InferTrade

Release 0.0.1

InferStat

May 26, 2021

CONTENTS:

CHAPTER

ONE

UTILITIES

1.1 utilities package

1.1.1 Submodules

1.1.2 utilities.operations module

Utility code for operations such as converting positions to price predictions and vice versa.

Created by: Joshua Mason Created date: 11/03/2021

class utilities.operations.**PositionsFromPricePrediction**

Bases: sklearn.base.TransformerMixin, sklearn.base.BaseEstimator

This class calculates the positions to take assuming Kelly Criterion.

fit(*X*, *y=None*)

transform(*X*, *y=None*)

class utilities.operations.**PricePredictionFromPositions**

Bases: sklearn.base.TransformerMixin, sklearn.base.BaseEstimator

This converts positions into implicit price predictions based on the Kelly Criterion and an assumed volatility.

fit(*X*, *y=None*)
Not used.

transform(*X: pandas.core.frame.DataFrame*, *y=None*)
Converts allocations into the forecast one-day price changes.

class utilities.operations.**PricePredictionFromSignalRegression**(*market_to_trade: Optional[str] = None*)

Bases: sklearn.base.TransformerMixin, sklearn.base.BaseEstimator

Class for creating price predictions from signal values.

fit(*X: numpy.array*, *y=None*)

transform(*X*, *y=None*)
We transform a signal input to a price prediction.

class utilities.operations.**ReturnsFromPositions**
Bases: sklearn.base.TransformerMixin, sklearn.base.BaseEstimator

This calculate returns from positions.

fit(*X*, *y=None*)
Not used.

transform(*X: pandas.core.frame.DataFrame*, *y=None*)
Converts positions into the cumulative portfolio return.

utilities.operations.**diff_log**(*x: Union[numpy.ndarray, pandas.core.series.Series]*) → numpy.ndarray
Differencing and log transformation between the current and a prior element.

utilities.operations.**dl_lag**(*x: Union[numpy.ndarray, pandas.core.series.Series]*, *shift: int = 1*) → numpy.ndarray
Differencing and log transformation of lagged series.

utilities.operations.**lag**(*x: Union[numpy.ndarray, pandas.core.series.Series]*, *shift: int = 1*) → numpy.ndarray
Lag (shift) series by desired number of periods.

utilities.operations.**log_price_minus_log_research**(*x: Union[numpy.ndarray, pandas.core.series.Series]*, *shift: int*) → numpy.ndarray
Difference of two lagged log series.

utilities.operations.**moving_average**(*x: Union[numpy.ndarray, pandas.core.series.Series]*, *window: int*) → numpy.ndarray
Calculate moving average of series for desired number of periods (window).

utilities.operations.**pct_chg**(*x: Union[numpy.ndarray, pandas.core.series.Series]*) → numpy.ndarray
Percentage change between the current and a prior element.

utilities.operations.**research_over_price_minus_one**(*x: Union[numpy.ndarray, pandas.core.series.Series]*, *shift: int*) → numpy.ndarray
Difference of two lagged log series.

utilities.operations.**zero_one_dl**(*x: Union[numpy.ndarray, pandas.core.series.Series]*) → numpy.ndarray
Returns ones for positive values of "diff-log" series, and zeros for negative values.

1.1.3 utilities.performance module

Performance calculation using the InferTrade inferface.

Author: Thomas Oliver Created: 11th March 2021

utilities.performance.**calculate_allocation_from_cash**(*last_cash_after_trade: float*, *last_securities_after_transaction: float*, *spot_price: float*) → float

Calculates the current allocation.

utilities.performance.**calculate_portfolio_performance_python**(*df_with_positions: pandas.core.frame.DataFrame*, *skip_checks: bool = False*, *show_absolute_bankruptcies: bool = False*, *annual_strategy_fee: float = 0.0*, *daily_spread_percent_override: float = 0.0*, *minimum_allocation_change_to_adjust: float = 0.0*, *detailed_output: bool = True*)

This is the main vanilla Python calculation of portfolio performance.

utilities.performance.**check_if_should_skip_return_calculation**(*previous_portfolio_return: float*, *spot_price: float*, *day: int*, *day_of_return_to_calculate: int*, *show_absolute_bankruptcies: bool*, *bankrupt: bool = False) -> (<class 'bool'>, <class 'float'>)*

This function checks if we should skip the returns calculation for the requested day.

utilities.performance.**portfolio_index**(*position_on_last_good_price: float, spot_price_usd: float, last_good_price_usd: Optional[float], current_bid_offer_spread_percent: float, target_allocation_perc: float, annual_strategy_fee_perc: float, last_securities_volume: float, last_cash_after_trade_usd: float, show_working: bool = False) -> (<class 'float'>, <class 'float'>, <class 'float'>)*

A function for calculating the cumulative return of the portfolio.

utilities.performance.**rounded_allocation_target**(*unconstrained_target_position: float*, *minimum_allocation_change_to_adjust: float*) → float

Determines what allocation size to take if using rounded targets.

1.1.4 utilities.simple_functions module

Simple functions used across the package.

Author: Thomas Oliver Created: 18th March 2021

utilities.simple_functions.**add_package**(*dictionary: dict*, *string_label: str*) → dict

Adds a string to every item.

1.1.5 Module contents

Utilties directory for functions that uses the infertrade interface.

Created by: Thomas Oliver Created date: 16th March 2021

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

u

INDEX