Package 'garchmodels'

April 12, 2021

Title The 'Tidymodels' Extension for GARCH Models

Version 0.1.1

Description Garch framework for use with the 'tidymodels' ecosystem. It includes both univariate and multivariate methods from the 'rugarch' and 'rmgarch' packages. These models include DCC-Garch, Copula Garch
and Go-GARCH among others.

License MIT + file LICENSE

Encoding UTF-8

LazyData true

Depends R (>= 3.5.0), parsnip, rugarch, rmgarch

Imports rlang (>= 0.1.2), magrittr, purrr, dplyr, stringr, tibble, tidyr, dials

Suggests tidymodels, tidyverse, modeltime, timetk, lubridate, knitr, rmarkdown, roxygen2, testthat (>= 3.0.0), covr

RoxygenNote 7.1.1

VignetteBuilder knitr

Config/testthat/edition 3

NeedsCompilation no

Author Alberto Almuiña [aut, cre]

Maintainer Alberto Almuiña <albertogonzalezalmuinha@gmail.com>

Repository CRAN

Date/Publication 2021-04-12 17:20:02 UTC

R topics documented:

`cgarch_rmgarch_multi_fit_impl` *Low-Level GARCH function for translating modeltime to forecast*

Description

Low-Level GARCH function for translating modeltime to forecast

Usage

```
cgarch_rmgarch_multi_fit_impl(
  formula,
  data,
  spec_type = "ugarchspec",
  period = "auto",
  ...
)
```

Arguments

`formula`	A dataframe of xreg (exogenous regressors)
`data`	A numeric vector of values to fit
`spec_type`	Must be ugarchspec
`period`	auto
`...`	Additional arguments passed to `forecast::Arima`

Value

A fitted model

dcc_rmgarch_multi_fit_impl

Low-Level GARCH function for translating modeltime to forecast

Description

Low-Level GARCH function for translating modeltime to forecast

Usage

```
dcc_rmgarch_multi_fit_impl(
  formula,
  data,
  spec_type = "ugarchspec",
  period = "auto",
  ...
)
```

Arguments

`formula`	A dataframe of xreg (exogenous regressors)
`data`	A numeric vector of values to fit
`spec_type`	Must be ugarchspec
`period`	Auto
`...`	Additional arguments passed to `forecast::Arima`

Value

A fitted model

dcc_rmgarch_multi_predict_impl

Bridge prediction function for GARCH models

Description

Bridge prediction function for GARCH models

Usage

```
dcc_rmgarch_multi_predict_impl(object, new_data, ...)
```

Arguments

object	An object of class model_fit
new_data	A rectangular data object, such as a data frame.
...	Additional arguments passed to stats::predict()

Value

A nested tibble

---

delete_attr *Delete Dimension Attribute*

Description

This function is a internal helper.

Usage

```
delete_attr(x)
```

Arguments

x	Data.frame

Value

A tibble

---

garch_multivariate_reg *General Interface for Multivariate GARCH Models*

Description

garch_multivariate_reg() allows you to model the volatility of various time series. This can be done with the multivariate equivalent of the univariate GARCH model. Estimating multivariate GARCH models turns out to be significantly more difficult than univariate GARCH models, but this function facilitates the task through different engines such as rugarch, dcc_rmgarch, gogar_rmgarch etc.

Usage

```
garch_multivariate_reg(mode = "regression", type = NULL)
```

Arguments

mode	A single character string for the type of model (Only regression is supported).
type	A single character string for the type of model or specification (See details below).
	Other options and argument can be set using set_engine() (See Engine Details below).

Details

Available engines:

- **rugarch (default)**: Connects to rugarch::multispec() and rugarch::multifit()
- **dcc_rmgarch**: Connects to rugarch::multispec(), rugarch::multifit(), rmgarch::dccspec() and rmgarch::dccfit().
- **c_rmgarch**: Connects to rugarch::multispec(), rugarch::multifit(), rmgarch::cgarchspec() and rmgarch::cgarchfit().
- **gogarch_rmgarch**: Connects to rmgarch::gogarchspec() and rmgarch::gogarchspec().

Value

A model specfication

Engine Details

rugarch (default)

The engine uses rugarch::multispec() and then rugarch::multifit()

Main Arguments

- type: You can choose between ugarchspec (default) or arfimaspec. Depending on which one you choose, you will select either a univariate GARCH model for each of your variables or an Arfima model as specification, which will then be passed to rugarch::multispec().

You **must** pass an argument through set_engine() called **specs** which will be a list consisting of the arguments to be passed to each of the specifications used in rugarch::multispec(). Other arguments that you wish to pass to rugarch::multifit() can also be passed through set_engine()

For example, imagine you have a data frame with 3 variables. For each of those variables you must define a specification (you can check the arguments you can use for a specification in ?rugarch::ugarchspec). Once the specifications have been decided, the way to pass it through set_engine would be as follows:

garch_multivariate_reg(mode = "regression") %>% set_engine("rugarch" , specs = list(spec1 = list(mean.model = list(armaOrder = c(1,0))), spec2 = list(mean.model = list(armaOrder = c(1,0))), spec3 = list(mean.model = list(armaOrder = c(1,0)))), out.sample = 10)

In the fit section we will see how to pass variables through parsnip::fit (See Fit Section below).

Parameter Notes:

- `xreg` - This engine does support xregs, but you have to provide them to each model in an array through set_engine. For more information see ?rugarch::ugarchspec. The xregs can be provided through `variance.model$external.regressors` or `mean.model$external.regressors` (or both) for the specifications of the desired variables.

dcc_rmgarch

The engine uses `rugarch::multispec()`, `rugarch::multifit()`, `rmgarch::dccspec()` and `rmgarch::dccfit()`.

Main Arguments

- `type`: Only `ugarchspec` is supported for this engine. This will then be passed to `rugarch::multispec()`.

You **must** pass an argument through `set_engine()` called **specs** which will be a list consisting of the arguments to be passed to each of the specifications used in `rugarch::multispec()`. Other arguments that you wish to pass to `rugarch::multifit()` can also be passed through `set_engine()`. To pass arguments to `dccfit()` you **must** pass a list through `set_engine` called **dcc_specs**.

For example, imagine you have a data frame with 3 variables. For each of those variables you must define a specification (you can check the arguments you can use for a specification in ?rugarch::ugarchspec). Once the specifications have been decided, the way to pass it through set_engine would be as follows:

garch_fit_model <- garch_multivariate_reg(type = "ugarchspec") %>% set_engine("dcc_rmgarch" , specs = list(spec1 = list(mean.model = list(armaOrder = c(1,0))), spec2 = list(mean.model = list(armaOrder = c(1,0))), spec3 = list(mean.model = list(armaOrder = c(1,0)))), dcc_specs = list(dccOrder = c(2,2), distribution = "mvlaplace"))

In the fit section we will see how to pass variables through parsnip::fit (See Fit Section below).

c_rmgarch

The engine uses `rugarch::multispec()`, `rugarch::multifit()`, `rmgarch::cgarchspec()` and `rmgarch::cgarchfit()`.

Main Arguments

- `type`: Only `ugarchspec` is supported for this engine. This will then be passed to `rugarch::multispec()`.

You **must** pass an argument through `set_engine()` called **specs** which will be a list consisting of the arguments to be passed to each of the specifications used in `rugarch::multispec()`. Other arguments that you wish to pass to `rugarch::multifit()` can also be passed through `set_engine()`. To pass arguments to `cgarchfit()` you **must** pass a list through `set_engine` called **c_specs**.

For example, imagine you have a data frame with 3 variables. For each of those variables you must define a specification (you can check the arguments you can use for a specification in ?rugarch::ugarchspec). Once the specifications have been decided, the way to pass it through set_engine would be as follows:

garch_fit_model <- garch_multivariate_reg(type = "arfima") %>% set_engine("c_rmgarch" , specs = list(spec1 = list(mean.model = list(armaOrder = c(1,0))), spec2 = list(mean.model = list(armaOrder = c(1,0))), spec3 = list(mean.model = list(armaOrder = c(1,0)))), c_specs = list(dccOrder = c(2,2))) %>% fit(value ~ date + id, data = rX_longer_train)

In the fit section we will see how to pass variables through parsnip::fit (See Fit Section below).

gogarch_rmgarch

The engine uses `rmgarch::gogarchspec()` and `rmgarch::gogarchfit()`.

Main Arguments

- type: Not available for this engine.

You **must** pass an argument through set_engine() called **gogarch_specs** which will be a list consisting of the arguments to be passed to each of the specifications used in rmgarch::gogarchspec(). Other arguments that you wish to pass to rmgarch::gogarchfit() can also be passed through set_engine().

For example, imagine you have a data frame with 3 variables. For each of those variables you must define a specification (you can check the arguments you can use for a specification in ?rugarch::ugarchspec). Once the specifications have been decided, the way to pass it through set_engine would be as follows:

model_fit_garch <- garch_multivariate_reg(type = "ugarchspec") %>% set_engine("gogarch_rmgarch" , gogarch_specs = list(variance.model = list(garchOrder = c(2,2)))) %>% fit(value ~ date + id, data = rX_longer_train)

In the fit section we will see how to pass variables through parsnip::fit (See Fit Section below).

See Also

fit.model_spec(), set_engine()

Examples

```
library(tidymodels)
library(garchmodels)
library(modeltime)
library(tidyverse)
library(timetk)
library(lubridate)

rX_longer <-  rX_longer %>%
    dplyr::mutate(date = as.Date(date)) %>%
    group_by(id) %>%
    future_frame(.length_out = 3, .bind_data = TRUE) %>%
    ungroup()

rX_longer_train  <- rX_longer %>% drop_na()
rX_longer_future <- rX_longer %>% filter(is.na(value))

#RUGARCH ENGINE

model_fit_garch <- garch_multivariate_reg() %>%
   set_engine("rugarch" , specs = list(spec1 = list(mean.model = list(armaOrder = c(1,0))),
                                       spec2 = list(mean.model = list(armaOrder = c(1,0))),
                                  spec3 = list(mean.model = list(armaOrder = c(1,0))))) %>%
    fit(value ~ date + id, data = rX_longer_train)

predict(model_fit_garch, new_data = rX_longer_future)


#DCC ENGINE
```

```
 model_fit_garch <- garch_multivariate_reg(type = "ugarchspec") %>%
set_engine("dcc_rmgarch" , specs = list(spec1 = list(mean.model = list(armaOrder = c(1,0))),
                                        spec2 = list(mean.model = list(armaOrder = c(1,0))),
                                       spec3 = list(mean.model = list(armaOrder = c(1,0)))),
           dcc_specs = list(dccOrder = c(2,2), distribution = "mvlaplace")) %>%
    fit(value ~ date + id, data = rX_longer_train)

predict(model_fit_garch, rX_longer_future)


# COPULA ENGINE

model_fit_garch <- garch_multivariate_reg(type = "ugarchspec") %>%
set_engine("c_rmgarch" , specs = list(spec1 = list(mean.model = list(armaOrder = c(1,0))),
                                        spec2 = list(mean.model = list(armaOrder = c(1,0))),
                                       spec3 = list(mean.model = list(armaOrder = c(1,0)))),
           c_specs = list(dccOrder = c(2,2))) %>%
    fit(value ~ date + id, data = rX_longer_train)


# GO-GARCH ENGINE

model_fit_garch <- garch_multivariate_reg(type = "ugarchspec") %>%
set_engine("gogarch_rmgarch" , gogarch_specs = list(variance.model = list(garchOrder = c(2,2)))) %>%
    fit(value ~ date + id, data = rX_longer_train)

predict(model_fit_garch, rX_longer_future)
```

---

garch_params *Tuning Parameters for Univariate Garch Models*

Description

Tuning Parameters for Univariate Garch Models

Usage

```
arch_order(range = c(0L, 3L), trans = NULL)

garch_order(range = c(0L, 3L), trans = NULL)

ar_order(range = c(0L, 5L), trans = NULL)

ma_order(range = c(0L, 5L), trans = NULL)
```

Arguments

range	A two-element vector holding the *defaults* for the smallest and largest possible values, respectively.

trans — A trans object from the scales package, such as scales::log10_trans() or scales::reciprocal_trans(). If not provided, the default is used which matches the units used in range. If no transformation, NULL.

Details

The main parameters for Univariate Garch models are:

- arch_order: The order corresponding to the ARCH part.
- garch_order: The order corresponding to the GARCH part.
- ar_order: The order of the non-seasonal auto-regressive (AR) terms.
- ma_order: The order of the non-seasonal moving average (MA) terms.

Value

A quant param

A quant param

A quant param

A quant param

Examples

```
arch_order()

garch_order()

ar_order()

ma_order()
```

---

garch_reg *General Interface for GARCH Models*

Description

General Interface for GARCH Models

Usage

```
garch_reg(
  mode = "regression",
  arch_order = NULL,
  garch_order = NULL,
  ar_order = NULL,
  ma_order = NULL,
  tune_by = NULL
)
```

Arguments

mode	A single character string for the type of model.
arch_order	An integer giving the order of the ARCH part for the variance model.
garch_order	An integer giving the order of the GARCH part for the variance model.
ar_order	An integer giving the order of the AR part for the mean model.
ma_order	An integer giving the order of the MA part for the mean model.
tune_by	Default is set to NULL, when no tuning. If you want to tune, you must choose between "seriesFor" or "sigmaFor" options. This will cause the function to not return a nested tibble and be able to tune. These arguments are converted to their specific names at the time that the model is fit. Other options and argument can be set using `set_engine()` (See Engine Details below).

Details

Available engines:

- **rugarch**: Connects to `rugarch::ugarchspec()` first and then to `rugarch::ugarchfit()`.

Value

A model specification

Engine Details

rugarch (default)

The engine uses `rugarch::ugarchspec()` and `rugarch::ugarchfit()`.

Function Parameters:

```
## Formal class 'standardGeneric' [package "methods"] with 8 slots
##  ..@ .Data    :function (variance.model = list(model = "sGARCH", garchOrder = c(1, 1),
##     submodel = NULL, external.regressors = NULL, variance.targeting = FALSE),
##    mean.model = list(armaOrder = c(1, 1), include.mean = TRUE, archm = FALSE,
##       archpow = 1, arfima = FALSE, external.regressors = NULL, archex = FALSE),
##     distribution.model = "norm", start.pars = list(), fixed.pars = list(),
##     ...)
##   ..@ generic   : chr "ugarchspec"
##   .. ..- attr(*, "package")= chr "rugarch"
##   ..@ package   : chr "rugarch"
##   ..@ group     : list()
##   ..@ valueClass: chr(0)
##  ..@ signature : chr [1:5] "variance.model" "mean.model" "distribution.model" "start.pars" ...
##  ..@ default  :Formal class 'derivedDefaultMethod' [package "methods"] with 4 slots
##  .. .. ..@ .Data :function (variance.model = list(model = "sGARCH", garchOrder = c(1, 1),
##     submodel = NULL, external.regressors = NULL, variance.targeting = FALSE),
```

```
##     mean.model = list(armaOrder = c(1, 1), include.mean = TRUE, archm = FALSE,
##       archpow = 1, arfima = FALSE, external.regressors = NULL, archex = FALSE),
##      distribution.model = "norm", start.pars = list(), fixed.pars = list(),
##      ...)
##   .. .. ..@ target :Formal class 'signature' [package "methods"] with 3 slots
##   .. .. .. .. ..@ .Data  : chr "ANY"
##   .. .. .. .. ..@ names  : chr "variance.model"
##   .. .. .. .. ..@ package: chr "methods"
##   .. .. ..@ defined:Formal class 'signature' [package "methods"] with 3 slots
##   .. .. .. .. ..@ .Data  : chr "ANY"
##   .. .. .. .. ..@ names  : chr "variance.model"
##   .. .. .. .. ..@ package: chr "methods"
##   .. .. ..@ generic: chr "ugarchspec"
##   .. .. .. ..- attr(*, "package")= chr "rugarch"
##  ..@ skeleton  : language (new("derivedDefaultMethod", .Data = function (variance.model = list(model
```

The Garch order for the variance model is provided using `arch_order` and `garch_order` parameters.. The ARMA order for the mean model is provided using `ar_order` and `ma_order` parameters. Other options and arguments can be set using `set_engine()`.

#' Parameter Notes:

- `xreg` - This engine supports xregs for both the variance model and the mean model. You can do this in two ways, either enter the matrices through set_engine parameters or as a formula in fit (note that the latter option is more limited, since you will not be able to pass two different xregs, one for each model). For simpler cases this is a compact option.
- order parameters - The parameters of rugarch::ugarchspec are lists containing several elements, some of them the commands that are the main arguments of the function. If you want to modify the parameter that encompasses such a list, you must know that the parameter passed in the function parameter will always prevail. (See Examples).

Fit Details

Date and Date-Time Variable

It's a requirement to have a date or date-time variable as a predictor. The `fit()` interface accepts date and date-time features and handles them internally.

- `fit(y ~ date)`

Univariate (No xregs, Exogenous Regressors):

For univariate analysis, you must include a date or date-time feature. Simply use:

- Formula Interface: `fit(y ~ date)` will ignore xreg's.

Multivariate (xregs, Exogenous Regressors)

The `xreg` parameter is populated using the `fit()` function:

- Only `factor`, ordered factor, and `numeric` data will be used as xregs.
- Date and Date-time variables are not used as xregs
- `character` data should be converted to factor.

Xreg Example: Suppose you have 3 features:

1. `y` (target)
2. `date` (time stamp),
3. `month.lbl` (labeled month as a ordered factor).

The `month.lbl` is an exogenous regressor that can be passed to the `garch_reg()` using `fit()`:

- `fit(y ~ date + month.lbl)` will pass `month.lbl` on as an exogenous regressor.

Note that date or date-time class values are excluded from `xreg`.

See Also

`fit.model_spec()`, `set_engine()`

Examples

```
library(tidymodels)
library(garchmodels)
library(modeltime)
library(tidyverse)
library(timetk)
library(lubridate)

rIBM_extended <- rIBM %>%
    future_frame(.length_out = 24, .bind_data = TRUE)

rIBM_train  <- rIBM_extended %>% drop_na()
rIBM_future <- rIBM_extended %>% filter(is.na(daily_returns))

model_garch_fit <-garchmodels::garch_reg(mode = "regression",
                                         arch_order = 1,
                                         garch_order = 1) %>%
    set_engine("rugarch") %>%
    fit(daily_returns ~ date, data = rIBM_train)

predict(model_garch_fit, rIBM_future)

model_garch_fit <-garchmodels::garch_reg(mode = "regression",
                                       arch_order = 2,
                                       garch_order = 2) %>%
    set_engine("rugarch", variance.model = list(model='gjrGARCH',
                                                garchOrder=c(1,1)),
               mean.model     = list(armaOrder=c(0,0))) %>%
    fit(daily_returns ~ date, data = rIBM_train)

predict(model_garch_fit, rIBM_future)
```

gogarch_rmgarch_multi_fit_impl *Low-Level GARCH function for translating modeltime to forecast*

Description

Low-Level GARCH function for translating modeltime to forecast

Usage

```
gogarch_rmgarch_multi_fit_impl(
  formula,
  data,
  spec_type = NULL,
  period = "auto",
  ...
)
```

Arguments

formula	A dataframe of xreg (exogenous regressors)
data	A numeric vector of values to fit
spec_type	NA
period	auto
...	Additional arguments passed to forecast::Arima

Value

A fitted model

gogarch_rmgarch_multi_predict_impl *Bridge prediction function for GARCH models*

Description

Bridge prediction function for GARCH models

Usage

```
gogarch_rmgarch_multi_predict_impl(object, new_data, ...)
```

Arguments

object	An object of class model_fit
new_data	A rectangular data object, such as a data frame.
...	Additional arguments passed to stats::predict()

Value

A nested tibble

---

new_modelgarch_bridge *Constructor for creating garchmodels models*

Description

These functions are used to construct new garchmodels bridge functions that connect the tidymodels infrastructure to time-series models containing date or date-time features.

Usage

```
new_modelgarch_bridge(class, models, data, extras = NULL, desc = NULL)
```

Arguments

class	A class name that is used for creating custom printing messages
models	A list containing one or more models
data	A data frame (or tibble) containing 4 columns: (date column with name that matches input data), .actual, .fitted, and .residuals.
extras	An optional list that is typically used for transferring preprocessing recipes to the predict method.
desc	An optional model description to appear when printing your modeltime objects

Value

A list with the constructor

rIBM *The IBM's Daily Returns*

Description

The IBM's Daily Returns

Usage

```
rIBM
```

Format

A tibble with 3523 rows and 2 variables:

- date Date. Timestamp information. Daily format.
- daily_returns Numeric. Value at the corresponding timestamp.

Examples

```
rIBM
```

rugarch_fit_impl *FIT - GARCH —-*

Description

#' Low-Level GARCH function for translating modeltime to forecast #' #' @param formula A dataframe of xreg (exogenous regressors) #' @param data A numeric vector of values to fit #' @param a The order of the non-seasonal auto-regressive (AR) terms. Often denoted "p" in pdq-notation. #' @param g The order of the non-seasonal auto-regressive (AR) terms. Often denoted "p" in pdq-notation. #' @param ... Additional arguments passed to forecast::Arima #' #' @export garch_fit_impl <- function(formula, data, a = 1, g = 1, ar_no_apply = NULL, ma_no_apply = NULL, period = "auto", ...)

```
# X & Y
others <- list(...)

 y <- all.vars(formula)[1]
 x <- attr(stats::terms(formula, data = data), "term.labels")

outcome <- data[[y]]
predictors <- data %>% dplyr::select(dplyr::all_of(x))
```

```
# INDEX & PERIOD
# Determine Period, Index Col, and Index
index_tbl <- modeltime::parse_index_from_data(predictors)
period    <- modeltime::parse_period_from_index(index_tbl, period)
idx_col   <- names(index_tbl)
idx       <- timetk::tk_index(index_tbl)

# XREGS
# Clean names, get xreg recipe, process predictors
# xreg_recipe <- create_xreg_recipe(predictor, prepare = TRUE)
# xreg_matrix <- juice_xreg_recipe(xreg_recipe, format = "matrix")

# FIT
outcome <- stats::ts(outcome, frequency = period)

fit_garch <- tseries::garch(outcome, order = c(a, g), ...)

# RETURN
modeltime::new_modeltime_bridge(
    class = "garch_fit_impl",

    # Models
    models = list(
        model_1 = fit_garch
    ),

   # Data - Date column (matches original), .actual, .fitted, and .residuals columns
    data = tibble::tibble(
        !! idx_col  := idx,
        .actual      =  as.numeric(outcome),
        .fitted      =  fit_garch$fitted.values[,1],
        .residuals   =  fit_garch$residuals
    ),

    extras = list(
        y_var    = y,
        period   = period,
        otros    = others
    ),

    # Description - Convert arima model parameters to short description
   desc = stringr::str_glue('GARCH ({fit_garch$order[1]}, {fit_garch$order[2]}) Model')
)
```

Usage

```
rugarch_fit_impl(
  formula,
  data,
```

```
  a = 1,
  g = 1,
  ar = 1,
  ma = 1,
  tune_by = NULL,
  period = "auto",
  ...
)
```

Arguments

formula	A dataframe of xreg (exogenous regressors)
data	A numeric vector of values to fit
a	The order of ARCH part
g	The order of GARCH part
ar	The order of the non-seasonal auto-regressive (AR) terms. Often denoted "p" in pdq-notation.
ma	The order of the non-seasonal auto-regressive (AR) terms. Often denoted "p" in pdq-notation.
tune_by	Parameter for tuning.
period	Period
...	Additional arguments passed to `forecast::Arima`

Details

#' @export print.garch_fit_impl <- function(x, ...) print(x$models$model_1) invisible(x)

Low-Level GARCH function for translating modeltime to forecast

Value

A fitted model

`rugarch_multi_fit_impl`

Low-Level GARCH function for translating modeltime to forecast

---

Description

Low-Level GARCH function for translating modeltime to forecast

Usage

```
rugarch_multi_fit_impl(
  formula,
  data,
  spec_type = "ugarchspec",
  period = "auto",
  ...
)
```

Arguments

formula	A dataframe of xreg (exogenous regressors)
data	A numeric vector of values to fit
spec_type	Must be ugarchspec or arfimaspec
period	Auto
...	Additional arguments passed to forecast::Arima

Value

A fitted model

---

rugarch_multi_predict_impl

Bridge prediction function for GARCH models

Description

Bridge prediction function for GARCH models

Usage

```
rugarch_multi_predict_impl(object, new_data, ...)
```

Arguments

object	An object of class model_fit
new_data	A rectangular data object, such as a data frame.
...	Additional arguments passed to stats::predict()

Value

A nested tibble

rugarch_predict_impl *Bridge prediction function for GARCH models*

Description

Bridge prediction function for GARCH models

Usage

```
rugarch_predict_impl(object, new_data, ...)
```

Arguments

object	An object of class model_fit
new_data	A rectangular data object, such as a data frame.
...	Additional arguments passed to stats::predict()

Value

A nested tibble

rX_longer *IBM, Google and BP's daily returns in long format*

Description

IBM, Google and BP's daily returns in long format

Usage

```
rX_longer
```

Format

A tibble with 8550 rows and 3 variables:

- id Factor. Unique series identifier
- date Date. Timestamp information. Daily format.
- value Numeric. Value at the corresponding timestamp.

Details

```
library(timetk)
m750_splits <- time_series_split(m750, assess = "2 years", cumulative = TRUE)
```

Examples

```
rX_longer
```

Index