InferTrade

Release 0.0.34

InferStat

Jun 07, 2021

CONTENTS:


InferTrade

CHAPTER

ONE

INFERTRADE

`infertrade` is an open source trading and investment strategy library designed for accessibility and compatibility.

The `infertrade` package seeks to achieve three objectives:

- Simplicity: a simple `pandas` to `pandas` interface that those experienced in trading but new to Python can easily use.
- Gateway to data science: classes that allow rules created for the infertrade simple interface to be used with `scikit-learn` functionality for prediction and calibration. (fit, transform, predict, pipelines, gridsearch) and `scikit-learn` compatible libraries, like `feature-engine`.
- The best open source trading strategies: wrapping functionality to allow strategies from any open source Python libraries with compatible licences, such as `ta` to be used with the `infertrade` interface.

The project is licenced under the Apache 2.0 licence.

1.1 Connection to InferTrade.com

Many thanks for looking into the `infertrade` package!

I created InferTrade.com to provide cutting edge statistical analysis in an accessible free interface. The intention was to help individuals and small firms have access to the same quality of analysis as large institutions for systematic trading and to allow more time to be spent on creating good signals rather than backtesting and strategy verification. If someone has done the hard work of gaining insights into markets I wanted them to be able to compete in a landscape of increasingly automated statistically-driven market participants. A huge amount of effort has been made by the trading and AI/ML communities to create open source packages with powerful diagnostic functionality, which means you do not need to build a large and complex in-house analytics library to be able to support your investment decisions with solid statistical machine learning. However there remain educational and technical barriers to using this community-created wealth if you are not an experience programmer or do not have mathematical training. I want InferTrade.com to allow everyone trading in markets to have access without barriers - cost, training or time - to be competitive, with an easy to use interface that both provides direct analysis and education insights to support your trading.

The initial impetus for the creation of this open source package, `infertrade` was to ensure any of our users finding an attractive strategy on InferTrade.com could easily implement the rule in Python and have full access to the code to fully understand every aspect of how it works. By adding wrapper for existing libraries we hope to support further independent backtesting by users with their own preferred choice of trading libraries. We at InferStat heavily use open source in delivering InferTrade.com's functionality and we also wanted to give something back to the trading and data science community. The Apache 2.0 licence is a permissive licence, so that you can use or build upon `infertrade` for your personal, community or commercial projects.

The `infertrade` package and InferTrade.com will be adding functionality each week, and we are continually seeking to improve the experience and support the package and website provides for traders, portfolio managers and other users. Gaining feedback on new features is extremely helpful for us to improve our UX and design, as are any

ideas for enhancements that would help you to trade better. If you would like to assist me in turning InferTrade into the leading open source trading platform we can offer participation in our Beta Testing programme (sign up link). You can also fork this repository and make direct improvements to the package.

Best, Tom Oliver

InferStat Founder and CEO

- https://github.com/ta-oliver
- https://www.linkedin.com/in/thomas-oliver-09487b9/

1.2 Contact Us

This was InferStat's first open source project and we welcome your thoughts for improvements to code structure, documentation or any changes that would support your use of the library.

If you would like assistance with using the `infertrade` you can email us at support@infertrade.com or book a video call

If you would like to contribute to the package, e.g. to add support for an additional package or library, please see our contributing information.

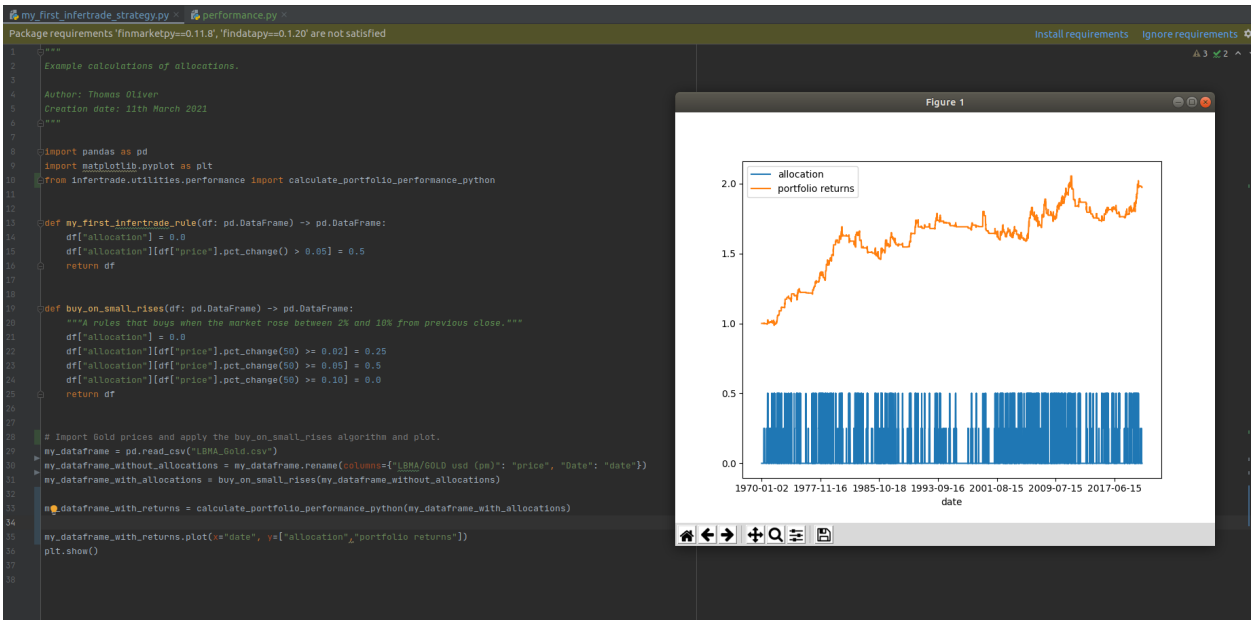
1.3 Quickstart

Please note the project requires Python 3.7 or higher due to dependent libraries used.

See Windows or Linux guides for installation details.

1.3.1 My First InferTrade Rule

```
import pandas as pd
import matplotlib.pyplot as plt

def my_first_infertrade_rule(df: pd.DataFrame) -> pd.DataFrame:
    df["allocation"] = 0.0
    df["allocation"][df.pct_change() > 0.02] = 0.5
    return df

my_dataframe = pd.read_csv("example_market_data.csv")
my_dataframe_with_allocations = my_first_infertrade_rule(my_dataframe)
my_dataframe_with_allocations.plot(["close"], ["allocation"])
plt.show()
```



1.3.2 Basic usage with community functions

"Community" functions are those declared in this repository, not retrieved from an external package. They are all exposed at `infertrade.algos.community`.

```
from infertrade.algos.community import normalised_close, scikit_signal_factory
from infertrade.data.simulate_data import simulated_market_data_4_years_gen
signal_transformer = scikit_signal_factory(normalised_close)
signal_transformer.fit_transform(simulated_market_data_4_years_gen())
```

1.3.3 Usage with TA

```
from infertrade.algos.community import scikit_signal_factory
from infertrade.data.simulate_data import simulated_market_data_4_years_gen
from infertrade.algos import ta_adaptor
from ta.trend import AroonIndicator
adapted_aroon = ta_adaptor(AroonIndicator, "aroon_down", window=1)
signal_transformer = scikit_signal_factory(adapted_aroon)
signal_transformer.fit_transform(simulated_market_data_4_years_gen())
```

1.3.4 Calculate positions with simple position function

```
from infertrade.algos.community.allocations import constant_allocation_size
from infertrade.utilities.operations import scikit_allocation_factory
from infertrade.data.simulate_data import simulated_market_data_4_years_gen

position_transformer = scikit_allocation_factory(constant_allocation_size)
position_transformer.fit_transform(simulated_market_data_4_years_gen())
# TODO add example with parameters
```

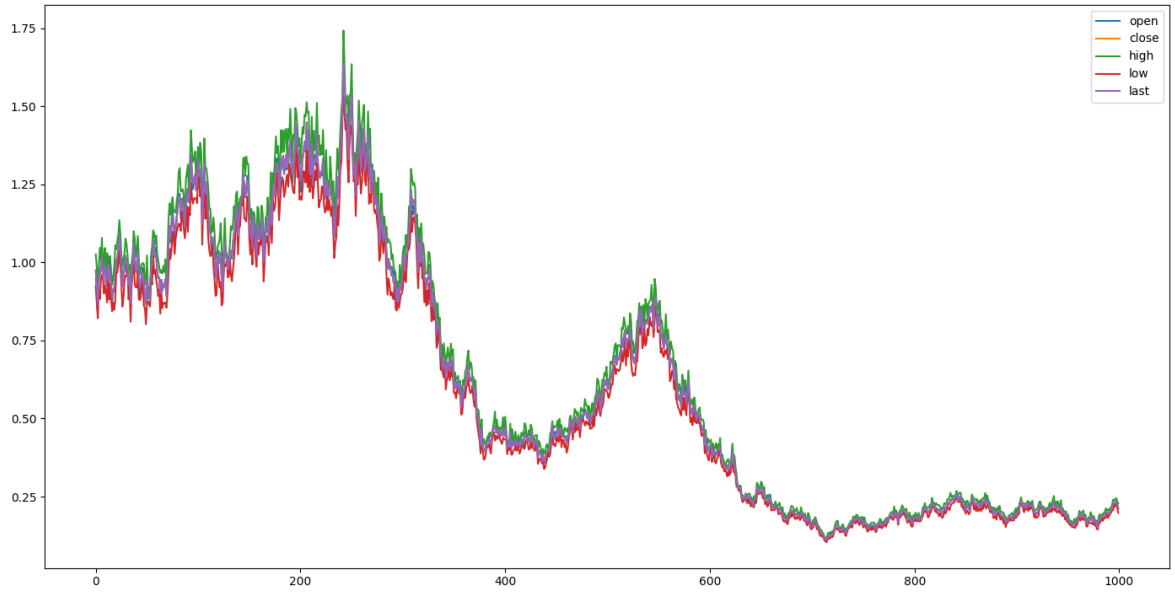
1.3.5 Example of position calculation via kelly just based on signal generation

```
from infertrade.algos.community import scikit_signal_factory
from infertrade.data.simulate_data import simulated_market_data_4_years_gen
from infertrade.utilities.operations import PositionsFromPricePrediction,␣
↪PricePredictionFromSignalRegression
from sklearn.pipeline import make_pipeline
from infertrade.algos import ta_adaptor
from ta.trend import AroonIndicator

adapted_aroon = ta_adaptor(AroonIndicator, "aroon_down", window=1)

pipeline = make_pipeline(scikit_signal_factory(adapted_aroon),
                         PricePredictionFromSignalRegression(),
                         PositionsFromPricePrediction()
                         )

pipeline.fit_transform(simulated_market_data_4_years_gen())
```

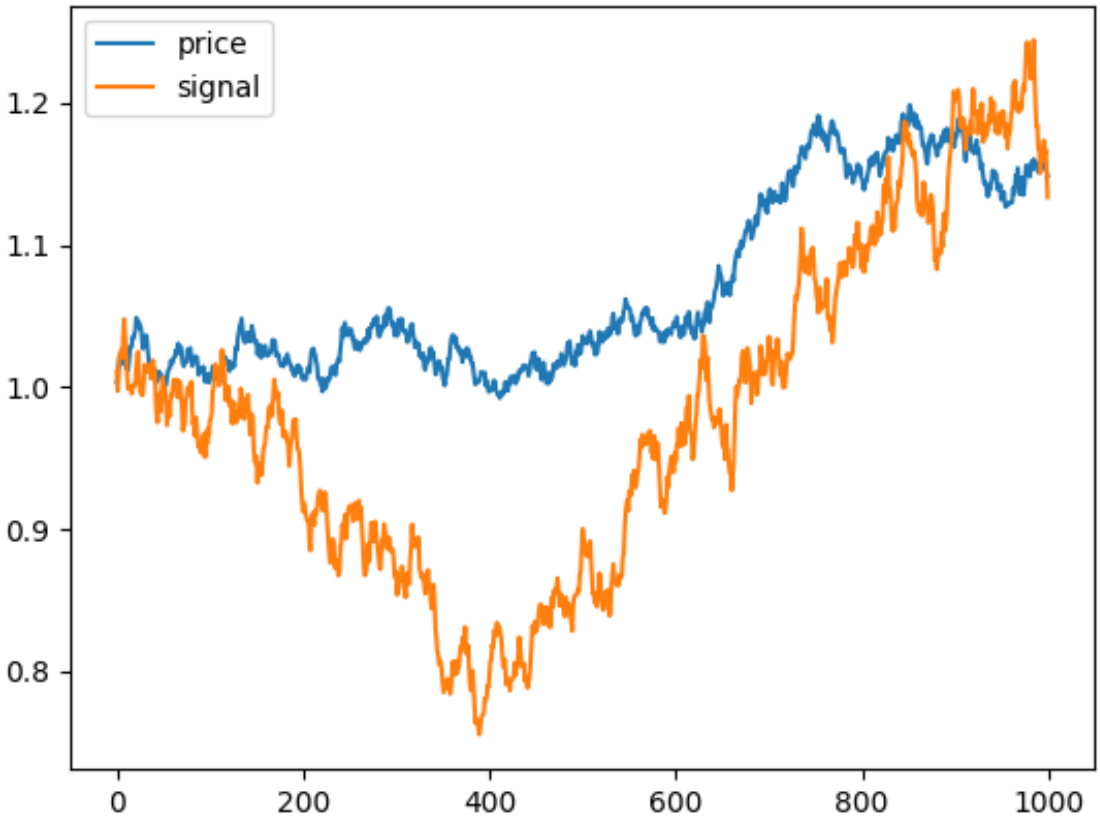
1.3.6 Creating simulated data for testing

For convenience, the `infertrade.data` module contains some basic functions for simulating market data.

```
import matplotlib.pyplot as plt
from infertrade.data.simulate_data import simulated_market_data_4_years_gen
simulated_market_data_4_years_gen().plot(y=["open", "close", "high", "low", "last"])
plt.show()
```




```
import matplotlib.pyplot as plt
from infertrade.data.simulate_data import simulated_correlated_equities_4_years_gen
simulated_correlated_equities_4_years_gen().plot(y=["price", "signal"])
plt.show()
```


price
signal
1.2
1.1
1.0
0.9
0.8
0
200
400
600
800
1000

CHAPTER

TWO

UTILITIES

2.1 utilities package

2.1.1 Submodules

2.1.2 utilities.operations module

This submodule includes facilities for operations such as converting positions to price predictions and vice versa.

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*)
This method is not used.

transform(*X: pandas.core.frame.DataFrame*, *y=None*) → pandas.core.frame.DataFrame
This method calculates the positions to be taken based on the forecast price, assuming the Kelly Criterion.

Args: X: A pandas.DataFrame object

Returns: A pandas.DataFrame object

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

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

fit(*X*, *y=None*)
This method is not used.

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

Args: X: A pandas.DataFrame object

Returns: A pandas.DataFrame object

class utilities.operations.**PricePredictionFromSignalRegression**(*market_to_trade: Optional[str] = None*)
Bases: sklearn.base.TransformerMixin, sklearn.base.BaseEstimator

This class creates price predictions from signal values.

Attributes: market_to_trade: The name of the column which contains the historical prices.

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

transform(*X: pandas.core.frame.DataFrame*, *y=None*) → pandas.core.frame.DataFrame
This method transforms a signal input to a price prediction.

Args: X: A pandas.DataFrame object

Returns: A pandas.DataFrame object

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

This class calculates returns from positions.

fit(*X*, *y=None*)
This method is not used.

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

Args: X: A pandas.DataFrame object

Returns: A pandas.DataFrame object

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.

Args: x: A numpy.ndarray or pandas.Series object

Returns: A numpy.ndarray with the results

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

Args: x: A numpy.ndarray or pandas.Series object shift: The number of periods by which to shift the input time series

Returns: A numpy.ndarray with the results

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

Args: x: A numpy.ndarray or pandas.Series object shift: The number of periods by which to shift the input time series

Returns: A numpy.ndarray with the results

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.

Args: x: A numpy.ndarray or pandas.Series object with exactly two columns shift: The number of periods by which the lag both series

Returns: A numpy.ndarray with the results

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).

Args: x: A numpy.ndarray or pandas.Series object window: The number of periods to be included in the moving average.

Returns: A numpy.ndarray with the results

utilities.operations.**pct_chg**(*x: Union[numpy.ndarray, pandas.core.series.Series]*) → numpy.ndarray

Percentage change between the current and a prior element.

Args: x: A numpy.ndarray or pandas.Series object

Returns: A numpy.ndarray with the results

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.

Args: x: A numpy.ndarray or pandas.Series object with exactly two columns shift: The number of periods by which the lag both series

Returns: A numpy.ndarray with the results

utilities.operations.**scikit_allocation_factory**(*allocation_function: callable*) → sklearn.preprocessing._function_transformer.FunctionTransforme

This function creates a SciKit Learn compatible Transformer embedding the position calculation.

Args: allocation_function: A function to be turned into a sklearn.preprocessing.FunctionTransformer

Returns: A sklearn.preprocessing.FunctionTransformer

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.

Args: x: A numpy.ndarray or pandas.Series object

Returns: A numpy.ndarray with the results

2.1.3 utilities.performance module

Performance calculation using the InferTrade interface.

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.

2.1.4 utilities.simple_functions module

Simple functions used across the package.

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

Adds a string to every item.

2.1.5 Module contents

Utilities directory for functions that uses the InferTrade interface.

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

u

INDEX