Package ‘lazytrade’

January 10, 2020

Type Package

Title Learn Computer and Data Science using Algorithmic Trading

Version 0.3.9

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.

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 7.0.2

Imports readr, stringr, dplyr, lubridate, magrittr, ggplot2,
grDevices, h2o, ReinforcementLearning, openssl

Suggests testthat (>= 2.1.0), covr

Depends R (>= 3.4.0)

NeedsCompilation no

Repository CRAN

Date/Publication 2020-01-10 02:20:02 UTC
### R topics documented:

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>aml_collect_data</td>
<td>3</td>
</tr>
<tr>
<td>aml_make_model</td>
<td>4</td>
</tr>
<tr>
<td>aml_score_data</td>
<td>6</td>
</tr>
<tr>
<td>aml_test_model</td>
<td>8</td>
</tr>
<tr>
<td>check_if_optimize</td>
<td>10</td>
</tr>
<tr>
<td>create_labelled_data</td>
<td>12</td>
</tr>
<tr>
<td>create_transposed_data</td>
<td>13</td>
</tr>
<tr>
<td>data_trades</td>
<td>14</td>
</tr>
<tr>
<td>decrypt_mykeys</td>
<td>14</td>
</tr>
<tr>
<td>DFR</td>
<td>15</td>
</tr>
<tr>
<td>EURUSDM15X75</td>
<td>16</td>
</tr>
<tr>
<td>evaluate_macroeconomic_event</td>
<td>16</td>
</tr>
<tr>
<td>evaluate_market_type</td>
<td>18</td>
</tr>
<tr>
<td>generate_RL_policy</td>
<td>19</td>
</tr>
<tr>
<td>generate_RL_policy_mt</td>
<td>20</td>
</tr>
<tr>
<td>get_profit_factorDF</td>
<td>21</td>
</tr>
<tr>
<td>import_data</td>
<td>22</td>
</tr>
<tr>
<td>import_data_mt</td>
<td>23</td>
</tr>
<tr>
<td>indicator_dataset</td>
<td>24</td>
</tr>
<tr>
<td>load_asset_data</td>
<td>24</td>
</tr>
<tr>
<td>log_RL_progress</td>
<td>25</td>
</tr>
<tr>
<td>log_RL_progress_mt</td>
<td>26</td>
</tr>
<tr>
<td>macd_df</td>
<td>27</td>
</tr>
<tr>
<td>opt_aggregate_results</td>
<td>27</td>
</tr>
<tr>
<td>opt_create_graphs</td>
<td>28</td>
</tr>
<tr>
<td>policy_tr_systDF</td>
<td>29</td>
</tr>
<tr>
<td>price_dataset</td>
<td>29</td>
</tr>
<tr>
<td>profit_factor</td>
<td>30</td>
</tr>
<tr>
<td>profit_factorDF</td>
<td>31</td>
</tr>
<tr>
<td>profit_factor_data</td>
<td>31</td>
</tr>
<tr>
<td>record_policy</td>
<td>32</td>
</tr>
<tr>
<td>record_policy_mt</td>
<td>33</td>
</tr>
<tr>
<td>result_prev</td>
<td>34</td>
</tr>
<tr>
<td>result_R</td>
<td>34</td>
</tr>
<tr>
<td>self_learn_ai_R</td>
<td>35</td>
</tr>
<tr>
<td>test_data_pattern</td>
<td>36</td>
</tr>
<tr>
<td>test_model</td>
<td>37</td>
</tr>
<tr>
<td>to_m</td>
<td>38</td>
</tr>
<tr>
<td>TradeStatePolicy</td>
<td>39</td>
</tr>
<tr>
<td>trading_systemDF</td>
<td>39</td>
</tr>
<tr>
<td>util_generate_password</td>
<td>40</td>
</tr>
<tr>
<td>writeCommandViaCSV</td>
<td>41</td>
</tr>
<tr>
<td>write_command_via_csv</td>
<td>42</td>
</tr>
<tr>
<td>write_control_parameters</td>
<td>43</td>
</tr>
<tr>
<td>write_control_parameters_mt</td>
<td>44</td>
</tr>
<tr>
<td>write_ini_file</td>
<td>45</td>
</tr>
</tbody>
</table>
**aml_collect_data**

Function to read new data, transform data, save data for further re-training of regression model for a single currency pair

**Description**

Function is collecting data from the files using dedicated function load_asset_data.R. One file with a prices of the asset and another file with the corresponding indicator pattern. Both data objects are transformed to be suitable for Regression Modelling. Indicator values will be placed into the column X1-X75 and price change is in the column 'LABEL' Result would be written to new or aggregated to the existing file

Function is also checking that generated dataset is not too big. Should the dataset is too big (e.g. > 1000000 rows), then only latest 950000 rows will be used. Note: the amount 1000000 rows is not verified in practice, further testing is required.

**Usage**

```r
aml_collect_data(
  price_dataset,
  indicator_dataset,
  symbol,
  num_bars,
  timeframe,
  path_data
)
```

**Arguments**

- `price_dataset`: Dataset containing assets prices. It will be used as a label
- `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
- `num_bars`: Number of bars used to detect pattern
- `timeframe`: Data timeframe e.g. 1 min
- `path_data`: Path where the aggregated historical data is stored, if exists in rds format

**Details**

Function is handling shift of the price and indicator datasets. New data will be always on the 'bottom' of the dataset

The amount of rows is customizable however it must be selected once for the function to start working. Other 'aml_*' functions will rely on this selections, use the same number accordingly!
AML_MAKE_MODEL

Value

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

Author(s)

(C) 2019 Vladimir Zhbanko

Examples

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

path_terminal <- system.file("extdata", package = "lazytrade")
macd <- load_asset_data(path_terminal = path_terminal, trade_log_file = "AI_Macd",
                        time_period = 15, data_depth = "300")

prices <- load_asset_data(path_terminal = path_terminal, trade_log_file = "AI_CP",
                        time_period = 15, data_depth = "300")

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

# data transformation using the custom function for one symbol
aml_collect_data(price_dataset = prices,
                 indicator_dataset = macd,
                 symbol = 'EURUSD',
                 num_bars = 75,
                 timeframe = 15,
                 path_data = path_data)

---

aml_make_model

Function to train Deep Learning regression model for a single currency pair

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 most recent indicator pattern. This is to mimic traditional algorithmic systems based on the indicator rule attempting to automate optimization process with AI.
Deep learning model structure is obtained from the 6 random combinations of neurons within 4 layers of the network, the most accurate model configuration will be automatically selected.

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.

Usage

```r
aml_make_model(
  symbol,
  num_bars,
  timeframe,
  path_model,
  path_data,
  force_update = FALSE
)
```

Arguments

- `symbol`: Character symbol of the asset for which to train the model
- `num_bars`: Number of bars used to detect pattern
- `timeframe`: Data timeframe e.g. 1 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
- `force_update`: Boolean, by setting this to TRUE function will generate new model (useful after h2o engine update)

Details

Function is using the dataset prepared by the function aml_collect_data.R. Function will start to train the model as soon as there are more than 100 rows in the dataset.

Value

Function is writing file object with the model

Author(s)

(C) 2019 Vladimir Zhenko

Examples

```r
library(dplyr)
library(readr)
library(h2o)
library(lazytrade)
```
aml_score_data <- normalizePath(tempdir(), winslash = "/")
path_data <- normalizePath(tempdir(), winslash = "/")
data(EURUSDM15X75)
write_rds(EURUSDM15X75, file.path(path_data, "EURUSDM15X75.rds"))

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

# performing Deep Learning Regression using the custom function
aml_make_model(symbol = "EURUSD",
               num_bars = 75,
               timeframe = 15,
               path_model = path_model,
               path_data = path_data)

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

---

**aml_score_data**

*Function to score new data and predict change for each single currency pair*

**Description**

Function is using the latest data from the financial assets indicator pattern and deep learning model. Prediction is a price change in the future for that asset will be used by the trading system.

**Usage**

```r
aml_score_data(
  symbol,
  num_bars,
  timeframe,
  path_model,
  path_data,
  path_sbxm,
  path_sbxs
)
```
Arguments

symbol Character symbol of the asset for which the model shall predict
num_bars Number of bars used to detect pattern
timeframe Data timeframe e.g. 1 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) 2019 Vladimir Zhbanko

Examples

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

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

data(EURUSDM15X75)
write_rds(EURUSDM15X75, file.path(path_data, 'EURUSDM15X75.rds'))

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

# performing Deep Learning Regression using the custom function
aml_make_model(symbol = 'EURUSD',
num_bars = 75,
timeframe = 15,
path_model = path_model,
aml_test_model

Function to test the model and conditionally decide to update existing model for a single currency pair

Description

Function is designed to test the trading decision generated by the Deep learning regression model. The outcome of this function will be used to perform update of existing model with a fresh data.

Usage

aml_test_model(symbol, num_bars, timeframe, path_model, path_data)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>symbol</td>
<td>Character symbol of the asset for which to train the model</td>
</tr>
<tr>
<td>num_bars</td>
<td>Number of bars used to detect pattern</td>
</tr>
<tr>
<td>timeframe</td>
<td>Data timeframe e.g. 1 min</td>
</tr>
<tr>
<td>path_model</td>
<td>Path where the models are be stored</td>
</tr>
<tr>
<td>path_data</td>
<td>Path where the aggregated historical data is stored, if exists in rds format</td>
</tr>
</tbody>
</table>
Details

Function is reading shifted 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, shift predicted value by 75 bars as we will hold the asset for 75 bars. Account for the real price change after 75 bars by creating a cumulative sum column. Verify obtained summary results on the model and obtain virtual real/simulated result is consolidated to calculate model quality. Whenever this value is less than 0 function is writing dedicated decision using simple *.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 folder

Author(s)

(C) 2019 Vladimir Zhbanko

Examples

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

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

data(EURUSDM15X75)
write_rds(EURUSDM15X75, file.path(path_data, 'EURUSDM15X75.rds'))

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

# performing Deep Learning Regression using the custom function
aml_make_model(symbol = 'EURUSD',
    num_bars = 75,
    timeframe = 15,
    path_model = path_model,
    path_data = path_data)

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 = 'EURUSD',
    num_bars = 75,
)```
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

Usage

check_if_optimize(
  x,
  path_trading_robot = "",
  num_trades_to_consider = 10,
  profit_factor_limit = 0.7,
  demo_mode = FALSE,
  write_mode = FALSE
)

Arguments

x - dataframe containing trading results
path_trading_robot - path of trading robot repository. must contain folder TEST with file Setup.csv. File Setup.csv contains a table with magic numbers under test
check_if_optimize

num_trades_to_consider
  - Number of trades to calculate profit factor

profit_factor_limit
  - Limit below which trading robot considered not working properly

demo_mode
  - When true function uses package test dataset

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 Vladimir Zhbanko

Examples

```r
library(lazytrade)
library(dplyr)
library(readr)
library(lubridate)
DFT1 <- import_data(trade_log_file = system.file("extdata", "OrdersResultsT1.csv", package = "lazytrade"),
                   demo_mode = TRUE)

# without writing to the file
DFT1 %>% check_if_optimize(num_trades_to_consider = 10,
                           profit_factor_limit = 1.2,
                           demo_mode = TRUE,
                           write_mode = FALSE)

# function will write to the temporary file
DFT1 %>% check_if_optimize(num_trades_to_consider = 10,
                           profit_factor_limit = 1.2,
                           demo_mode = TRUE,
                           write_mode = TRUE)
```
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

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

Examples

library(dplyr)
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.

Learn by example how to manipulate data

**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(readr)

# 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
- **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

decrypt_mykeys

Function that decrypt encrypted content

Description
Function that decrypt encrypted content

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(openssl)

# Consumer API keys
ConsumerAPIkeys <- decrypt_mykeys(path_encrypted_content = file.path(path_encrypted_keys, "ConsumerAPIkeys.enc.rds"), path_private_key = path_private_key)
```

---

**DFR**

*Table with aggregated trade results*

**Description**

Table with aggregated trade results

**Usage**

DFR

**Format**

A dataframe with one column

- **MagicNumber** 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
- **CUMSUM_PNL** Cumulative sum of the ordered data
**EURUSD15X75**

*Table with indicator and price change dataset*

**Description**

Table with indicator and price change dataset

**Usage**

EURUSD15X75

**Format**

A dataframe with several columns

- **X1** future price change
- **X2-X76** Values of the macd indicator

**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 directories. These files will either enable or disable opening of new positions in the trading robots #'

**Usage**

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

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

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

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

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_file_name = "01_MacroeconomicEvent.csv",
    path_T1 = dir, path_T3 = dir)
**evaluate_market_type**  
*Function to score data and predict current market type using pretrained 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**

```r
evaluate_market_type(x, model_path, num_cols)
```

**Arguments**

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

**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**

```r
library(h2o)

# start h2o - don't start/stop within for loop
h2o.init()

# Use function to score the data to the model
data(macd_df)
x <- macd_df
num_cols <- 64
models_path <- tempdir()
remain_path <- "~/regression.bin/DL_Classification"
model_path <- file.path(models_path, remain_path)

my_market_prediction <- evaluate_market_type(x = df,
```
**generate_RL_policy**

Function performs RL and generates model policy

**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).

**Usage**

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

**Arguments**

- `x` - Dataframe containing trading data
- `states` - Selected states of the System
- `actions` - Selected actions executed under environment
- `control` - 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 Vladimir Zhbanko
Examples

```r
library(dplyr)
library(ReinforcementLearning)
library(magrittr)

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

---

**generate_RL_policy_mt**  
*Function performs RL and generates model policy for each Market Type*

**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).

**Usage**

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

**Arguments**

- **x** - Dataframe containing trading data
- **states** - possible states for Reinforcement Learning
- **actions** - possible actions
- **control** - control parameters

**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

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

get_profit_factorDF

Function that returns the profit factors of the systems in a form of a DataFrame

Description

Function that returns the profit factors of the systems in a form of a DataFrame

Usage

get_profit_factorDF(x, num_orders)

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(profit_factorDF, 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.

**Usage**

```r
import_data(path_terminal, trade_log_file, demo_mode = FALSE)
```

**Arguments**

- **path_terminal** - path of the Trading Terminal where the file with data is written
- **trade_log_file** - File name where the order results are written
- **demo_mode** - When true function uses data stored in the package data folder

**Value**

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

**Author(s)**

(C) 2019 Vladimir Zhbanko

**Examples**

```r
library(lazytrade)
library(dplyr)
library(readr)
library(lubridate)
DFT1 <- import_data(trade_log_file = system.file("extdata", "OrdersResultsT1.csv", package = "lazytrade"), demo_mode = TRUE)
```
**import_data_mt**

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

**Usage**

```r
import_data_mt(path_terminal, trade_log_file, system_number, demo_mode = FALSE)
```

**Arguments**

- `path_terminal` - path to the sandbox
- `trade_log_file` - direct path to the log file (used for demo purposes)
- `system_number` - magic number id of the trading system
- `demo_mode` - when true, uses sample datafile stored in the package

**Value**

function returns the data frame with 3 columns including market type code

**Author(s)**

(C) 2019 Vladimir Zhbanko

**Examples**

```r
library(dplyr)
library(readr)
import_data_mt(trade_log_file = system.file("extdata", "MarketTypeLog8132101.csv", package = "lazytrade"),
               demo_mode = TRUE)
```
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

load_asset_data  

Load and Prepare Asset Data

Description

Function imports file with financial asset data. Each column represent one asset, rows represent observations. Values in specific columns will be normalized by dividing them by 100. This is specifically done for pairs with JPY. In addition, X1 column will be converted to the ymd_hms format

Usage

load_asset_data(
    path_terminal,
    trade_log_file,
    time_period = 1,
    data_deepth = 50000
)

Arguments

path_terminal  - path to the MT4 terminal, string
trade_log_file  - csv file name where the data is stored, without ".csv"
time_period  - data periodicity in minutes, can be 1, 15, 60
data_deepth  - collected data deepth in rows. describe how many rows in original file to read
**log_RL_progress**

Details

Works for both price and indicator values, function parameters allowing to import different files. File names are selected to account different time periodicity and amount of the data.

Value

- dataframe with asset data in columns where X1 column is in a POSIXct format

Examples

```r
library(readr)
library(dplyr)
library(lubridate)
library(magrittr)
path_terminal <- system.file("extdata", package = "lazytrade")

# load and prepare prices data
prices <- load_asset_data(path_terminal = path_terminal,
                          trade_log_file = "AI_CP",
                          time_period = 60,
                          data_depth = "300")

# load and prepare indicator data
macd <- load_asset_data(path_terminal = path_terminal,
                        trade_log_file = "AI_Macd",
                        time_period = 60,
                        data_depth = "300")
```

---

**log_RL_progress**  
*Function to log RL progress.*

Description

Function will record Q values during updating of the model. These values will be used by another function.

Usage

`log_RL_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
log_RL_progress_mt

Function to log RL progress, dedicated to Market Types

Description

Function will record Q values during updating of the model. These values will be used by another function.

Usage

log_RL_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
### Examples

```r
# retrieve RL model Q values progress
library(ReinforcementLearning)
library(dplyr)
library(magrittr)
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)

log_RL_progress_mt(x = x, states = states, actions = actions, control = control)
```

---

**macd_df**  
*Table with one column indicator dataset*

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table with one column indicator dataset</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>macd_df</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>A dataframe with one column</td>
</tr>
</tbody>
</table>

| CADCHF | Indicator values of the asset |

---

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

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PURPOSE: Read multiple files stored in different folders Store results to the intermediate dataframe.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>opt_aggregate_results(fold_path)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Arguments</th>
</tr>
</thead>
<tbody>
<tr>
<td>fold_path</td>
</tr>
</tbody>
</table>
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

library(lazytrade)
library(readr)
library(dplyr)
library(magrittr)
library(lubridate)
DFOLDER <- system.file("extdata/RES", package = "lazytrade")
#dir <- normalizePath(tempdir(), winslash = "/")
opt_aggregate_results(fold_path = DFOLDER)

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
policy_tr_systDF

Examples

```r
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')
```

<table>
<thead>
<tr>
<th>policy_tr_systDF</th>
<th>Table with Market Types and sample of actual policy for those states</th>
</tr>
</thead>
</table>

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

<table>
<thead>
<tr>
<th>price_dataset</th>
<th>Table with price dataset</th>
</tr>
</thead>
</table>

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

profit_factor Calculate Profit Factor

Description

Calculate profit factor using a data vector with the trading results

Usage

profit_factor(x)

Arguments

x column vector with profit or loss of the orders for one system

Value

function should calculate profit factor for this vector and return one value also as vector

Author(s)

(C) 2019 Vladimir Zhbanko

Examples

library(magrittr)
library(dplyr)
data(profit_factor_data)
profit_factor_data %>%
  group_by(X1) %>%
  summarise(PnL = sum(X5),
            NumTrades = n(),
            PrFact = profit_factor(X5)) %>%
  select(PrFact) %>% head(1) %>% as.vector() %>% round(3)
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

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
record_policy

Record Reinforcement Learning Policy.

Description

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

Usage

record_policy(
  x,
  last_result,
  trading_system,
  pathterminal,
  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'

Value

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

Examples

library(stringr)
library(magrittr)
library(dplyr)
data(TradeStatePolicy)
dir <- normalizePath(tempdir(), winslash = "/")

record_policy(x = TradeStatePolicy,
              last_result = "tradewin",
              trading_system = 8118101,
              path_terminal = dir,
              fileName = "SystemControlRL")
**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

### Usage

```r
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'

### Value

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

### Examples

```r
library(stringr)
data(policy_tr_systDF)
dir <- normalizePath(tempdir(), winslash = "/")
record_policy_mt(x = policy_tr_systDF,
                 trading_system = 8118101,
                 path_terminal = dir,
                 fileName = "SystemControlMT")
```
### result_prev

**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

**Description**

Table with predicte price change

**Usage**

`result_R`

**Format**

A dataframe with one column

- **predict** Predicted future price change
Function to train Deep Learning regression model

**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 most recent indicator pattern.

**Usage**

```r
self_learn_ai_R(
  price_dataset,
  indicator_dataset,
  num_bars,
  timeframe,
  path_model,
  setup_mode = FALSE,
  research_mode = FALSE,
  write_log = TRUE
)
```

**Arguments**

- `price_dataset`: Dataset containing assets prices. It will be used as a label
- `indicator_dataset`: Dataset containing assets indicator which pattern will be used as predictor
- `num_bars`: Number of bars used to detect pattern
- `timeframe`: Data timeframe e.g. 1 min
- `path_model`: Path where the models are be stored
- `setup_mode`: When TRUE function will attempt to write model to the disk without checking it
- `research_mode`: When TRUE model will be saved and model result will be stored as well. To be used at the first run.
- `write_log`: Writes results of the newly trained model and previously used model to the file

**Details**

Performs data manipulation and training of the model. Function is handling shift of the price and indicator datasets. Function will also check how the model predict by using trading objective.

**NOTE:** Always run parameter research_mode = TRUE for the first time

Because of the function is intended to periodically re-train the model it would always check how the previous model was working. In case new model is better, the better model will be used.

Function can also write a log files with a results of the strategy test
**Value**

Function is writing files into Decision Support System folder

**Author(s)**

(C) 2019 Vladimir Zhbanko

**Examples**

```r
library(dplyr)
library(readr)
# start h2o engine (using all CPU's by default)
h2o.init()

# performing Deep Learning Regression using the custom function
self_learn_ai_R(price_dataset = prices,
                 indicator_dataset = macd,
                 num_bars = 75,
                 timeframe = 1,
                 path_model = path_model,
                 write_log = TRUE)

# stop h2o engine
h2o.shutdown(prompt = F)
```

---

**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

test_model  
*Test model using independent price data.*

Description

Goal of the function is to verify how good predicted results are.

Usage

```python
test_model(test_dataset, predictor_dataset, test_type)
```

Arguments

- `test_dataset` - Dataset containing the column 'LABEL' which will correspond to the real outcome of Asset price change. This column will be used to verify the trading strategy
- `predictor_dataset` - Dataset containing the column 'predict'. This column is corresponding to the predicted outcome of Asset change. This column will be used to verify strategy outcomes
- `test_type` can be either "regression" or "classification" used to distinguish which type of model is being used

Details

This function should work to backtest any possible dataset length. It could be that we will need to use it for testing 1 week or 1 month. It should also work for both Regression and Classification models. Note: strategy outcomes assumes trading on all 28 major forex pairs

Value

Function will return a data frame with several quality score metrics for the best model. In case quality score is positive or more than 1 the model would likely be working good. In case the score will be negative then the model is not predicting good. Internal logic will test several predictor thresholds and will indicate the best one
Examples

```r
library(dplyr)
data(result_prev)
data(test_data_pattern)

## evaluate hypothetical results of trading using the model
test_model(test_dataset = test_data_pattern,
            predictor_dataset = result_prev,
            test_type = "regression")
```

---

### to_m

**Convert time series data to matrix with defined number of columns**

**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.

**Usage**

```r
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**

```r
library(magrittr)
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_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.

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 Vladimir Zhbanko
Examples

```r
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)
```

writeCommandViaCSV

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'

Usage

```r
writeCommandViaCSV(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'

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 Vladimir Zhbanko
Examples

```r
library(dplyr)
library(readr)
library(lubridate)
DFT1 <- import_data(trade_log_file = system.file("extdata",
    "OrdersResultsT1.csv",
    package = "lazytrade"),
    demo_mode = TRUE)

dir <- normalizePath(tempdir(), winslash = "/")
DFT1 %>%
  group_by(MagicNumber) %>%
  select(MagicNumber) %>%
  mutate(IsEnabled = 0) %>%
  # write commands to disable systems
  writeCommandViaCSV(path_terminal = file.path(dir))
```

---

**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'

### 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'

### 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 Vladimir Zhbanko
**write_control_parameters**

*Function to find and write the best control parameters.*

**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.

**Usage**

```
write_control_parameters(x, path_control_files)
```

**Arguments**

- **x** - dataset containing the trading results for one trading robot
- **path_control_files** - path where control parameters will be saved

**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

---

**Examples**

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

DFT1 <- import_data(trade_log_file = system.file("extdata", 
  "OrdersResultsT1.csv", 
  package = "lazytrade"), 
  demo_mode = TRUE)

dir <- normalizePath(tempdir(), winslash = "/")
DFT1 %>%
  group_by(MagicNumber) %>%
  select(MagicNumber) %>%
  mutate(IsEnabled = 0) %>%
  # write commands to disable/enable systems
  write_command_via_csv(path_terminal = file.path(dir))
```
Author(s)
(C) 2019 Vladimir Zhbanko

Examples

#test lasts 15 sec:
library(dplyr)
library(readr)
library(ReinforcementLearning)
library(magrittr)
data(data_trades)
write_control_parameters(data_trades, path_control_files = tempfile())

write_control_parameters_mt

Function to find and write the best control parameters.

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.

Usage
write_control_parameters_mt(x, path_control_files)

Arguments
x - dataset containing the trading results for one trading robot
path_control_files - path where control parameters will be saved

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 Vladimir Zhbanko
Examples

# test lasts 15 sec:
library(dplyr)
library(readr)
library(ReinforcementLearning)
library(magrittr)
data(trading_systemDF)

# use optimal control parameters found by auxiliary function
write_control_parameters_mt(trading_systemDF, path_control_files = tempfile())

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.

Usage

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 = ""
mt4_TestVisualEnable = "false"
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_ExpertsEnable string, enable/disable experts.

mt4_ExpertsDllImport string, enable/disable DLL imports

mt4_ExpertsExpImport string, enable/disable import of functions from external experts or MQL4 libraries.
write_ini_file

mt4_ExertsTrades
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.

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.
write_ini_file

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

added value of this function is the ability to generate multiple files to backtest several robots for several timeframes. For example it is 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

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_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,
               mt4_TestToDate = TO,
               mt4_TestReport = "EURUSD_Report",
               mt4_TestReplaceReport = "false",
               mt4_TestShutdownTerminal = "true",
               mt4_TestVisualEnable = "false",
               dss_inifilepath = dir,
               dss_inifilename = "backtest.ini",
               dss_mode = "backtest")

---

x_test_model

<table>
<thead>
<tr>
<th>Table with a dataset to test the Model</th>
</tr>
</thead>
</table>

Description

Table with a dataset to test the Model
Usage

x_test_model

Format

A dataframe with several columns

X1  future price change
X2-X76  Values of the macd indicator
# Index

**Topic datasets**
- data_trades, 14
- DFR, 15
- EURUSD15X75, 16
- indicator_dataset, 24
- macd_df, 27
- policy_tr_systDF, 29
- price_dataset, 29
- profit_factor_data, 31
- profit_factorDF, 31
- result_prev, 34
- result_R, 34
- test_data_pattern, 36
- TradeStatePolicy, 39
- trading_systemDF, 39
- x_test_model, 49

aml_collect_data, 3
aml_make_model, 4
aml_score_data, 6
aml_test_model, 8

check_if_optimize, 10
create_labelled_data, 12
create_transposed_data, 13

data_trades, 14
decrypt_mykeys, 14
DFR, 15

EURUSD15X75, 16
evaluate_macroeconomic_event, 16
evaluate_market_type, 18

generate_RL_policy, 19
generate_RL_policy_mt, 20
get_profit_factorDF, 21

import_data, 22
import_data_mt, 23
indicator_dataset, 24

load_asset_data, 24
log_RL_progress, 25
log_RL_progress_mt, 26
macd_df, 27

opt_aggregate_results, 27
opt_create_graphs, 28

policy_tr_systDF, 29
price_dataset, 29
profit_factor, 30
profit_factor_data, 31
profit_factorDF, 31

record_policy, 32
record_policy_mt, 33
result_prev, 34
result_R, 34

self_learn_ai_R, 35

test_data_pattern, 36
test_model, 37
to_m, 38
TradeStatePolicy, 39
trading_systemDF, 39

util_generate_password, 40

write_command_via_csv, 42
write_control_parameters, 43
write_control_parameters_mt, 44
write_ini_file, 45
writeCommandViaCSV, 41

x_test_model, 49