# Package: Dyn4cast (via r-universe)

September 12, 2024

**Title** Dynamic Modeling and Machine Learning Environment

**Version** 11.11.24

**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.

**License** MIT + file LICENSE

**Encoding** UTF-8

**LazyData** true

**Roxygen** list(markdown = TRUE)

**RoxygenNote** 7.3.2

**Imports** stats, forecast, lubridate, splines, Metrics, tidyr, ggplot2, magrittr, formattable, utils, lifecycle, zoo, ModelMetrics, broom, dplyr, modelsummary, caret, corrplot, marginaleffects, tibble, purrr

**Depends** tidyverse, R (>= 2.10)

**Suggests** testthat (>= 3.0.0), rmarkdown, covr, qpdf, readr, kableExtra, knitr, spelling, psych

**Config/testthat/edition** 3

**URL** https://github.com/JobNmadu/Dyn4cast, https://jobnmadu.github.io/Dyn4cast/

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

**VignetteBuilder** knitr

**Language** en-US

**Repository** https://jobnmadu.r-universe.dev

**RemoteUrl** https://github.com/JobNmadu/Dyn4cast

**RemoteRef** HEAD

**RemoteSha** 10212ade8a644ba6ad6d39361394fe1731dc6e81

# Contents



---

`constrainedforecast` *Constrained Forecast of One-sided Integer Response Model*

---

### Description

This function estimates the lower and upper 80% and 95% forecasts of the Model. The final values are within the lower and upper limits of the base data. Used in conjunction with <scaled_logit> and <inv_scaled_logit> functions, they are adapted from Hyndman & Athanasopoulos (2021) and modified for independent use rather than be restricted to be used with a particular package.

### Usage

```
constrainedforecast(Model, lower, upper)
```

### Arguments

| | |
|---|---|
| `Model` | This is the exponential values from the `invscaledlogit` function. |
| `lower` | The lower limit of the forecast |
| `upper` | The upper limit of the forecast |

## Value

A list of forecast values within 80% and 95% confidence band. The values are:

| | |
|---|---|
| `Lower 80%` | Forecast at lower 80% confidence level. |
| `Upper 80%` | Forecast at upper 80% confidence level. |
| `Lower 95%` | Forecast at lower 95% confidence level. |
| `Upper 95%` | Forecast at upper 95% confidence level. |

## Examples

```
library(Dyn4cast)
library(splines)
library(forecast)
lower <- 1
upper <- 37
Model   <- lm(states ~ bs(sequence, knots = c(30, 115)), data = Data)
FitModel <- scaledlogit(x = fitted.values(Model), lower = lower,
 upper = upper)
ForecastModel <- forecast(FitModel, h = length(200))
ForecastValues <- constrainedforecast(Model = ForecastModel, lower, upper)
```

---

# `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 `log` transformation, `2` is for `min-max` 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

`tata_transformed`
: A new `data.frame` containing the transformed values

## Examples

```
library(Dyn4cast)
# 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)
```

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

### Description

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

### Usage

```
estimate_plot(Model, limit)
```

### Arguments

| | |
|---|---|
| Model | 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

```
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:

| | |
|---|---|
| `Data mean table` | Table of data ranked using simple average. |
| `Garrett ranked data` | Table of data ranked using Garrett mean score. |
| `Garrett value` | Table of ranking Garrett values |

### Examples

```
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)
```

---

## invscaledlogit — *Exponential Values after One-Sided Response Integer Variable Forecasting*

---

### Description

This function is used to estimate exponential lower (80% and 95%) and upper (80% and 95%) values from the outcome of the `scaledlogit` function. The exponentiation ensures that the forecast does not go beyond the upper and lower limits of the base data.

### Usage

```
invscaledlogit(x, lower, upper)
```

### Arguments

| | |
|---|---|
| `x` | The forecast values from constrained forecast package. Please specify the appropriate column containing the forecast values. |
| `lower` | Lower limits of the forecast values |
| `upper` | Upper limits of the forecast values |

### Examples

```
x <- 1:35
lower <- 1
upper <- 35
invscaledlogit(x = x, lower = lower, upper = upper)
```

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(Model, y, x, type, Nlevels = 0)
```

### Arguments

- `Model` 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(Model = Model, y = y, x = x, type = Type, Nlevels = 0)
```

---

`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, Model, K, Name, Form, kutuf, TTy)
```

### Arguments

| | |
|---|---|
| `Observed` | The Observed data in a data frame format |
| `yvalue` | The Response variable of the estimated Model |
| `Model` | 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.

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

---

`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 because for frame argument, the per cent value of each cell is calculated before the sign is affixed. |
| `Type` | The type of data. The default arguments are *Value* for single numeric data of *Frame* for a numeric vector or data frame data. In the case of vector or data frame, 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
Percent(Data = sample, Type = "Frame")  # Value, Frame
```

---

## `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 a simplified 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, Up, Down, 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` | The cut-off point for Likert-type data |
| `Up` | The top Likert-type scale, for example, `Agree`, `Constraints` etc which would appear in the remark column. |

| | |
|---|---|
| `Down` | The lower Likert-type scale, for example, `Disagree`, `Not a Constraint` etc which would appear in the remark column. |
| `ci` | Confidence interval which is defaults to 0.95. |

### Value

The function returns a formatted Table of the Quick summary

| | |
|---|---|
| `ANS` | The formatted Table of the summary |

### Examples

```
# Likert-type data
Up <- "Constraint"
Down <- "Not a constraint"
quicksummary(x = Quicksummary, Type = 2, Cut = 2.60, Up = Up, Down = Down)

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

---

## `scaledlogit` *Scale Parameter for Integer Modeling and Forecast*

---

### Description

This function is a wrapper for scaling the fitted (predicted) values of a one-sided (positive or negative only) integer response variable of supported models. The scaling involves some log transformation of the fitted (predicted) values.

### Usage

```
scaledlogit(x, lower, upper)
```

### Arguments

| | |
|---|---|
| `x` | The parameter to be scaled, which is the fitted values from supported models. The scaled parameter is used mainly for constrained forecasting of a response variable *positive (0 - inf) or negative (-inf - 0)*. The scaling involves log transformation of the parameter |
| `lower` | Integer or variable representing the lower limit for the scaling (-inf or 0) |
| `upper` | Integer or variable representing the upper limit for the scaling (0 or inf) |

### Examples

```
library(Dyn4cast)
library(splines)
lower <- 1
upper <- 37
Model   <- lm(states ~ bs(sequence, knots = c(30, 115)), data = Data)
scaledlogit(x = fitted.values(Model), lower = lower,
 upper = upper)
```

---

## `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 excellent materials dealing with each of the treatment effects, please see

### Usage

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

### Arguments

| | |
|---|---|
| `Treatment` | Vector of binary data (0, 1) LHS for the treatment effects estimation |
| `x_data` | Data frame of explanatory variables for the RHS of the 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

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

# Index