Package 'Dyn4cast'

November 18, 2026

Title Dynamic Modeling and Machine Learning Environment

Version 11.11.28-1

Date 2026-11-05

Description Estimates, predict and forecast dynamic models as well as Machine Learning metrics which assists in model selection for further analysis. The package also have capabilities to provide tools and metrics that are useful in machine learning and modeling. For example, there is quick summary, percent sign, Mallow's Cp tools and others. The ecosystem of this package is analysis of economic data for national development. The package is so far stable and has high reliability and efficiency as well as time-saving. The package is a variety but the following references are important guide to the major themes in the package (Hyndman & Athanasopoulos (2014 ISBN 978-0-9875071-0-5), Alkire & Santos (2014, doi.org/10.1016/j.worlddev.2014.01.026)).

License GPL (>= 3)

Encoding UTF-8

LazyLoad true

LazyData true

Config/Needs/website rmarkdown

VignetteBuilder knitr, rmarkdown

Imports stats, splines, Metrics, tidyr, ggplot2, generics, magrittr, formattable, utils, zoo, ModelMetrics, dplyr, modelsummary, corrplot, marginaleffects, tibble, purrr, lifecycle, tidyselect

Depends R (>= 4.2)

Suggests testthat (>= 3.0.0), tidyverse, rmarkdown, covr, caret, kableExtra, knitr, spelling, psych, readr, MetBrewer, data.table, ggtext, lubridate, forecast, MASS, mlogit, nnet, betareg, mvProbit, miscTools

Config/testthat/edition 3

URL https://github.com/JobNmadu/Dyn4cast, https://jobnmadu.github.io/software.html, https://jobnmadu.r-universe.dev/Dyn4cast

BugReports https://github.com/JobNmadu/Dyn4cast/issues

Language en-US

Config/roxygen2/version 8.1.1

NeedsCompilation no

Author Job Nmadu [aut, cre] (ORCID: <https://orcid.org/0000-0002-1320-8957>)

Maintainer Job Nmadu <job.nmadu@gmail.com>

Repository CRAN

Date/Publication 2026-11-18 15:30:08 UTC

Contents

corplot 2
data_transform 3
Dyn4cast-deprecated 5
DynamicForecast 5
estimate_plot 8
formattedcut 8
garrett_ranking 9
gender 10
index_construction 11
Linearsystems 12
MallowsCp 15
mdi 16
MLMetrics 18
model_factors 21
odds_summary 22
Percent 23
plot_mdi 24
quicksummary 25
relative_likert 26
treatment_model 28

Index **30**

`corplot` *Custom plot of correlation matrix*

Description

This is a custom plot for correlation matrix in which the coefficients are displayed along with graphics showing the magnitude of each coefficient.

Usage

```
corplot(r)
```

Arguments

r	Correlation matrix of the data for the plot

Value

The function returns a custom plot of the correlation matrix

corplot	The custom plot of the correlation matrix

---

data_transform *Standardize* data.frame *for comparable* **Machine Learning** *prediction and visualization*

Description

Often economic and other **Machine Learning** data are of different units or sizes making either estimation, interpretation or visualization difficult. The solution to these issues can be handled if the data can be transformed into *unitless* or data of similar magnitude. This is what data_transform is set to do. It is simple and straight forward to use.

Usage

```
data_transform(data, method, margin = 2)
```

Arguments

data	A data.frame with numeric data for transformation. All columns in the data are transformed
method	The type of transformation. There three options. 1 is for min-max transformation, 2 is for log transformation and 3 is for mean-SD transformation.
margin	Option to either transform the data 2 == column-wise or 1 == row-wise. Defaults to column-wise transformation if no option is indicated.

Value

This function returns the output of the data transformation process as

data_transformed	A new data.frame containing the transformed values

Examples

```
library(Dyn4cast)
library(tidyverse)

# View the data without transformation

data0 <- Transform %>%
pivot_longer(!X, names_to = "Factors", values_to = "Data")

ggplot(data = data0, aes(x = X, y = Data, fill = Factors, color = Factors)) +
  geom_line() +
  scale_fill_brewer(palette = "Set1") +
  scale_color_brewer(palette = "Set1") +
  labs(y = "Data", x = "Series", color = "Factors") +
  theme_bw(base_size = 12)

# Example 1: Transformation by `min-max` method.
# You could also transform the `X column` but is is better not to.

data1 <- data_transform(Transform[, -1], 1)
data1 <- cbind(Transform[, 1], data1)
data1 <- data1 %>%
  pivot_longer(!X, names_to = "Factors", values_to = "Data")

ggplot(data = data1, aes(x = X, y = Data, fill = Factors, color = Factors)) +
  geom_line() +
  scale_fill_brewer(palette = "Set1") +
  scale_color_brewer(palette = "Set1") +
  labs(y = "Data", x = "Series", color = "Factors") +
  theme_bw(base_size = 12)

# Example 2: `log` transformation

data2 <- data_transform(Transform[, -1], 2)
data2 <- cbind(Transform[, 1], data2)
data2 <- data2 %>%
  pivot_longer(!X, names_to = "Factors", values_to = "Data")

ggplot(data = data2, aes(x = X, y = Data, fill = Factors, color = Factors)) +
  geom_line() +
  scale_fill_brewer(palette = "Set1") +
  scale_color_brewer(palette = "Set1") +
  labs(y = "Data", x = "Series", color = "Factors") +
  theme_bw(base_size = 12)

# Example 3: `Mean-SD` transformation

data3 <- data_transform(Transform[, -1], 3)
data3 <- cbind(Transform[, 1], data3)
data3 <- data3 %>%
  pivot_longer(!X, names_to = "Factors", values_to = "Data")
```

```
ggplot(data = data3, aes(x = X, y = Data, fill = Factors, color = Factors)) +
  geom_line() +
  scale_fill_brewer(palette = "Set1") +
  scale_color_brewer(palette = "Set1") +
  labs(y = "Data", x = "Series", color = "Factors") +
  theme_bw(base_size = 12)
```

`Dyn4cast-deprecated` *Deprecated functions in Dyn4cast*

Description

These functions have been removed (defunct) as indicated against their names.

Usage

```
Model_factors(...)
```

Arguments

`...`	Not used.

Value

None, function now defunct.

Examples

```
# Non
```

`DynamicForecast` *Dynamic Forecast of Five Models and their Ensembles*

Description

The function estimates, predict and forecast time series data with models, and also make subset forecasts within the length of the entire trend of the data. However, the forecast is constrained to lower and upper 80% and 95% forecasts of the of the data for `integer series` in line with Hyndman & Athanasopoulos (2021). The recognized models are lm, smooth spline, polynomial splines with or without knots, quadratic polynomial, and ARIMA. The robust output include the models' estimates, time-varying forecasts and plots based on themes from ggplot. The main attraction of this function is the use of the newly introduced *equal number of trend to forecast from the model*. The function takes `daily, monthly and yearly data sets for now`.

Usage

```
DynamicForecast(Data, date, series, dyrima, Trend, Type, MaximumDate, x = 0,
x100 = 0, BREAKS = 0, ORIGIN = NULL, origin = "1970-01-01", Length = 0, ...)
```

Arguments

date	A vector containing the dates for which the data is collected. Must be the same length with series. The date must be in 'YYYY-MM-DD'. If the data is monthly series, the recognized date format is the last day of the month of the dataset e.g. 2021-02-28. If the data is a yearly series, the recognized date format is the last day of the year of the data set e.g. 2020-12-31. There is no format for Quarterly data for now.
x	**[Deprecated]**
series	A vector containing observations for estimation and forecasting. Must be the same length with date.
dyrima	**ARIMA** object of the series obtained from auto.rima in forecast package.
x100	vector of optional dataset that is to be added to the model for forecasting. The modeling and forecasting is still done if not provided. Must be the same length with series.
BREAKS	A vector of numbers indicating points of breaks for estimation of the spline models.
MaximumDate	**[Deprecated]**. The date indicating the maximum date (last date) in the data frame, meaning that forecasting starts the next date following it. The date must be a recognized date format. Note that for forecasting, the date origin is set to 1970-01-01.
Trend	The type of trend. There are three options **Day, Month and Year**.
Type	The type of response variable. There are two options **Continuous and Integer**. For integer variable, the forecasts are constrained between the minimum and maximum value of the response variable.
Length	The length for which the forecast would be made. If not given, would default to the length of the dataset i.e. sample size.
origin	default date origin which is **1970-01-01** used to position the date of data so that the forecasts are in tandem with the period of the observations.
ORIGIN	date origin of the dataset and if different from **origin** must be in the format "YYYY-MM-DD". This is used to position the date of the data to properly date the forecasts.
Data	**[Deprecated]**. Now broken into three vectors date, series and x100.
...	Additional arguments that may be passed to the function.

Value

A list with the following components:

Spline without knots
: The estimated spline model without the breaks (knots).

`Spline with knots`
: The estimated spline model with the breaks (knots).

`Smooth Spline`
: The smooth spline estimates.

`ARIMA`
: Estimated Auto Regressive Integrated Moving Average model.

`Quadratic`
: The estimated quadratic polynomial model.

`Ensembled with equal weight`
: Estimated Ensemble model with equal weight given to each of the models. To get this, the fitted values of each of the models is divided by the number of models and summed together.

`Ensembled based on weight`
: Estimated Ensemble model based on weight of each model. To do this, the fitted values of each model served as independent variable and regressed against the trend with interaction among the variables.

`Ensembled based on summed weight`
: Estimated Ensemble model based on summed weight of each model. To do this, the fitted values of each model served as independent variable and is regressed against the trend.

`Ensembled based on weight of fit`
: Estimated Ensemble model. The fit of each model is measured by the rmse.

`Unconstrained Forecast`
: The forecast if the response variable is continuous. The number of forecasts is equivalent to the length of the dataset (equal days forecast).

`Constrained Forecast`
: The forecast if the response variable is integer. The number of forecasts is equivalent to the length of the dataset (equal days forecast).

`RMSE`
: Root Mean Square Error (rmse) for each forecast.

`Unconstrained forecast Plot`
: The combined plots of the unconstrained forecasts using ggplot.

`Constrained forecast Plot`
: The combined plots of the constrained forecasts using ggplot.

`Date`
: This is the date range for the forecast.

`Fitted plot`
: This is the plot of the fitted models.

`Estimated coefficients`
: This is the estimated coefficients of the various models in the forecast.

Examples

```
library(readr)
library(forecast)
COVID19$Date <- zoo::as.Date(COVID19$Date, format = '%m/%d/%Y')
 #The date is formatted to R format
LEN <- length(COVID19$Case)
Dss <- seq(COVID19$Date[1], by = "day", length.out = LEN)
 #data length for forecast
ORIGIN = "2020-02-29"
lastdayfo21 <- Dss[length(Dss)] # The maximum length # uncomment to run
```

```
Data <- COVID19[COVID19$Date <= lastdayfo21 - 28, ]
# desired length of forecast
BREAKS <- c(70, 131, 173, 228, 274) # The default breaks for the data
dyrima <- auto.arima(Data$Case)
DynamicForecast(date = Data$Date, series = Data$Case, dyrima = dyrima,
BREAKS = BREAKS, Trend = "Day", Length = 0, Type = "Integer", x100 = 0)
```

`estimate_plot` *Plot of Order of Significance of Estimated Regression Coefficients*

Description

This function provides graphic displays of the estimated coefficients in the order of their significance in the models. This would assists in accessing models to decide which can be used for further analysis, prediction and policy consideration.

Usage

```
estimate_plot(model25, limit)
```

Arguments

`model25`	Estimated model for which the estimated coefficients would be plotted
`limit`	Number of variables to be included in the coefficients plots

Value

The function returns a plot of the order of importance of the estimated coefficients

`estimate_plot`	The plot of the order of importance of estimated coefficients

---

`formattedcut` *Convert continuous vector variable to formatted factors*

Description

Often, when a continuous data is converted to factors using the `base R` cut function, the resultant `Class Interval` column provide data with scientific notation which normally appears confusing to interpret, especially to casual data scientist. This function provide a more user-friendly output and is provided in a formatted manner. It is a easy to implement function.

Usage

```
formattedcut(data, breaks, cut = FALSE)
```

Arguments

data	A vector of the data to be converted to factors if not cut already or the vector of a cut data
breaks	Number of classes to break the data into
cut	Logical to indicate if the cut function has already being applied to the data, defaults to FALSE.

Value

The function returns a data frame with three or four columns i.e Lower class, Upper class, Class interval and Frequency (if the cut is FALSE).

Cut	The data frame

Examples

```
library(tidyverse)
DD <- rnorm(100000)
formattedcut(DD, 12, FALSE)
DD1 <- cut(DD, 12)
DDK <- formattedcut(DD1, 12, TRUE)
DDK
# if data is not from a data frame, the frequency distribution is required.
as.data.frame(DDK %>%
group_by(`Lower class`, `Upper class`, `Class interval`) %>%
tally())
```

---

garrett_ranking *Garrett Ranking of Categorical Data*

Description

There are three main types of ranking: Standard competition, Ordinal and Fractional. Garrett's Ranking Technique is the application of fractional ranking in which the data points are ordered and given an ordinal number/rank. The ordering and ranking provide additional information which may not be available from frequency distribution. Again, the ordering is based on the level of seriousness or severity of the data point from the view point of the respondent. Ranking enables ease of comparison and makes grouping more meaningful. It is used in social science, psychology and other survey types of research. This functions performs Garrett Ranking of up to 15 ranks.

Usage

```
garrett_ranking(data, num_rank, ranking = NULL, m_rank = c(2:15))
```

Arguments

data	The data for the Garrett Ranking, must be a data.frame.
num_rank	A vector representing the number of ranks applied to the data. If the data is a five-point Likert-type data, then number of ranks is 5.
ranking	A vector of list representing the ranks applied to the data. If not available, positional ranks are applied.
m_rank	The scope of the ranking methods which is between 2 and 15.

Value

A list with the following components:

RII	Relative importance index.
Garrett ranked data	Table of data ranked using Garrett mean score.
Garrett value	Table of ranking Garrett values

Examples

```
library(readr)
garrett_data <- data.frame(garrett_data)
ranking <- c("Serious constraint", "Constraint",
"Not certain it is a constraint", "Not a constraint",
"Not a serious constraint")

## ranking is supplied
garrett_ranking(garrett_data, 5, ranking)

# ranking not supplied
garrett_ranking(garrett_data, 5)

# you can rank subset of the data
garrett_ranking(garrett_data, 8)

garrett_ranking(garrett_data, 4)
```

---

gender *Create Gender Variable*

Description

Often, there is need to differentiate between sex and gender. Many wonder if there is any difference at all. This function will create clarity between them.

Usage

```
gender(data)
```

Arguments

data	data frame containing **Age** and **Sex** variables

Value

The data.frame with:

Gender	data frame with two additional variables.

Examples

```
df <- data.frame(Age = c(49, 30, 44, 37, 29, 56),
 Sex = c("male", "female", "female", "male", "Prefer not to say",
  "Non-binary/third gender"))
 gender(df)
```

---

index_construction *Index Construction for estimation of* **Exposure** *or* **Sensitivity**

Description

Vulnerability or to be vulnerable means the state or quality of being susceptible to physical or emotional harm, damage, or attack, usually indicating a lack of defense or protection, making someone or some systems more likely to be affected by external factors or threats. Therefore, vulnerability index is a quantitative or standardized framework of such state or quality which then makes comparisons between households, communities or systems possible. The index is made up of three main components: exposure, sensitivity and adaptive capacity. Each component has multiple indicators from wide ranges including social, medical, psychological and various extreme events like floods, drought, earthquakes etc. This function is for conversion of indicators exposure and sensitivity into a vector of index through normalization and weighting. The resulting index from each of the component is then combined via an appropriate model into vulnerability index.

Usage

```
index_construction(data)
```

Arguments

data	Data frame of indicators of Exposure or Sensitivity. The data frame must be numeric.

Value

A list with the following components:

Indexed data	dataframe of indices corresponding to the supplied data.
Index	A vector of indices representing the variable of interest, either **Exposure** or **Sensitivity**.

Examples

```
library(readr)
garrett_data <- data.frame(garrett_data)
index_construction(garrett_data)
```

`Linearsystems` *Linear Model and various Transformations for Efficiency*

Description

The linear model still remains a reference point towards advanced modeling of some datasets as foundation for **Machine Learning**, **Data Science** and **Artificial Intelligence** in spite of some of her weaknesses. The major task in **modeling** is to compare various models before a selection is made for one or for advanced modeling. Often, some trial and error methods are used to decide which model to select. This is where this function is unique. It helps to estimate 14 different linear models and provide their coefficients in a formatted Table for quick comparison so that time and energy are saved. The interesting thing about this function is the simplicity, and it is a *one line* code.

Usage

```
Linearsystems(y, x, mod, limit, Test = NA)
```

Arguments

`y`	Vector of the dependent variable. This must be numeric.
`x`	Data frame of the explanatory variables.
`mod`	The group of linear models to be estimated. It takes value from 0 to 6. 0 = EDA (correlation, summary tables, Visuals means); 1 = Linear systems, 2 = power models, 3 = polynomial models, 4 = root models, 5 = inverse models, 6 = all the 14 models
`limit`	Number of variables to be included in the coefficients plots
`Test`	test data to be used to predict y. If not supplied, the fitted y is used hence may be identical with the fitted value. It is important to be cautious if the data is to be divided between train and test subsets in order to train and test the model. If the sample size is not sufficient to have enough data for the test, errors are thrown up.

Value

A list with the following components:

`Visual means of the numeric variable`
: Plot of the means of the *numeric* variables.

`Correlation plot`
: Plot of the Correlation Matrix of the *numeric* variables. To recover the plot, please use this canonical form *object*$`Correlation plot`*$plot()*.

`Linear` The full estimates of the Linear Model.

`Linear with interaction`
The full estimates of the Linear Model with full interaction among the *numeric* variables.

`Semilog` The full estimates of the Semilog Model. Here the independent variable(s) is/are log-transformed.

`Growth` The full estimates of the Growth Model. Here the dependent variable is log-transformed.

`Double Log` The full estimates of the double-log Model. Here the both the dependent and independent variables are log-transformed.

`Mixed-power model`
The full estimates of the Mixed-power Model. This is a combination of linear and double log models. It has significant gains over the two models separately.

`Translog model` The full estimates of the double-log Model with full interaction of the *numeric* variables.

`Quadratic` The full estimates of the Quadratic Model. Here the square of *numeric* independent variable(s) is/are included as independent variables.

`Cubic model` The full estimates of the Cubic Model. Here the third-power (x^3) of *numeric* independent variable(s) is/are included as independent variables.

`Inverse y` The full estimates of the Inverse Model. Here the dependent variable is inverse-transformed (1 / y).

`Inverse x` The full estimates of the Inverse Model. Here the independent variable is inverse-transformed (1 / x).

`Inverse y & x` The full estimates of the Inverse Model. Here the dependent and independent variables are inverse-transformed 1 / y & 1 / x).

`Square root` The full estimates of the Square root Model. Here the independent variable is square root-transformed (x^0.5).

`Cubic root` The full estimates of the cubic root Model. Here the independent variable is cubic root-transformed (x^1 / 3).

`Significant plot of Linear`
Plots of order of importance and significance of estimates coefficients of the model.

`Significant plot of Linear with interaction`
Plots of order of importance and significance of estimates coefficients of the model.

`Significant plot of Semilog`
Plots of order of importance and significance of estimates coefficients of the model.

`Significant plot of Growth`
Plots of order of importance and significance of estimates coefficients of the model.

`Significant plot of Double Log`
Plots of order of importance and significance of estimates coefficients of the model.

`Significant plot of Mixed-power model`
: Plots of order of importance and significance of estimates coefficients of the model.

`Significant plot of Translog model`
: Plots of order of importance and significance of estimates coefficients of the model.

`Significant plot of Quadratic`
: Plots of order of importance and significance of estimates coefficients of the model.

`Significant plot of Cubic model`
: Plots of order of importance and significance of estimates coefficients of the model.

`Significant plot of Inverse y`
: Plots of order of importance and significance of estimates coefficients of the model.

`Significant plot of Inverse x`
: Plots of order of importance and significance of estimates coefficients of the model.

`Significant plot of Inverse y & x`
: Plots of order of importance and significance of estimates coefficients of the model.

`Significant plot of Square root`
: Plots of order of importance and significance of estimates coefficients of the model.

`Significant plot of Cubic root`
: Plots of order of importance and significance of estimates coefficients of the model.

`Model Table`
: Formatted Tables of the coefficient estimates of all the models

`Machine Learning Metrics`
: Metrics (47) for assessing model performance and metrics for diagnostic analysis of the error in estimation.

`Table of Marginal effects`
: Tables of marginal effects of each model. Because of computational limitations, if you choose to estimate all the 14 models, the Tables are produced separately for the major transformations. They can easily be compiled into one.

`Fitted plots long format`
: Plots of the fitted estimates from each of the model.

`Fitted plots wide format`
: Plots of the fitted estimates from each of the model.

`Prediction plots long format`
: Plots of the predicted estimates from each of the model.

`Prediction plots wide format`
: Plots of the predicted estimates from each of the model.

`Naive effects plots long format`
: Plots of the `lm` effects. May be identical with plots of marginal effects if performed.

`Naive effects plots wide format`
: Plots of the `lm` effects. May be identical with plots of marginal effects if performed.

`Summary of numeric variables`
: of the dataset.

`Summary of character variables`
: of the dataset.

Examples

```
library(tidyverse)
library(ggtext)

y <- linearsystems$MKTcost
x <- select(linearsystems, -MKTcost)
x <- sampling[, -1]
y <- sampling$qOutput
limit <- 20
mod <-3
Test <- NA
Linearsystems(y, x, 3, 15)
```

`MallowsCp` *Computation of MallowsCp*

Description

Mallow's Cp is one of the very useful metrics and selection criteria for machine learning algorithms (models). It is used to estimate the closest number to the number of predictors and the intercept (approximate number of explanatory variables) of linear and non-linear based models. The function inherits `residuals` from the estimated model. The uniqueness of this function compared to other procedures for computing Mallow's Cp is that it does not require nested models for computation and it is not limited to `lm` based models only.

Usage

```
MallowsCp(model2, y, x, type, Nlevels = 0)
```

Arguments

`model2`	The estimated **model** from which the Mallows Cp would be computed
`y`	The vector of the **LHS** variable of the estimated model
`x`	The matrix of the **RHS** variable of the estimated model. Note that if the model adds additional factor variables into the output, then the number of additional factors `Nlevels` is required otherwise the computed Cp would be biased.

type — The type of model (LM, ALM, GLM,N-LM, nls, ARDL, SMOOTH, SPLINE, ARIMA, plm) for which Cp would be computed broadly divided in to linear (LM, ALM, GLM, ARDL, SMOOTH, SPLINE, ARIMA, plm) and non-linear (GLM,N-LM, nls). The type of model must be specified as indicated. Supported models are LM, ALM, GLM (for binary based models), N-LM (not linear for models not clearly defined as linear or non-linear especially some of the essemble models that are merely **computed** not **estimated**) or nls for other non linear models, ARDL, SMOOTH for **smooth.spline**, SPLINE for bs spline models, ARIMA and plm.

Nlevels — Optional number of additional variables created if the model has categorical variables that generates additional dummy variables during estimation or the number of additional variables created if the model involves interaction terms.

Value

A list with the following components

MallowsCp — of the Model.

Examples

```
library(Dyn4cast)
ctl <- c(4.17, 5.58, 5.18, 6.11, 4.50, 4.61, 5.17, 4.53, 5.33, 5.14)
trt <- c(4.81, 4.17, 4.41, 3.59, 5.87, 3.83, 6.03, 4.89, 4.32, 4.69)
x <- gl(2, 10, 20, labels = c("Ctl", "Trt"))
y <- c(ctl, trt)
Model <- lm(y ~ x)
Type <- "LM"
MallowsCp(model2 = Model, y = y, x = x, type = Type, Nlevels = 0)
```

---

mdi — *Sequential Computation of Dynamic Multidimensional Indices (MDI)*

Description

This function computes the indices and all associated measures of multidimensional index sequentially in a dynamic way. Sequentially the function computes *Incidence (H = q / n)*, *Adjusted incidence (H / (q / D))*, *Deprivation Score* of each dimension in the computation, *Intensity (A)*, *Multidimensional index (MDI = H * A)*, the *Contribution* in % of each of the dimensions to MDI, and *Average deprivation among the deprived (A * D)*. Dynamically, it computes the various indices for between three and nine dimensions (D). The first five dimensions included in the computations are *Health*, *Education*, *Living standard*, *Social security* and, *Employment and Income* depending on the choice of the user. Four additional dimensions can be included in the computations. The computations are carried out either for the national sample data or can be dis-aggregated based on grouping factors, like region, sex, gender, marital status or any suitable one. The cut-off mark demarcating poor (q) and non-poor (n-q) members in the sample (n) is defaulted to 0.4 but can be varied as may be dictated by the interests or the need for the computation. The computations are in line with various procedures already outlined in literature starting with the work of Alkire et. al, (2015) but has been expanded from three dimensions to nine. Each dimension is given

`equal weight` in the computation but all indicators are weighted in line with existing guidelines in Alkire & Foster (2011) and Alkire & Santos (2010). See also Alkire & Santos (2014) and Chan & Wong (2024).

Usage

```
mdi(
  data,
  dm,
  Bar = 0.4,
  id_addn = NULL,
  Factor = NULL,
  plots = NULL,
  id = c("Health", "Education", "Living standard"),
  id_add = "Social security",
  id_add1 = "Employment and Income",
  Echo = FALSE
)
```

Arguments

`data`	`data frame` containing all the variables for the computation. Note that the variables to be used for the computation must be coded `(0,1)`.
`dm`	list of vectors of *indicators* making up each *dimension* to be computed
`Bar`	an optional vector of cut-of used to divide the population into those in the poverty category and those that are not. Defaults to 0.4 if not supplied.
`id_addn`	an optional vector of additional dimensions to be used for the computation up to a **maximum of four**.
`Factor`	an optional grouping factor for the computation which must be a variable in the *data*. If not supplied, only the national MDI will be computed.
`plots`	plots of the various measures. For this to be possible, the number of options in the `Factor` argument must be less than 41. The default is `NULL`. To produce the plots, any character string will overwrite the default.
`id`	a vector of the first three dimensions used in the computation given as **Health**, **Education** and **Living standard**. Can be redefined but must match the indicators and cannot be `NULL`.
`id_add`	a vector of the fourth dimension in the computation given as **Social security**. Can be re-defined but never `NULL`.
`id_add1`	a vector of the fifth dimension in the computation given as **Employment and Income**. Can be re-defined but never `NULL`.
`Echo`	Optional indicating whether the progress note is visible defaults to FALSE.

Value

A list with the following components:

MDI_p	Publication-ready table of the factor and national MDI prepared with summarymodels package. Will not *return* if only national computation is carried out.
MDI	Data frame of the factor and national MDI. Will not *return* if only national computation is carried out.
MDI mean	Data frame of the mean MDI. Will not *return* if only national computation is carried out.
MDI SD	Data frame of the SD of MDI. Will not *return* if only national computation is carried out.
national	Data frame of national MDI with mean and SD.
dimensions	Data frame of the scores for each dimension in the computation.
Score	Data frame of the scores for each indicator in the computation.

References

Alkire, S. & Foster, J. (2011). Counting and Multidimensional Poverty Measurement. Journal of Public Economics 95(7-8): 476–87. https://doi.org/10.1016/j.jpubeco.2010.11.006.

Alkire, S., Foster, J. E., Seth, S., Santos, M. E., Roche, J., & Ballon, P. (2015). Multidimensional poverty measurement and analysis. Oxford University Press.

Alkire, S. & Santos, M. E. (2010). Acute Multidimensional Poverty: A New Index for Developing Countries. Oxford Poverty and Human Development Initiative (OPHI) Working Paper No. 38.

Alkire, S. & Santos, M. E. (2014). Measuring Acute Poverty in the Developing World: Robustness and Scope of the Multidimensional Poverty Index. World Development 59:251-274. https://doi.org/10.1016/j.worlddev.2014.01.026.

Siu Ming Chan & Hung Wong (2024): Measurement and determinants of multidimensional poverty: the case of Hong Kong, Journal of Asian Public Policy, DOI: 10.1080/17516234.2024.2325857

Examples

```
# data from `mpitbR` package
data <- mdpi2
dm <- list(d1 = c("d_nutr","d_cm"),
           d2 = c("d_satt","d_educ"),
           d3 = c("d_elct","d_sani","d_wtr","d_hsg","d_ckfl","d_asst"))
mdi(data, dm, Factor = "region")
```

MLMetrics *Collection of Machine Learning Model Metrics for Easy Reference*

Description

This function estimates over 40 Metrics for assessing the quality of Machine Learning Models. The purpose is to provide a wrapper which brings all the metrics on the table and makes it easier to use them to select a model.

Usage

```
MLMetrics(Observed, yvalue, modeli, K, Name, Form, kutuf, TTy)
```

Arguments

Observed	The Observed data in a data frame format
yvalue	The Response variable of the estimated Model
modeli	The Estimated Model (*Model* = a + bx)
K	The number of variables in the estimated Model to consider
Name	The Name of the Models that need to be specified. They are *ARIMA*, *Values* if the model computes the fitted value without estimation like *Essembles*, *SMOOTH* (smooth.spline), *Logit*, Ensembles based on weight - *EssemWet*, *QUADRATIC* polynomial, *SPLINE* polynomial.
Form	Form of the Model Estimated (LM, ALM, GLM, N-LM, ARDL)
kutuf	Cutoff for the Estimated values (defaults to 0.5 if not specified)
TTy	Type of response variable (Numeric or Response - like *binary*)

Value

A list with the following components:

Absolute Error	of the Model.
Absolute Percent Error	of the Model.
Accuracy	of the Model.
Adjusted R Square	of the Model.
'Akaike's' Information Criterion AIC	of the Model.
Area under the ROC curve (AUC)	of the Model.
Average Precision at k	of the Model.
Bias	of the Model.
Brier score	of the Model.
Classification Error	of the Model.
F1 Score	of the Model.
fScore	of the Model.
GINI Coefficient	of the Model.
kappa statistic	of the Model.
Log Loss	of the Model.

‘Mallow’s‘ cp of the Model.

Matthews Correlation Coefficient
of the Model.

Mean Log Loss of the Model.

Mean Absolute Error
of the Model.

Mean Absolute Percent Error
of the Model.

Mean Average Precision at k
of the Model.

Mean Absolute Scaled Error
of the Model.

Median Absolute Error
of the Model.

Mean Squared Error
of the Model.

Mean Squared Log Error
of the Model.

Model turning point error
of the Model.

Negative Predictive Value
of the Model.

Percent Bias of the Model.

Positive Predictive Value
of the Model.

Precision of the Model.

Predictive Residual Sum of Squares
of the Model.

R Square of the Model.

Relative Absolute Error
of the Model.

Recall of the Model.

Root Mean Squared Error
of the Model.

Root Mean Squared Log Error
of the Model.

Root Relative Squared Error
of the Model.

Relative Squared Error
of the Model.

‘Schwarz’s‘ Bayesian criterion BIC
of the Model.

Sensitivity of the Model.

specificity of the Model.

Squared Error of the Model.

Squared Log Error
: of the Model.

Symmetric Mean Absolute Percentage Error
: of the Model.

Sum of Squared Errors
: of the Model.

True negative rate
: of the Model.

True positive rate
: of the Model.

Examples

```
library(splines)
library(readr)
Model   <- lm(states ~ bs(sequence, knots = c(30, 115)), data = Data)
MLMetrics(Observed = Data, yvalue = Data$states, modeli = Model, K = 2,
 Name = "Linear", Form = "LM", kutuf = 0, TTy = "Number")
```

---

model_factors *Latent Factors Recovery from Variables Loadings*

Description

This function retrieves the latent factors and their variable loadings which can be used as R objects to perform other analysis.

Usage

```
model_factors(data, DATA)
```

Arguments

data	An R object obtained from exploratory factor analysis (EFA) using the fa function in psych package.
DATA	A data.frame, the raw data used to carry out the parallel analysis to obtain data object.

Value

A list with the following components:

Loadings data
: dataframe of the factor loadings from the data.

Factors extracted
: dataframe of retrieved latent factors.

factored data
: dataframe of latent data based the product of recovered latent factors on the raw data.

Factors list
: A list of vectors of individual latent factors recovered from the data. However, to make it usable, the vector should be bind with the names of the variables in the data and the NA removed.

Resilence capacity
: A vector of the resilience capacity.

Examples

```
library(psych)
library(readr)
Data <- Quicksummary
GGn <- names(Data)
GG <- ncol(Data)
GGx <- c(paste0('x0', 1 : 9), paste("x", 10 : ncol(Data), sep = ""))
names(Data) <- GGx
lll <- fa.parallel(Data, fm = "minres", fa = "fa")
dat <- fa(Data, nfactors = lll[["nfact"]], rotate = "varimax",fm = "minres")

model_factors(data = dat, DATA = Data)
```

---

odds_summary *Odds-Based Measures for Binary and Categorical Models*

Description

This function computes odds ratios, percentage changes, and confidence intervals from fitted binary and categorical regression models. It standardizes statistical inference outputs and highlights significant predictors for rapid interpretation. It is a *one-line*, *one-argument* code!

Usage

```
odds_summary(model)
```

Arguments

model
: An R object of estimates from models covered. For now only glm, betareg, mlogit, multimon, mvProbit and polr models are covered.

Value

A list or a data.frame depending on which model. The model must converged otherwise there will be no any return and an error is thrown up

Examples

```
library(Dyn4cast)
library(tidyverse)

counts <- c(18,17,15,20,10,20,25,13,12)
outcome <- gl(3,1,9)
treatment <- gl(3,3)
ddc <- data.frame(treatment, outcome, counts) # showing data
glm.D93 <- glm(counts ~ ., data = ddc, family = poisson())
odds_summary(glm.D93)
```

`Percent` *Attach Per Cent Sign to Data*

Description

This function is a wrapper for easy affixing of the per cent sign (%) to a value or a vector or a data frame of values.

Usage

```
Percent(Data, Type, format = "f", ...)
```

Arguments

`Data`	The Data which the percent sign is to be affixed. The data must be in the raw form for *frame* argument since the per cent value of each cell is calculated before the sign is affixed.
`Type`	The type of data. The options for this argument are *Value* for scalar or vector numeric data or *Frame* for a numeric vector or *data.frame* data. In the case of *frame*, whether vector or columns, the per cent value of each cell is calculated before the per cent sign is affixed.
`format`	The format of the output which is internal and the default is a character factor
`...`	Additional arguments that may be passed to the function

Value

This function returns the result as

`percent`	values with the percentage sign (%) affixed.

Examples

```
Data <- c(1.2, 0.5, 0.103, 7, 0.1501)
Percent(Data = Data, Type = "Frame")  # Value, Frame
Data <- 1.2
Percent(Data = Data, Type = "Value")  # Value, Frame
df <- data.frame(c(A = 2320, 5760, 4800, 2600, 5700, 7800, 3000, 6300, 2400,
10000, 2220, 3740),
B = c(0, 0, 1620, 3600, 1200, 1200, 1200, 4250, 14000, 10000, 1850, 1850),
C = c(3000, 3000, 7800, 5400, 3900, 7800, 1950, 2400, 2400, 7000, 1850, 1850),
D = c(2900, 5760, 3750, 5400, 4095, 3150, 2080, 7800, 1920, 1200, 5000, 1950),
E = c(2900, 2030, 0, 5400, 5760, 1800, 2000, 1950, 1850, 3600, 5200, 5760),
F = c(2800, 5760, 1820, 4340, 7500, 2400, 2300, 1680, 1850, 0, 2800, 8000),
G = c(5760, 4600, 13000, 7800, 6270, 1200, 1440, 8000, 1200, 2025, 4800, 2600),
H = c(2100, 5760, 8250, 3900, 1800, 1200, 4800, 1800, 7800, 2035, 8000, 3000))
Percent(Data = df, Type = "Frame")  # Value, Frame
```

---

plot_mdi *Plots of Multidimensional Index Measures*

Description

Plots of Multidimensional Index Measures

Usage

```
plot_mdi(data, kala, dma, factor = NULL)
```

Arguments

data	Data frame of Multidimensional Index measures which is an object from mdi
kala	color palette with at least 15 colors but must be equal or higher than the number of options in the factor argument
dma	number of Dimensions involved in the computation of Multidimensional Index measures.
factor	the optional grouping factor used in the computation measures. If not supplied only the national plots will be produced irrespective of whether the factor was used in the computation.

Value

A list of the following plots:

Multidimensional index
: plot.

Deprivation Score
: plot.

Adjusted incidence
: plot.

`Intensity` plot.

`Average deprivation among the deprived` plot.

`Contribution of each Dimension` plot.

`combined dimensions` plot.

`national` plot.

`combined dimensions of national` plot.

Examples

```
# data from `mpitbR` package
data <- mdpi2
dm <- list(d1 = c("d_nutr","d_cm"),
           d2 = c("d_satt","d_educ"),
           d3 = c("d_elct","d_sani","d_wtr","d_hsg","d_ckfl","d_asst"))
dp <-  mdi(data, dm, plots = "t")
library(MetBrewer)
kala <- met.brewer("OKeeffe1", 20, type = "continuous")
dma <- 3
plot_mdi(dp$national, kala, dma)
```

`quicksummary` *Quick Formatted Summary of Machine Learning Data*

Description

There is increasing need to make user-friendly and production ready Tables for machine learning data. This function is simplified and quick summary; and the output is a formatted table. This is very handy for those who do not have the time to write codes for user-friendly summaries.

Usage

```
quicksummary(
  x,
  Type,
  Cut = deprecated(),
  Up = deprecated(),
  Down = deprecated(),
  Dig = 2,
  ci = 0.95
)
```

Arguments

x	The data to be summarised. Only numeric data is allowed.
Type	The type of data to be summarised. There are two options here 1 or 2, 1 = Continuous and 2 = Likert-type
Cut	**[Deprecated]**
Up	**[Deprecated]**
Down	**[Deprecated]**
Dig	Number of significant digits which is defaults to 2.
ci	Confidence interval which is defaults to 0.95.

Value

The function returns formatted tables of the Quick summary

Summary	List of two data.frames

Examples

```
library(tidyverse)
# Likert-type data
quicksummary(x = Quicksummary, Type = 2)

# Continuous data
x <- select(linearsystems, 1:6)
quicksummary(x = x, Type = 1)
```

---

relative_likert *Convert Likert Data to Relative Scores and knowledge-based* **Adaptive Capacity**

Description

Adaptive capacity refers to the ability of systems—biological, social, or institutional—to adjust to environmental changes, capitalize on emerging opportunities, and mitigate potential threats in order to preserve essential functions. In the context of climate change, adaptive capacity denotes the competence of social-ecological systems to cope with present variability and prepare for uncertain future conditions.

From a machine learning perspective, adaptive capacity is closely linked to the system's ability to process large-scale, heterogeneous data sources, identify patterns, and support the development of predictive models and adaptive strategies. Key components of adaptive capacity include access to relevant and reliable information, computational infrastructure, financial and human capital, and strong social and institutional networks.

Machine learning can enhance adaptive capacity by enabling dynamic learning from historical and real-time data, improving climate risk assessments, and optimizing adaptation strategies. Moreover, the iterative nature of model refinement and feedback integration mirrors the learning processes

inherent in adaptive systems. Thus, adaptive capacity in this context involves not only the ability to design and implement effective interventions but also to learn from outcomes and continuously update strategies in light of new data and evolving conditions.

This function is for knowledge-based **Adaptive Capacity**. The indices from the various knowledge areas like Awareness, availability, affordability, accessibility, benefits, adequacy, usage, effectiveness, etc can be obtained individually and converted to adaptive capacity. Adaptive capacity based on financial and human capital, strong social and institutional networks can be obtained from `model_factors()`.

Usage

```
relative_likert(
  data,
  Likert = NULL,
  Ranks = NULL,
  Option = "text",
  Echo = FALSE
)
```

Arguments

`data`	Data frame of likert data either in **text** or **scores**.
`Likert`	Vector of likert-type factors in **descending order** as in the data frame which must be given if the data frame is in **text**.
`Ranks`	Optional vector of number of levels which is required if the data frame is in scores rather than text. There are only four choices i.e. 3, 5, 7, 9.
`Option`	Optional vector indicating whether the data frame is in text or scores format. Defaults to **text** if not given.
`Echo`	Optional indicating whether the progress note is visible defaults to FALSE.

Value

A list with the following components:

`Likert-scores` `dataframe` of likert scores

`Relative-likert-scores`
`dataframe` of relative likert scores

`Summary-of-relative-scores`
`dataframe` summary of relative scores based on factors.

`Vector-of-indices`
A vector of indices for **Adaptive Capacity**.

Examples

```
library(readr)
garrett_data <- data.frame(garrett_data)
relative_likert(garrett_data, Ranks = 3, Option = "sccore")
relative_likert(garrett_data, Ranks = 5, Option = "sccore")
```

```
relative_likert(garrett_data, Ranks = 7, Option = "sccore")
relative_likert(garrett_data, Ranks = 9, Option = "sccore")

relative_likert(Quicksummary, Ranks = 5, Option = "sccore")

library(tidyverse)
data_l <- garrett_data %>%
pivot_longer(cols = everything()) %>%
 mutate(value = case_when(value == 5 ~ "Serious constraint",
                          value == 4 ~ "Constraint",
                          value == 3 ~ "Not certain it is a constraint",
                          value == 2 ~ "Not a constraint",
                          value == 1 ~ "Not a serious constraint",
                          .default = "None")) %>%
 group_by(name) %>%
 mutate(row = row_number()) %>%
 pivot_wider(names_from = name, values_from = value) %>%
 select(-row) %>%
 unnest(cols = everything())

 ranking <- c("Serious constraint", "Constraint",
"Not certain it is a constraint", "Not a constraint",
"Not a serious constraint")

 relative_likert(data_l, Likert = ranking)
```

`treatment_model` *Enhanced Estimation of Treatment Effects of Binary Data from Randomized Experiments*

Description

Observational study involves the evaluation of outcomes of participants not randomly assigned treatments or exposures. To be able to assess the effects of the outcome, the participants are matched using propensity scores (PSM). This then enables the determination of the effects of the treatments on those treated against those who were not treated. Most of the earlier functions available for this analysis only enables the determination of the average treatments effects on the treated (ATT) while the other treatment effects are optional. This is where this functions is unique because five different average treatment effects are estimated simultaneously, in spite of the **one line code arguments**. The five treatment effects are:

1. Average treatment effect for the entire (ATE) population
2. Average treatment effect for the treated (ATT) population
3. Average treatment effect for the controlled (ATC) population
4. Average treatment effect for the evenly matched (ATM) population
5. Average treatment effect for the overlap (ATO) population.

There are excellent materials dealing with each of the treatment effects, please see Understanding propensity score weighting

Usage

```
treatment_model(Treatment, x_data)
```

Arguments

Treatment	Vector of binary data (0 = control population, 1 = treated population), the LHS of the model for treatment effects estimation
x_data	Data frame of explanatory variables for the RHS of the model for treatment effects estimation

Value

A list with the following components:

Model	Estimated treatment effects model.
Effect	Data frame of the estimated various treatment effects.
P_score	Vector of estimated propensity scores from the model
Fitted_estimate	Vector of fitted values from the model
Residuals	Residuals of the estimated model
'Experiment plot'	Plot of the propensity scores from the model faceted into Treated and control populations
'ATE plot'	Plot of the average treatment effect for the **entire** population
'ATT plot'	Plot of the average treatment effect for the **treated** population
'ATC plot'	Plot of the average treatment effect for the **controlled** population
'ATM plot'	Plot of the average Treatment effect for the **evenly** population
'ATO plot'	Plot of the average Treatment effect for the **overlap** population
weights	Estimated weights for each of the treatment effects

Examples

```
library(readr)
Treatment = treatments$treatment
data = treatments[, c(2:3)]
treatment_model(Treatment, data)
```

Index

corplot, 2
COVID19 *(*DynamicForecast*)*, 5

Data *(*MLMetrics*)*, 18
data_transform, 3
Dyn4cast-deprecated, 5
DynamicForecast, 5

estimate_plot, 8

formattedcut, 8

garrett_data *(*garrett_ranking*)*, 9
garrett_ranking, 9
garrett_table *(*garrett_ranking*)*, 9
gender, 10

index_construction, 11

Linearsystems, 12
linearsystems *(*Linearsystems*)*, 12

MallowsCp, 15
mdi, 16
mdpi1 *(*mdi*)*, 16
mdpi2 *(*mdi*)*, 16
MLMetrics, 18
Model_factors *(*Dyn4cast-deprecated*)*, 5
model_factors, 21

odds_summary, 22

Percent, 23
plot_mdi, 24

Quicksummary *(*quicksummary*)*, 25
quicksummary, 25

relative_likert, 26

sampling *(*Linearsystems*)*, 12

Transform *(*data_transform*)*, 3
treatment_model, 28
treatments *(*treatment_model*)*, 28