Package ‘lazytrade’
December 15, 2021

Type Package
Title Learn Computer and Data Science using Algorithmic Trading
Version 0.5.3
Author Vladimir Zhbanko
Maintainer Vladimir Zhbanko <vladimir.zhbanko@gmail.com>
Description Provide sets of functions and methods to learn and practice data science using idea of algorithmic trading.
Main goal is to process information within "Decision Support System" to come up with analysis or predictions.
There are several utilities such as dynamic and adaptive risk management using reinforcement learning
and even functions to generate predictions of price changes using pattern recognition deep regression learning.
Summary of Methods used: Awesome H2O tutorials: <https://github.com/h2oai/awesome-h2o>,
Market Type research of Van Tharp Institute: <https://www.vantharp.com/>,
Reinforcement Learning R package: <https://CRAN.R-project.org/package=ReinforcementLearning>.
License MIT + file LICENSE
URL https://vladdsm.github.io/myblog_attempt/topics/lazy%20trading/,
https://github.com/vzhomeexperiments/lazytrade
BugReports https://github.com/vzhomeexperiments/lazytrade/issues
Encoding UTF-8
LazyData true
RoxygenNote 7.1.2
Imports readr, stringr, dplyr, tibble, lubridate, ggplot2, grDevices,
h2o, ReinforcementLearning, openssl, stats, cluster, lifecycle
Suggests testthat (>= 2.1.0), covr, magrittr, data.table, bit64
Depends R (>= 3.6.0)
NeedsCompilation no
Repository CRAN
Date/Publication 2021-12-15 18:10:07 UTC
### R topics documented:

- `aml_collect_data` .................................................. 3
- `aml_consolidate_results` ........................................ 5
- `aml_make_model` .................................................... 7
- `aml_score_data` .................................................... 11
- `aml_simulation` .................................................... 13
- `aml_test_model` .................................................... 15
- `check_if_optimize` ................................................ 17
- `create_labelled_data` ............................................ 19
- `create_transposed_data` ......................................... 20
- `data_trades` ........................................................ 21
- `decrypt_mykeys` .................................................... 22
- `DFR` ................................................................. 23
- `dlog` ................................................................. 23
- `encrypt_api_key` ................................................... 24
- `EURUSD15X75` ....................................................... 25
- `evaluate_macroeconomic_event` ............................... 26
- `get_profit_factorDF` ............................................. 27
- `import_data` ........................................................ 28
- `indicator_dataset` ................................................ 29
- `macd_100` ........................................................... 30
- `macd_df` ............................................................. 31
- `macd_ML60M` ......................................................... 31
- `mt_evaluate` ........................................................ 32
- `mt_import_data` .................................................... 33
- `mt_make_model` ..................................................... 34
- `mt_stat_evaluate` .................................................. 37
- `mt_stat_transf` ..................................................... 39
- `opt_aggregate_results` ........................................... 41
- `opt_create_graphs` ............................................... 42
- `policy_tr_systDF` ................................................ 43
- `price_dataset` ....................................................... 43
- `price_dataset_big` ............................................... 44
- `profit_factorDF` .................................................. 44
- `profit_factor_data` ............................................... 45
- `result_prev` ........................................................ 45
- `result_R` ............................................................ 46
- `result_R1` .......................................................... 46
- `rl_generate_policy` ............................................... 47
- `rl_generate_policy_mt` ......................................... 48
- `rl_log_progress` ................................................... 49
- `rl_log_progress_mt` ............................................... 50
- `rl_record_policy` .................................................. 51
- `rl_record_policy_mt` .............................................. 52
- `rl_write_control_parameters` ................................... 53
- `rl_write_control_parameters_mt` ................................ 54
- `test_data_pattern` ................................................ 55
aml_collect_data

Function to read, transform, aggregate and save data for further re-training of regression model for a single asset

Description

Function is collecting data from the csv files. Data objects are transformed to be suitable for Regression Modelling. Price change will be in the column 'LABEL', column X1 will keep the time index. Result will be written to a new or aggregated to the existing '.rds' file.

Function is keeping generated dataset to be not larger than specified by the user.

[Stable]

Usage

aml_collect_data(
  indicator_dataset,
  symbol,
  timeframe = 60,
  path_data,
  max_nrows = 2500
)

Arguments

  indicator_dataset  Dataset containing assets indicator which pattern will be used as predictor
  symbol             Character symbol of the asset for which to train the model
  timeframe          Data timeframe e.g. 1 min
  path_data          Path where the aggregated historical data is stored, if exists in rds format
  max_nrows          Integer, Maximum number of rows to collect
Details

Function is not handling shift of the price and indicator datasets.
This function is relying on the data collection from the dedicated data robot Other 'aml_*' functions will work based on the data processed by this function

Value

Function is writing files into Decision Support System folder, mainly file object with the model

Author(s)

(C) 2020, 2021 Vladimir Zhbanko

Examples

```r
# write examples for the function
library(dplyr)
library(readr)
library(lubridate)
library(lazytrade)
library(magrittr)

# sample dataset
ind = system.file("extdata", "AI_RSIADXUSDJPY60.csv", package = "lazytrade") %>% read_csv(col_names = FALSE)

# convert to POSIX format
ind$X1 <- ymd_hms(ind$X1)

# create temporary path (check output of tempdir() to check the result)
path_data <- normalizePath(tempdir(), winslash = "/")

# add tick data to the folder
tick = system.file("extdata", "TickSize.AI_RSIADX.csv", package = "lazytrade") %>% read_csv(col_names = FALSE)

write_csv(tick, file.path(path_data, "TickSize.AI_RSIADX.csv"), col_names = FALSE)

# data transformation using the custom function for one symbol
aml_collect_data(indicator_dataset = ind,
                 symbol = 'USDJPY',
                 timeframe = 60,
                 path_data = path_data)
```
Description

Function is designed to evaluate test results of multiple models. This is done to select only group of models with the best performance. In addition, function will provide facility to generate logs hence to allow tracking of long term model performance

[Experimental]

Usage

aml_consolidate_results(
  timeframe = 15,
  used_symbols,
  path_model,
  path_sbmx,
  path_sbxs,
  min_quality = 0.75,
  get_quantile = FALSE,
  log_results = FALSE,
  path_logs = NULL
)

Arguments

timeframe Integer, Data timeframe interval in minutes e.g. 60 min

used_symbols Vector, containing several financial instruments that were previously used to test the model

path_model String, User path where the test results were stored

path_sbmx String, User path to the sandbox where file with strategy test results should be written (master terminal)

path_sbxs String, User path to the sandbox where file with strategy test results should be written (slave terminal)

min_quality Double, value typically from 0.25 to 0.95 to select the min threshold value

get_quantile Bool, whether or not function should return an overall value of model performances this will be used to conditionally update only less performant models

log_results Bool, option to write logs with cumulative results obtained for all models

path_logs String, User path to the folder where to log results

Details

Provide a modular facility to aggregate and update files, write performance logs.
Value

Function is writing files into Decision Support System folders

Author(s)

(C) 2021 Vladimir Zhbanko

Examples

```r
library(dplyr)
library(magrittr)
library(readr)
library(lazytrade)
library(stats)

testpath <- normalizePath(tempdir(), winslash = "/")
path_model <- file.path(testpath, "_Model")
path_sbxm <- file.path(testpath, "_T1")
path_sbxs <- file.path(testpath, "_T3")
path_logs <- file.path(testpath, "_LOGS")
dir.create(path_model)
dir.create(path_sbxm)
dir.create(path_sbxs)
dir.create(path_logs)

file.copy(from = system.file("extdata", "StrTest-EURGBP15.csv", package = "lazytrade"),
          to = file.path(path_model, "StrTest-EURGBP15.csv"), overwrite = TRUE)

file.copy(from = system.file("extdata", "StrTest-EURJPY15.csv", package = "lazytrade"),
          to = file.path(path_model, "StrTest-EURJPY15.csv"), overwrite = TRUE)

file.copy(from = system.file("extdata", "StrTest-EURNZD15.csv", package = "lazytrade"),
          to = file.path(path_model, "StrTest-EURNZD15.csv"), overwrite = TRUE)

file.copy(from = system.file("extdata", "StrTest-EURUSD15.csv", package = "lazytrade"),
          to = file.path(path_model, "StrTest-EURUSD15.csv"), overwrite = TRUE)

Pairs <- c("EURGBP", "EURJPY", "EURNZD", "EURUSD")

aml_consolidate_results(timeframe = 15,
                         used_symbols = Pairs,
                         path_model = path_model,
                         path_sbxm = path_sbxm,
                         path_sbxs = path_sbxs,
                         min_quality = 0.75,
                         get_quantile = FALSE)

aml_consolidate_results(timeframe = 15,
                         used_symbols = Pairs,
```
aml_make_model

Function to train Deep Learning regression model for a single asset

Description

Function is training h2o deep learning model to match future prices of the asset to the indicator pattern. Main idea is to be able to predict future prices by solely relying on the recently retrieved indicator pattern. This is to mimic traditional algorithmic systems based on the indicator rule and to attempt automated optimization with AI.

[Experimental]

Usage

aml_make_model(
    symbol,
    timeframe = 60,
    path_model,
    path_data,
    force_update = FALSE,
    objective_test = FALSE,
    num_nn_options = 12,
    fixed_nn_struct = c(100, 100),
    num_epoch = 100,
    num_bars_test = 600,
    num_bars_ahead = 34,
    num_cols_used = 16,
    min_perf = 0.3
)
Arguments

symbol  Character symbol of the asset for which to train the model

timeframe  Integer, value in minutes, e.g. 60 min

path_model  Path where the models shall be stored

path_data  Path where the aggregated historical data is stored, if exists in rds format

force_update  Boolean, by setting this to TRUE function will generate new model (useful after h2o engine update)

objective_test  Boolean, option to use trading objective test as a parameter to validate best model, defaults to FALSE

num_nn_options  Integer, value from 0 to 24 or more. Used to change number of variants of the random neural network structures, when value is 0 uses fixed structure. Higher number may lead to long code execution. Select value multiple of 3 otherwise function will generate warning. E.g. 12, 24, 48, etc

fixed_nn_struct  Integer vector with numeric elements, see par hidden in ?h2o.deeplearning, default value is c(100,100). Note this will only work if num_nn_options is 0

num_epoch  Integer, see parameter epochs in ?h2o.deeplearning, default value is 100. Higher number may lead to long code execution

num_bars_test  Integer, value of bars used for model testing

num_bars_ahead  Integer, value to specify how far should the function predict. Default 34 bars.

num_cols_used  Integer, number of columns to use for training the model, defaults to 16

min_perf  Double, value greater than 0. Used to set minimum value of model performance. Higher value will increase computation time

Details

Deep learning model structure is obtained from several random combinations of neurons within 3 hidden layers of the network. The most accurate model configuration will be automatically selected based either RMSE or Objective Test. In addition, the function will check if there is a need to update the model. To do that function will check results of the function aml_test_model.R.

Function is using the dataset prepared by the function aml_collect_data.R. Note that function will start to train the model as soon as there are more than 1000 rows in the dataset

Value

Function is writing a file object with the best Deep Learning Regression model

Author(s)

(C) 2020, 2021 Vladimir Zhbanko
```
library(dplyr)
library(readr)
library(h2o)
library(lazytrade)
library(lubridate)
library(magrittr)

path_model <- normalizePath(tempdir(), winslash = "/")
path_data <- normalizePath(tempdir(), winslash = "/")

ind = system.file("extdata", "AI_RSIADXUSDJPY60.csv",
                  package = "lazytrade") %>% read_csv(col_names = FALSE)

ind$X1 <- ymd_hms(ind$X1)

tick = system.file("extdata", "TickSize_AI_RSIADX.csv",
                   package = "lazytrade") %>% read_csv(col_names = FALSE)

write_csv(tick, file.path(path_data, "TickSize_AI_RSIADX.csv"), col_names = FALSE)

# data transformation using the custom function for one symbol
aml_collect_data(indicator_dataset = ind,
                 symbol = '/quotesingle.VarUSDJPY/quotesingle.Var',
                 timeframe = 60,
                 path_data = path_data)

# dataset will be written to the temp directory

# start h2o engine
h2o.init(nthreads = 2)

# performing Deep Learning Regression using 2 random neural network structures and objective test
aml_make_model(symbol = '/quotesingle.VarUSDJPY/quotesingle.Var',
                timeframe = 60,
                path_model = path_model,
                path_data = path_data,
                force_update=FALSE,
                objective_test = TRUE,
                num_nn_options = 6,
                num_epoch = 10,
                min_perf = 0,
                num_bars_test = 600,
                num_bars_ahead = 34,
                num_cols_used = 16)

# performing DL Regression using 2 random neural network structures
```
AML make model

# with objective test, all columns
aml_make_model(symbol = 'USDJPY',
              timeframe = 60,
              path_model = path_model,
              path_data = path_data,
              force_update=FALSE,
              objective_test = TRUE,
              num_nn_options = 6,
              num_epoch = 10,
              min_perf = 0,
              num_bars_test = 600,
              num_bars_ahead = 34,
              num_cols_used = 0)

# performing Deep Learning Regression using the custom function
aml_make_model(symbol = 'USDJPY',
              timeframe = 60,
              path_model = path_model,
              path_data = path_data,
              force_update=FALSE,
              objective_test = FALSE,
              num_nn_options = 6,
              num_epoch = 10,
              min_perf = 0,
              num_bars_test = 600,
              num_bars_ahead = 34,
              num_cols_used = 16)

# performing Deep Learning Regression, fixed mode
aml_make_model(symbol = 'USDJPY',
              timeframe = 60,
              path_model = path_model,
              path_data = path_data,
              force_update=TRUE,
              num_nn_options = 0,
              fixed_nn_struct = c(100, 100),
              num_epoch = 10,
              min_perf = 0)

# stop h2o engine
h2o.shutdown(prompt = FALSE)

# set delay to insure h2o unit closes properly before the next test
Sys.sleep(5)
Description

Function is using the latest data from the financial assets indicator pattern and deep learning model. Prediction is a future price change for that asset

[Stable]

Usage

aml_score_data(symbol, timeframe, path_model, path_data, path_sbxm, path_sbxs)

Arguments

symbol Character symbol of the asset for which the model shall predict
timeframe Data timeframe e.g. 60 min
path_model Path where the models are be stored
path_data Path where the aggregated historical data is stored, if exists in rds format
path_sbxm Path to the sandbox where file with predicted price should be written (master terminal)
path_sbxs Path to the sandbox where file with predicted price should be written (slave terminal)

Details

Performs fresh data reading from the rds file

Value

Function is writing file into Decision Support System folder, mainly file with price change prediction in pips

Author(s)

(C) 2020 Vladimir Zhbanko

Examples

# test of function aml_make_model is duplicated here
library(dplyr)
library(readr)
library(lubridate)
library(h2o)
library(magrittr)
library(lazytrade)

path_model <- normalizePath(tempdir(), winslash = "/")
path_data <- normalizePath(tempdir(), winslash = "/")

ind = system.file("extdata", "AI_RSIADXUSDJPY60.csv", package = "lazytrade") %>% read_csv(col_names = FALSE)

ind$X1 <- ymd_hms(ind$X1)
write_csv(ind, file.path(path_data, "AI_RSIADXUSDJPY60.csv"), col_names = FALSE)

# add tick data to the folder
tick = system.file("extdata", "TickSize_AI_RSIADX.csv", package = "lazytrade") %>% read_csv(col_names = FALSE)
write_csv(tick, file.path(path_data, "TickSize_AI_RSIADX.csv"), col_names = FALSE)

# data transformation using the custom function for one symbol
aml_collect_data(indicator_dataset = ind, symbol = quotesingle.Var USDJPY quotesingle.Var, timeframe = 60, path_data = path_data)

# start h2o engine (using all CPU's by default)
h2o.init(nthreads = 2)

# performing Deep Learning Regression using the custom function
aml_make_model(symbol = quotesingle.Var USDJPY quotesingle.Var, timeframe = 60, path_model = path_model, path_data = path_data, force_update = FALSE, num_nn_options = 2)

path_sbxm <- normalizePath(tempdir(), winslash = "/")
path_sbxs <- normalizePath(tempdir(), winslash = "/")

# score the latest data to generate predictions for one currency pair
aml_score_data(symbol = quotesingle.Var USDJPY quotesingle.Var, timeframe = 60, path_model = path_model, path_data = path_data, path_sbxm = path_sbxm, path_sbxs = path_sbxs)

# stop h2o engine
aml_simulation

h2o.shutdown(prompt = FALSE)

#set delay to insure h2o unit closes properly before the next test
Sys.sleep(5)

---

aml_simulation:

*Function to simulate multiple input structures*

---

**Description**

Function is designed to evaluate several different inputs.

*[Experimental]*

**Usage**

```r
aml_simulation(
    timeframe = 60,
    path_sim_input,
    path_sim_result,
    par_simulate1 = 10,
    par_simulate2 = 16,
    demo_mode = FALSE
)
```

**Arguments**

- **timeframe**: Integer, Data timeframe e.g. 60 min. This will be equal to 1 bar
- **path_sim_input**: String, Path to the folder where csv files will be placed, typically AI_RSIADXAUDCAD60.csv
- **path_sim_result**: String, Path to the folder where all results from simulations shall be written
- **par_simulate1**: Integer, Parameter that can be used in simulation
- **par_simulate2**: Integer, Parameter that can be used in simulation
- **demo_mode**: Boolean, Simplify function test. When TRUE no simulation will be made

**Details**

Function is using several other functions to perform sets of operations designed to test several inputs. Designed to validate model settings.

Update: New function structure to allow quicker simulation of parameters to find best performing input.
Value

Function is writing file into Decision Support System folders

Author(s)

(C) 2021 Vladimir Zhbanko

Examples

```r
library(dplyr)
library(magrittr)
library(readr)
library(h2o)
library(lazytrade)
library(lubridate)
library(stats)

path_input <- normalizePath(tempdir(), winslash = "/")
path_sim_input <- file.path(path_input, "path_sim_input")
dir.create(path_sim_input)
path_sim_result <- file.path(path_input, "path_sim_result")
dir.create(path_sim_result)

file.copy(from = system.file("extdata", "AI_RSIADXCADCHF60.csv", package = "lazytrade"),
          to = file.path(path_sim_input, "AI_RSIADXCADCHF60.csv"), overwrite = TRUE)
file.copy(from = system.file("extdata", "AI_RSIADXEURNZD60.csv", package = "lazytrade"),
          to = file.path(path_sim_input, "AI_RSIADXEURNZD60.csv"), overwrite = TRUE)

# start h2o engine
h2o.init(nthreads = 2)

# simulation of different epoch values
aml_simulation(timeframe = 60,
               path_sim_input = path_sim_input,
               path_sim_result = path_sim_result,
               par_simulate1 = 10,
               par_simulate2 = 10,
               demo_mode = FALSE)

Sys.sleep(5)
# stop h2o engine
h2o.shutdown(prompt = FALSE)

# set delay to insure h2o unit closes properly before the next test
Sys.sleep(5)
```
Description

Function is designed to test the trading decision generated by the Deep learning regression model. It is doing so by simulating trading strategy outcome. The outcome of this function will be also used to define best trigger to join into the trading opportunity

[Experimental]

Usage

```r
aml_test_model(  
  symbol,  
  num_bars,  
  timeframe,  
  path_model,  
  path_data,  
  path_sbxm = path_sbxm,  
  path_sbxs = path_sbxs  
)
```

Arguments

- **symbol**: Character symbol of the asset for which to train the model
- **num_bars**: Integer, Number of (rows) bars used to test the model
- **timeframe**: Integer, Data timeframe e.g. 60 min. This will be equal to 1 bar
- **path_model**: String, User path where the models are be stored
- **path_data**: String, User path where the aggregated historical data is stored, if exists in rds format
- **path_sbxm**: String, User path to the sandbox where file with strategy test results should be written (master terminal)
- **path_sbxs**: String, User path to the sandbox where file with strategy test results should be written (slave terminal)

Details

Function is reading price data and corresponding indicator. Starting from the trained model function will test the trading strategy using simplified trading approach. Trading approach will entail using the last available indicator data, predict the price change for every row, trade will be simulating by holding the asset for 3, 5, 10 and 34 hours. Several trigger points will be evaluated selecting the most optimal trading trigger. Function is writing most optimal decision into *.csv file Such file will be used by the function aml_make_model.R to decide whether model must be updated...
Value

Function is writing file into Decision Support System folders

Author(s)

(C) 2020, 2021 Vladimir Zhbanko

Examples

```r
library(dplyr)
library(magrittr)
library(readr)
library(h2o)
library(lazytrade)
library(lubridate)

path_model <- normalizePath(tempdir(),winslash = "/")
path_data <- normalizePath(tempdir(),winslash = "/")

ind = system.file("extdata", "AI_RSIADXUSDJPY60.csv",
                  package = "lazytrade") %> read_csv(col_names = FALSE)

ind$X1 <- ymd_hms(ind$X1)

tick = system.file("extdata", "TickSize_AI_RSIADX.csv",
                   package = "lazytrade") %> read_csv(col_names = FALSE)

write_csv(ind, file.path(path_data, "AI_RSIADXUSDJPY60.csv"), col_names = FALSE)
write_csv(tick, file.path(path_data, "TickSize_AI_RSIADX.csv"), col_names = FALSE)

# data transformation using the custom function for one symbol
aml_collect_data(indicator_dataset = ind,
                 symbol = "USDJPY",
                 timeframe = 60,
                 path_data = path_data)

# start h2o engine
h2o.init(nthreads = 2)

# performing Deep Learning Regression using the custom function
aml_make_model(symbol = "USDJPY",
               timeframe = 60,
               path_model = path_model,
               path_data = path_data,
               force_update=FALSE,
               num_nn_options = 3)
```
check_if_optimize <- function(x, system_list, path_data, path_model, path_sbxm, path_sbxs) {
  # score the latest data to generate predictions for one currency pair
  aml_score_data(symbol = 'USDJPY',
                  timeframe = 60,
                  path_model = path_model,
                  path_data = path_data,
                  path_sbxm = path_sbxm,
                  path_sbxs = path_sbxs)

  # test the results of predictions
  aml_test_model(symbol = 'USDJPY',
                 num_bars = 600,
                 timeframe = 60,
                 path_model = path_model,
                 path_data = path_data,
                 path_sbxm = path_sbxm,
                 path_sbxs = path_sbxs)

  # stop h2o engine
  h2o.shutdown(prompt = FALSE)

  # set delay to insure h2o unit closes properly before the next test
  Sys.sleep(5)
}

check_if_optimize

Function check_if_optimize.

Description

Purpose of this function is to verify trading system functionality by analysing profit factor on the last trades. Whenever trading robot has profit factor value below certain limit function will write a file log indicating which trading systems need to be maintained.

Learn by example how to manipulate data

[Stable]

Usage

check_if_optimize(
  x,
  system_list,
  path_data,
num_trades_to_consider = 3,
profit_factor_limit = 0.7,
write_mode = FALSE
)

Arguments

x • dataframe containing trading results
system_list • dataframe containing a table with magic numbers used by robots. Stored in file Setup.csv
path_data • string, path to the folder where optimization file should be written
num_trades_to_consider • Number of trades to calculate profit factor
profit_factor_limit • Limit below which trading robot is considered not working properly
write_mode • When true function will write result to the file located in the temporary directory

Details

Whenever there will be not enough trades then empty file will be written to the destination

Value

function returns a dataframe with systems that should be optimized

Author(s)

(C) 2019,2021 Vladimir Zhbanko

Examples

library(lazytrade)
library(magrittr)
library(dplyr)
library(readr)
library(lubridate)

path_data <- normalizePath(tempdir(), winslash = "/")

file.copy(from = system.file("extdata", "Setup.csv", package = "lazytrade"),
          to = file.path(path_data, "Setup.csv"), overwrite = TRUE)

system_list <- read_csv(file.path(path_data, "Setup.csv"))

data(profit_factorDF)

# without writing to the file
check_if_optimize(x = profit_factorDF,
    system_list = system_list,
    path_data,
    num_trades_to_consider = 3,
    profit_factor_limit = 0.8,
    write_mode = TRUE)

---

**create_labelled_data**  
*Create labelled data*

**Description**

FUNCTION `create_labelled_data`. PURPOSE: function gets price data of every currency in each column. It is splitting this data by periods and transposes the data. Additionally function is capable to label the data based on the simple logic. Each row will be assigned into 2 categories based on the difference between beginning and end of the row elements. Finally all data will be stacked on top and joined into the table.

Learn by example how to manipulate data

[Superseded]

**Usage**

`create_labelled_data(x, n = 50, type = "regression")`

**Arguments**

- **x**
  - data set containing a table where 1st column is a Time index and other columns containing financial asset price values
- **n**
  - number of rows we intend to split and transpose the data to
- **type**
  - type of the label required. Can be either "classification" or "regression". "classification" will return either "BU" or "BE", "regression" will return the difference between first value and the last value in each row (in pips)

**Details**

see more info in the udemy course self-learning-trading-robot

**Value**

function returns transposed data. One column called 'LABEL' indicate achieved value of the label. Transposed values from every column are stacked one to each other
create_transposed_data

Examples

```r
library(dplyr)
library(magrittr)
library(readr)
library(lazytrade)

# using a sample data
data(price_dataset)

# price change as a label
create_labelled_data(x = price_dataset, n = 75, type = "regression")

# factors 'BU'/ 'BE' as a label
create_labelled_data(x = price_dataset, n = 75, type = "classification")
```

create_transposed_data

Create Transposed Data

Description

PURPOSE: function gets indicator data in each column. Goal is to splitting this data into periods and transpose the data.

[Superseded]

Usage

```r
create_transposed_data(x, n = 50)
```

Arguments

- `x`: data set containing a table where 1st column is a Time index and other columns containing financial asset indicator values
- `n`: number of rows we intend to split and transpose the data

Details

Each column contains records of the indicator value of the assets every column will be split into chunks of n observations and transposed into rows this repeated for all the columns coming up with a matrix. Function works in combination with a function create_labelled_data

Value

function returns transposed data. Transposed values from every column are stacked one to each other
Examples

```r
library(dplyr)
library(magrittr)
library(lazytrade)

# using a sample data
data(indicator_dataset)

create_transposed_data(indicator_dataset, n = 75)
```

---

data_trades Table with Trade results samples

Description

Table with Trade results samples

Usage

data_trades

Format

A dataframe with several columns

- **MagicNumber**: Unique identifiers of the Trading Robots
- **TicketNumber**: Ticket Number of closed position
- **OrderStart Time**: Date and Time when order started
- **OrderClose Time**: Date and Time when order closed
- **Profit**: Monetary result of the trade
- **Symbol**: Symbol of the Asset e.g. EURUSD
- **OrderType**: Order Type 0 - buy, 1 - sell
decrypt_mykeys  

*Function that decrypt encrypted content*

**Description**

Cryptography facility

[Stable]

**Usage**

decrypt_mykeys(path_encrypted_content, path_private_key)

**Arguments**

- path_encrypted_content
  - path to the encrypted content of the API key
- path_private_key
  - path to the private RSA key, should be without password

**Details**

It is possible to generate private/public key pair using R-Studio Project Options Menu. Alternatively possible to use `openssl` R package

**Value**

- a string with decrypted key

**Examples**

```r
library(dplyr)
library(magrittr)
library(openssl)
library(readr)

path_ssh <- normalizePath(tempdir(), winslash = "/")
rsa_keygen() %>% write_pem(path = file.path(path_ssh, "id_api"))
# extract and write your public key
read_key(file = file.path(path_ssh, "id_api"), password = "") %>%
  
"pubkey" %>% write_pem(path = file.path(path_ssh, "id_api.pub"))

path_private_key <- file.path(path_ssh, "id_api")
path_public_key <- file.path(path_ssh, "id_api.pub")

# encrypting string 'my_key'...
encrypt_api_key(api_key = 'my_key', enc_name = 'api_key.enc.rds', path_ssh = path_ssh)

# encrypted content
```
out <- read_rds(file.path(path_ssh, "api_key.enc.rds"))

# Consumer API keys
ConsumerAPIkeys <- decrypt_mykeys(path_encrypted_content = file.path(path_ssh, 
  'api_key.enc.rds'),
  path_private_key = path_private_key)

---

**DFR**  
**Table with predicted price change**

**Description**  
Table with predicted price change

**Usage**  
DFR

**Format**  
A dataframe with one column

"MagicNumber.x"  
Unique identifiers of the Trading Robots from Trade Log

TicketNumber  
Ticket Number of closed position

OrderStart Time  
Date and Time when order started

OrderCloseTime  
Date and Time when order closed

Profit  
Monetary result of the trade

Symbol  
Symbol of the Asset e.g. EURUSD

OrderType  
Order Type 0 - buy, 1 - sell

"CUMSUM_PNL"  
Cumulative sum of Profit and Loss

---

**dlog**  
**Create log difference distribution**

**Description**  
Calculate log distribution and calculate difference within rows

[Stable]

**Usage**  
dlog(x)
Arguments

- matrix with one or more columns

Value

- dataframe

Examples

```r
library(magrittr)
library(lazytrade)
m <- seq(1:1000) %>% as.matrix(10) %>% dlog()
```

Description

Provide easy interface to encrypt the api key. In order to use function simply provide a string with an API key. In addition provide the path to the .ssh folder and names of the private and public keys.

[Stable]

Usage

```r
encrypt_api_key(
  api_key,
  enc_name = "api_key.enc.rds",
  path_ssh = "path_ssh",
  file_rsa = "id_api",
  file_rsa_pub = "id_api.pub"
)
```

Arguments

- `api_key` String with API key
- `enc_name` String with a name of the file with encrypted key. Default name is `api_key.enc.rds`
- `path_ssh` String with path to the file with rsa keys. Same place will be used to store encrypted data
- `file_rsa` String with a name of the file with a private key. Default name is `id_api`
- `file_rsa_pub` String with a name of the file with a public key. Default name is `id_api.pub`

Details

Make sure to clean the history of the R session
Value

 Writes a file with encrypted key

References

 for more info on how to use RSA cryptography in R check my course https://www.udemy.com/course/keep-your-secrets-under-control/?referralCode=5B78D58E7C06AFFD80AE

Examples

```
library(openssl)
library(magrittr)
library(readr)

path_ssh <- normalizePath(tempdir(), winslash = "/")

rsa_keygen() %>% write_pem(path = file.path(path_ssh, 'id_api'))
# extract and write your public key
read_key(file = file.path(path_ssh, 'id_api'), password = "") %>%
  `[["pubkey"] %>% write_pem(path = file.path(path_ssh, 'id_api.pub'))

path_private_key <- file.path(path_ssh, "id_api")
path_public_key <- file.path(path_ssh, "id_api.pub")

# encrypting string 'my_key'
encrypt_api_key(api_key = 'my_key', enc_name = 'api_key.enc.rds', path_ssh = path_ssh)

out <- read_rds(file.path(path_ssh, "api_key.enc.rds"))

# decrypting the password using public data list and private key
api_key <- decrypt_envelope(out$data,
                           out$iv,
                           out$s session,
                           path_private_key, password = "") %>%
                 unserialize()

# outcome of the encryption will be a string 'my_key'
```

---

**EURUSDM15X75**  

*Table with indicator and price change dataset*

**Description**

 Table with indicator and price change dataset

**Usage**

 EURUSDM15X75
evaluate_macroeconomic_event

Function used to evaluate market type situation by reading the file with Macroeconomic Events and writing a trigger to the trading robot

Description

Function is reading the content of the file 01_MacroeconomicEvent.csv. Content of the file can be either 1 or 0. 1 - when Macro Economic event is present, 0 - when it’s not. Function will also read magic number of the trading robots. This is indicated in the file 'Setup.csv'. Final outcome of the function is the series of files written to the destination directory. These files will either enable or disable opening of new positions in the trading robots

[Stable]

Usage

evaluate_macroeconomic_event(  
    setup_file_path,  
    setup_file_name = "Setup.csv",  
    macro_event_path,  
    macro_file_name = "01_MacroeconomicEvent.csv",  
    path_T1,  
    path_T3  
)

Arguments

setup_file_path
    string, path to the folder with Setup.csv file
setup_file_name
    string, name of the file 'Setup.csv'
macro_event_path
    string, path to the folder with a file '01_MacroeconomicEvent.csv'
macro_file_name
    string, name of the file '01_MacroeconomicEvent.csv'
path_T1
    Path of the Terminal 1
path_T3
    Path of the Terminal 3
**get_profit_factorDF**

**Details**

This function is used exclusively with Market Type recognition system. Final evaluation will consist in writing a dedicated file with a simple information:

When Macro economic even is not present:
"Magic","IsEnabled" 8139125,1
or, when Macro economic event is present:
"Magic","IsEnabled" 8139125,0

**Value**

Function will write files indicating to enable or disable trading systems to open new orders

**Examples**

```r
# evaluate data on macroeconomic event (required to start trading)
library(dplyr)
library(readr)
library(lazytrade)

dir <- normalizePath(tempdir(), winslash = "/")

evaluate_macroeconomic_event(setup_file_path = system.file(’extdata’, package = “lazytrade”),
             setup_file_name = “Setup.csv”,
             macro_event_path = system.file(’extdata’, package = “lazytrade”),
             macro_event_name = “01_MacroeconomicEvent.csv”,
             path_T1 = dir, path_T3 = dir)
```

**Description**

Calculation of profit factor using dplyr verbs

[Superseded]

**Usage**

`get_profit_factorDF(x, num_orders)`
import_data

Arguments

x • data frame with orders. Note x must contain MagicNumber and Profit columns!
num_orders • desired number of orders to base profit factor calculation

Value

• Function returns dataframe with column PrFact with calculated profit factor value for each trading robot

Examples

library(lazytrade)
library(dplyr)
library(magrittr)
data(profit_factorDF)
get_profit_factorDF(x = profit_factorDF,
num_orders = 10)

import_data

Import Data file with Trade Logs to R.

Description

Function is capable to import file with executed trades log. Files do not have column headers hence function will take care to name columns as well as to perform relevant cleansing

[Stable]

Usage

import_data(path_sbxm, trade_log_file)

Arguments

path_sbxm • String, Path to the sandbox with the log file where the file with data is written
trade_log_file • String, File name where the order results are written

Value

Function will return the dataframe with trade data and automatically set proper column types
**Examples**

```r
library(lazytrade)
library(dplyr)
library(readr)
library(lubridate)

path_sbxm <- normalizePath(tempdir(), winslash = "/")

file.copy(from = system.file("extdata", "OrdersResultsT1.csv", package = "lazytrade"),
          to = file.path(path_sbxm, "OrdersResultsT1.csv"), overwrite = TRUE)

DFT1 <- import_data(path_sbxm = path_sbxm,
                     trade_log_file = "OrdersResultsT1.csv")
```

---

### indicator_dataset

*Table with indicator dataset*

#### Description

Table with indicator dataset

#### Usage

`indicator_dataset`

#### Format

A dataframe with several columns

- **X1** Date and time of the indicator sample
- **X2-X29** Values of the assets
Description

Table with indicator only used to train model, 128 col 1646 rows

Usage

macd_100

Format

A dataframe with several columns

- **EURUSD**  Values of the macd indicator
- **GBPUSD**  Values of the macd indicator
- **AUDUSD**  Values of the macd indicator
- **NZDUSD**  Values of the macd indicator
- **USDCAD**  Values of the macd indicator
- **USDCHF**  Values of the macd indicator
- **USDJPY**  Values of the macd indicator
- **EURGBP**  Values of the macd indicator
- **EURJPY**  Values of the macd indicator
- **EURCHF**  Values of the macd indicator
- **EURNZD**  Values of the macd indicator
- **EURCAD**  Values of the macd indicator
- **EURAUD**  Values of the macd indicator
- **GBPUSD**  Values of the macd indicator
- **GBPCAD**  Values of the macd indicator
- **GBPCHF**  Values of the macd indicator
- **GBPJPY**  Values of the macd indicator
- **GBPUSD**  Values of the macd indicator
- **AUDCAD**  Values of the macd indicator
- **AUDCHF**  Values of the macd indicator
- **AUDJPY**  Values of the macd indicator
- **AUDNZD**  Values of the macd indicator
- **CADJPY**  Values of the macd indicator
- **CHFJPY**  Values of the macd indicator
- **NZDJPY**  Values of the macd indicator
- **NZDCAD**  Values of the macd indicator
- **NZDCHF**  Values of the macd indicator
- **CADCHF**  Values of the macd indicator
**macd_df**

Table with one column indicator dataset

**Description**

Table with one column indicator dataset

**Usage**

macd_df

**Format**

A dataframe with one column

**CADCHF**  Indicator values of the asset

---

**macd_ML60M**

Table with indicator and market type category used to train model

**Description**

Table with indicator and market type category used to train model

**Usage**

macd_ML60M

**Format**

A dataframe with several columns

**X1-X64**  Values of the macd indicator

**M_T**  Category of Market Type
mt_evaluate  

Function to score data and predict current market type using pre-trained classification model

Description

PURPOSE: Function that uses Deep Learning model and Time Series Column of the dataframe to find out specific market type of the financial asset it will also discard bad result outputting -1 if it is the case

Usage

mt_evaluate(x, path_model, num_cols, timeframe)

Arguments

x
  - dataframe with one column containing asset indicator in the time descending order, typically 64 or more values
path_model
  - String, path to the model
num_cols
  - Integer, number of columns (features) in the final vector input to the model
timeframe
  - Integer, timeframe in Minutes.

Details

it is mandatory to switch on the virtual h2o machine with h2o.init() also to shut it down with h2o.shutdown(prompt = F)

Value

dataframe with predicted value of the market type

Examples

library(h2o)
library(magrittr)
library(dplyr)
library(readr)
library(lazytrade)

path_model <- normalizePath(tempdir(), winslash = "/")
path_data <- normalizePath(tempdir(), winslash = "/")

data(macd_ML60M)

# start h2o engine (using all CPU’s by default)
h2o.init(nthreads = 2)

# performing Deep Learning Regression using the custom function
# this function stores model to the temp location
mt_make_model(indicator_dataset = macd_ML60M,
               num_bars = 64,
               timeframe = 60,
               path_model = path_model,
               path_data = path_data,
               activate_balance = TRUE,
               num_nn_options = 3)

# Use sample data
data(macd_100)

# use one column for testing
x <- macd_100[,2]

mt_evaluate(x = x,
            path_model = path_model,
            num_cols = 64,
            timeframe = 60)

h2o.shutdown(prompt = FALSE)

#set delay to insure h2o unit closes properly before the next test
Sys.sleep(5)

---

**mt_import_data**

Import Market Type related Data to R from the Sandbox

**Description**

Function imports file from the MetaTrader sandbox. Function performs necessary cleansing of the data column types

[Stable]

**Usage**

mt_import_data(path_sbxm, system_number)

**Arguments**

- **path_sbxm**: String, Path to the sandbox with the log file (master terminal)
- **system_number**: magic number id of the trading system
mt_make_model

Function to train Deep Learning Classification model for Market Type recognition

Description

Function is training h2o deep learning model to match classified patterns of the financial indicator. Main idea is to be able to detect Market Type by solely relying on the current indicator pattern. This is in the attempt to evaluate current market type for trading purposes.

Selected Market Periods could be manually classified according to the theory from Van K. Tharp:

1. Bull normal, BUN
2. Bull volatile, BUV
3. Bear normal, BEN
4. Bear volatile, BEV
5. Sideways quiet, RAN
6. Sideways volatile, RAV

For automatic classification, could only be used: BUN, BEN, RAN market types

[Experimental]
Usage

```r
mt_make_model(
  indicator_dataset,
  num_bars = 64,
  timeframe = 60,
  path_model,
  path_data,
  activate_balance = TRUE,
  num_nn_options = 24,
  fixed_nn_struct = c(100, 100),
  num_epoch = 100,
  is_cluster = FALSE
)
```

Arguments

- `indicator_dataset`  
  Data frame, Data set containing indicator patterns to train the model
- `num_bars`  
  Integer, Number of bars used to detect pattern
- `timeframe`  
  Integer, Data time frame in minutes.
- `path_model`  
  String, Path where the models are be stored
- `path_data`  
  String, Path where the aggregated historical data is stored, if exists, in rds format
- `activate_balance`  
  Boolean, option to choose to balance market type classes or not, default TRUE
- `num_nn_options`  
  Integer, value from 0 to 24 or more as multiple of 3. Used to change number of variants for 3 hidden layer structure. Random neural network structures will be generated. When value 0 is set then a fixed structure will be used as defined by parameter fixed_nn_struct. To avoid warnings make sure to set this value as multiple of 3. Higher values will increase computation time.
- `fixed_nn_struct`  
  Integer vector with numeric elements, see par hidden in ?h2o.deeplearning, default value is c(100,100). Note this will only work if num_nn_options is 0
- `num_epoch`  
  Integer, see parameter epochs in ?h2o.deeplearning, default value is 100 Higher number may lead to long code execution
- `is_cluster`  
  Boolean, set TRUE to use automatically clustered data

Details

Function is using labeled dataset and tries several different random neural network structures. Once the best neural network is found then the better model is selected and stored. Dataset can be either manually labelled or generated using function mt_stat_transf.R. In the latter case parameter is_cluster shall be set to TRUE.

Value

Function is writing file object with the model
Author(s)

(C) 2020, 2021 Vladimir Zhbanko

Examples

```r
library(dplyr)
library(magrittr)
library(readr)
library(h2o)
library(lazytrade)
library(stats)

path_model <- normalizePath(tempdir(), winslash = "/")
path_data <- normalizePath(tempdir(), winslash = "/")

data(macd_ML60M)
Sys.sleep(5)

# start h2o engine
h2o.init(nthreads = 2)

# performing Deep Learning Classification using manually labelled data
mt_make_model(indicator_dataset = macd_ML60M,
              num_bars = 64,
              timeframe = 60,
              path_model = path_model,
              path_data = path_data,
              activate_balance = TRUE,
              num_nn_options = 3,
              num_epoch = 10)

data(price_dataset_big)
data <- head(price_dataset_big, 5000) # reduce computational time

ai_class <- mt_stat_transf(indicator_dataset = data,
                            num_bars = 64,
                            timeframe = 60,
                            path_data = path_data,
                            mt_classes = c("BUN", "BEN", "RAN"))

# performing Deep Learning Classification using the custom function auto clustered data
mt_make_model(indicator_dataset = ai_class,
              num_bars = 64,
              timeframe = 60,
              path_model = path_model,
              path_data = path_data,
              activate_balance = TRUE,
```

mt_stat_evaluate

num_nn_options = 6,
num_epoch = 10,
is_cluster = TRUE)

# performing Deep Learning Classification using the custom function auto clustered data
# and fixed nn structure
mt_make_model(indicator_dataset = ai_class,
    num_bars = 64,
    timeframe = 60,
    path_model = path_model,
    path_data = path_data,
    activate_balance = TRUE,
    num_nn_options = 0,
    fixed_nn_struct = c(10, 10),
    num_epoch = 10,
    is_cluster = TRUE)

# stop h2o engine
h2o.shutdown(prompt = FALSE)

#set delay to insure h2o unit closes properly before the next test
Sys.sleep(5)

mt_stat_evaluate Function to prepare and score data, finally predict current market type using pre-trained classification model

Description

PURPOSE: Function that uses Deep Learning model and Time Series Column of the dataframe to find out specific market type of the financial asset it will also discard bad result outputting -1 if it is the case

[Stable]

Usage

mt_stat_evaluate(x, path_model, num_bars, timeframe)

Arguments

x • dataframe with one column containing asset indicator in the time descending order, typically 64 or more values
path_model String, path to the model
num_bars Integer, Number of bars used to perform transformation
timeframe Integer, timeframe in Minutes.
Value

dataframe with predicted value of the market type

Author(s)

(C) 2021 Vladimir Zhbanko

Examples

```r
library(h2o)
library(magrittr)
library(dplyr)
library(readr)
library(lazytrade)
library(stats)

path_model <- normalizePath(tempdir(), winslash = "/")
path_data <- normalizePath(tempdir(), winslash = "/")

# start h2o engine (using all CPU's by default)
h2o.init(nthreads = 2)

data(price_dataset_big)
data <- head(price_dataset_big, 500) # reduce computational time

ai_class <- mt_stat_transf(indicator_dataset = data,
    num_bars = 64,
    timeframe = 60,
    path_data = path_data,
    mt_classes = c('BUN', 'BEN', 'RAN'))

# performing Deep Learning Classification using the custom function auto clustered data
mt_make_model(indicator_dataset = ai_class,
    num_bars = 64,
    timeframe = 60,
    path_model = path_model,
    path_data = path_data,
    activate_balance = TRUE,
    num_nn_options = 3,
    num_epoch = 10,
    is_cluster = TRUE)

# Use sample data
data(price_dataset)

# use one column for testing
x <- price_dataset[,2]
```
mt_stat_transf

mt_stat_evaluate(x = x,
    path_model = path_model,
    num_bars = 64,
    timeframe = 60)

h2o.shutdown(prompt = FALSE)

#set delay to insure h2o unit closes properly before the next test
Sys.sleep(5)

---

**mt_stat_transf**

*Perform Statistical transformation and clustering of Market Types on the price data*

---

**Description**

Function features methods of statistical data transformation and clustering of the price data. Multiple statistical properties are calculated for a defined time interval. Once combined, unsupervised learning (clustering) is performed to assign several classes, see function `mt_make_model`. Function allows to fully automatize financial periods classification. It is possible to choose two clustering methods either kmeans or hierarchical clustering.

*[Stable]*

**Usage**

```r
mt_stat_transf(
    indicator_dataset,
    num_bars = 64,
    timeframe = 60,
    path_data,
    mt_classes,
    clust_method = "kmeans",
    clust_opt = "complete",
    rule_opt = TRUE
)
```

**Arguments**

- **indicator_dataset**: Dataframe, multiple column dataset containing price data in each column. Each row is a time index, multiple columns are required but not strictly needed
- **num_bars**: Integer, Number of bars used to perform transformation
- **timeframe**: Integer, Data timeframe in Minutes, only used for naming convention
**mt_stat_transf**

- **path_data**  
  String, User path where the dataset could be stored for the future use by other function

- **mt_classes**  
  Character Vector, with 2 or more Market Type classes

- **clust_method**  
  Character, option to select which clustering method to choose. Could be either 'kmeans' or 'hclust'. Default value is 'kmeans'

- **clust_opt**  
  Character, option to select how to perform h clustering "average", "single", "complete", "ward". Default value is 'complete'

- **rule_opt**  
  Boolean, option to perform rule-based Market Type Assignment, defaults to TRUE

**Details**

User can define several market type classes names however function will randomly assign Market Period labels based on Unsupervised Learning. This is inconvenient however that should be compensated by automated way of doing such data classification

**Value**

Dataframe with statistically transformed and classified dataset for classification modeling

**Author(s)**

(C) 2021 Vladimir Zhbanko

**Examples**

```r
library(dplyr)
library(stats)
library(magrittr)
library(readr)
library(lazytrade)

path_data <- normalizePath(tempdir(), winslash = "/")

data(price_dataset_big)

#option
#mt_classes = c('BUN', 'BEN', 'RAN', 'BUV', 'BEV', 'RAV')
#mt_classes = c('BUN', 'BEN', 'RAN')
#clust_method = 'hclust'
#clust_opt = 'ward'

#build dataset for Market Type detection without rule based check
ai_class_rand <- mt_stat_transf(indicator_dataset = price_dataset_big,
num_bars = 64,
timeframe = 60,
path_data = path_data,
mt_classes = c('BUN', 'BEN', 'RAN'),
clust_method = 'kmeans',
```
opt_aggregate_results

```r
clust_opt = 'complete',
rule_opt = FALSE)

# use rule base check
ai_class_rule <- mt_stat_transf(indicator_dataset = price_dataset_big,
num_bars = 64,
timeframe = 60,
path_data = path_data,
mt_classes = c('BUN', 'BEN', 'RAN'),
clust_method = 'kmeans',
clust_opt = 'complete',
rule_opt = TRUE)
```

---

**opt_aggregate_results**  *Function to aggregate trading results from multiple folders and files*

**Description**

Read multiple `.csv` files stored in different folders. Store results to the intermediate dataframe.  
[Deprecated]

**Usage**

```r
opt_aggregate_results(path_data)
```

**Arguments**

- `path_data`  
  • String, path to the folder containing subfolders

**Details**

User must provide the path to the files in the folders. All files in subfolders are read and aggregated into one data object. Data object is sorted in descending order by order close time.

**Value**

Dataframe with trading results

**Examples**

```r
library(lazytrade)
library(readr)
library(dplyr)
library(magrittr)
library(lubridate)
```
opt_create_graphs

Function to create summary graphs of the trading results

Description
Create graphs and store them into pdf file

[Stable]

Usage
opt_create_graphs(x, outp_path, graph_type = "pdf")

Arguments
x  
• dataframe with aggregated trading results
outp_path  
• path to the folder where to write file
graph_type  
• character, one of the options c('ts', 'bars', 'pdf')

Details
bar graph and time series optionally written to the pdf file. File is named with a date of analysis to the location specified by the user

Value
graphic output

Examples

library(lazytrade)
library(readr)
library(dplyr)
library(magrittr)
library(lubridate)
library(ggplot2)
data(DFR)
dir <- normalizePath(tempdir(),winslash = "/")
# create pdf file with two graphs
opt_create_graphs(x = DFR, outp_path = dir)

# only show time series plot
opt_create_graphs(x = DFR, graph_type = 'ts')

---

**policy_tr_systDF**  
*Table with Market Types and sample of actual policy for those states*

**Description**

Table with Market Types and sample of actual policy for those states

**Usage**

`policy_tr_systDF`

**Format**

A dataframe with 2 columns:

- **MarketType**  Current Market Type status
- **Policy**  Policy choice

---

**price_dataset**  
*Table with price dataset*

**Description**

Table with price dataset

**Usage**

`price_dataset`

**Format**

A dataframe with several columns

- **X1**  Date and time of the price sample
- **X2-X29**  Values of the assets
### price_dataset_big

**Table with price dataset, 30000 rows**

**Description**

Table with price dataset, 30000 rows

**Usage**

`price_dataset_big`

**Format**

A dataframe with several columns

- **X1** Date and time of the price sample
- **X2-X29** Values of the assets

### profit_factorDF

**Table with Trade results samples**

**Description**

Table with Trade results samples

**Usage**

`profit_factorDF`

**Format**

A dataframe with several columns

- **MagicNumber** Unique identifiers of the Trading Robots
- **TicketNumber** Ticket Number of closed position
- **OrderStartTime** Date and Time when order started
- **OrderCloseTime** Date and Time when order closed
- **Profit** Monetary result of the trade
- **Symbol** Symbol of the Asset e.g. EURUSD
- **OrderType** Order Type 0 - buy, 1 - sell
**profit_factor_data**  
*Table with Trade results samples*

**Description**
Table with Trade results samples

**Usage**
profit_factor_data

**Format**
A dataframe with several columns

- **X1** Unique identifiers of the Trading Robots
- **X2** Ticket Number of closed position
- **X3** Date and Time when order started
- **X4** Date and Time when order closed
- **X5** Monetary result of the trade
- **X6** Symbol of the Asset e.g. EURUSD
- **X7** Order Type 0 - buy, 1 - sell

**result_prev**  
*Table with one column as result from the model prediction*

**Description**
Table with one column as result from the model prediction

**Usage**
result_prev

**Format**
A dataframe with one column

- **predict** Predicted values from the model
result_R  

*Table with predicted price change*

**Description**

Table with predicted price change

**Usage**

result_R

**Format**

A dataframe with one column

predict  predicted future price change

---

result_R1  

*Table with aggregated trade results*

**Description**

Table with aggregated trade results

**Usage**

result_R1

**Format**

A dataframe with one column

predict  predicted price change
**Description**

This function will perform Reinforcement Learning using Trading Data. It will suggest whether or not it is better to keep using trading systems or not. Function is just using results of the past performance to generate the recommendation (not a holy grail).

[Stable]

**Usage**

```r
rl_generate_policy(x, states, actions, control)
```

**Arguments**

- `x` • Dataframe containing trading data
- `states` • Character vector, Selected states of the System
- `actions` • Character vector, Selected actions executed under environment
- `control` • List, control parameters as defined in the Reinforcement Learning Package

**Details**

Initial policy is generated using a dummy zero values. This way function starts working directly from the first observation. However policy ‘ON’ value will only be generated once the Q value is greater than zero

**Value**

Function returns data frame with reinforcement learning model policy

**Author(s)**

(C) 2019,2021 Vladimir Zhbanko

**Examples**

```r
library(dplyr)
library(RelreinforcementLearning)
library(magrittr)
library(lazytrade)

data(data_trades)
states <- c("tradewin", "tradeloss")
actions <- c("ON", "OFF")
```
control <- list(alpha = 0.7, gamma = 0.3, epsilon = 0.1)
rl_generate_policy(x = data_trades,
    states, actions, control)

### Description

This function will perform Reinforcement Learning using Trading Data. It will suggest whether or not it is better to keep using trading systems or not. Function is just using results of the past performance to generate the recommendation (not a holy grail).

[Stable]

### Usage

```r
rl_generate_policy_mt(x, states, actions, control)
```

### Arguments

- **x**
  - Dataframe containing trading data
- **states**
  - Character vector, Selected states of the System
- **actions**
  - Character vector, Selected actions executed under environment
- **control**
  - List, control parameters as defined in the Reinforcement Learning Package

### Details

Initial policy is generated using a dummy zero values. This way function starts working directly from the first observation. However policy 'ON' value will only be generated once the Q value is greater than zero.

### Value

Function returns data frame with reinforcement learning model policy

### Examples

```r
library(dplyr)
library(magrittr)
library(ReinforcementLearning)
library(lazytrade)
data(trading_systemDF)
states <- c("BUN", "BUY", "BEN", "BEV", "RAN", "RAV")
actions <- c("ON", "OFF")
```
control <- list(alpha = 0.7, gamma = 0.3, epsilon = 0.1)

rl_generate_policy_mt(x = trading_systemDF,
    states = states,
    actions = actions,
    control = control)

---

**Description**

Function will record Q values during the model update. These values will be used by another function Function was developed to help to estimate best control parameters during optimisation process

**Usage**

```r
rl_log_progress(x, states, actions, control)
```

**Arguments**

- `x` • dataframe containing trading results
- `states` • Selected states of the System
- `actions` • Selected actions executed under environment
- `control` • control parameters as defined in the Reinforcement Learning Package

**Value**

dataframe with log of RL model reward sequences during model update

**Examples**

```r
# retrieve RL model Q values progress
library(ReinforcementLearning)
library(dplyr)
library(magrittr)
library(lazytrade)
data(data_trades)
x <- data_trades
states <- c("tradewin", "tradeloss")
actions <- c("ON", "OFF")
control <- list(alpha = 0.7, gamma = 0.3, epsilon = 0.1)

rl_log_progress(x = x, states = states, actions = actions, control = control)
```
Function to retrieve and help to log Q values during RL progress. This function is dedicated to the situations when Market Types are used as a 'states' for the Environment.

Description

Function will record Q values during the model update. These values will be used by another function. Function was developed to help to estimate best control parameters during optimisation process.

[Stable]

Usage

rl_log_progress_mt(x, states, actions, control)

Arguments

- x: dataframe containing trading results
- states: Selected states of the System
- actions: Selected actions executed under environment
- control: control parameters as defined in the Reinforcement Learning Package

Value

dataframe with log of RL model reward sequences during model update

Author(s)

(C) 2020, 2021 Vladimir Zhbanko

Examples

```r
# retrieve RL model Q values progress
library(ReinforcementLearning)
library(dplyr)
library(magrittr)
library(lazytrade)
data(trading_systemDF)
x <- trading_systemDF
states <- c("BUN", "BUV", "BEN", "BEV", "RAN", "RAV")
actions <- c("ON", "OFF") # 'ON' and 'OFF' are referring to decision to trade with Slave system
control <- list(alpha = 0.7, gamma = 0.3, epsilon = 0.1)
rl_log_progress_mt(x = x, states = states, actions = actions, control = control)
```
rl_record_policy

Record Reinforcement Learning Policy.

Description

Function will write a policy 'decision' to the csv file specific for each Expert Advisor.

Usage

```r
rl_record_policy(
  x,  
  last_result, 
  trading_system, 
  path_terminal, 
  fileName = "SystemControl"
)
```

Arguments

- `x` • Dataframe containing columns MarketType and Policy
- `last_result` • character vector of the last result of the trade
- `trading_system` • character vector of length 1 with Trading System Magic Number information
- `path_terminal` • path to the sandbox where this Policy/Decision must be written
- `fileName` • string, desired control file prefix e.g. 'SystemControl'

Details

It is imperative that terminal path contains exact word Terminal3

Value

nothing is returned but function will write csv file to the supplied directory

Author(s)

(C) 2019,2021 Vladimir Zhbanko

Examples

```r
library(stringr)
library(magrittr)
library(dplyr)
data(TradeStatePolicy)
```
rl_record_policy_mt  

Record Reinforcement Learning Policy for Market Types

Description

Function will write a policy 'decision' to the csv file specific for each Expert Advisor

[Stable]

Usage

rl_record_policy_mt(
  x,
  trading_system,
  path_terminal,
  fileName = "SystemControlMT"
)

Arguments

x  • Dataframe containing columns MarketType and Policy
trading_system  • numeric vector of length 1 with Trading System Magic Number information
path_terminal  • string, path to the terminal where this Policy/Decision must be written
fileName  • string, desired control file prefix e.g. 'SystemControlMT'

Details

It is imperative that terminal path contains exact word Terminal3

Value

nothing is returned but function will write csv file to the supplied directory
Examples

```r
library(stringr)
library(lazytrade)
data(policy_tr_systDF)

dir <- normalizePath(tempdir(), winslash = "/")

rl_record_policy_mt(x = policy_tr_systDF,
  trading_system = 8118101,
  path_terminal = dir,
  fileName = "SystemControlMT")
```

**Description**

This function is supposed to run on a weekly basis. Purpose of this function is to perform RL and trading simulation and find out the best possible control parameters for the RL function.

[Stable]

**Usage**

```r
rl_write_control_parameters(
  x,
  path_control_files,
  num_trades_to_consider = 100
)
```

**Arguments**

- `x`  
  • dataset containing the trading results for one trading robot
- `path_control_files`  
  • path where control parameters will be saved
- `num_trades_to_consider`  
  • number of last trades to use for RL modeling simulations, default value 100

**Details**

Function is used by the R script Adapt_RL_control.R
Value

Function writes best control parameters to be used by the Reinforcement Learning Function

Author(s)

(C) 2019 Vladimir Zhbanko

Examples

dir <- normalizePath(tempdir(), winslash = "/")
# test lasts 15 sec:
library(dplyr)
library(readr)
library(ReinforcementLearning)
library(magrittr)
library(lazytrade)
data(data_trades)
x <- data_trades
rl_write_control_parameters(x = data_trades,
    path_control_files = dir,
    num_trades_to_consider = 20)

Description

This function is supposed to run on a weekly basis. Purpose of this function is to perform RL and trading simulation and find out the best possible control parameters for the RL function.

[Stable]

Usage

rl_write_control_parameters_mt(
    x,
    path_control_files,
    num_trades_to_consider = 100
)
Arguments

- **x**  
  dataset containing the trading results for one trading robot

- **path_control_files**  
  path where control parameters will be saved

- **num_trades_to_consider**  
  number of last trades to use for RL modeling simulations, default value 100

Details

Function is used by the R script Adapt_RL_MT_control.R

Value

Function writes best control parameters to be used by the Reinforcement Learning Function

Author(s)

(C) 2019, 2021 Vladimir Zhbanko

Examples

```r
# test lasts 15 sec:
dir <- normalizePath(tempdir(), winslash = "/")
library(dplyr)
library(readr)
library(ReinforcementLearning)
library(magrittr)
library(lazytrade)
data(trading_systemDF)

# use optimal control parameters found by auxiliary function
rl_write_control_parameters_mt(x = trading_systemDF, 
                               path_control_files = dir, 
                               num_trades_to_consider = 100)
```

test_data_pattern

Table with several columns containing indicator values and Label values

Description

Table with several columns containing indicator values and Label values
Usage

test_data_pattern

Format

A dataframe with several columns

**LABEL** Asset values as were recorded in the future

**V1-V49** Transposed values of the indicator

---

<table>
<thead>
<tr>
<th>to_m</th>
<th>Convert time series data to matrix with defined number of columns</th>
</tr>
</thead>
</table>

Description

Transforms Time Series Column of the dataframe to the matrix with specified number of columns. Number of rows will be automatically found. Eventually not complete last row will be discarded.

[Superseded]

Usage

```
to_m(x, n_cols)
```

Arguments

- **x** • dataframe with one column
- **n_cols** • number of columns in the matrix

Value

- matrix with specified amount of rows

Examples

```
library(magrittr)
library(lazytrade)
macd_m <- seq(1:1000) %>% as.data.frame() %>% to_m(64)
```
**TradeStatePolicy**

Table with Trade States and sample of actual policy for those states

**Description**

Table with Trade States and sample of actual policy for those states

**Usage**

TradeStatePolicy

**Format**

A dataframe with 2 columns:

- **TradeState**  Current trade state status
- **Policy**  Policy choice

---

**trading_systemDF**  

Table with trade data and joined market type info

**Description**

Table with trade data and joined market type info

**Usage**

trading_systemDF

**Format**

A dataframe with several columns

- "MagicNumber.x"  Unique identifiers of the Trading Robots from Trade Log
- **TicketNumber**  Ticket Number of closed position
- **OrderStartTime**  Date and Time when order started
- **OrderCloseTime**  Date and Time when order closed
- **Profit**  Monetary result of the trade
- **Symbol**  Symbol of the Asset e.g. EURUSD
- **OrderType**  Order Type 0 - buy, 1 - sell
- "MagicNumber.y"  Unique identifiers of the Trading Robots from Ticket Opening Log
- "MarketType"  Logged Market Type of the asset at the moment of Ticket Opening
util_find_pid  
*R function to find PID of active applications*

**Description**
Utility function to find PID of the working terminal.exe application. Function is created to generate a system call to programmatically close any given application.

[Experimental]

**Usage**
util_find_pid(tasks_running = t_running, pid_pattern = "terminal.exe")

**Arguments**
tasks_running  
• string, vector with system tasks

pid_pattern  
• string, pattern value to search application with

**Details**
Function is executing a system command to get all processes running on the OS. Retrieved data is cleaned and organized to filter on required process. Function can also be used to track specific applications defined by the user.

**Value**
string with system kill command to close selected application

**Author(s)**
(C) 2021 Vladimir Zhbanko

**Examples**
```r
library(magrittr)
library(tibble)
library(stringr)
library(dplyr)
#library(readr)

dir <- normalizePath(tempdir(), winslash = "/")

#tasks_running <- system("tasklist", intern = TRUE)
#writelines(tasks_running, con = file.path(dir, 'tasks_running.txt'))
```
util_generate_password

R function to generate random passwords for MT4 platform or other needs

Description

Utility function to generate random passwords. Wrapper of cryptographic functions from 'openssl' library in R. Password length can be customized. By default function just output randomly generated 8 symbol password suitable for MT4 logins. It is also possible to create other passwords and include special symbols. When required, it's possible to write resulting password to the txt file. Once generated, password is written to the destination supplied by the user.

[Stable]

Usage

util_generate_password(
    salt = "something random",
    pass_len = 8,
    write_file = FALSE,
    file_name = "",
    special_symbols = FALSE
)

Arguments

salt string, random text supplied by the user
pass_len integer, number specifying how long should the password be
write_file bool, if true writes result to the txt file
file_name string, indicate path of the file where to write text result
special_symbols bool, if true adds special symbols
Details

Passwords are generated using sha512 cryptographic function from openssl package. System date and user 'salt' is used to supply initial text for cryptographic function. Hashing function is using additional 'salt' which will be based on the current System time. Additionally, only a part of generated string is selected and used for password. Some letters of generated string are converted from lower to upper case.

Value

string or text file with password

Author(s)

(C) 2019, 2021 Vladimir Zhbanko

Examples

```r
library(stringr)
library(magrittr)
library(openssl)
library(readr)

dir <- normalizePath(tempdir(), winslash = "/")
file_path <- file.path(dir, 'p.txt')

# write to file
util_generate_password(salt = 'random text', file_name = file_path)

# generate 8digit
util_generate_password(salt = 'random text')

# generate password with special symbols
util_generate_password(salt = 'random text', special_symbols = TRUE)

# generate longer password with special symbols
util_generate_password(salt = 'random text', pass_len = 10, special_symbols = TRUE)
```

util_profit_factor  Calculate Profit Factor

Description

Calculate profit factor using a data vector with the trading results

[Stable]
**write_command_via_csv**

Write csv files with indicated commands to the external system

**Description**

Function is capable to read the data and writing multiple files e.g. 'SystemControl8139124.csv'

[Stable]

**Usage**

```r
write_command_via_csv(x, path_terminal = "", fileName = "SystemControl")
```

**Arguments**

- ```x``` • dataframe object with resulting command e.g. 1 - enable; 0 - disable
- ```path_terminal``` • path to the terminal
- ```fileName``` • desired control file prefix e.g. 'SystemControl'

**Usage**

```r
call(write_command_via_csv, x, path_terminal = "", fileName = "SystemControl")
```

**Examples**

```r
library(magrittr)
library(dplyr)
library(lazytrade)
data(profit_factor_data)
profit_factor_data %>%
  group_by(X1) %>%
  summarise(PnL = sum(X5),
             NumTrades = n(),
             PrFact = util_profit_factor(X5)) %>%
  select(PrFact) %>% head(1) %>% round(3)
```
write_ini_file

Create initialization files to launch MT4 platform with specific configuration

Description

Function generate initialization files suitable for launching MT4 terminal with specific parameters. Several options available for generating files specific for each purpose. Option 'prod' will just use existing profile and connect to the broker server Option 'backtest' will generate file for the robot backtest Option 'opt' will generate file needed for the robot optimization Option 'full' allows to specify any desired parameter

[Stable]

Value

Function is writing multiple files e.g. 'SystemControl8139124.csv' to the Sandbox
typical content of the file: "Magic","IsEnabled" 8139124.1

Author(s)

(C) 2019, 2021 Vladimir Zhbanko

Examples

```
library(dplyr)
library(readr)
library(lubridate)
library(lazytrade)

path_sbxm <- normalizePath(tempdir(), winslash = "/")

file.copy(from = system.file("extdata", "OrdersResultsT1.csv", package = "lazytrade"),
          to = file.path(path_sbxm, "OrdersResultsT1.csv"), overwrite = TRUE)

DFT1 <- import_data(path_sbxm = path_sbxm,
                     trade_log_file = "OrdersResultsT1.csv")

DFT1 %>%
group_by(MagicNumber) %>% select(MagicNumber) %>% mutate(IsEnabled = 0) %>%
# head to shorten time of this example
head(2) %>%
# write commands to disable/enable systems
write_command_via_csv(path_terminal = path_sbxm)
```
**Usage**

```python
write_ini_file(
    mt4_Profile = "Default",
    mt4_MarketWatch = "Forex.set",
    mt4_Login = "1234567",
    mt4_Password = "xxxxxXX",
    mt4_Server = "BrokerServerName",
    mt4_AutoConfiguration = "false",
    mt4_EnableNews = "false",
    mt4_ExpertsEnable = "true",
    mt4_ExpertsDllImport = "true",
    mt4_ExpertsExpImport = "true",
    mt4_ExpertsTrades = "true",
    mt4_Symbol = "EURUSD",
    mt4_Period = "H1",
    mt4_Template = "Default",
    mt4_Expert = "",
    mt4_ExpertParameters = "",
    mt4_Script = "",
    mt4_ScriptParameters = "",
    mt4_TestExpert = "",
    mt4_TestExpertParameters = "",
    mt4_TestSymbol = "EURUSD",
    mt4_TestPeriod = "H1",
    mt4_TestModel = "",
    mt4_TestSpread = "",
    mt4_TestOptimization = "false",
    mt4_TestDateEnable = "true",
    mt4_TestFromDate = "",
    mt4_TestToDate = "",
    mt4_TestReport = "test report",
    mt4_TestReplaceReport = "false",
    mt4_TestShutdownTerminal = "",
    dss_inifilepath = "",
    dss_inifilename = "test.ini",
    dss_mode = "prod"
)
```

**Arguments**

- **mt4_Profile**: string, the subdirectory name in the /profiles directory. The charts will be opened in the client terminal according to the given profile. If this parameter is not specified, the current profile will be opened.

- **mt4_MarketWatch**: string, file name (the symbolsets directory) that contains the symbol list to be shown in the Market Watch window.
mt4_Login  string, the number of the account to connect to at startup. If this parameter is not specified, the current login will be used.

mt4_Password  string, the password that allows entering the system. This parameter will be ignored if the client terminal stores personal data on the disk and the account to be connected is in the list.

mt4_Server  string, the name of the trade server to be connected to. The server name is the same as the name of the corresponding .srv file stored in the /config directory.

mt4_AutoConfiguration  string, "true" or "false" depending on whether the autoconfiguration of Data Center setting should be enabled or not. If this parameter is not specified, the value from the current server settings will be used.

mt4_EnableNews  string, either 'false' or 'true'.

mt4_ExperimentsEnable  string, enable/disable experts.

mt4_ExperimentsDllImport  string, enable/disable DLL imports.

mt4_ExperimentsExpImport  string, enable/disable import of functions from external experts or MQL4 libraries.

mt4_ExperimentsTrades  string, enable/disable the experts trading.

mt4_Symbol  string, the symbol of the security the chart of which should be opened immediately after the terminal startup.

mt4_Period  string, the chart timeframe (M1, M5, M15, M30, H1, H4, D1, W1, MN). If this parameter is not specified, H1 is used.

mt4_Template  string, the name of the template file (the templates directory), which should be applied to the chart.

mt4_Expert  string, the name of the expert that should be launched after the client terminal has started.

mt4_ExpertParameters  string, the name of the file containing the expert parameters (the MQL4 Presets directory).

mt4_Script  string, the name of the script, which must be launched after the client terminal startup.

mt4_ScriptParameters  string, the name of the file containing the script parameters (the MQL5 Presets directory).

mt4_TestExpert  string, the name of the expert to be launched for testing. If this parameter has not been specified, no testing is launched.

mt4_TestExpertParameters  string, the name of the file containing parameters (the tester directory).

mt4_TestSymbol  string, the name of the symbol used for the expert testing. If this parameter has not been specified, the latest value used in the tester is used.
**write_ini_file**

**mt4_TestPeriod** string, the chart period (M1, M5, M15, M30, H1, H4, D1, W1, MN). If this parameter has not been specified, H1 is used.

**mt4_TestModel** string, 0, 1, or 2, depending on the testing model (Every tick, Control points, Open prices only). If this parameter has not been specified, 0 is used (Every tick)

**mt4_TestSpread** string, spread value that will be used for modeling Ask prices during testing. If 0 value is specified, the strategy tester will use the current spread of a symbol at the beginning of testing.

**mt4_TestOptimization** string, enable/disable optimization. The values that can be taken are "true" or "false". If this parameter had not been specified, the "false" value is used.

**mt4_TestDateEnable** string, enable/disable the "Use date" flag. The values that can be taken are "true" or "false". If this parameter had not been specified, the "false" value is used.

**mt4_TestFromDate** string, the date, from which to start testing, appeared as YYYY.MM.DD. If this parameter has not been specified, this date is 1970.01.01.

**mt4_TestToDate** string, the date, on which to finish testing, appeared as YYYY.MM.DD. If this parameter has not been specified, this date is 1970.01.01.

**mt4_TestReport** string, the name of the test report file. The file will be created in the client terminal directory. A relative path can be specified, for example: tester\MovingAverageReport". If the extension has not been specified in the file name, the ".htm" will be set automatically. If this parameter has not been specified, the test report will not be formed.

**mt4_TestReplaceReport** string, enable/disable the repeated report file record. The values that can be taken are "true" or "false".

**mt4_TestShutdownTerminal** string, enable/disable shutdown of the terminal after the testing has been finished.

**mt4_TestVisualEnable** string, enable (true) or disable (false) the visual test mode. If the parameter is not specified, the current setting is used.

**dss_inifilepath** string, path on the computer where file will be stored.

**dss_inifilename** string, file name that should be written.

**dss_mode** string.

**Details**

The added value of this function is the ability to generate multiple files to backtest several robots for several timeframes. For example, it solves the problem of doing repetitive tasks to 'backtest' robots for several currencies and repeat this procedure over time.

Most of the variables present in the function are starting with a prefix mt4_, the remainder of the name comes from the platform documentation, see references.
Remaining variables are named with a prefix ‘dss_’ stands for ‘Decision Support System’, as these are the variables used for further automation purposes.

Note that for simplicity reasons not all parameters are present in this function. e.g. FTP Settings and Proxy Server settings are not present.

**Value**

output is a file with desired parameters

**Author(s)**

(C) 2019 Vladimir Zhbanko

**References**

All parameters used are taken from the reference documentation https://www.metatrader4.com/en/trading-platform/help/service/start_conf_file

**Examples**

```r
library(lazytrade)

dir <- normalizePath(tempdir(), winslash = "/")

# test file to launch MT4 terminal with parameters
write_ini_file(mt4_Profile = "Default",
              mt4_Login = "12345678",
              mt4_Password = "password",
              mt4_Server = "BrokerServerName",
              dss_inifilepath = dir,
              dss_inifilename = "prod_T1.ini",
              dss_mode = "prod")

# test file to launch robot backtest
TO <- format(as.Date(Sys.Date()), "%Y.%m.%d")
FROM <- format(as.Date(Sys.Date() - 60), "%Y.%m.%d")

# test file for MT4 use for backtesting
write_ini_file(mt4_Profile = "Default",
              mt4_Login = "12345678",
              mt4_Password = "password",
              mt4_Server = "BrokerServerName",
              mt4_TestExpert = "FALCON_D\Falcon_D",
              mt4_TestExpertParameters = "Falcon_D.set",
              mt4_TestSymbol = "EURUSD",
              mt4_TestPeriod = "H1",
              mt4_TestModel = "2",
              mt4_TestSpread = "20",
              mt4_TestOptimization = "false",
              mt4_TestDateEnable = "true",
              mt4_TestFromDate = FROM,
```
### x_test_model

Table with a dataset to test the Model

<table>
<thead>
<tr>
<th>Description</th>
<th>Table with a dataset to test the Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Usage</td>
<td>x_test_model</td>
</tr>
<tr>
<td>Format</td>
<td>A dataframe with several columns</td>
</tr>
<tr>
<td></td>
<td>LABEL  future price change</td>
</tr>
<tr>
<td></td>
<td>X1-X75 Values of the macd indicator</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>y</th>
<th>Table with indicators and price change which is used to train model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Table with indicators and price change which is used to train model</td>
</tr>
<tr>
<td>Usage</td>
<td>y</td>
</tr>
<tr>
<td>Format</td>
<td>A dataframe with several columns</td>
</tr>
<tr>
<td></td>
<td>X1 Time index</td>
</tr>
<tr>
<td></td>
<td>X2 Closed price now</td>
</tr>
<tr>
<td></td>
<td>X3 Closed price 34 bars ago</td>
</tr>
<tr>
<td></td>
<td>X4-X19 Series of Indicator values</td>
</tr>
<tr>
<td></td>
<td>LABEL Price difference, difference between X3 and X2</td>
</tr>
</tbody>
</table>
Index

* datasets
  data_trades, 21
  DFR, 23
  EURUSDM15X75, 25
  indicator_dataset, 29
  macd_100, 30
  macd_df, 31
  macd_ML60M, 31
  policy_tr_systDF, 43
  price_dataset, 43
  price_dataset_big, 44
  profit_factor_data, 45
  profit_factorDF, 44
  result_prev, 45
  result_R, 46
  result_R1, 46
  test_data_pattern, 55
  TradeStatePolicy, 57
  trading_systemDF, 57
  x_test_model, 67
  y, 67

* see https://docs.h2o.ai/h2o-tutorials/latest-stable/tutorials/deeplearning/index.html
  aml_make_model, 7
  aml_collect_data, 3
  aml_consolidate_results, 5
  aml_make_model, 7
  aml_score_data, 11
  aml_simulation, 13
  aml_test_model, 15

check_if_optimize, 17
create_labelled_data, 19
create_transposed_data, 20

data_trades, 21
decrypt_mykeys, 22
DFR, 23
dlog, 23
encrypt_api_key, 24
EURUSDM15X75, 25
evaluate_macroeconomic_event, 26
get_profit_factorDF, 27
import_data, 28
indicator_dataset, 29
macd_100, 30
macd_df, 31
macd_ML60M, 31
mt_evaluate, 32
mt_import_data, 33
mt_make_model, 34
mt_stat_evaluate, 37
mt_stat_transf, 39
opt_aggregate_results, 41
opt_create_graphs, 42

policy_tr_systDF, 43
price_dataset, 43
price_dataset_big, 44
profit_factor_data, 45
profit_factorDF, 44
result_prev, 45
result_R, 46
result_R1, 46

rl_generate_policy, 47
rl_generate_policy_mt, 48
rl_log_progress, 49
rl_log_progress_mt, 50
rl_record_policy, 51
rl_record_policy_mt, 52
rl_write_control_parameters, 53
rl_write_control_parameters_mt, 54

test_data_pattern, 55
to_m, 56
INDEX

TradeStatePolicy, 57
trading_systemDF, 57
util_find_pid, 58
util_generate_password, 59
util_profit_factor, 60
write_command_via_csv, 61
write_ini_file, 62
x_test_model, 67
y, 67