Package 'strand'

May 26, 2020

Type Package

Title A Framework for Investment Strategy Simulation

Version 0.1.3

Date 2020-05-24

Description Provides a framework for performing discrete (share-level) simulations of investment strategies. Simulated portfolios optimize exposure to an input signal subject to constraints such as position size and factor exposure.

License GPL-3

URL `https://github.com/strand-tech/strand`

BugReports `https://github.com/strand-tech/strand/issues`

Depends R (>= 3.5.0)

Imports R6, Matrix, Rglpk, dplyr, tidyr, feather, lubridate, rlang, yaml, ggplot2

Suggests testthat, knitr, rmarkdown, shiny, DT, Rsymphony

Encoding UTF-8

LazyData true

VignetteBuilder knitr

RoxygenNote 7.1.0

NeedsCompilation no

Author Jeff Enos [cre, aut],
David Kane [aut],
Strand Technologies, Inc. [cph]

Maintainer Jeff Enos `<jeff@strand.tech>`

Repository CRAN

Date/Publication 2020-05-26 10:10:02 UTC

R topics documented:

strand-package *strand: a framework for investment strategy simulation*

Description

The strand package provides a framework for performing discrete (share-level) simulations of investment strategies. Simulated portfolios optimize exposure to an input signal subject to constraints such as position size and factor exposure.

For an introduction to running simulations using the package, see `vignette("strand")`. For details on available methods see the documentation for the `Simulation` class.

Author(s)

Jeff Enos <jeff@strand.tech> and David Kane <david@strand.tech>

Examples

```
# Load up sample data
data(sample_secref)
data(sample_pricing)
data(sample_inputs)

# Load sample configuration
config <- example_strategy_config()

# Create the Simulation object and run
sim <- Simulation$new(config,
                      raw_input_data = sample_inputs,
                      raw_pricing_data = sample_pricing,
                      security_reference_data = sample_secref)
sim$run()

# Print overall statistics
sim$overallStatsDf()

# Access tabular result data
head(sim$getSimSummary())
head(sim$getSimDetail())
head(sim$getPositionSummary())
head(sim$getInputStats())
head(sim$getOptimizationSummary())
head(sim$getExposures())

# Plot results
## Not run:
sim$plotPerformance()
sim$plotMarketValue()
sim$plotCategoryExposure("category_1")
sim$plotFactorExposure(c("factor_1", "factor_2", "factor_3"))
sim$plotNumPositions()

## End(Not run)
```

example_shiny_app *Run an example shiny app*

Description

Runs a shiny app that allows interactively configuring and running a simulation. Once the simulation is finished results, such as performance statistics and plots of exposures, are available in a results panel.

Usage

```
example_shiny_app()
```

Examples

```
if (interactive()) {
  example_shiny_app()
}
```

example_strategy_config *Load example strategy configuration*

Description

Loads an example strategy configuration file for use in examples.

Usage

```
example_strategy_config()
```

Value

An object of class `list` that contains the example configuration. The list object is the result of loading the package's example yaml configuration file `application/strategy_config.yaml`.

Examples

```
config <- example_strategy_config()
names(config$strategies)
show(config$strategies$strategy_1)
```

`sample_inputs` *Sample security inputs for examples and testing*

Description

A dataset containing sample (fake) security input data for 500 securities and 63 weekdays, from 2019-01-02 to 2019-03-29.

Usage

```
sample_inputs
```

Format

A data frame with 31500 rows and 9 variables:

id security identifier

date input date

average_volume measurement of average security trading volume, in shares

alpha_1 sample numeric alpha input

alpha_2 sample numeric alpha input

factor_1 sample numeric factor input

factor_2 sample numeric factor input

factor_3 sample numeric factor input

factor_4 sample numeric factor input

`sample_pricing` *Sample pricing data for examples and testing*

Description

A dataset containing sample (fake) pricing data for 500 securities and 63 weekdays, from 2019-01-02 to 2019-03-29.

Usage

```
sample_pricing
```

Format

A data frame with 31500 rows and 8 variables:

id security identifier

date pricing date

price_unadj the unadjusted price of the security

prior_close_unadj the unadjusted prior closing price of the security

dividend_unadj the dividend for the security on an unadjusted basis

distribution_unadj the distribution (e.g., spin-off) for the security on an unadjusted basis

volume trading volume for the security, in shares

adjustment_ratio the adjustment ratio for the security

`sample_secref` *Sample security reference data for examples and testing*

Description

A dataset containing sample (fake) security reference data for 500 securities.

Usage

```
sample_secref
```

Format

A data frame with 500 rows and 4 variables:

id security identifier

symbol human-readable trading symbol

category_1 categorical variable with values A-F

category_2 categorical variable with values A-L

`Simulation` *Simulation class*

Description

Class for running a simulation and getting results.

Details

The `Simulation` class is used to set up and run a daily simulation over a particular period. Portfolio construction parameters and other simulator settings can be configured in a yaml file that is passed to the object's constructor. See `vignette("strand")` for information on configuration file setup.

Methods

Public methods:

- `Simulation$new()`
- `Simulation$setVerbose()`
- `Simulation$setShinyCallback()`
- `Simulation$getSecurityReference()`
- `Simulation$run()`
- `Simulation$getSimDates()`
- `Simulation$getSimSummary()`
- `Simulation$getSimDetail()`
- `Simulation$getPositionSummary()`
- `Simulation$getInputStats()`
- `Simulation$getLooseningInfo()`
- `Simulation$getOptimizationSummary()`
- `Simulation$getExposures()`
- `Simulation$getDelistings()`
- `Simulation$getSingleStrategySummaryDf()`
- `Simulation$plotPerformance()`
- `Simulation$plotMarketValue()`
- `Simulation$plotCategoryExposure()`
- `Simulation$plotFactorExposure()`
- `Simulation$plotNumPositions()`
- `Simulation$overallStatsDf()`
- `Simulation$print()`
- `Simulation$writeFeather()`
- `Simulation$readFeather()`
- `Simulation$clone()`

Method `new()`**:** Create a new `Simulation` object.

Usage:

```
Simulation$new(
  config = NULL,
  raw_input_data = NULL,
  raw_pricing_data = NULL,
  security_reference_data = NULL,
  delisting_dates_data = NULL
)
```

Arguments:

`config` An object of class `list` or `character`, or `NULL`. If the value passed is a character vector, it should be of length 1 and specify the path to a yaml configuration file that contains the object's configuration info. If the value passed is of class list(), the list should contain the object's configuration info in list form (e.g, the return value of calling `yaml.load_file` on the configuration file). If the value passed is `NULL`, then there will be no configuration information associated with the simulation and it will not possible to call the `run` method. Setting `config` = `NULL` is useful when creating simulation objects into which results will be loaded with `readFeather`.

`raw_input_data` A data frame that contains all of the input data (for all periods) for the simulation. The data frame must have a `date` column. Data supplied using this parameter will only be used if the configuration option `simulator/input_data/type` is set to `object`. Defaults to `NULL`.

`raw_pricing_data` A data frame that contains all of the input data (for all periods) for the simulation. The data frame must have a `date` column. Data supplied using this parameter will only be used if the configuration option `simulator/pricing_data/type` is set to `object`. Defaults to `NULL`.

`security_reference_data` A data frame that contains reference data on the securities in the simulation, including any categories that are used in portfolio construction constraints. Note that the simulator will throw an error if there are input data records for which there is no entry in the security reference. Data supplied using this parameter will only be used if the configuration option `simulator/secref_data/type` is set to `object`. Defaults to `NULL`.

`delisting_dates_data` A data frame that contains the dates on which securities are delisted. It must contain two columns: id (character) and delisting_date (Date). The date in the delisting_date column means the day on which a stock will be removed from the simulation portfolio, at the beginning of the day, due to delisting. Data supplied using this parameter will only be used if the configuration option `simulator/delisting_data/type` is set to `object`. Defaults to `NULL`.

Returns: A new `Simulation` object.

Method `setVerbose()`**:** Set the verbose flag to control info output.

Usage:

```
Simulation$setVerbose(verbose)
```

Arguments:

`verbose` Logical flag indicating whether to be verbose or not.

Returns: No return value, called for side effects.

Method `setShinyCallback()`**:** Set the callback function for updating progress when running a simulation in shiny.

Usage:

```
Simulation$setShinyCallback(callback)
```

Arguments:

`callback` A function suitable for updating a shiny Progress object. It must have two parameters: `value`, indicating the progress amount, and detail, and `detail`, a text string for display on the progress bar.

Returns: No return value, called for side effects.

Method `getSecurityReference()`**:** Get security reference information.

Usage:

```
Simulation$getSecurityReference()
```

Returns: An object of class `data.frame` that contains the security reference data for the simulation.

Method `run()`**:** Run the simulation.

Usage:

```
Simulation$run()
```

Returns: No return value, called for side effects.

Method `getSimDates()`**:** Get a list of all date for the simulation.

Usage:

```
Simulation$getSimDates()
```

Returns: A vector of class `Date` over which the simulation currently iterates: all weekdays between the 'from' and 'to' dates in the simulation's config.

Method `getSimSummary()`**:** Get summary information.

Usage:

```
Simulation$getSimSummary()
```

Returns: An object of class `data.frame` that contains summary data for the simulation, by period, at the joint and strategy level. The data frame contains the following columns:

strategy Strategy name, or 'joint' for the aggregate strategy.

sim_date Date of the summary data.

market_fill_nmv Total net market value of fills that do not net down across strategies.

transfer_fill_nmv Total net market value of fills that represent "internal transfers", i.e., fills in one strategy that net down with fills in another. Note that at the joint level this column by definition is 0.

market_order_gmv Total gross market value of orders that do not net down across strategies.

market_fill_gmv Total gross market value of fills that do not net down across strategies.

transfer_fill_gmv Total gross market value of fills that represent "internal transfers", i.e., fills in one strategy that net down with fills in another.

start_nmv Total net market value of all positions at the start of the period.

start_lmv Total net market value of all long positions at the start of the period.

start_smv Total net market value of all short positions at the start of the period.

end_nmv Total net market value of all positions at the end of the period.

end_gmv Total gross market value of all positions at the end of the period.

end_lmv Total net market value of all long positions at the end of the period.

end_smv Total net market value of all short positions at the end of the period.

end_num Total number of positions at the end of the period.

end_num_long Total number of long positions at the end of the period.

end_num_short Total number of short positions at the end of the period.

position_pnl The total difference between the end and start market value of positions.

trading_pnl The total difference between the market value of trades at the benchmark price and at the end price. Note: currently assuming benchmark price is the closing price, so trading P&L is zero.

gross_pnl Total P&L gross of costs, calculated as position_pnl + trading_pnl.

trade_costs Total trade costs (slippage).

financing_costs Total financing/borrow costs.

net_pnl Total P&L net of costs, calculated as gross_pnl - trade_costs - financing_costs.

fill_rate_pct Total fill rate across all market orders, calculated as 100 * market_fill_gmv / market_order_gmv.

Method `getSimDetail()`**:** Get detail information.

Usage:

```
Simulation$getSimDetail(
  sim_date = NULL,
  strategy_name = NULL,
  security_id = NULL
)
```

Arguments:

`sim_date` Vector of length 1 of class Date or character that specifies the period for which to get detail information. If `NULL` then data from all periods is returned. Defaults to `NULL`.

`strategy_name` Character vector of length 1 that specifies the strategy for which to get detail data. If `NULL` data for all strategies is returned. Defaults to `NULL`.

`security_id` Character vector of length 1 that specifies the security for which to get detail data. If `NULL` data for all securities is returned. Defaults to `NULL`.

Returns: An object of class `data.frame` that contains detail data for the simulation at the joint and strategy level. Detail data is at the security level. The data frame contains the following columns:

id Security identifier.

strategy Strategy name, or 'joint' for the aggregate strategy.

sim_date Date to which the data pertains.

shares Shares at the start of the period.

int_shares Shares at the start of the period that net down with positions in other strategies.

ext_shares Shares at the start of the period that do not net down with positions in other strategies.

order_shares Order, in shares.

market_order_shares Order that does not net down with orders in other strategies, in shares.

transfer_order_shares Order that nets down with orders in other strategies, in shares.

fill_shares Fill, in shares.

market_fill_shares Fill that does not net down with fills in other strategies, in shares.

transfer_fill_shares Fill that nets down with fills in other strategies, in shares.

end_shares Shares at the end of the period.

end_int_shares Shares at the end of the period that net down with positions in other strategies.

end_ext_shares Shares at the end of the period that do not net down with positions in other strategies.

start_price Price for the security at the beginning of the period.

end_price Price for the security at the end of the period.

dividend Dividend for the security, if any, for the period.

distribution Distribution (e.g., spin-off) for the security, if any, for the period.

position_pnl Position P&L, calculated as shares * (end_price + dividend + distribution - start_price)

trading_pnl The difference between the market value of trades at the benchmark price and at the end price. Note: currently assuming benchmark price is the closing price, so trading P&L is zero.

trade_costs Trade costs, calculated as a fixed percentage (set in the simulation configuration) of the notional of the market trade (valued at the close).

financing_costs Financing cost for the position, calculated as a fixed percentage (set in the simulation configuration) of the notional of the starting value of the portfolio's external positions. External positions are positions held on the street and are recorded in the ext_shares column.

gross_pnl Gross P&L, calculated as position_pnl + trading_pnl.

net_pnl Net P&L, calculated as gross_pnl - trade_costs - financing_costs.

market_order_nmv Net market value of the order that does not net down with orders in other strategies.

market_fill_gmv Gross market value of the order that does not net down with orders in other strategies.

market_fill_nmv Net market value of the fill that does not net down with orders in other strategies.

market_fill_gmv Gross market value of the fill that does not net down with orders in other strategies.

transfer_fill_nmv Net market value of the fill that nets down with fills in other strategies.

transfer_fill_gmv Gross market value of the fill that nets down with fills in other strategies.

start_nmv Net market value of the position at the start of the period.

end_nmv Net market value of the position at the end of the period.

end_gmv Gross market value of the position at the end of the period.

Method `getPositionSummary()`**:** Get summary information by security. This method can be used, for example, to calculate the biggest winners and losers over the course of the simulation.

Usage:

```
Simulation$getPositionSummary(strategy_name = NULL)
```

Arguments:

`strategy_name` Character vector of length 1 that specifies the strategy for which to get detail data. If `NULL` data for all strategies is returned. Defaults to `NULL`.

Returns: An object of class `data.frame` that contains summary information aggregated by security. The data frame contains the following columns:

id Security identifier.

strategy Strategy name, or 'joint' for the aggregate strategy.

gross_pnl Gross P&L for the position over the entire simulation.

gross_pnl Net P&L for the position over the entire simulation.

average_market_value Average net market value of the position over days in the simulation where the position was not flat.

total_trading Total gross market value of trades for the security.

trade_costs Total cost of trades for the security over the entire simulation.

trade_costs Total cost of financing for the position over the entire simulation.

days_in_portfolio Total number of days there was a position in the security in the portfolio over the entire simulation.

Method `getInputStats()`**:** Get input statistics.

Usage:

```
Simulation$getInputStats()
```

Returns: An object of class `data.frame` that contains statistics on select columns of input data. Statistics are tracked for the columns listed in the configuration variable `simulator/input_data/track_metadata`. The data frame contains the following columns:

period Period to which statistics pertain.

input_rows Total number of rows of input data, including rows carried forward from the previous period.

cf_rows Total number of rows carried forward from the previous period.

num_na_*column* Number of NA values in *column*. This measure appears for each element of `track_metadata`.

cor_*column* Period-over-period correlation for *column*. This measure appears for each element of `track_metadata`.

Method `getLooseningInfo()`**:** Get loosening information.

Usage:

```
Simulation$getLooseningInfo()
```

Returns: An object of class `data.frame` that contains, for each period, which constraints were loosened in order to solve the portfolio optimization problem, if any. The data frame contains the following columns:

date Date for which the constraint was loosened.

constraint_name Name of the constraint that was loosened.

pct_loosened Percentage by which the constraint was loosened, where 100 means loosened fully (i.e., the constraint is effectively removed).

Method `getOptimizationSummary()`**:** Get optimization summary information.

Usage:

```
Simulation$getOptimizationSummary()
```

Returns: An object of class `data.frame` that contains optimization summary information, such as starting and ending factor constraint values, at the strategy and joint level. The data frame contains the following columns:

strategy Strategy name, or 'joint' for the aggregate strategy.

sim_date Date to which the data pertains.

order_gmv Total gross market value of orders generated by the optimization.

start_smv Total net market value of short positions at the start of the optimization.

start_lmv Total net market value of long positions at the start of the optimization.

end_smv Total net market value of short positions at the end of the optimization.

end_lmv Total net market value of long positions at the end of the optimization.

start_*factor* Total net exposure to *factor* at the start of the optimization, for each factor constraint.

end_*factor* Total net exposure to *factor* at the start of the optimization, for each factor constraint.

Method `getExposures()`**:** Get end-of-period exposure information.

Usage:

```
Simulation$getExposures()
```

Returns: An object of class `data.frame` that contains end-of-period exposure information for the simulation portfolio. The units of the exposures are portfolio weight relative to strategy_captial (i.e., net market value of exposure divided by strategy capital). The data frame contains the following columns:

strategy Strategy name, or 'joint' for the aggregate strategy.

sim_date Date of the exposure data.

category_level Exposure to *level* within *category*, for all levels of all category constraints, at the end of the period.

factor Exposure to *factor*, for all factor constraints, at the end of the period.

Method `getDelistings()`**:** Get information on positions removed due to delisting.

Usage:

```
Simulation$getDelistings()
```

Returns: An object of class `data.frame` that contains a row for each position that is removed from the simulation portfolio due to a delisting. Each row contains the size of the position on the day on which it was removed from the portfolio.

Method `getSingleStrategySummaryDf()`**:** Get summary information for a single strategy suitable for plotting input.

Usage:

```
Simulation$getSingleStrategySummaryDf(strategy_name, include_zero_row = TRUE)
```

Arguments:

`strategy_name` Strategy for which to return summary data.

`include_zero_row` Logical flag indicatiing whether to prepend a row to the summary data with starting values at zero. Defaults to `TRUE`.

Returns: A data frame that contains summary information for the desired strategy, as well as columns for cumulative net and gross total return, calculated as pnl divided by ending gross market value.

Method `plotPerformance()`**:** Draw a plot of cumulative gross and net return by date.

Usage:

```
Simulation$plotPerformance()
```

Method `plotMarketValue()`**:** Draw a plot of total gross, long, short, and net market value by date.

Usage:

```
Simulation$plotMarketValue()
```

Method `plotCategoryExposure()`**:** Draw a plot of exposure to all levels in a category by date.

Usage:

```
Simulation$plotCategoryExposure(in_var)
```

Arguments:

`in_var` Category for which exposures are plotted.

Method `plotFactorExposure()`**:** Draw a plot of exposure to factors by date.

Usage:

```
Simulation$plotFactorExposure(in_var)
```

Arguments:

`in_var` Factors for which exposures are plotted.

Method `plotNumPositions()`**:** Draw a plot of number of long and short positions by date.

Usage:

```
Simulation$plotNumPositions()
```

Method `overallStatsDf()`**:** Calculate overall simulation summary statistics, such as total P&L, Sharpe, average market values and counts, etc.

Usage:

```
Simulation$overallStatsDf()
```

Returns: A data frame that contains summary statistics, suitable for reporting.

Method `print()`**:** Print overall simulation statistics.

Usage:

```
Simulation$print()
```

Method `writeFeather()`**:** Write the data in the object to feather files.

Usage:

```
Simulation$writeFeather(out_loc)
```

Arguments:

`out_loc` Directory in which output files should be created.

Returns: No return value, called for side effects.

Method `readFeather()`**:** Load files created with `writeFeather` into the object. Note that because detail data is not re-split by period, it will not be possible to use the `sim_date` parameter when calling `getSimDetail` on the populated object.

Usage:

```
Simulation$readFeather(in_loc)
```

Arguments:

`in_loc` Directory that contains files to be loaded.

Returns: No return value, called for side effects.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
Simulation$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Index