lon-lat grid that is coarser or of the same resolution. It uses conservative interpolation (rather than bilinear interpolation) which is the better choice for upscaling, see details below. If the fine grid and coarse grid are of the same resolution but shifted, results are (almost) identical to bilinear interpolation (almost because bilinear interpolation does not account for the fact that grid cells get smaller towards the pole, which this function does).

The function addresses the following major challenges:

- The fine grid does not need to be nested in the coarse grid, creating different partial overlap scenarios. Therefore, the value of each fine grid cell may contribute to multiple (up to four) coarse grid cells.
- Grid cell area varies with latitude, grid cells at the equator are much larger than at the poles. This affects the contribution of grid cells (grid cells closer to the pole contribute less to the coarse grid cell average).
- Frequently, it is required to upscale *repeated* data between the same grids, for example when you want to upscale observations for many different years. In this case, the calculation of grid cell overlaps is only done once, and not repeated every time.
- For coarse grid cells that are only partially covered, a minimal required fraction of coverage can be specified.

- It is memory efficient: Naive merging of data tables or distance-based matching of grid cells is avoided, since it results in unnecessary large lookup tables that may not fit into memory when both your fine and your coarse grid are high-resolution.

Usage

```
upscale_regular_lon_lat(
  dt,
  coarse_grid,
  uscols,
  bycols = setdiff(dimvars(dt), c("lon", "lat")),
  save_weights = NULL,
  req_frac_of_coverage = 0
)
```

Arguments

`dt`	data table containing the data you want to upscale.
`coarse_grid`	data table containing lons/lats of the grid you want to upscale to.
`uscols`	column name(s) of the data you want to upscale (can take multiple columns at once, but assumes that the different columns have missing values at the same position).
`bycols`	optional column names for grouping if you have repeated data on the same grid, e.g. use bycols = 'date' if your data table contains observations for many dates on the same grid (and the column specifying the date is in fact called 'date').
`save_weights`	optional file name for saving the weights for upscaling. Used for the CHIRPS data.
`req_frac_of_coverage`	Numeric value between 0 and 1. All coarse grid cells with less coverage than this value get assigned a missing value. In particular, setting this to 0 (the default) means a value is assigned to each coarse grid cell that overlaps with at least one fine grid cell. Setting this to 1 means only coarse grid cells are kept for which we have full coverage.

Details

Bilinear interpolation is generally not appropriate for mapping data from finer to coarser grids. The reason is that in BI, the value of a coarse grid cell only depends on the four fine grid cells surrounding its center coordinate, even though many fine grid cells may overlap the coarse grid cell). Conservative interpolation calculates the coarse grid cell value by averaging all fine grid cells overlapping with it, weighted by the fraction of overlap. This is the appropriate way of upscaling when predictions and observations constitute grid point averages, which is usually the case (Göber et al. 2008).

The grids are assumed to be *regular*, but are not required to be *complete* (see `set_spatial_grid`). The function is faster when missing-data grid points are not contained in `dt` (then fewer grid points need to be matched).

Value

A data table with the upscaled values.

References

Göber, M., Ervin Z., and Richardson, D.S. (2008): *"Could a perfect model ever satisfy a naïve forecaster? On grid box mean versus point verification."* Meteorological Applications: A journal of forecasting, practical applications, training techniques and modelling 15, no. 3 (2008): 359-365.

`ver_map` *Plot a verification map of percentiles*

Description

For each location, the map shows whether the observed value was normal, below, or above. This makes it possible to visually compare to the usual tercile forecsst

Usage

```
ver_map(
  dt,
  o = obs_cols(dt),
  yy = dt[, max(year)],
  climatology_period = unique(dt[, year]),
  out_file = NULL
)
```

Arguments

`dt`	input data table. This has to contain the observations for the year to plot, as well as for many other years (which are used to calculate the climatological reference). The data table should have coumns named `lon`, `lat`, `year`, and an observation column, the name of which is passed as value of `o` to the function, see below. For each level of `lon`, `lat`, and `year`, the table should only contain one row (this is checked by the function).
`o`	name of the column containing the observation.
`yy`	The year for which to show the verification map. Defaults to the last year available in dt
`climatology_period`	which years should the climatology be calculated on? Defaults to all years (except `yy`) in `dt`
`out_file`	optional path and file name (including valid filetype, like .pdf or .png) for saving the file. If not provided, the function just shows the plot in the running R session.

Value

a gg object

Examples

```
# takes some time:
pp = ver_map(chirps_monthly[month == 11],yy = 2018)
if(interactive()) plot(pp)
```

`ver_map_chirps` *Plot a verification map of percentiles based on precomputed CHIRPS quantiles.*

Description

The quantiles should be computed and saved by the function `chirps_ver_map_quantiles`.

Usage

```
ver_map_chirps(
  mm = month(Sys.Date() - 60),
  yy = year(Sys.Date() - 60),
  version = "UCSB",
  resolution = "low",
  ...
)
```

Arguments

yy, mm	The year and month for which to show the verification map. Defaults to the month 60 days ago (in order to avoid using preliminary data).
version	which CHIRPS version to use.
resolution	Spatial resolution, 'high' or 'low'
...	passed on to ver_map.

Value

A gg object

Examples

```
 # takes a while:
if(interactive()) ver_map_chirps(mm = 12,yy = 2022)
```

Index

∗ **datasets**
    chirps_monthly, 10
    ecmwf_monthly, 29

add_climatology, 4
add_country, 4
add_country(), *15*
add_country_names, *4*, 5, *29*, *58*
add_tercile_cat, 6, *72*
add_tercile_cat(), *15*
add_tercile_probs, 7, *73*
are_all_elements_within_eps, 7

by_cols_ens_fc_score, 8
by_cols_terc_fc_score, 8
by_cols_terc_fc_score_sp, 9

checks_ens_fc_score, 9
checks_terc_fc_score, 9
chirps_dir, 10
chirps_monthly, 10
chirps_ver_map_quantiles, 11
climatology_ens_forecast, 12
climatology_threshold_exceedence, 12
combine, 13
complete_regular_grid, 14
convert_monthly_to_seasonal, 14
convert_monthly_to_seasonal(), *66*
CPA, 16
create_diagram_by_level, 17
CRPS, 18
crps_aux, 20
crps_aux_esc, 20
CRPSS, 19

data_dir, 21
delete_redundant_files, 21
dimvars, 22
dimvars(), *15*
disc_score_dt, 22
DISS, 23
download_chirps_monthly, 24
download_chirps_monthly_high, 25
download_chirps_monthly_low, 26
download_chirps_prelim_aux, 27
dt_to_netcdf, 28

EA_country_names, *5*, 29, *59*
ecmwf_monthly, 29
EIR, 30

fc_cols, 31

get_mask, 32
get_quantiles, 32, *33*, *34*
get_terciles, 33
ggplot2::scale_fill_continuous(), *75*
ggplot2::scale_fill_steps(), *75*
ggplot_dt, 34, *37*
ggplot_dt_gha_map *(*gha_plot*)*, 37
ggplot_dt_shf *(*gha_plot*)*, 37
GHA_extent, 36
gha_plot, 37
grid_info, 37, *67*

HS, 38
HSS, 39

IGS, 40
IGSS, 41
indicator_times_value_aux, 42

load_chirps, *32*, 42
lt_cols, 43

MB, 44
MBS, 45
merge, *13*
modify_dt_map_plotting, 46
MSD_to_YM, 46
MSE, 47

MSES, 48

netcdf_to_dt, 49
netcdf_to_dt(), *28*

obs_cols, 50
obs_dimvars, 51

PCC, 51
profit_graph, 52

REL, 53
rel_diag, 55
rel_diag_vec, 56
RES, 57
restrict_to_confer_region
(restrict_to_GHA), 59
restrict_to_country, 58, *59*
restrict_to_GHA, 59
ROC_curve, 60
roc_curve_vec, 61
roc_score_vec, 62
ROCS, 59
round_probs, 63
RPS, 63
RPSS, 64
run_dimension_check_ens_fc_score, 65
run_dimension_check_terc_forecast, 65

season_strings_to_int, 66
set_spatial_grid, *14*, *37*, 66, *78*
space_dimvars, 68
SRC, 69

tc_cols, 70
tc_cols(), *15*
tendency_diag, 70
tercile_plot, 71
tfc_from_efc, 72
tfc_from_efc(), *15*
tfc_gha_plot, 73
tfc_plot, 74
tfc_plot(), *73*
time_dimvars, 76

upscale_chirps, 76
upscale_regular_lon_lat, 77

ver_map, 79
ver_map_chirps, 80