stock-bot-gav

Release 0.0.1

barbarich vi, taranenko gs, tsoi as, burkina es

Dec 27, 2022

CONTENTS:

CHAPTER

ONE

STOCK-BOT

1.1 simulation package

1.1.1 Module contents

1.2 src package

1.2.1 Subpackages

src.structures package

Submodules

src.structures.st_portfolio module

class Portfolio

Bases: object

__init__(*init_balance=100000*, *tickers=None*, *weights=None*, *strategy=None*, *type_process='sim'*)

Инициализация портфеля

Parameters

- **init_balance** (*[<class 'int'>, <class 'float'>]*) – начальный баланс портфеля;
- **tickers** (*Optional[list[str]]*) – список тикеров акций, которые будут входить в портфель. Если не указан, то портфель будет состоять из всех акций moex russia index;
- **weights** (*Optional[list[float]]*) – веса акций в портфеле. Если не указаны, то все акции будут иметь одинаковый вес;
- **strategy** (*Optional[callable]*) – стратегия, которая будет использоваться для обновления портфеля.
- **type_process** (*str*) – тип процесса, в котором работает портфель. Может быть 'sim' или 'real', соответственно симуляция или реальный процесс

async calc_amount(*strategy_response*)

Вычисляет количество бумаг для покупки / продажи

Parameters
strategy_response (StrategyResponse) – ответ стратегии;

Returns
количество бумаг для покупки

Return type
float

async calc_amount_buy(*strategy_response*)

Расчет количества свободных денег для покупки

Parameters
strategy_response (StrategyResponse) – ответ от стратегии на покупку;

Returns
количество свободных денег

Return type
float

async calc_amount_sell(*strategy_response*)

Расчет количества бумаг для продажи

Parameters
strategy_response (StrategyResponse) – ответ от стратегии на продажу;

Returns
количество бумаг

Return type
float

static calc_shares(*tickers*)

Расчет весов акций по ковариации

Parameters
tickers (*list[str]*) – датафрейм с ценами закрытия акций;

Returns
вектор весов акций

Return type
ndarray

async call_strategy()

Вызывает стратегию

Parameters
dtime_now – текущее время;

Returns
None

Return type
[<class 'src.structures.st_strategies.StrategyResponse'>]

async check_st_tp()

Проверяет стоп-лосс и тейк профит

Returns
продает или покупает бумаги

property free_balance: float

property full_balance: float

property history: str

async log_history(*timestamp*, *current_structure*, *received_structure*, *new_structure*)

После покупки логирует историю.

Parameters

- **timestamp** (*str*) – время совершения сделки. Формат '%Y-%m-%d %H:%M:%S' или '%Y-%m-%d'
- **current_structure** (Securities) – текущее состояние портфеля
- **received_structure** (Securities) – полученное состояние портфеля
- **new_structure** (Securities) – обновленное состояние портфеля

property securities: dict

async sell_all(*dtime_now*)

Продать все имеющиеся бумаги

Parameters
dtime_now (*str*) – текущее время;

Returns
None

Return type
None

property type_process: str

async update_securities(**args*)

Обновляет бумаги в портфеле.

Parameters
args (StrategyResponse) – Если позиция короткая, то количество отрицательное. Если бумаги были проданы или куплены количество соответственно положительное или отрицательное. После обновления цена будет перевзвешена в соответствии с количеством. Если мы стоим в короткой позиции, то покупка просто сократит количество бумаг, а на баланс может поступить положительная разница, при ее наличии. Если в короткой позиции мы продаем бумаги, то их цена перевзвешивается;

Returns
обновляет внутренний портфель

Return type
None

class PortfolioHistory

Bases: object

__init__()

Конструктор класса, отвечающего за историю по портфелю

get_history()

log_history(*balance*, *timestamp*, *current_structure*, *received_structure*, *new_structure*)

Логирование состояния портфеля

Parameters

- **balance** (*[<class 'float'>, <class 'int'>]*) – новый баланс портфеля
- **timestamp** (*str*) – дата и время изменения
- **current_structure** (Securities) – текущее состояние портфеля
- **received_structure** (Securities) – полученное состояние портфеля
- **new_structure** (Securities) – обновленное состояние портфеля

Returns

только обновляет историю

class Securities

Bases: defaultdict

get_json()

src.structures.st_purchase module

class DataMessage

Bases: object

Класс, в котором определена структура сообщения с данными.

__init__(*message_code=0*, *message_text=''*)

Конструктор класса

Parameters

- **message_code** (*int*) – флаг. 0 - все окей данные есть. 1 - данных нет, так как скорее всего торги не ведутся. 2 - тикер не найден;
- **message_text** (*str*) – сообщение

class StockPurchaseProcessMoex

Bases: object

__init__(*purchase_requests*)

Parameters

purchase_requests (*list[src.structures.st_purchase.StockPurchaseRequest]*) – запросы от стратегии

async static calc_purchase_quantity(*market_price*, *amt_assets*)

Рассчитывает количество акций, которое можно купить на указанную сумму

```
>>> import asyncio
>>> asyncio.run(StockPurchaseProcessMoex.calc_purchase_quantity(market_
↪price=100, amt_assets=1000))
10
>>> asyncio.run(StockPurchaseProcessMoex.calc_purchase_quantity(market_
↪price=100, amt_assets=90))
0
>>> asyncio.run(StockPurchaseProcessMoex.calc_purchase_quantity(market_
```

(continues on next page)

(continued from previous page)

```
↪price=100, amt_assets=-1000))
10
```

Parameters

- **market_price** (*float*) – цена акции;
- **amt_assets** (*float*) – сумма, которую можно потратить на покупку;

Returns
словарь с информацией о покупке

Return type
int

async static generate_message(*moex_dict*)

Генерирует сообщение о покупке

Parameters
moex_dict (*dict*) – словарь с информацией о покупке;

Returns
код + сообщение о том, как прошла покупка

Return type
DataMessage

class StockPurchaseRequest

Bases: object

__init__(*ticker*, *type_action*, *amt_assets*, *price=None*, *dtime_now=None*)

Parameters

- **ticker** (*str*) – тикер инструмента;
- **type_action** (TypeAction) – тип операции. 0 - ничего, -1 - продажа, 1 - покупка;
- **amt_assets** (*float*) – сумма, которая доступна к покупке. Должна быть указана в валюте покупки;
- **price** (*Optional[float]*) – цена, за которую нужно купить. Если None, то покупка по рыночной цене;
- **dtime_now** (*Optional[str]*) – дата и время, когда был совершен запрос на покупку. Если None, то текущее время. Для симуляций необходимо указывать дату и время, когда был совершен запрос на покупку. Дата-время должны быть в формате '%Y-%m-%d %H:%M:%S' или '%Y-%m-%d'

get_state()

```
>>> StockPurchaseRequest('sBer', TypeAction.BUY, amt_assets=100, dtime_now=
↪'2022-12-17').get_state()
{'ticker': 'SBER', 'type_action': 1, 'price': None, 'amt_assets': 100, 'dtime_
↪now': '2022-12-17'}
```

Returns
Возвращает словарь с информацией о покупке

Return type
dict

class StockPurchaseResponse

Bases: object

__init__(*message*, *ticker*, *market_price*, *quantity*, *lot_quantity*, *dt_purchase=None*, *exchange_fee=0*)

Parameters

- **message** (DataMessage) – код + сообщение о том, как прошла покупка;
- **ticker** (*str*) – Тикер бумаги
- **market_price** (*float*) – Актуальная цена бумаги на рынке
- **quantity** (*int*) – Количество купленных / проданных бумаг
- **lot_quantity** (*int*) – Количество лотов, которое было куплено / продано
- **dt_purchase** (*Optional[str]*) – Дата и время покупки
- **exchange_fee** (*float*) – Комиссия брокера

src.structures.st_securities module

class InfoSecurityRequest

Bases: object

__init__(*ticker*, *start*, *end*, *time_step*, *columns*)

Конструктор класса для запроса данных с биржи.

Parameters

- **ticker** (*str*) – название бумаги;
- **start** (*str*) – дата начала запроса;
- **end** (*str*) – дата конца запроса;
- **time_step** (*str*) – биржевой тик;
- **columns** (*list[str]*) – список столбцов

class SecurityState

Bases: object

__init__(*quantity=0*, *price=0*, *stop_loss=None*, *take_profit=None*)

Конструктор класса, который будет хранить состояние конкретной бумаги.

Note: Стоп лосс и тейк профит пока исполняют заявку по бумаге в полном объеме

Parameters

- **quantity** (*int*) – количество актива;
- **stop_loss** (*Optional[float]*) – Цена с целью ограничить свои убытки;
- **take_profit** (*Optional[float]*) – Цена, при которой мы получаем таргетированную выгоду;

- **price** (*float*) – цена актива

security_value()

Возвращает стоимость всех бумаг: quantity * price

```
>>> SecurityState(quantity=2, price=12.5).security_value()
25.0
```

Returns
стоимость всех бумаг

Return type
float

short_state()

```
>>> security_state = SecurityState(1, 2, 3, 4)
>>> security_state.short_state()
{'quantity': 1, 'price': 2, 'sl': 3, 'tp': 4}
```

Returns
возвращает состояние бумаги без ее истории. Не изменяет объект

Return type
SecurityState

update_state(*new_quantity=0*, *new_price=0*, *sl=None*, *tp=None*)

Обновляет состояние бумаги

```
>>> security_state = SecurityState(1, 2, 3, 4)
>>> security_state.update_state(5, 6)
>>> security_state
{'quantity': 5, 'price': 6, 'sl': 3, 'tp': 4}

>>> security_state.history_security
[{'quantity': 1, 'price': 2, 'sl': 3, 'tp': 4}, {'quantity': 5, 'price': 6,
↪'sl': 3, 'tp': 4}]
```

Parameters

- **new_quantity** (*int*) – количество актива;
- **new_price** (*float*) – цена актива
- **sl** (*Optional[float]*) – Цена с целью ограничить свои убытки;
- **tp** (*Optional[float]*) – Цена, при которой мы получаем таргетированную выгоду;

Return type
None

class StockSecurityPrice

Bases: object

__init__ (*ticker*, *market_price*)

Конструктор класс рыночной цены по бумаге

Parameters

- **ticker** (*str*) – тикер бумаги
- **market_price** (*float*) – рыночная цена

src.structures.st_strategies module

class BaseStrategy

Bases: object

Базовый класс стратегии. При создании экземпляра класса, в качестве аргументов передаются: Необходимые параметры для работы конкретной стратегии.

__init__ (**args*, ***kwargs*)

Parameters

- **args** –
- **kwargs** –

get_decision (**args*, ***kwargs*)

Метод, который возвращает решение стратегии.

Parameters

- **args** –
- **kwargs** –

Returns
StrategyResponse

Return type
StrategyResponse

get_request (**args*, ***kwargs*)

Метод, который возвращает запрос стратегии.

Parameters

- **args** –
- **kwargs** –

Returns
DataRequest

Return type
DataRequest

class DataRequest

Bases: object

__init__ (*tickers=None*, *dt_start=None*, *dt_end=None*, *dt_frequency=None*, *columns=('TRADEDATE', 'CLOSE')*)

Конструктор базового класса запроса стратегии.

Parameters

- **tickers** (*Optional[list[str]]*) – список тикеров, для которых нужно получить данные
- **dt_start** (*Optional[Union[Timestamp, str]]*) – дата и время начала периода
- **dt_end** (*Optional[Union[Timestamp, str]]*) – дата и время конца периода
- **dt_frequency** (*Optional[str]*) – частота данных
- **columns** (*list[str]*) – необходимые колонки, по умолчанию дата-время + цена закрытия

get_json()

Метод, который возвращает json-представление класса

Returns

json-представление класса

class StrategyResponse

Bases: object

Класс, в котором определена общая структура ответа от стратегии

__init__(*ticker=None*, *type_action=0*, *price=None*, *quantity=None*, *dtime_now=None*, *stop_loss=None*, *take_profit=None*, *comment=None*)

Конструктор класса

Parameters

- **ticker** (*Optional[str]*) – название инструмента / стратегии
- **type_action** (TypeAction) – флаг действия. 1 - покупка, -1 - продажа, 0 - ничего не делать
- **price** (*Optional[float]*) – цена
- **quantity** (*Optional[int]*) – количество акций
- **dtime_now** (*Optional[str]*) – дата и время
- **stop_loss** (*Optional[float]*) – стоп-лосс
- **take_profit** (*Optional[float]*) – тейк-профит
- **comment** (*Optional[str]*) – комментарий, описание действия, которое нужно совершить

class TypeAction

Bases: object

Класс, в котором определены флаги действий

BUY = 1

NOTHING = 0

SELL = -1

Module contents

1.2.2 Submodules

1.2.3 src.bot module

1.2.4 Module contents

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

S

INDEX