Package ‘linea’

August 19, 2022

Title Linear Regression Interface

Version 0.0.3

Description An interface to accelerate linear regression (Ordinary least squares) modelling, which allows users to build models quickly, while automatically generating interactive visualizations of the results.

Non-linear models specification (e.g. \( y = b_1 x_1 + b_2 \log(x_2) \)) can be easily constructed using user-defined transformations.

Functions for testing wide ranges of model specifications (e.g. \( y = b \log(x,10) \), \( y = b \log(x,20) \), ...), all at once, are also available.

Finally, models can be imported and exported as Excel files where all the information necessary for re-running the models is stored in separate sheets.

License GPL-3

Encoding UTF-8

Imports anytime, dplyr, ggplot2, gtrendsR, lubridate, magrittr, methods, openxlsx, plotly, purrr, RColorBrewer, readr, readxl, reshape2, sjmisc, stringr, tibble, tidyR, tidyverse, tis, zoo

RoxygenNote 7.2.0

Suggests testthat (>= 3.0.0)

Config/testthat/edition 3

NeedsCompilation no

Author Claudio Paladini [aut, cre] (<https://orcid.org/0000-0002-5551-1380>)

Maintainer Claudio Paladini <claudio.paladini@bath.edu>

Repository CRAN

Date/Publication 2022-08-19 08:20:02 UTC

R topics documented:

  acf_chart ................................................................. 3
  apply_normalisation .................................................. 3
<table>
<thead>
<tr>
<th>R topics documented:</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>apply_transformation</td>
<td>4</td>
</tr>
<tr>
<td>build_formula</td>
<td>5</td>
</tr>
<tr>
<td>build_model_table</td>
<td>6</td>
</tr>
<tr>
<td>check_model_file</td>
<td>6</td>
</tr>
<tr>
<td>check_trans_df</td>
<td>7</td>
</tr>
<tr>
<td>check_ts</td>
<td>8</td>
</tr>
<tr>
<td>color_palette</td>
<td>8</td>
</tr>
<tr>
<td>decay</td>
<td>9</td>
</tr>
<tr>
<td>decomp</td>
<td>9</td>
</tr>
<tr>
<td>decomp_chart</td>
<td>10</td>
</tr>
<tr>
<td>default_trans_df</td>
<td>11</td>
</tr>
<tr>
<td>diminish</td>
<td>12</td>
</tr>
<tr>
<td>export_model</td>
<td>13</td>
</tr>
<tr>
<td>first_last_dates</td>
<td>13</td>
</tr>
<tr>
<td>fit_chart</td>
<td>14</td>
</tr>
<tr>
<td>get_seasonality</td>
<td>15</td>
</tr>
<tr>
<td>get_variable_t</td>
<td>16</td>
</tr>
<tr>
<td>get_vector_from_str</td>
<td>17</td>
</tr>
<tr>
<td>gt_f</td>
<td>17</td>
</tr>
<tr>
<td>heteroskedasticity_chart</td>
<td>18</td>
</tr>
<tr>
<td>hill_function</td>
<td>19</td>
</tr>
<tr>
<td>import_model</td>
<td>20</td>
</tr>
<tr>
<td>is_daily</td>
<td>20</td>
</tr>
<tr>
<td>is_uniform_ts</td>
<td>21</td>
</tr>
<tr>
<td>is_weekly</td>
<td>21</td>
</tr>
<tr>
<td>lag</td>
<td>22</td>
</tr>
<tr>
<td>ma</td>
<td>22</td>
</tr>
<tr>
<td>read_xcsv</td>
<td>23</td>
</tr>
<tr>
<td>resid_hist_chart</td>
<td>24</td>
</tr>
<tr>
<td>response_curves</td>
<td>25</td>
</tr>
<tr>
<td>re_run_model</td>
<td>26</td>
</tr>
<tr>
<td>run_combo_model</td>
<td>27</td>
</tr>
<tr>
<td>run_model</td>
<td>29</td>
</tr>
<tr>
<td>run_text</td>
<td>30</td>
</tr>
<tr>
<td>trans_test</td>
<td>31</td>
</tr>
<tr>
<td>TRY</td>
<td>32</td>
</tr>
<tr>
<td>vapply_transformation</td>
<td>32</td>
</tr>
<tr>
<td>what_combo</td>
<td>33</td>
</tr>
<tr>
<td>what_next</td>
<td>34</td>
</tr>
<tr>
<td>what_trans</td>
<td>35</td>
</tr>
</tbody>
</table>
acf_chart

Description

Bar chart of autocorrelation function

Usage

acf_chart(
    model = NULL,
    decomp_list,
    pool = NULL,
    color = "black",
    verbose = FALSE
)

Arguments

model     Model object
decomp_list list object generated by the decomping function.
pool      string specifying a group within the pool column to be filtered
color     string specifying bar color
verbose   A boolean to specify whether to print warnings

Details

A bar chart meant to assess the correlation of the residuals with lagged versions of themselves.

Value

a plotly bar chart of the model’s ACF

apply_normalisation

Description

Normalise data based on pool mean
apply_transformation

Usage

apply_normalisation(
  raw_data = NULL,
  meta_data = NULL,
  model_table = NULL,
  dv = NULL,
  verbose = FALSE
)

Arguments

raw_data data.frame containing data for analysis
meta_data data.frame mapping variable names to their roles (i.e. POOL)
model_table data.frame as created in the build_model_table function
dv string specifying the dependent variable name
verbose A boolean to specify whether to print warnings

Details

Normalise data by dividing all values in each pool by that pool’s mean

Value

list containing a tibble of normalised data and a tibble of pool means

apply_transformation    #' apply_transformation

Description

Transform data based on model table

Usage

apply_transformation(
  data = NULL,
  model_table = NULL,
  trans_df = NULL,
  meta_data = NULL,
  verbose = FALSE
)
**Arguments**

- **data** data.frame containing data for analysis
- **model_table** data.frame as created in the `build_model_table` function
- **trans_df** data.frame defining the non-linear transformations to apply
- **meta_data** data.frame mapping variable names to their roles (i.e. POOL)
- **verbose** A boolean to specify whether to print warnings

**Details**

Transform data based on the model table by applying the transformation functions (e.g. decay, diminish, lag, and ma) with the specified parameters to the respective variable.

**Value**

tibble of raw_data with added transformed variables

---

**Description**

Build a formula (e.g. y ~ x1 + x2)

**Usage**

```r
build_formula(dv, ivs, model_table = NULL)
```

**Arguments**

- **dv** string of dependent variable name
- **ivs** character vector of independent variable names
- **model_table** tibble/data.frame as created in the `build_model_table` function

**Details**

Build a formula (e.g. y ~ x1 + x2) based on a dependent variable name and a model table or independent variables' names' vector.

**Value**

- a formula object
build_model_table

Description
Build an empty model table

Usage
build_model_table(ivs, trans_df = NULL, ts = TRUE)

Arguments
ivs character vector of variables
trans_df data.frame defining the non-linear transformations to apply
ts boolean to specify if time-series or not

Details
Build an empty table as a template to capture model predictors, and transformation parameters

Value
tibble of model table

Examples
build_model_table(c('x1','x2'))
build_model_table(colnames(mtcars))

---

check_model_file

description
Check the excel model file

Usage
check_model_file(model_file, verbose = FALSE, return_list = TRUE)

Arguments
model_file File path to the model file as string
verbose Boolean to specify whether to return the checked file
return_list Boolean to specify whether to print warnings
check_trans_df

Details

Check the model file contains all the needed sheets.

Value

Messages and the checked file

default_trans_df() %>% check_trans_df()
check_ts

Description
Check time series dataframe

Usage
check_ts(data, date_col, allow_non_num = TRUE, verbose = FALSE)

Arguments
data The dataframe containing the column specified
date_col The date column name as a string
allow_non_num A boolean to specify whether to include only date and numeric columns
verbose A boolean to specify whether to print warnings

Details
Check if dataframe contains specified date column and that its data-type

Value
checked data.frame

color_palette
color_palette

Description
A pre-loaded color palette

Usage
color_palette()

Details
A pre-loaded color palette that can be used in charting functions

Value
character vector of colors in hexadecimal notation
Description

Time series decay

Usage

decay(v, decay)

Arguments

v numeric vector
decay The rate of decay as a numeric decimal

Details

Applies the specified decay on the input vector, v

Value

The transformed vector v

Examples

decay(c(1,0,0,1,0,0,0,2), 0.5)
decay(c(1,0,0,1,0,0,0,2), 0.1)

decomping
decomping

Description

Variable decomposition of linear regression

Usage

decomping(
    model = NULL,
    de_normalise = TRUE,
    raw_data = NULL,
    categories = NULL,
    id_var = NULL,
    verbose = FALSE
)

Arguments

model       Model object
de_normalise  A boolean to specify whether to apply the normalisation
raw_data    data.frame containing data for analysis
categories  data.frame mapping variables to groups
id_var      string of id variable name (e.g. date)
verbose     A boolean to specify whether to print warnings

Details

Calculates the decomposition of the independent variables based on an input model object. This can be expanded by leveraging id variables (e.g. date) and categories (i.e. groups of variables).

Value

a list of 3 data.frame's representing the variable and category decomposition, and the fitted values.

Examples

run_model(data = mtcars, dv = 'mpg', ivs = c('wt', 'cyl', 'disp'), decompose=FALSE) %>% decomping()
**Arguments**

- **model**
  - Model object
- **decomp_list**
  - list object generated by the decomposing function.
- **pool**
  - string specifying a group within the pool column to be filtered
- **colors**
  - character vector of colors in hexadecimal notation
- **variable_decomp**
  - boolean specifying whether the chart should be based on the variable_decomp or the category_decomp from the decomposing function.
- **verbose**
  - A boolean to specify whether to print warnings

**Details**

Plot the variable, or category, decomposition as stacked bars over the id variable which can be supplied to the decomposing function.

**Value**

a plotly bar chart of the model's decomposition

---

**default_trans_df**

**Usage**

default_trans_df(ts = TRUE)

**Arguments**

- **ts**
  - boolean to specify if time-series or not

**Details**

Generate the default trans_df data frame with functions from linea:

- decay
- hill_function
- ma
- lag

**Value**

data.frame of trans_df
Examples

```r
default_trans_df()
default_trans_df(ts = TRUE)
```

diminish(1,0,0,0,1,0,0,0,2, 1)
diminish(c(1,0,0,0,1,0,0,0,2), 1, FALSE)

### Description

Negative exponential (Diminish returns)

### Usage

```r
diminish(v, m, abs = TRUE)
```

### Arguments

- **v**: Numeric vector
- **m**: Scale of diminishing as a numeric integer or decimal
- **abs**: Boolean to determine if diminishing scale `m` is a percentage or absolute value

### Details

Applies the negative exponential \(1 - \exp(-x/m)\) on the input vector, \(v\)

### Value

The transformed vector \(v\)

### Examples

```r
diminish(c(1,0,0,0,1,0,0,0,2), 1)
diminish(c(1,0,0,0,1,0,0,0,2), 1, FALSE)
```
Description

Export model to excel file

Usage

\[
\text{export_model}(\text{model}, \text{path} = \text{"model.xlsx"}, \text{overwrite} = \text{FALSE})
\]

Arguments

- \text{model} Model object
- \text{path} File path to the model file as string
- \text{overwrite} Boolean to specify whether to overwrite the file in the specified path

Details

Export a model to an excel file

Description

Check if a time-series is uniform

Usage

\[
\text{first_last_dates}(\text{date_values}, \text{date_type})
\]

Arguments

- \text{date_values} a date-type or numeric vector
- \text{date_type} The date column type as either of the following strings: 'weekly starting', 'weekly ending', 'daily'

Details

Check if a time-series is uniform, where the step (e.g. days(1), weeks(7)) is consistent

Value

list of first and last daily dates
Description

Dependent Variable, Predictions and Residuals Line Chart

Usage

```r
code
fit_chart(
  model = NULL,
  decomp_list = NULL,
  pool = NULL,
  verbose = FALSE,
  colors = NULL
)
```

Arguments

- **model**: Model object
- **decomp_list**: list object generated by the decomping function.
- **pool**: string specifying a group within the pool column to be filtered
- **verbose**: A boolean to specify whether to print warnings
- **colors**: character vector of colors in hexadecimal notation

Details

Plot the dependent variable, predictions and Residuals as a line chart over the id variable which can be supplied to the decomping function.

Value

A plotly line chart of the model’s prediction and actual

Examples

```r
code
run_model(data = mtcars, dv = 'mpg', ivs = 'cyl') %>% fit_chart()
```
get_seasonality

Description

generate seasonality variables

Usage

get_seasonality(
  data,
  date_col_name,
  date_type = "weekly starting",
  verbose = FALSE,
  keep_dup = FALSE,
  pool_var = NULL
)

Arguments

data: data.frame containing data for analysis

date_col_name: The date column name as a string

date_type: The date column type as either of the following strings: ‘weekly starting’, ‘weekly ending’, ‘daily’

verbose: A boolean to specify whether to print warnings

keep_dup: A boolean to specify whether to keep duplicate columns between seasonal and data

pool_var: The pool (group) column name as a string (e.g. ‘country’)

Details

generate seasonality variables from a data.frame containing a date-type variable.

Value

data.frame with added variables

Examples

read_xcsv("https://raw.githubusercontent.com/paladinic/data/main/ecomm_data.csv") %>%
  get_seasonality(date_col_name = 'date')
Description

Generate more specific variable names

Usage

```r
get_variable_t(
  model_table,
  excl_intercept = TRUE,
  excl_dup = TRUE,
  excl_blanks = FALSE,
  trans_df = NULL
)
```

Arguments

- `model_table` tibble/data.frame as created in the `build_model_table` function
- `excl_intercept` Boolean to specify whether to drop the "(Intercept)" row of the model table
- `excl_dup` Boolean to specify whether to drop the duplicated rows of the model table
- `excl_blanks` Boolean to specify whether to drop the blank rows of the model
- `trans_df` data.frame defining the non-linear transformations

Details

Generate variable names that capture the transformations applied

Value

tibble of model table with added `variable_t` column

Examples

```r
build_model_table(colnames(mtcars)) %>%
  get_variable_t()
```
**get_vector_from_str**

**Description**

get a numeric vector from string (e.g. '1,2,3')

**Usage**

```r
get_vector_from_str(string, sep = ",", zero = TRUE)
```

**Arguments**

- `string`: a string containing separated by a separator
- `sep`: a string representing the separator
- `zero`: a boolean defining whether the output vector must contain a zero

**Details**

Get a numeric vector from string containing numbers separated by a separator (e.g. '1,2,3').

**Value**

numeric vector from the string

**Examples**

```r
get_vector_from_str('1,2,3')
get_vector_from_str('0.1')
get_vector_from_str('1;2;3', sep=';')
```

---

**gt_f**

**Description**

Normalise data based on pool mean

**Usage**

```r
gt_f(
  data,
  kw,
  date_col = "date",
  date_type = "weekly starting",
  geo = "all",
  append = FALSE
)
```
Arguments

data data.frame containing data for analysis
kw a string of the search keyword
date_col a string specifying the date column name
date_type The date column type as either of the following strings: 'weekly starting', 'weekly ending', 'daily'
geo a string specifying the country code of the search found in countrycode::codelist
append a boolean specifying whether to return the original data.frame as well as the added column

Details

Normalise data by dividing all values in each pool by that pool’s mean

Value

data.frame of the original data with the added google trend column

Examples

data = read_xcsv("https://raw.githubusercontent.com/paladinic/data/main/ecomm_data.csv") %>%
gt_f(kw = 'covid') %>%
gt_f(kw = 'bitcoin')

differentiate_chart

differentiate_chart

Description

Scatter of Residuals over dependent Variable

Usage

heteroskedasticity_chart(
  model = NULL,
  decomp_list = NULL,
  pool = NULL,
  color = "black",
  verbose = FALSE
)
**Arguments**

- **model**: Model object
- **decomp_list**: list object generated by the decomping function.
- **pool**: string specifying a group within the pool column to be filtered
- **color**: string specifying bar color
- **verbose**: A boolean to specify whether to print warnings

**Details**

Plot a scatter chart of residuals over the dependent variable. This is meant to assess the consistency of the residuals’ variance across the dependent variable.

**Value**

A plotly scatter chart of the model’s dependent variable over residuals

---

**Description**

Hill Function

**Usage**

```r
hill_function(v, k = 1, m = 5, abs = TRUE)
```

**Arguments**

- **v**: Numeric vector
- **k**: Numeric integer or decimal
- **m**: Numeric integer or decimal
- **abs**: Boolean to determine if diminishing scale m is a percentage or absolute value

**Details**

Applies the Hill Function 1 - (k^m)(k^m + v^m) on the input vector, v

**Value**

The transformed vector v

**Examples**

```r
hill_function(c(1,0,0,0,10,0,0,0,20), k=10)
hill_function(c(1,0,0,0,10,0,0,0,20), k=0.1, abs = FALSE)
hill_function(c(1,0,0,0,10,0,0,0,20), k=10, m = 3)
```
Description
Import and run the excel model file

Usage
import_model(path, verbose = FALSE)

Arguments
path File path to the model file as string
verbose Boolean to specify whether to return the checked file

Details
Import and run the excel model file using check_model_file and run_model functions

Value
model object

Description
Check if a time-series is daily

Usage
is_daily(dates)

Arguments
dates a date-type or numeric vector

Details
Check if a time-series is daily return boolean

Value
boolean to specify whether the time series has a daily frequency
is_uniform_ts

Description
Check if a time-series is uniform

Usage
is_uniform_ts(dates)

Arguments
dates a date-type or numeric vector

Details
Check if a time-series is uniform, where the step (e.g. days(1), weeks(7)) is consistent

Value
boolean to specify whether the time series is uniform

is_weekly

Description
Check if a time-series is weekly

Usage
is_weekly(dates)

Arguments
dates a date-type or numeric vector

Details
Check if a time-series is weekly return boolean

Value
boolean to specify whether the time series has a weekly frequency
### lag

**Description**

Lag by 1

**Usage**

```r
lag(v, l, strategy = "extremes")
```

**Arguments**

- `v`: Numeric vector
- `l`: Lag as an integer
- `strategy`: String to determine the NAs generated by the lag should be filled with zeros or with the extremities' values

**Details**

Applies a lag of 1 on the input vector, v

**Value**

The lagged vector v

**Examples**

```r
lag(c(1,0,0,0,1,0,0,0,2), 1)
lag(c(1,0,0,0,1,0,0,0,2), -2)
lag(c(1,0,0,0,1,0,0,0,2), -2, strategy = "zero")
```

---

### ma

**Description**

Moving Average

**Usage**

```r
ma(v, width, align = "center", zero = TRUE)
```
Arguments

- **v**  
  Numeric vector
- **width**  
  Width of moving average window as an integer, v
- **align**  
  Either string "center", "left", or "right"
- **zero**  
  Boolean to determine the NAs generated by the moving average should be filled with zeros or with the vector's mean.

Details

Applies a moving average on the input vector. The type of moving average is defined by the argument align.

Value

The modified vector v

Examples

- `ma(c(1,0,0,0,1,0,0,0,2), 3)`
- `ma(c(1,0,0,0,1,0,0,0,2), 3, align = "right")`
- `ma(c(1,0,0,0,1,0,0,0,2), 3, zero = FALSE)`

---

**Description**

Reads flat files: either csv or excel

**Usage**

`read_xcsv(file, sheet = NULL, verbose = FALSE)`

**Arguments**

- **file**  
  The file path that points to either a csv or excel file ending in csv, xls, xlsx, or xlsm
- **sheet**  
  For excel files, the sheet name as a string or number as an integer
- **verbose**  
  A boolean to specify whether to print warnings

**Details**

Reads csv or excel files with the suffixes csv, xls, xlsx, xlsm

**Value**

data.frame from flatfile
Examples

```r
read_xcsv("https://raw.githubusercontent.com/paladinic/data/main/ecomm_data.csv")
```

---

Description

Histogram of Model Residuals

Usage

```r
resid_hist_chart(
  model = NULL,
  decomp_list = NULL,
  pool = NULL,
  color = "black",
  verbose = FALSE
)
```

Arguments

- `model`: Model object
- `decomp_list`: list object generated by the `decomping` function. 
- `pool`: string specifying a group within the pool column to be filtered
- `color`: string specifying bar color
- `verbose`: A boolean to specify whether to print warnings

Details

Plot a histogram to visualise the distribution of residuals. This is meant to assess the residual distribution’s normality.

Value

- a `plotly` histogram of the model’s residuals
Description

Line chart of variable response curves

Usage

```r
response_curves(
  model,
  x_min = NULL,
  x_max = NULL,
  y_min = NULL,
  y_max = NULL,
  interval = NULL,
  trans_only = FALSE,
  colors = color_palette(),
  plotly = TRUE,
  verbose = FALSE,
  table = FALSE,
  add_intercept = FALSE,
  points = FALSE
)
```

Arguments

- **model**: Model object
- **x_min**: number specifying horizontal axis min
- **x_max**: number specifying horizontal axis max
- **y_min**: number specifying vertical axis min
- **y_max**: number specifying vertical axis max
- **interval**: number specifying interval between points of the curve
- **trans_only**: a boolean specifying whether to display non-linear only \( y = b \times \text{dim}_\text{rest}(x) \)
- **colors**: character vector of colors in hexadecimal notation
- **plotly**: A boolean to specify whether to include use ggplot over plotly
- **verbose**: A boolean to specify whether to print warnings
- **table**: A boolean to specify whether to return a data.frame of the response curves
- **add_intercept**: A boolean to specify whether to include the intercept when calculating the curves
- **points**: A boolean to specify whether to include the points from the data on the curve
Details

Line chart of variable response curves visualising the relationship of each independent variable with the dependent variable

Value

a plotly line chart of the model’s response curves

Examples

```r
model = run_model(data = mtcars, dv = 'mpg', ivs = c('disp'))
model %>%
  response_curves()
model = run_model(data = mtcars, dv = 'mpg', ivs = c('wt', 'cyl', 'disp'))
model %>%
  response_curves()
run_model(data = scale(mtcars) %>%
  data.frame(),
  dv = 'mpg',
  ivs = c('wt', 'cyl', 'disp')) %>%
  response_curves()
```

Description

Re-run a linear regression model

Usage

```r
re_run_model(
  model,
  data = NULL,
  dv = NULL,
  ivs = NULL,
  trans_df = NULL,
  meta_data = NULL,
  id_var = NULL,
  model_table = NULL,
  normalise_by_pool = FALSE,
  verbose = FALSE,
  decompose = TRUE
)
```
**Arguments**

- **model**
  - the model object used as the starting point of the re-run
- **data**
  - data.frame containing variables included in the model specification
- **dv**
  - string of the dependent variable name
- **ivs**
  - character vector of the independent variables names
- **trans_df**
  - data.frame defining the non-linear transformations to apply
- **meta_data**
  - data.frame mapping variable names to their roles (i.e. POOL)
- **id_var**
  - string of id variable name (e.g. date)
- **model_table**
  - data.frame as created in the `build_model_table` function
- **normalise_by_pool**
  - A boolean to specify whether to apply the normalisation
- **verbose**
  - A boolean to specify whether to print warnings
- **decompose**
  - A boolean to specify whether to generate the model decomposition

**Details**

Re-run a linear regression model using the function output of running `linea::run_model`.

**Value**

Model object

**Examples**

```r
model = run_model(
  data = read_xcsv("https://raw.githubusercontent.com/paladinic/data/main/ecomm_data.csv"),
  dv = 'ecommerce',
  ivs = c('christmas','black.friday'))
re_run_model(model,ivs = c('disp','cyl','wt'))
```

---

**Description**

generate the mode object from the output of `linea::what_combo()`

**Usage**

```r
run_combo_model(combos, model, model_null = FALSE, results_row = 1)
```
Arguments

- **combos**: output of `linea::what_combo()` function
- **model**: Model object
- **model_null**: a boolean to specify whether the model should be used as starting point
- **results_row**: numeric value of the model (i.e. row from `what_combo()$results`) to run

Details

Generate the mode object from the output of `linea::what_combo()` Using the specs from the output of `linea::what_combo()` a new model is run.

Value

list of two `data.frame` mapping variables’ transformations to the respective model’s statistics.

Examples

```r
# using a model object
data = read_xcsv("https://raw.githubusercontent.com/paladinic/data/main/ecomm_data.csv")
dv = 'ecommerce'
ivs = c('christmas','black.friday')

trans_df = data.frame(
  name = c('diminish', 'decay', 'hill', 'exp'),
  ts = c(FALSE,TRUE,FALSE,FALSE),
  func = c(
    'linea::diminish(x,a)',
    'linea::decay(x,a)',
    "linea::hill_function(x,a,b,c)",
    '(x^a)'
  ),
  order = 1:4
)

trans_df = dplyr::mutate(offline_media = dplyr::if_else(condition = name == 'hill',
  '(1,50),(1),(1,100)',
  offline_media))
trans_df = dplyr::mutate(offline_media = dplyr::if_else(condition = name == 'decay',
  '.1,.7',
  offline_media))

model = run_model(data = data,dv = dv,ivs = ivs, trans_df = trans_df)
combos = what_combo(model = model,trans_df = trans_df)
```
run_model

combos %>%
  run_combo_model(model,1)

run_model run_model

Description

Run a linear regression model

Usage

run_model(
  data = NULL,
  dv = NULL,
  ivs = NULL,
  trans_df = NULL,
  meta_data = NULL,
  id_var = NULL,
  model_table = NULL,
  verbose = FALSE,
  normalise_by_pool = FALSE,
  save_raw_data = TRUE,
  decompose = TRUE,
  categories = NULL
)

Arguments

data data.frame containing variables included in the model specification
dv string of the dependent variable name
ivs character vector of the independent variables names
trans_df data.frame defining the non-linear transformations to apply
meta_data data.frame mapping variable names to their roles (i.e. POOL)
id_var string of id variable name (e.g. date)
model_table data.frame as created in the build_model_table function
verbose A boolean to specify whether to print warnings
normalise_by_pool A boolean to specify whether to apply the normalisation
save_raw_data A boolean to specify whether to save all input data variables to the model object
decompose A boolean to specify whether to generate the model decomposition
categories data.frame mapping variables to groups
Details

Run a linear regression model that captures the transformations applied in the `model_table` and the normalisation described in the `meta_data`. A model can be run also by only supplying a dependent variable name `dv`, a vector of independent variable names `ivs`, and the data that contains these.

Value

Model object

Examples

```r
trans_df = data.frame(
   name = c('diminish', 'decay', 'hill', 'exp'),
   func = c(
   'linea::diminish(x,a)',
   'linea::decay(x,a)',
   "linea::hill_function(x,a,b,c)",
   '(x^a)'
   ),
   order = 1:4
)
data = read_xcsv("https://raw.githubusercontent.com/paladinic/data/main/ecomm_data.csv")
dv = 'ecommerce'
ivs = c('christmas','black.friday')

run_model(data = data,
   dv = dv,
   ivs = ivs,
   trans_df = trans_df)
run_model(data = mtcars,dv = 'mpg',ivs = c('disp','cyl'))
```

Description

run text as R code

Usage

```r
run_text(text, env)
```
Arguments
text  Code to run as string
env   environment object specifying the environment

Details
Run a text string as R code

Value
text expression output

trans_test  trans_tester

Description
Transformation Tester

Usage
trans_test(f, p = NULL)

Arguments
f  the function to test
p  an optional list of parameters and values

Details
Tests a mathematical transformation function for errors. The function must accept an input vector v as well as, optionally, additional parameters.

Value
a character vector of messages

Examples
trans_test(log)
**TRY**

**Description**

TRY or NULL

**Usage**

```r
TRY(x, verbose = FALSE)
```

**Arguments**

- `x`: The expression to try
- `verbose`: boolean to specify whether to print the error

**Details**

A tryCatch implementation that returns NULL when an error is thrown

**Value**

expression output or NULL

---

**vapply_transformation**

**Description**

Transform vector based on transformation parameters

**Usage**

```r
vapply_transformation(v, trans_df = NULL, verbose = FALSE)
```

**Arguments**

- `v`: Numeric vector to be transformed
- `trans_df`: data.frame defining the non-linear transformations to apply
- `verbose`: A boolean to specify whether to print warnings

**Details**

Transform vector based on the transformation parameters of the `trans_df`

**Value**

Transformed numeric vector
**Description**

run models across combinations of transformations and variables

**Usage**

```r
what_combo(
  model = NULL,
  trans_df = NULL,
  data = NULL,
  dv = NULL,
  r2_diff = TRUE,
  return_model_objects = FALSE,
  verbose = FALSE
)
```

**Arguments**

- `model`: Model object
- `trans_df`: data.frame containing the transformations, variables and parameter values
- `data`: data.frame containing data from analysis
- `dv`: string specifying the dependent variable name
- `r2_diff`: A boolean to determine whether to add a column to compare new and original model R2
- `return_model_objects`: A boolean to specify whether to return model objects
- `verbose`: A boolean to specify whether to print warnings

**Details**

Run a separate model for each combination of transformations specified. The combinations are defined by the possible transformation parameters specified in the `trans_df`. Then, for each model run, return that model’s fit and the variables’ statistics.

**Value**

list of two data.frame mapping variables’ transformations to the respective model’s statistics.
Examples

```r
# using a model object
data = read_xcsv("https://raw.githubusercontent.com/paladinic/data/main/ecomm_data.csv")
dv = 'ecommerce'
ivs = c('christmas','black.friday')

trans_df = data.frame(
  name = c('diminish', 'decay', 'hill', 'exp'),
ts = c(FALSE,TRUE,TRUE,FALSE),
func = c(
  'linea::diminish(x,a)',
  'linea::decay(x,a)',
  "linea::hill_function(x,a,b,c)",
  '(x^a)'
),
order = 1:4
) %>%
dplyr::mutate(offline_media = dplyr::if_else(condition = name == 'hill',
  '(1,50),(1),(1,100)',
  offline_media)) %>%
dplyr::mutate(offline_media = dplyr::if_else(condition = name == 'decay',
  '.1,.7',
  offline_media)) %>%
dplyr::mutate(online_media = dplyr::if_else(condition = name == 'decay',
  '.1,.7',
  offline_media)) %>%
dplyr::mutate(promo = '')

model = run_model(data = data,dv = dv,ivs = ivs, trans_df = trans_df)

compos = what_combo(model = model,trans_df = trans_df)

#using the trans_df, data, and dv
what_combo(trans_df = trans_df, data = data, dv = dv)
```

---

Description

run model with other variables from the data

Usage

```r
what_next(model = NULL, data = NULL, verbose = FALSE, r2_diff = TRUE)
```
what_trans

Arguments

model: Model object
data: data.frame containing data for analysis
verbose: A boolean to specify whether to print warnings
r2_diff: A boolean to determine whether to add a column to compare new and original model R2

details

Run a separate model for each numeric variable in the data provided. Then, for each model run, return that model’s fit and the variables’ statistics.

Value

data.frame mapping variables' to the respective model’s statistics.

Examples

```r
run_model(data = mtcars, dv = 'mpg', ivs = c('disp', 'cyl')) %>% what_next()
```

Description

run models with additional (transformed) variables from the data

Usage

```r
what_trans(
  model = NULL,
  trans_df = NULL,
  variable = NULL,
  data = NULL,
  r2_diff = TRUE,
  verbose = FALSE
)
```

Arguments

model: Model object
trans_df: data.frame
variable: string or character vector of variable names contained in raw_data data.frame
data: data.frame containing data from analysis
r2_diff: A boolean to determine whether to add a column to compare new and original model R2
verbose: A boolean to specify whether to print warnings
Details

Run a separate model for each combination of transformations specified. Then, for each model run, return that model’s fit and the variables’ statistics.

Value

data.frame mapping variables’ transformations to the respective model’s statistics.

Examples

```r
model = run_model(data = mtcars, dv = 'mpg', ivs = c('disp', 'cyl'))

trans_df = data.frame(
  name = c('diminish', 'decay', 'lag', 'ma', 'log', 'hill', 'sin', 'exp'),
  ts = c(FALSE, TRUE, TRUE, TRUE, FALSE, FALSE, FALSE, FALSE),
  func = c('linea::diminish(x,a)',
    'linea::decay(x,a)',
    'linea::lag(x,a)',
    'linea::ma(x,a)',
    'log(x,a)',
    "linea::hill_function(x,a,b,c)",
    'sin(x*a)',
    '(x^a)'),
  order = 1:8)

variable = 'cyl'

model %>%
  what_trans(variable = variable, trans_df = trans_df)
```
Index

acf_chart, 3
apply_normalisation, 3
apply_transformation, 4
build_formula, 5
build_model_table, 6
check_model_file, 6
check_trans_df, 7
check_ts, 8
color_palette, 8
decay, 9
decomps_chart, 10
decomping, 9
default_trans_df, 11
diminish, 12
export_model, 13
first_last_dates, 13
fit_chart, 14
get_seasonality, 15
gt_6, 16
gt_vector_from_str, 17
gt_f, 17
heteroskedasticity_chart, 18
hill_function, 19
import_model, 20
is_daily, 20
is_uniform_ts, 21
is_weekly, 21
lag, 22
ma, 22
re_run_model, 26
read_xcsv, 23
resid_hist_chart, 24
response_curves, 25
run_combo_model, 27
run_model, 29
run_text, 30
trans_test, 31
TRY, 32
vapply_transformation, 32
what_combo, 33
what_next, 34
what_trans, 35