Package ‘TSstudio’

January 21, 2020

Type Package

Title Functions for Time Series Analysis and Forecasting

Version 0.1.6

Maintainer Rami Krispin <rami.krispin@gmail.com>

Description Provides a set of tools for descriptive and predictive analysis of time series data. That includes functions for interactive visualization of time series objects and as well utility functions for automation time series forecasting.

License MIT + file LICENSE

Encoding UTF-8

LazyData true

Depends R (>= 3.0.2)

Imports data.table(>= 1.11.2), dplyr(>= 0.7.5), forecast (>= 8.2), forecastHybrid(>= 2.0.10), future(>= 1.10.0), future.apply(>= 1.0.1), lubridate (>= 1.6.0), magrittr (>= 1.5), plotly (>= 4.7.1), purrr(>= 0.2.5), RColorBrewer(>= 1.1-2), reshape2 (>= 1.4.2), scales(>= 1.0.0), tidyr(>= 0.8.1), tsibble(>= 0.8.2), viridis (>= 0.5.1), xts (>= 0.12-0), zoo (>= 1.8-0)

Suggests devtools, DT, knitr, quantmod, rmarkdown, UKgrid

VignetteBuilder knitr

RoxygenNote 6.1.1

URL https://github.com/RamiKrispin/TSstudio

BugReports https://github.com/RamiKrispin/TSstudio/issues

NeedsCompilation no

Author Rami Krispin [aut, cre]

Repository CRAN

Date/Publication 2020-01-21 05:30:02 UTC

R topics documented:

arima_diag *Diagnostic Plots for ARIMA Models*

Description

Diagnostic Plots for ARIMA Models

Usage

```
arima_diag(ts.obj, method = list(first = list(diff = 1, log = TRUE, title
  = "First Difference with Log Transformation")), cor = TRUE)
```

Arguments

ts.obj	A ts object
method	A list, defines the transformation parameters of each plot. Each plot should be defined by a list, where the name of the list defines the plot ID. The plot parameters are: diff - an integer, defines the degree of diffrence log - a boolean, optional, defines if log transformation should be used title - optional, the plot title
cor	A boolean, if TRUE (default), will plot the series ACF and PACF

Details

The arima_diag function provides a set of diagnostic plots for identify the ARIMA model parameters. The ACF and PACF can assist in identifying the AR and MA process, and the diffrence plotting hel in idenitfying the degree of differencing that required to make the series stationary

Value

A plot

Examples

```
data(USgas)

arima_diag(ts.obj = USgas)

# Can define more than one differencing plot using the 'method' argument

arima_diag(ts.obj = USgas,
           cor = TRUE,
           method = list(first = list(diff = 1,
                                      log = TRUE,
                                      title = "First Diff with Log Transformation"),
                         Second = list(diff = c(1,1),
                                       log = TRUE,
                                       title = "Second Diff with Log Transformation")))
```

ccf_plot *Time Series Cross Correlation Lags Visualization*

Description

Visualize the series y against the series x lags (according to the setting of the lags argument) and return the corresponding cross-correlation value for each lag

Usage

```
ccf_plot(x, y, lags = 0:12, margin = 0.02, n_plots = 3,
  Xshare = TRUE, Yshare = TRUE, title = NULL)
```

Arguments

x	A univariate time series object of a class "ts"
y	A univariate time series object of a class "ts"
lags	An integer, set the lags range, by default will plot the two series along with the first 12 lags
margin	Plotly parameter, either a single value or four values (all between 0 and 1). If four values provided, the first will be used as the left margin, the second will be used as the right margin, the third will be used as the top margin, and the fourth will be used as the bottom margin. If a single value provided, it will be used as all four margins.
n_plots	An integer, define the number of plots per row
Xshare	Plotly parameter, should the x-axis be shared amongst the subplots?
Yshare	Plotly parameter, should the y-axis be shared amongst the subplots?
title	A character, optional, set the plot title

Value

Plot

Examples

```
data("USUnRate")
data("USVSales")

ccf_plot(x = USVSales, y = USUnRate)

#Plotting the first 6 lead and lags of the USVSales with the USUnRate
ccf_plot(x = USVSales, y = USUnRate, lags = -6:6)

# Setting the plot margin and number of plots in each raw
ccf_plot(x = USVSales, y = USUnRate, lags = c(0, 6, 12, 24),
margin = 0.01,  n_plots = 2)
```

check_res — *Visualization of the Residuals of a Time Series Model*

Description

Provides a visualization of the residuals of a time series model. That includes a time series plot of the residuals, and the plots of the autocorrelation function (acf) and histogram of the residuals

Usage

```
check_res(ts.model, lag.max = 36)
```

Arguments

`ts.model`	A time series model (or forecasted) object, support any model from the forecast package with a residuals output
`lag.max`	The maximum number of lags to display in the residuals' autocorrelation function plot

Examples

```
library(forecast)
data(USgas)

# Create a model
fit <- auto.arima(USgas)

# Check the residuals of the model
check_res(fit)
```

Coffee_Prices — *Coffee Prices: Robusta and Arabica*

Description

Coffee Prices: Robusta and Arabica: 1960 - 2018. Units: Dollars per Kg

Usage

```
Coffee_Prices
```

Format

Time series data - 'mts' object

Source

WIKI Commodity Prices - Quandle

Examples

```
ts_plot(Coffee_Prices)
```

`create_model` *A Functional Approach for Building the `train_model` Components*

Description

Add, edit, or remove the components of the `train_model` function

Usage

```
create_model()

add_input(model.obj, input)

add_methods(model.obj, methods)

remove_methods(model.obj, method_ids)

add_train_method(model.obj, train_method)

add_horizon(model.obj, horizon)

build_model(model.obj)

set_error(model.obj, error)

add_xreg(model.obj, xreg)

add_level(model.obj, level)
```

Arguments

`model.obj`	The train_model skeleton, created by the create_model function or edited by add_input, add_methods, remove_methods, add_train_method or add_horizon
`input`	A univariate time series object (ts class)
`methods`	A list, defines the models to use for training and forecasting the series. The list must include a sub list with the model type, and the model's arguments (when applicable) and notes about the model. The sub-list name will be used as the model ID. Possible models: `arima` - model from the stats package

auto.arima - model from the forecast package

ets - model from the forecast package

HoltWinters - model from the stats package

nnetar - model from the forecast package

tslm - model from the forecast package (note that the 'tslm' model must have the formula argument in the 'method_arg' argument)

method_ids
: A character, defines the IDs of the model methods to be remove with the remove_methods function

train_method
: A list, defines the train approach, either using a single testing partition (sample out) or use multiple testing partitions (backtesting). The list should include the training method argument, (please see 'details' for the structure of the argument)

horizon
: An integer, defines the forecast horizon

error
: A character, defines the error metrics to be used to sort the models leaderboard. Possible metric - "MAPE" or "RMSE"

xreg
: Optional, a list with two vectors (e.g., data.frame or matrix) of external regressors, one vector corresponding to the input series and second to the forecast itself (e.g., must have the same length as the input and forecast horizon, respectively)

level
: An integer, set the confidence level of the prediction intervals

Examples

```
 ## Not run:
### Building train_model function by adding its different components
# Create a skeleton model
md <- create_model()

class(md)

# Add input
data(USgas)
md <- add_input(model.obj = md, input = USgas)

# Add methods
methods <- list(ets1 = list(method = "ets",
                            method_arg = list(opt.crit = "lik"),
                            notes = "ETS model with opt.crit = lik"),
                ets2 = list(method = "ets",
                            method_arg = list(opt.crit = "amse"),
                            notes = "ETS model with opt.crit = amse"),
                arima1 = list(method = "arima",
                              method_arg = list(order = c(1,1,1),
                                        seasonal = list(order = c(1,0,1))),
                              notes = "SARIMA(1,1,1)(1,0,1)"))

md <- add_methods(model.obj = md, methods = methods)

# Add additional methods
methods2 <- list(arima2 = list(method = "arima",
                              method_arg = list(order = c(2,1,2),
```

```
                                    seasonal = list(order = c(1,1,1))),
                             notes = "SARIMA(2,1,2)(1,1,1)"),
                hw = list(method = "HoltWinters",
                          method_arg = NULL,
                          notes = "HoltWinters Model"),
                tslm = list(method = "tslm",
                    method_arg = list(formula = input ~ trend + season),
                    notes = "tslm model with trend and seasonal components"))

md <- add_methods(model.obj = md, methods = methods2)

# Remove methods
md <- remove_methods(model.obj = md, method_ids = c("ets2"))

# Add train method
md <- add_train_method(model.obj = md, train_method = list(partitions = 6,
                                                       sample.out = 12,
                                                       space = 3))


# Set the forecast horizon
md <- add_horizon(model.obj = md, horizon = 12)

# Add the forecast prediction intervals confidence level
md <- add_level(model.obj = md, level = c(90, 95))

### Alternatively, pipe the function with the magrittr package

library(magrittr)

md <- create_model() %>%
      add_input(input = USgas) %>%
      add_methods(methods = methods) %>%
      add_methods(methods = methods2) %>%
      add_train_method(train_method = list(partitions = 4,
                                           sample.out = 12,
                                           space = 3)) %>%
       add_horizon(horizon = 12) %>%
       add_level(level = c(90, 95))

# Run the model
fc <- md %>% build_model()

## End(Not run)
```

EURO_Brent *Crude Oil Prices: Brent - Europe*

Description

Crude Oil Prices: Brent - Europe: 1987 - 2019. Units: Dollars per Barrel

Usage

```
EURO_Brent
```

Format

Time series data - 'zoo' object

Source

U.S. Energy Information Administration, Crude Oil Prices: Brent - Europe [MCOILBRENTEU], retrieved from FRED, Federal Reserve Bank of St. Louis; https://fred.stlouisfed.org/series/MCOILBRENTEU, January 8, 2018.

Examples

```
ts_plot(EURO_Brent)
ts_decompose(EURO_Brent, type = "both")
```

---

forecast_sim *Forecasting simulation*

Description

Creating different forecast paths for forecast objects (when applicable), by utilizing the underline model distribution with the `simulate` function

Usage

```
forecast_sim(model, h, n, sim_color = "blue", opacity = 0.05,
  plot = TRUE)
```

Arguments

model	A forecasting model supporting `Arima`, `auto.arima`, `ets`, and `nnetar` models from the **forecast** package
h	An integer, defines the forecast horizon
n	An integer, set the number of iterations of the simulation
sim_color	Set the color of the simulation paths lines
opacity	Set the opacity level of the simulation path lines
plot	Logical, if TRUE will desplay the output plot

Value

The baseline series, the simulated values and a plot

Examples

```
 ## Not run:
library(forecast)
data(USgas)

# Create a model
fit <- auto.arima(USgas)

# Simulate 100 possible forecast path, with horizon of 60 months
forecast_sim(model = fit, h = 60, n = 100)

## End(Not run)
```

`Michigan_CS` *University of Michigan Consumer Survey, Index of Consumer Sentiment*

Description

University of Michigan Consumer Survey, Index of Consumer Sentiment: 1980 - 2019. Units: Index 1966:Q1=100

Usage

```
Michigan_CS
```

Format

Time series data - 'xts' object

Source

University of Michigan, University of Michigan: Consumer Sentiment

Examples

```
ts_plot(Michigan_CS)
ts_heatmap(Michigan_CS)
```

plot_error — *Plot the Models Error Rates on the Testing Partitions*

Description

Plot the Models Error Rates on the Testing Partitions

Usage

```
plot_error(model.obj, error = "MAPE", palette = "Set1")
```

Arguments

`model.obj`	A train_model object
`error`	A character, defines the type of error metrics to plot, possible metric - "MAPE" or "RMSE"
`palette`	A character, defines the color type to used on the plot, use row.names(RColorBrewer::brewer.pal.info) to view possible color palletes

Details

The plot_model provides a visualization of the models performance on the testing paritions for the train_model function output

Value

A plot with a summery of the models error rate by testing partition

Examples

```
 ## Not run:
# Defining the models and their arguments
methods <- list(ets1 = list(method = "ets",
                            method_arg = list(opt.crit = "lik"),
                            notes = "ETS model with opt.crit = lik"),
                ets2 = list(method = "ets",
                            method_arg = list(opt.crit = "amse"),
                            notes = "ETS model with opt.crit = amse"),
                arima1 = list(method = "arima",
                              method_arg = list(order = c(2,1,0)),
                              notes = "ARIMA(2,1,0)"),
                arima2 = list(method = "arima",
                              method_arg = list(order = c(2,1,2),
                                                seasonal = list(order = c(1,1,1))),
                              notes = "SARIMA(2,1,2)(1,1,1)"),
                hw = list(method = "HoltWinters",
                          method_arg = NULL,
                          notes = "HoltWinters Model"),
```

```
                tslm = list(method = "tslm",
                            method_arg = list(formula = input ~ trend + season),
                            notes = "tslm model with trend and seasonal components"))
# Training the models with backtesting
md <- train_model(input = USgas,
                  methods = methods,
                  train_method = list(partitions = 6,
                                      sample.out = 12,
                                      space = 3),
                  horizon = 12,
                  error = "MAPE")

# Plot the models performance on the testing partitions
plot_error(model.obj = md)

## End(Not run)
```

---

plot_forecast *Plotting Forecast Object*

Description

Visualization functions for forecast package forecasting objects

Usage

```
plot_forecast(forecast_obj, title = NULL, Xtitle = NULL,
  Ytitle = NULL, color = NULL, width = 2)
```

Arguments

forecast_obj	A forecast object from the forecast, forecastHybrid, or bsts packages
title	A character, a plot title, optional
Xtitle	Set the X axis title, default set to NULL
Ytitle	Set the Y axis title, default set to NULL
color	A character, the plot, support both name and expression
width	An Integer, define the plot width, default is set to 2

Examples

```
data(USgas)
library(forecast)
fit <- ets(USgas)
fc<- forecast(fit, h = 60)
plot_forecast(fc)
```

plot_grid *Visualizing Grid Search Results*

Description

Visualizing Grid Search Results

Usage

```
plot_grid(grid.obj, top = NULL, highlight = 0.1, type = "parcoords",
  colors = list(showscale = TRUE, reversescale = FALSE, colorscale =
  "Jet"))
```

Arguments

`grid.obj`	A ts_grid output object
`top`	An integer, set the number of hyper-parameters combinations to visualize (ordered by accuracy). If set to NULL (default), will plot the top 100 combinations
`highlight`	A proportion between 0 (excluding) and 1, set the number of hyper-parameters combinations to highlight (by accuracy), if the type argument is set to "parcoords"
`type`	The plot type, either "3D" for 3D plot or "parcoords" for parallel coordinates plot. Note: the 3D plot option is applicable whenever there are three tuning parameters, otherwise will use a 2D plot for two tuning parameters.
`colors`	A list of plotly arguments for the color scale setting: showscale - display the color scale if set to TRUE. reversescale - reverse the color scale if set to TRUE colorscale set the color scale of the plot, possible palettes are: Greys, YlGnBu, Greens , YlOrRd, Bluered, RdBu, Reds, Blues, Picnic, Rainbow, Portland, Jet, Hot, Blackbody, Earth, Electric, Viridis, Cividis

plot_model *Plot the Models Performance on the Testing Partitions*

Description

Plot the Models Performance on the Testing Partitions

Usage

```
plot_model(model.obj, model_ids = NULL)
```

Arguments

model.obj
: A train_model object

model_ids
: A character, defines the trained models to plot, if set to NULL (default), will plot all the models

Details

The plot_model provides a visualization of the models performance on the testing paritions for the train_model function output

Value

Animation of models forecast on the testing partitions compared to the actuals

Examples

```
 ## Not run:
# Defining the models and their arguments
methods <- list(ets1 = list(method = "ets",
                            method_arg = list(opt.crit = "lik"),
                            notes = "ETS model with opt.crit = lik"),
                ets2 = list(method = "ets",
                            method_arg = list(opt.crit = "amse"),
                            notes = "ETS model with opt.crit = amse"),
                arima1 = list(method = "arima",
                              method_arg = list(order = c(2,1,0)),
                              notes = "ARIMA(2,1,0)"),
                arima2 = list(method = "arima",
                              method_arg = list(order = c(2,1,2),
                                                seasonal = list(order = c(1,1,1))),
                              notes = "SARIMA(2,1,2)(1,1,1)"),
                hw = list(method = "HoltWinters",
                          method_arg = NULL,
                          notes = "HoltWinters Model"),
                tslm = list(method = "tslm",
                            method_arg = list(formula = input ~ trend + season),
                            notes = "tslm model with trend and seasonal components"))
# Training the models with backtesting
md <- train_model(input = USgas,
                  methods = methods,
                  train_method = list(partitions = 6,
                                      sample.out = 12,
                                      space = 3),
                  horizon = 12,
                  error = "MAPE")
# Plot the models performance on the testing partitions
plot_model(model.obj = md)

# Plot only the ETS models
plot_model(model.obj = md , model_ids = c("ets1", "ets2"))
```

```
## End(Not run)
```

---

res_hist *Histogram Plot of the Residuals Values*

Description

Histogram plot of the residuals values

Usage

```
res_hist(forecast.obj)
```

Arguments

forecast.obj	A fitted or forecasted object (of the forecast package) with residuals output

Examples

```
## Not run:
library(forecast)
data(USgas)

# Set the horizon of the forecast
h <- 12

# split to training/testing partition
split_ts <- ts_split(USgas, sample.out  = h)
train <- split_ts$train
test <- split_ts$test

# Create forecast object
fc <- forecast(auto.arima(train, lambda = BoxCox.lambda(train)), h = h)

# Plot the fitted and forecasted vs the actual values
res_hist(forecast.obj = fc)

## End(Not run)
```

`test_forecast` *Visualize of the Fitted and the Forecasted vs the Actual Values*

Description

Visualize the fitted values of the training set and the forecast values of the testing set against the actual values of the series

Usage

```
test_forecast(actual, forecast.obj, train = NULL, test, Ygrid = FALSE,
  Xgrid = FALSE, hover = TRUE)
```

Arguments

`actual`	The full time series object (supports "ts", "zoo" and "xts" formats)
`forecast.obj`	The forecast output of the training set with horizon align to the length of the testing (support forecasted objects from the “forecast” package)
`train`	Training partition, a subset of the first n observation in the series (not requiredthed)
`test`	The testing (hold-out) partition
`Ygrid`	Logic,show the Y axis grid if set to TRUE
`Xgrid`	Logic,show the X axis grid if set to TRUE
`hover`	If TRUE add tooltip with information about the model accuracy

Examples

```
## Not run:
library(forecast)
data(USgas)

# Set the horizon of the forecast
h <- 12

# split to training/testing partition
split_ts <- ts_split(USgas, sample.out  = h)
train <- split_ts$train
test <- split_ts$test

# Create forecast object
fc <- forecast(auto.arima(train, lambda = BoxCox.lambda(train)), h = h)

# Plot the fitted and forecasted vs the actual values
test_forecast(actual = USgas, forecast.obj = fc, test = test)

## End(Not run)
```

train_model *Train, Test, Evaluate, and Forecast Multiple Time Series Forecasting Models*

Description

Method for train test and compare multiple time series models using either one partition (i.e., sample out) or multipe partitions (backtesting)

Usage

```
train_model(input, methods, train_method, horizon, error = "MAPE",
  xreg = NULL, level = c(80, 95))
```

Arguments

`input` A univariate time series object (ts class)

`methods` A list, defines the models to use for training and forecasting the series. The list must include a sub list with the model type, and the model's arguments (when applicable) and notes about the model. The sub-list name will be used as the model ID. Possible models:

`arima` - model from the stats package

`auto.arima` - model from the forecast package

`ets` - model from the forecast package

`HoltWinters` - model from the stats package

`nnetar` - model from the forecast package

`tslm` - model from the forecast package (note that the 'tslm' model must have the formula argument in the 'method_arg' argument)

`train_method` A list, defines the backtesting parameters:

partitions - an integer, set the number of training and testing partitions to be used in the backtesting process, where when partition is set to 1 it is a simple holdout training approach

space - an integer, defines the length of the backtesting window expansion

sample.in - an integer, optional, defines the length of the training partitions, and therefore the backtesting window structure. By default, it set to NULL and therefore, the backtesting using expending window. Otherwise, when the sample.in defined, the window structure is sliding

sample.in - an integer, optional, defines the length of the training partitions, and therefore the type of the backtesting window. By default, is set to NULL, which implay that the backtesting is using an expending window. Otherwise, when defining the size of the training partition, th defines the train approach, either using a single testing partition (sample out) or use multiple testing partitions (backtesting). The list should include the training method argument, (please see 'details' for the structure of the argument)

`horizon` An integer, defines the forecast horizon

error	A character, defines the error metrics to be used to sort the models leaderboard. Possible metric - "MAPE" or "RMSE"
xreg	Optional, a list with two vectors (e.g., data.frame or matrix) of external regressors, one vector corresponding to the input series and second to the forecast itself (e.g., must have the same length as the input and forecast horizon, respectively)
level	An integer, set the confidence level of the prediction intervals

Examples

```
 ## Not run:
# Defining the models and their arguments
methods <- list(ets1 = list(method = "ets",
                            method_arg = list(opt.crit = "lik"),
                            notes = "ETS model with opt.crit = lik"),
                ets2 = list(method = "ets",
                            method_arg = list(opt.crit = "amse"),
                            notes = "ETS model with opt.crit = amse"),
                arima1 = list(method = "arima",
                              method_arg = list(order = c(2,1,0)),
                              notes = "ARIMA(2,1,0)"),
                arima2 = list(method = "arima",
                              method_arg = list(order = c(2,1,2),
                                                seasonal = list(order = c(1,1,1))),
                              notes = "SARIMA(2,1,2)(1,1,1)"),
                hw = list(method = "HoltWinters",
                          method_arg = NULL,
                          notes = "HoltWinters Model"),
                tslm = list(method = "tslm",
                            method_arg = list(formula = input ~ trend + season),
                            notes = "tslm model with trend and seasonal components"))
# Training the models with backtesting
md <- train_model(input = USgas,
                  methods = methods,
                  train_method = list(partitions = 4,
                                      sample.out = 12,
                                      space = 3),
                  horizon = 12,
                  error = "MAPE")
# View the model performance on the backtesting partitions
md$leaderboard

## End(Not run)
```

---

ts_cor *An Interactive Visualization of the ACF and PACF Functions*

Description

An Interactive Visualization of the ACF and PACF Functions

Usage

```
ts_cor(ts.obj, type = "both", seasonal = TRUE, ci = 0.95,
  lag.max = NULL, seasonal_lags = NULL)
```

Arguments

`ts.obj`	A univariate time series object class 'ts'
`type`	A character, defines the plot type - 'acf' for ACF plot, 'pacf' for PACF plot, and 'both' (default) for both ACF and PACF plots
`seasonal`	A boolean, when set to TRUE (default) will color the seasonal lags
`ci`	The significant level of the estimation - a numeric value between 0 and 1, default is set for 0.95
`lag.max`	maximum lag at which to calculate the acf. Default is 10*log10(N/m) where N is the number of observations and m the number of series. Will be automatically limited to one less than the number of observations in the series
`seasonal_lags`	A vector of integers, highlight specific cyclic lags (besides the main seasonal lags of the series). This is useful when working with multiseasonal time series data. For example, for a monthly series (e.g., frequency 12) setting the argument to 3 will highlight the quarterly lags

Examples

```
data(USgas)

ts_cor(ts.obj = USgas)

# Setting the maximum number of lags to 72
ts_cor(ts.obj = USgas, lag.max = 72)

# Plotting only ACF
ts_cor(ts.obj = USgas, lag.max = 72, type = "acf")
```

---

ts_decompose *Visualization of the Decompose of a Time Series Object*

Description

Interactive visualization the trend, seasonal and random components of a time series based on the decompose function from the stats package.

Usage

```
ts_decompose(ts.obj, type = "additive", showline = TRUE)
```

Arguments

`ts.obj`	a univariate time series object of a class "ts", "zoo" or "xts"
`type`	Set the type of the seasonal component, can be set to either "additive", "multiplicative" or "both" to compare between the first two options (default set to “additive”)
`showline`	Logic, add a separation line between each of the plot components (default set to TRUE)

Examples

```
# Defualt decompose plot
ts_decompose(AirPassengers)

# Remove the sepration lines between the plot components
ts_decompose(AirPassengers, showline = FALSE)

# Plot side by side a decompose of additive and multiplicative series
ts_decompose(AirPassengers, type = "both")
```

---

ts_grid *Tuning Time Series Forecasting Models Parameters with Grid Search*

Description

Tuning time series models with grid search approach using backtesting method. If set to "auto" (default), will use all available cores in the system minus 1

Usage

```
ts_grid(ts.obj, model, optim = "MAPE", periods, window_length = NULL,
  window_space, window_test, hyper_params, parallel = TRUE,
  n.cores = "auto")
```

Arguments

`ts.obj`	A univariate time series object of a class "ts"
`model`	A string, defines the model c("HoltWinters"), currently support only Holt-Winters model
`optim`	A string, set the optimization method - c("MAPE", "RMSE")
`periods`	A string, set the number backtesting periods
`window_length`	An integer, defines the length of the backtesting training window. If set to NULL (default) will use an expending window starting the from the first observation, otherwise will use a sliding window.
`window_space`	An integer, set the space length between each of the backtesting training partition

window_test	An integer, set the length of the backtesting testing partition
hyper_params	A list, defines the tuning parameters and their range
parallel	Logical, if TRUE use multiple cores in parallel
n.cores	Set the number of cores to use if the parallel argument is set to TRUE. If set to "auto" (default), will use n-1 of the available cores

Value

A list

Examples

```
## Not run:
 data(USgas)

 # Starting with a shallow search (sequence between 0 and 1 with jumps of 0.1)
 # To speed up the process, will set the parallel option to TRUE
 # to run the search in parallel using 8 cores

 hw_grid_shallow <- ts_grid(ts.obj = USgas,
                            periods = 6,
                            model = "HoltWinters",
                            optim = "MAPE",
                            window_space = 6,
                            window_test = 12,
                            hyper_params = list(alpha = seq(0.01, 1,0.1),
                                                beta =  seq(0.01, 1,0.1),
                                                gamma = seq(0.01, 1,0.1)),
                            parallel = TRUE,
                            n.cores = 8)


 # Use the parameter range of the top 20 models
 # to set a narrow but more agressive search

 a_min <- min(hw_grid_shallow$grid_df$alpha[1:20])
 a_max <- max(hw_grid_shallow$grid_df$alpha[1:20])

 b_min <- min(hw_grid_shallow$grid_df$beta[1:20])
 b_max <- max(hw_grid_shallow$grid_df$beta[1:20])

 g_min <- min(hw_grid_shallow$grid_df$gamma[1:20])
 g_max <- max(hw_grid_shallow$grid_df$gamma[1:20])

 hw_grid_second <- ts_grid(ts.obj = USgas,
                           periods = 6,
                           model = "HoltWinters",
                           optim = "MAPE",
                           window_space = 6,
                           window_test = 12,
                           hyper_params = list(alpha = seq(a_min, a_max,0.05),
```

```
                                         beta =  seq(b_min, b_max,0.05),
                                         gamma = seq(g_min, g_max,0.05)),
                              parallel = TRUE,
                              n.cores = 8)

md <- HoltWinters(USgas,
                  alpha = hw_grid_second$alpha,
                  beta = hw_grid_second$beta,
                  gamma = hw_grid_second$gamma)

library(forecast)

fc <- forecast(md, h = 60)

plot_forecast(fc)


## End(Not run)
```

`ts_heatmap` *Heatmap Plot for Time Series*

Description

Heatmap plot for time series object by it periodicity (currently support only daily, weekly, monthly and quarterly frequencies)

Usage

```
ts_heatmap(ts.obj, last = NULL, wday = TRUE, color = "Blues",
  title = NULL, padding = TRUE)
```

Arguments

`ts.obj`	A univariate time series object of a class "ts", "zoo", "xts", and the data frame family (data.frame, data.table, tbl, tibble, etc.) with a Date column and at least one numeric column. This function support time series objects with a daily, weekly, monthly and quarterly frequencies
`last`	An integer (optional), set a subset using only the last observations in the series
`wday`	An boolean, provides a weekday veiw for daily data (relevent only for objects with dates such as xts, zoo, data.frame, etc.)
`color`	A character, setting the color palette of the heatmap. Corresponding to any of the RColorBrewer palette or any other arguments of the `col_numeric` function. By default using the "Blues" palette
`title`	A character (optional), set the plot title
`padding`	A boolean, if TRUE will add to the heatmap spaces between the observations

Examples

```
data(USgas)
ts_heatmap(USgas)

# Show only the last 4 years
ts_heatmap(USgas, last = 4 *12)
```

`ts_info` *Get the Time Series Information*

Description

Returning the time series object main characteristics

Usage

```
ts_info(ts.obj)
```

Arguments

`ts.obj`	A time series object of a class "ts", "mts", "xts", or "zoo"

Value

Text

Examples

```
# ts object
data("USgas")
ts_info(USgas)

# mts object
data("Coffee_Prices")
ts_info(Coffee_Prices)

# xts object
data("Michigan_CS")
ts_info(Michigan_CS)
```

ts_lags *Time Series Lag Visualization*

Description

Visualization of series with its lags, can be used to identify a correlation between the series and it lags

Usage

```
ts_lags(ts.obj, lags = 1:12, margin = 0.02, Xshare = TRUE,
  Yshare = TRUE, n_plots = 3)
```

Arguments

ts.obj	A univariate time series object of a class "ts", "zoo" or "xts"
lags	An integer, set the lags range, by default will plot the first 12 lags
margin	Plotly parameter, either a single value or four values (all between 0 and 1). If four values provided, the first will be used as the left margin, the second will be used as the right margin, the third will be used as the top margin, and the fourth will be used as the bottom margin. If a single value provided, it will be used as all four margins.
Xshare	Plotly parameter, should the x-axis be shared amongst the subplots?
Yshare	Plotly parameter, should the y-axis be shared amongst the subplots?
n_plots	An integer, define the number of plots per row

Examples

```
data(USgas)

# Plot the first 12 lags (default)
ts_lags(USgas)

# Plot the seasonal lags for the first 4 years (hence, lag 12, 24, 36, 48)
ts_lags(USgas, lags = c(12, 24, 36, 48))

# Setting the margin between the plot
ts_lags(USgas, lags = c(12, 24, 36, 48), margin = 0.01)
```

ts_ma *Moving Average Method for Time Series Data*

Description

Calculate the moving average (and double moving average) for time series data

Usage

```
ts_ma(ts.obj, n = c(3, 6, 9), n_left = NULL, n_right = NULL,
  double = NULL, plot = TRUE, show_legend = TRUE, multiple = FALSE,
  separate = TRUE, margin = 0.03, title = NULL, Xtitle = NULL,
  Ytitle = NULL)
```

Arguments

ts.obj	a univariate time series object of a class "ts", "zoo" or "xts" (support only series with either monthly or quarterly frequency)
n	A single or multiple integers (by default using 3, 6, and 9 as inputs), define a two-sides moving averages by setting the number of past and future to use in each moving average window along with current observation.
n_left	A single integer (optional argument, default set to NULL), can be used, along with the n_right argument, an unbalanced moving average. The n_left defines the number of lags to includes in the moving average.
n_right	A single integer (optional argument, default set to NULL), can be used, along with the n_left argument, to set an unbalanced moving average. The n_right defines the number of negative lags to includes in the moving average.
double	A single integer, an optional argument. If not NULL (by default), will apply a second moving average process on the initial moving average output
plot	A boolean, if TRUE will plot the results
show_legend	A boolean, if TRUE will show the plot legend
multiple	A boolean, if TRUE (and n > 1) will create multiple plots, one for each moving average degree. By default is set to FALSE
separate	A boolean, if TRUE will separate the orignal series from the moving average output
margin	A numeric, set the plot margin when using the multiple or/and separate option, default value is 0.03
title	A character, if not NULL (by default), will use the input as the plot title
Xtitle	A character, if not NULL (by default), will use the input as the plot x - axis title
Ytitle	A character, if not NULL (by default), will use the input as the plot y - axis title

Details

A one-side moving averages (also known as simple moving averages) calculation for Y[t] (observation Y of the series at time t):

MA[t|n] = (Y[t-n] + Y[t-(n-1)] +...+ Y[t]) / (n + 1),

where n defines the number of consecutive observations to be used on each rolling window along with the current observation

Similarly, a two-sided moving averages with an order of (2*n + 1) for Y[t]:

MA[t|n] = (Y[t-n] + Y[t-(n-1)] +...+ Y[t] +...+ Y[t+(n-1)] + Y[t+n]) / (2*n + 1)

Unbalanced moving averages with an order of (k1 + k2 + 1) for observation Y[t]:

MA[t|k1 & k2] = (Y[t-k1] + Y[t-(k1-1)] +...+ Y[t] +...+ Y[t+(k2-1)] + Y[t+k2]) / (k1 + k2 + 1)

The unbalanced moving averages is a special case of two-sides moving averages, where k1 and k2 represent the number of past and future periods, respectively to be used in each rolling window, and k1 != k2 (otherwise it is a normal two-sided moving averages function)

Value

A list with the original series, the moving averages outputs and the plot

Examples

```
## Not run:
# A one-side moving average order of 7
USgas_MA7 <- ts_ma(USgas, n_left = 6, n = NULL)

# A two-sided moving average order of 13
USgas_two_side_MA <- ts_ma(USgas, n = 6)

# Unbalanced moving average of order 12
 USVSales_MA12 <- ts_ma(USVSales, n_left = 6, n_right = 5, n = NULL,
 title = "US Monthly Total Vehicle Sales - MA",
 Ytitle = "Thousand of Units")

# Adding double MA of order 2 to balanced the series:
USVSales_MA12 <- ts_ma(USVSales, n_left = 6, n_right = 5, n = NULL,
 double = 2,
 title = "US Monthly Total Vehicle Sales - MA",
 Ytitle = "Thousand of Units")

# Adding several types of two-sided moving averages along with the unblanced
# Plot each on a separate plot
USVSales_MA12 <- ts_ma(USVSales, n_left = 6, n_right = 5, n = c(3, 6, 9),
double = 2, multiple = TRUE,
title = "US Monthly Total Vehicle Sales - MA",
Ytitle = "Thousand of Units")

## End(Not run)
```

ts_plot *Plotting Time Series Objects*

Description

Visualization functions for time series object

Usage

```
ts_plot(ts.obj, line.mode = "lines", width = 2, dash = NULL,
  color = NULL, slider = FALSE, type = "single", Xtitle = NULL,
  Ytitle = NULL, title = NULL, Xgrid = FALSE, Ygrid = FALSE)
```

Arguments

ts.obj	A univariate or multivariate time series object of class "ts", "mts", "zoo", "xts", or any data frame object with a minimum of one numeric column and either a Date or POSIXt class column
line.mode	A plotly argument, define the plot type, c("lines", "lines+markers", "markers")
width	An Integer, define the plot width, default is set to 2
dash	A plotly argument, define the line style, c(NULL, "dot", "dash")
color	The color of the plot, support both name and expression
slider	Logic, add slider to modify the time axis (default set to FALSE)
type	A character, optional, if having multiple tims series object, will plot all series in one plot when set to "single" (default), or plot each series on a separate plot when set to "multiple"
Xtitle	A character, set the X axis title, default set to NULL
Ytitle	A character, set the Y axis title, default set to NULL
title	A character, set the plot title, default set to NULL
Xgrid	Logic,show the X axis grid if set to TRUE
Ygrid	Logic,show the Y axis grid if set to TRUE

Examples

```
data(USVSales)
ts_plot(USVSales)

# adding slider
ts_plot(USVSales, slider = TRUE)
```

ts_polar *Polor Plot for Time Series Object*

Description

Polor plot for time series object (ts, zoo, xts), currently support only monthly and quarterly frequency

Usage

```
ts_polar(ts.obj, title = NULL, width = 600, height = 600,
  left = 25, right = 25, top = 25, bottom = 25)
```

Arguments

ts.obj	A univariate time series object of a class "ts", "zoo" or "xts" (support only series with either monthly or quarterly frequency)
title	Add a title for the plot, default set to NULL
width	The widht of the plot in pixels, default set to 600
height	The height of the plot pixels, default set to 600
left	Set the left margin of the plot in pixels, default set to 25
right	Set the right margin of the plot in pixels, default set to 25
top	Set the top margin of the plot in pixels, default set to 25
bottom	Set the bottom margin of the plot in pixels, default set to 25

Examples

```
data(USgas)
ts_polar(USgas)
```

ts_quantile *Quantile Plot for Time Series*

Description

A quantile plot of time series data, allows the user to display a quantile plot of a series by a subset period

Usage

```
ts_quantile(ts.obj, upper = 0.75, lower = 0.25, period = NULL,
  n = 1, title = NULL, Xtitle = NULL, Ytitle = NULL)
```

Arguments

`ts.obj`	A univariate time series object of a class "zoo", "xts", or data frame family ("data.frame", "data.table", "tbl")
`upper`	A numeric value between 0 and 1 (excluding 0, and greater than the "lower" argument) set the upper bound of the quantile plot (using the "probs" argument of the `quantile` function). By default set to 0.75
`lower`	A numeric value between 0 and 1 (excluding 1, and lower than the "upper" argument) set the upper bound of the quantile plot (using the "probs" argument of the `quantile` function). By default set to 0.25
`period`	A character, set the period level of the data for the quantile calculation and plot representation. Must be one level above the input frequency (e.g., an hourly data can represent by daily, weekdays, monthly, quarterly and yearly). Possible options c("daily", "weekdays", "monthly", "quarterly", "yearly")
`n`	An integer, set the number of plots rows to display (by setting the nrows argument in the `subplot` function), must be an integer between 1 and the frequency of the period argument.
`title`	A character, set the plot title, default set to NULL
`Xtitle`	A character, set the X axis title, default set to NULL
`Ytitle`	A character, set the Y axis title, default set to NULL

Examples

```
## Not run:

# Loading the UKgrid package to pull a multie seasonality data
require(UKgrid)

UKgrid_half_hour <- extract_grid(type = "xts", aggregate = NULL)

# Plotting the quantile of the UKgrid dataset
# No period subset
ts_quantile(UKgrid_half_hour,
period = NULL,
title = "The UK National Grid Net Demand for Electricity - Quantile Plot")

# Plotting the quantile of the UKgrid dataset
# Using a weekday subset
ts_quantile(UKgrid_half_hour,
period = "weekdays",
title = "The UK National Grid Net Demand for Electricity - by Weekdays")

# Spacing the plots by setting the
# number of rows of the plot to 2
ts_quantile(UKgrid_half_hour,
period = "weekdays",
title = "The UK National Grid Net Demand for Electricity - by Weekdays",
n = 2)
```

```
## End(Not run)
```

`ts_reshape` *Transform Time Series Object to Data Frame Format*

Description

Transform time series object into data frame format

Usage

```
ts_reshape(ts.obj, type = "wide", frequency = NULL)
```

Arguments

`ts.obj`	a univariate time series object of a class "ts", "zoo", "xts", and the data frame family (data.frame, data.table, tbl, tibble, etc.) with a Date column and at least one numeric column. This function support time series objects with a daily, weekly, monthly or quarterly frequencies
`type`	The reshape type - "wide" set the years as the columns and the cycle units (months or quarter) as the rows, or "long" split the time object to year, cycle unit and value
`frequency`	An integer, define the series frequency when more than one option is avaiable and the input is one of the data frame family. If set to NULL will use the first option by default when applicable - daily = c(7, 365)

Examples

```
data(USgas)
USgas_df <- ts_reshape(USgas)
```

`ts_seasonal` *Seasonality Visualization of Time Series Object*

Description

Visualize time series object by it periodicity, currently support time series with daily, monthly and quarterly frequency

Usage

```
ts_seasonal(ts.obj, type = "normal", title = NULL, Ygrid = TRUE,
  Xgrid = TRUE, last = NULL, palette = "Set1",
  palette_normal = "viridis")
```

Arguments

`ts.obj`	Input object, either a univariate time series object of a class "ts", "zoo", "xts", or a data frame object of a class "data.frame", "tbl", "data.table" as long as there is at least one "Date"/"POSIXt" and a "numeric" objects (if there are more then one, by defualt will use the first of each). Currently support only daily, weekly, monthly, and quarterly frequencies
`type`	The type of the seasonal plot - "normal" to split the series by full cycle units, or "cycle" to split by cycle units (applicable only for monthly and quarterly data), or "box" for box-plot by cycle units, or "all" for all the three plots together
`title`	Plot title - Character object
`Ygrid`	Logic,show the Y axis grid if set to TRUE (default)
`Xgrid`	Logic,show the X axis grid if set to TRUE (defualt)
`last`	Subset the data to the last number of observations
`palette`	A character, the color palette to be used when the "cycle" or "box" plot are being selected (by setting the type to "cycle", "box", or "all"). All the palettes in the RColorBrewer and viridis packages are available to be use, the default option is "Set1" from the RColorBrewer package
`palette_normal`	A character, the color palette to be used when the "normal" plot is being selected (by setting the type to "normal" or "all"). All the palettes in the RColorBrewer and viridis packages are available to be used, the default palette is "viridis" from the RColorBrewer package

Examples

```
data(USgas)
ts_seasonal(USgas)

# Seasonal box plot
ts_seasonal(USgas, type = "box")

# Plot all the types
ts_seasonal(USgas, type = "all")
```

`ts_split` *Split Time Series Object for Training and Testing Partitions*

Description

Split a time series object into training and testing partitions

Usage

```
ts_split(ts.obj, sample.out = NULL)
```

Arguments

ts.obj
: A univariate time series object of a class "ts" or "tsibble"

sample.out
: An integer, set the number of periods of the testing or sample out partition, defualt set for 30 percent of the lenght of the series

Examples

```
## Split the USgas dataset into training and testing partitions

## Set the last 12 months as a testing partition

## and the rest as a training partition

data(USgas, package = "TSstudio")

split_USgas <- ts_split(ts.obj = USgas, sample.out = 12)

training <- split_USgas$train
testing <- split_USgas$test

length(USgas)

length(training)
length(testing)
```

---

ts_sum *Summation of Multiple Time Series Objects*

Description

A row sum function for multiple time series object ("mts"), return the the summation of the "mts" object as a "ts" object

Usage

```
ts_sum(mts.obj)
```

Arguments

mts.obj
: A multivariate time series object of a class "mts"

Examples

```
x <- matrix(c(1:100, 1:100, 1:100), ncol = 3)
mts.obj <- ts(x, start = c(2000, 1), frequency = 12)
ts_total <- ts_sum(mts.obj)
```

`ts_surface` *3D Surface Plot for Time Series*

Description

3D surface plot for time series object by it periodicity (currently support only monthly and quarterly frequency)

Usage

```
ts_surface(ts.obj)
```

Arguments

`ts.obj`	a univariate time series object of a class "ts", "zoo" or "xts" (support only series with either monthly or quarterly frequency)

Examples

```
ts_surface(USgas)
```

`ts_to_prophet` *Transform Time Series Object to Prophet input*

Description

Transform a time series object to Prophet data frame input format

Usage

```
ts_to_prophet(ts.obj, start = NULL)
```

Arguments

`ts.obj`	A univariate time series object of a class "ts", "zoo", "xts", with a daily, weekly, monthly , quarterly or yearly frequency
`start`	A date object (optional), if the starting date of the series is known. Otherwise, the date would be derive from the series index

Value

A data frame object

Examples

```
data(USgas)

ts_to_prophet(ts.obj = USgas)

# If known setting the start date of the input object

ts_to_prophet(ts.obj = USgas, start = as.Date("2000-01-01"))
```

`USgas` *US monthly natural gas consumption*

Description

US monthly natural gas consumption: 2000 - 2019. Units: Billion Cubic Feet

Usage

```
USgas
```

Format

Time series data - 'ts' object

Source

U.S. Bureau of Transportation Statistics, Natural Gas Consumption [NATURALGAS], retrieved from FRED, Federal Reserve Bank of St. Louis; https://fred.stlouisfed.org/series/NATURALGAS, January 7, 2018.

Examples

```
ts_plot(USgas)
ts_seasonal(USgas, type = "all")
```

USUnRate — *US Monthly Civilian Unemployment Rate*

Description

US monthly civilian unemployment rate: 1948 - 2019. Units: Percent

Usage

```
USUnRate
```

Format

Time series data - 'ts' object

Source

U.S. Bureau of Labor Statistics, Civilian Unemployment Rate [UNRATENSA], retrieved from FRED, Federal Reserve Bank of St. Louis; https://fred.stlouisfed.org/series/UNRATENSA, January 6, 2018.

Examples

```
ts_plot(USUnRate)
ts_seasonal(USUnRate)
```

USVSales — *US Monthly Total Vehicle Sales*

Description

US monthly total vehicle sales: 1976 - 2019. Units: Thousands of units

Usage

```
USVSales
```

Format

Time series data - 'ts' object

Source

U.S. Bureau of Economic Analysis, Total Vehicle Sales [TOTALNSA], retrieved from FRED, Federal Reserve Bank of St. Louis; https://fred.stlouisfed.org/series/TOTALNSA, January 7, 2018.

Examples

```
ts_plot(USVSales)
ts_seasonal(USVSales)
```

US_indicators *US Key Indicators - data frame format*

Description

Monthly total vehicle sales and unemployment rate: 1976 - 2019. Units: Dollars per Kg

Usage

```
US_indicators
```

Format

Time series data - 'data.frame' object

Source

U.S. Bureau of Economic Analysis, Total Vehicle Sales [TOTALNSA], retrieved from FRED, Federal Reserve Bank of St. Louis; https://fred.stlouisfed.org/series/TOTALNSA, January 7, 2018. U.S. Bureau of Labor Statistics, Civilian Unemployment Rate [UNRATENSA], retrieved from FRED, Federal Reserve Bank of St. Louis; https://fred.stlouisfed.org/series/UNRATENSA, January 6, 2018.

Examples

```
ts_plot(US_indicators)
```

---

xts_to_ts *Converting 'xts' object to 'ts' object*

Description

Converting 'xts' object to 'ts' object

Usage

```
xts_to_ts(xts.obj, frequency = NULL, start = NULL)
```

Arguments

`xts.obj`	A univariate 'xts' object
`frequency`	A character, optional, if not NULL (default) set the frequency of the series
`start`	A Date or POSIXct/lt object, optional, can be used to set the starting date or time of the series

Examples

```
data(Michigan_CS)
class(Michigan_CS)
ts_plot(Michigan_CS)
Michigan_CS_ts <- xts_to_ts(Michigan_CS)
ts_plot(Michigan_CS_ts)


# Defining the frequency and starting date of the series
Michigan_CS_ts1 <- xts_to_ts(Michigan_CS, start = as.Date("1980-01-01"), frequency = 12 )
ts_plot(Michigan_CS_ts1)
```

---

zoo_to_ts *Converting 'zoo' object to 'ts' object*

Description

Converting 'zoo' object to 'ts' object

Usage

```
zoo_to_ts(zoo.obj)
```

Arguments

`zoo.obj`	a univariate 'zoo' object

Examples

```
data("EURO_Brent", package = "TSstudio")
class(EURO_Brent)
ts_plot(EURO_Brent)
EURO_Brent_ts <- zoo_to_ts(EURO_Brent)
class(EURO_Brent_ts)
ts_plot(EURO_Brent_ts)
```

Index