czsc

Release 0.9.50

zengbin93

May 12, 2024

目录

CHAPTER

ONE

学习资料

- 缠中说禅交易指南
- 缠中说禅图解分析示范
- 缠中说禅技术原理
- 缠中说禅重新编排版《论语》（整理版）

CHAPTER

TWO

CZSC API

2.1 Subpackages

2.1.1 czsc.data package

czsc.data.base Module

Functions

gm_symbol_to_jq(symbol)		掘金代码转聚宽代码
gm_symbol_to_tdx(symbol)		掘金代码转通达信代码
gm_symbol_to_ts(symbol)		掘金代码转 Tushare 代码
jq_symbol_to_gm(symbol)		聚宽代码转掘金代码
jq_symbol_to_tdx(symbol)		将聚宽的代码转成通达信代码
jq_symbol_to_ts(symbol)		将聚宽代码转成 Tushare 代码
save_symbols_to_ebk(symbols, source])	file_ebk[,	将股票代码列表保存到 EBK 文件，用来导入标的到同花顺、通达信软件中
tdx_symbol_to_gm(symbol)		将通达信的代码转成掘金代码
tdx_symbol_to_jq(symbol)		将通达信的代码转成聚宽代码
tdx_symbol_to_ts(symbol)		将通达信的代码转成 Tushare 代码
ts_symbol_to_gm(symbol)		将 Tushare 代码转成掘金代码
ts_symbol_to_jq(symbol)		将 Tushare 代码转成聚宽代码
ts_symbol_to_tdx(symbol)		将 Tushare 代码转成通达信代码

gm_symbol_to_jq

czsc.data.base.**gm_symbol_to_jq**(*symbol: str*) → str

掘金代码转聚宽代码

gm_symbol_to_tdx

czsc.data.base.**gm_symbol_to_tdx**(*symbol: str*) → str

掘金代码转通达信代码

gm_symbol_to_ts

czsc.data.base.**gm_symbol_to_ts**(*symbol: str*) → str

掘金代码转 Tushare 代码

jq_symbol_to_gm

czsc.data.base.**jq_symbol_to_gm**(*symbol: str*) → str

聚宽代码转掘金代码

jq_symbol_to_tdx

czsc.data.base.**jq_symbol_to_tdx**(*symbol*)

将聚宽的代码转成通达信代码

jq_symbol_to_ts

czsc.data.base.**jq_symbol_to_ts**(*symbol*)

将聚宽代码转成 Tushare 代码

save_symbols_to_ebk

czsc.data.base.**save_symbols_to_ebk**(*symbols*, *file_ebk*, *source='ts'*)

将股票代码列表保存到 EBK 文件，用来导入标的到同花顺、通达信软件中

Parameters

- **symbols** –股票代码列表
- **file_ebk** –EBK 结果文件

- **source** –代码格式

Returns

tdx_symbol_to_gm

czsc.data.base.**tdx_symbol_to_gm**(*symbol*)

将通达信的代码转成掘金代码

tdx_symbol_to_jq

czsc.data.base.**tdx_symbol_to_jq**(*symbol*)

将通达信的代码转成聚宽代码

tdx_symbol_to_ts

czsc.data.base.**tdx_symbol_to_ts**(*symbol*)

将通达信的代码转成 Tushare 代码

ts_symbol_to_gm

czsc.data.base.**ts_symbol_to_gm**(*symbol*)

将 Tushare 代码转成掘金代码

ts_symbol_to_jq

czsc.data.base.**ts_symbol_to_jq**(*symbol*)

将 Tushare 代码转成聚宽代码

ts_symbol_to_tdx

czsc.data.base.**ts_symbol_to_tdx**(*symbol*)

将 Tushare 代码转成通达信代码

czsc.data Package

Functions

deprecated(*args, **kwargs)		This is a decorator which can be used to mark functions as deprecated.
get_symbols(dc, step)		获取择时策略投研不同阶段对应的标的列表
gm_symbol_to_jq(symbol)		掘金代码转聚宽代码
gm_symbol_to_tdx(symbol)		掘金代码转通达信代码
gm_symbol_to_ts(symbol)		掘金代码转 Tushare 代码
jq_symbol_to_gm(symbol)		聚宽代码转掘金代码
jq_symbol_to_tdx(symbol)		将聚宽的代码转成通达信代码
jq_symbol_to_ts(symbol)		将聚宽代码转成 Tushare 代码
save_symbols_to_ebk(symbols, source])	file_ebk[,	将股票代码列表保存到 EBK 文件，用来导入标的到同花顺、通达信软件中
tdx_symbol_to_gm(symbol)		将通达信的代码转成掘金代码
tdx_symbol_to_jq(symbol)		将通达信的代码转成聚宽代码
tdx_symbol_to_ts(symbol)		将通达信的代码转成 Tushare 代码
ts_symbol_to_gm(symbol)		将 Tushare 代码转成掘金代码
ts_symbol_to_jq(symbol)		将 Tushare 代码转成聚宽代码
ts_symbol_to_tdx(symbol)		将 Tushare 代码转成通达信代码

get_symbols

czsc.data.**get_symbols**(*dc:* TsDataCache, *step*)

获取择时策略投研不同阶段对应的标的列表

Parameters

- **dc** –数据缓存
- **step** –投研阶段

Returns

gm_symbol_to_jq

czsc.data.**gm_symbol_to_jq**(*symbol: str*) → str

掘金代码转聚宽代码

gm_symbol_to_tdx

czsc.data.**gm_symbol_to_tdx**(*symbol: str*) → str

掘金代码转通达信代码

gm_symbol_to_ts

czsc.data.**gm_symbol_to_ts**(*symbol: str*) → str

掘金代码转 Tushare 代码

jq_symbol_to_gm

czsc.data.**jq_symbol_to_gm**(*symbol: str*) → str

聚宽代码转掘金代码

jq_symbol_to_tdx

czsc.data.**jq_symbol_to_tdx**(*symbol*)

将聚宽的代码转成通达信代码

jq_symbol_to_ts

czsc.data.**jq_symbol_to_ts**(*symbol*)

将聚宽代码转成 Tushare 代码

save_symbols_to_ebk

czsc.data.**save_symbols_to_ebk**(*symbols*, *file_ebk*, *source='ts'*)

将股票代码列表保存到 EBK 文件，用来导入标的到同花顺、通达信软件中

Parameters

- **symbols** –股票代码列表
- **file_ebk** –EBK 结果文件

- **source** –代码格式

Returns

tdx_symbol_to_gm

czsc.data.**tdx_symbol_to_gm**(*symbol*)

将通达信的代码转成掘金代码

tdx_symbol_to_jq

czsc.data.**tdx_symbol_to_jq**(*symbol*)

将通达信的代码转成聚宽代码

tdx_symbol_to_ts

czsc.data.**tdx_symbol_to_ts**(*symbol*)

将通达信的代码转成 Tushare 代码

ts_symbol_to_gm

czsc.data.**ts_symbol_to_gm**(*symbol*)

将 Tushare 代码转成掘金代码

ts_symbol_to_jq

czsc.data.**ts_symbol_to_jq**(*symbol*)

将 Tushare 代码转成聚宽代码

ts_symbol_to_tdx

czsc.data.**ts_symbol_to_tdx**(*symbol*)

将 Tushare 代码转成通达信代码

Classes

TsDataCache(*args, **kwargs)	Tushare 数据缓存

TsDataCache

class czsc.data.**TsDataCache**(**args*, ***kwargs*)

Bases: object

Tushare 数据缓存

Methods Summary

cctv_news([date])	新闻联播
clear()	清空缓存
daily_basic(ts_code, start_date, end_date)	每日指标
daily_basic_new(trade_date)	股票每日指标接口整合
get_all_ths_members([exchange, type_])	获取同花顺 A 股全部概念列表
get_dates_span(sdt, edt[, is_open])	获取日期区间列表
get_next_trade_dates(date[, n, m])	获取将来的交易日期
hk_hold([trade_date])	沪深港股通持股明细
index_weight(index_code, trade_date)	指数成分和权重
limit_list(trade_date)	https://tushare.pro/document/2?doc_id=198
pro_bar(ts_code[, start_date, end_date, ...])	获取日线以上数据
pro_bar_minutes(ts_code[, sdt, edt, freq, ...])	获取分钟线
stock_basic()	获取基础信息数据，包括股票代码、名称、上市日期、退市日期等
stocks_daily_bars([sdt, edt, adj])	读取 A 股全部历史日线
stocks_daily_basic_new(sdt, edt)	读取 A 股 sdt ~ edt 时间区间的全部历史 daily_basic_new
ths_daily(ts_code[, start_date, end_date, ...])	获取同花顺概念板块的日线行情
ths_index([exchange, type_])	获取同花顺概念
ths_member(ts_code)	获取同花顺概念成分股
trade_cal()	https://tushare.pro/document/2?doc_id=26

Methods Documentation

cctv_news(*date='20190625'*)

新闻联播

https://tushare.pro/document/2?doc_id=154

clear()

清空缓存

daily_basic(*ts_code: str*, *start_date: str*, *end_date: str*)

每日指标

https://tushare.pro/document/2?doc_id=32

daily_basic_new(*trade_date: str*)

股票每日指标接口整合

每日指标：https://tushare.pro/document/2?doc_id=32 备用列表：https://tushare.pro/document/2?doc_id=262

Parameters

trade_date –交易日期

Returns

get_all_ths_members(*exchange='A'*, *type_='N'*)

获取同花顺 A 股全部概念列表

get_dates_span(*sdt: str*, *edt: str*, *is_open: bool = True*) → List[str]

获取日期区间列表

Parameters

- **sdt** –开始日期
- **edt** –结束日期
- **is_open** –是否是交易日

Returns

日期区间列表

get_next_trade_dates(*date*, *n: int = 1*, *m: int | None = None*)

获取将来的交易日期

如果 m = None，返回基准日期后第 n 个交易日；否则返回基准日期后第 n ~ m 个交易日

Parameters

- **date** –基准日期
- **n** –

- **m** –

Returns

hk_hold(*trade_date='20190625'*)

沪深港股通持股明细

https://tushare.pro/document/2?doc_id=188

index_weight(*index_code: str*, *trade_date: str*)

指数成分和权重

https://tushare.pro/document/2?doc_id=96

limit_list(*trade_date: str*)

https://tushare.pro/document/2?doc_id=198

Parameters

trade_date –交易日期

Returns

每日涨跌停统计

pro_bar(*ts_code*, *start_date=None*, *end_date=None*, *freq='D'*, *asset='E'*, *adj='qfq'*, *raw_bar=True*)

获取日线以上数据

https://tushare.pro/document/2?doc_id=109

Parameters

- **ts_code** –
- **start_date** –
- **end_date** –
- **freq** –
- **asset** –资产类别：E 股票 I 沪深指数 C 数字货币 FT 期货 FD 基金 O 期权 CB 可转债（v1.2.39），默认 E
- **adj** –资产类别：E 股票 I 沪深指数 C 数字货币 FT 期货 FD 基金 O 期权 CB 可转债（v1.2.39），默认 E
- **raw_bar** –

Returns

pro_bar_minutes(*ts_code*, *sdt=None*, *edt=None*, *freq='60min'*, *asset='E'*, *adj=None*, *raw_bar=True*)

获取分钟线

https://tushare.pro/document/2?doc_id=109

Parameters

- **ts_code** –标的代码
- **sdt** –开始时间，精确到分钟
- **edt** –结束时间，精确到分钟
- **freq** –分钟周期，可选值 1min, 5min, 15min, 30min, 60min
- **asset** –资产类别：E 股票 I 沪深指数 C 数字货币 FT 期货 FD 基金 O 期权 CB 可转债（v1.2.39），默认 E
- **adj** –复权类型，None 不复权，qfq: 前复权，hfq: 后复权
- **raw_bar** –是否返回 RawBar 对象列表

Returns

stock_basic()

获取基础信息数据，包括股票代码、名称、上市日期、退市日期等

https://tushare.pro/document/2?doc_id=25

Returns

stocks_daily_bars(*sdt='20190101'*, *edt='20220218'*, *adj='hfq'*)

读取 A 股全部历史日线

Parameters

- **sdt** –开始日期
- **edt** –结束日期
- **adj** –复权类型

Returns

stocks_daily_basic_new(*sdt: str*, *edt: str*)

读取 A 股 sdt ~ edt 时间区间的全部历史 daily_basic_new

Parameters

- **sdt** –开始日期
- **edt** –结束日期

Returns

ths_daily(*ts_code*, *start_date=None*, *end_date=None*, *raw_bar=True*)

获取同花顺概念板块的日线行情

ths_index(*exchange='A'*, *type_='N'*)

获取同花顺概念

https://tushare.pro/document/2?doc_id=259

ths_member(*ts_code*)

获取同花顺概念成分股

https://tushare.pro/document/2?doc_id=261 :param ts_code: :return:

trade_cal()

https://tushare.pro/document/2?doc_id=26

2.2 czsc Package

2.2.1 Functions

cal_trade_price(bars[, decimals])	计算给定品种基础周期 K 线数据的交易价格
calculate_bi_info(bars, **kwargs)	计算笔的特征
check_abnormal_volume(df, **kwargs)	检查是否存在异常成交量。
check_freq_and_market(time_seq[, freq])	检查时间序列是否为同一周期，是否为同一市场
check_high_low(df)	检查是否存在 high < low 的情况。
check_price_gap(df, **kwargs)	检查是否存在超过阈值的大幅度缺口。
check_signals_acc(bars, signals_config[, ...])	输入基础周期 K 线和想要验证的信号，输出信号识别结果的快照
check_zero_volume(df)	计算零成交量的 K 线占比。
clear_cache([path, subs, recreate])	清空缓存文件夹
clear_strategy(strategy_name[, redis_url, ...])	删除策略所有记录
combine_dates_and_pairs(dates, pairs, ...)	结合大盘日期择时和择时策略开平交易进行分析
combine_holds_and_pairs(holds, pairs, ...)	结合股票池和择时策略开平交易进行分析
create_grid_params([prefix, multiply])	创建 grid search 参数组合
cross_sectional_ic(df[, x_col, y_col, method])	分析 df 中 x_col 和 y_col 列的截面相关性（IC）
cross_sectional_ranker(df, x_cols, y_col, ...)	截面打分排序
daily_performance(daily_returns)	采用单利计算日收益数据的各项指标
dill_dump(data, file)	
dill_load(file)	
disk_cache([path, suffix, ttl])	缓存装饰器，支持多种数据格式
empty_cache_path()	
feature_adjust(df, fcol, method, **kwargs)	特征调整函数：对特征进行调整，使其符合持仓权重的定义
feture_cross_layering(df, x_col, **kwargs)	对因子数据在时间截面上进行分层处理

continues on next page

Table 1 – continued from previous page

find_most_similarity(vector, matrix[, n, metric])	寻找向量在矩阵中最相似的 n 个向量
format_standard_kline(df, freq)	格式化标准 K 线数据为 CZSC 标准数据结构 RawBar 列表
freq_end_time(dt, freq[, market])	A 股与期货市场精确的获取 dt 对应的 K 线周期结束时间
freqs_sorted(freqs)	K 线周期列表排序并去重，第一个元素是基础周期
generate_czsc_signals(bars, signals_config)	使用 CzscSignals 生成信号
get_dir_size(path)	获取目录大小，单位：Bytes
get_ensemble_weight(trader[, method])	获取 CzscTrader 中所有 positions 按照 method 方法集成之后的权重
get_heartbeat_time([strategy_name, ...])	获取策略的最近一次心跳时间
get_intraday_times([freq, market])	获取指定市场的交易时间段
get_py_namespace(file_py[, keys])	获取 python 脚本文件中的 namespace
get_signals_config(signals_seq[, signals_module])	获取信号列表对应的信号函数配置
get_signals_freqs(signals_seq)	获取信号列表对应的 K 线周期列表
get_strategy_mates([redis_url, ...])	获取 Redis 中的策略元数据
get_strategy_weights(strategy_name[, ...])	获取策略的持仓权重
get_sub_elements(elements[, di, n])	获取截止到倒数第 di 个元素的前 n 个元素
get_trading_dates(sdt[, edt])	获取两个日期之间的所有交易日
get_unique_signals(bars, signals_config, ...)	获取信号函数中定义的所有信号列表
get_url_token(url)	获取指定 URL 数据接口的凭证码
holds_concepts_effect(holds, concepts[, ...])	股票持仓列表的板块效应
holds_performance(df, **kwargs)	组合持仓权重表现
import_by_name(name)	通过字符串导入模块、类、函数
index_composition(klines[, weights, base_point])	设置基点，按收益率加权合成指数
is_event_feature(df, col, **kwargs)	事件类因子的判断函数
is_trading_date([date])	判断是否是交易日
is_trading_time([dt, market])	判断指定时间是否是交易时间
long_short_equity(factors, returns[, ...])	根据截面因子值与收益率，回测分析多空对冲组合的收益率
net_value_stats(nv[, exclude_zero, sub_cost])	统计净值曲线的年化收益、夏普等
next_trading_date([date, n])	获取未来第 N 个交易日
normalize_corr(df, fcol[, ycol])	标准化因子与收益相关性为正数
normalize_feature(df, x_col, **kwargs)	因子标准化：缩尾，然后标准化
normalize_ts_feature(df, x_col[, n])	对时间序列数据进行归一化处理
optuna_good_params(study[, keep])	获取 optuna 优化结果中的最优参数
optuna_study(objective[, direction, n_trials])	使用 optuna 进行参数优化

continues on next page

Table 1 – continued from previous page

overlap(df, col, **kwargs)	给定 df 和 col，计算 col 中相同值的连续出现次数
prev_trading_date([date, n])	获取过去第 N 个交易日
print_df_sample(df[, n])	
psi(df, factor, segment, **kwargs)	PSI 群体稳定性指标，反映数据在不同分箱中的分布变化
read_json(file)	
resample_bars(df, target_freq[, raw_bars])	将给定的 K 线数据重新采样为目标周期的 K 线数据
resample_to_daily(df[, sdt, edt, ...])	将非日线数据转换为日线数据，以便进行日线级别的分析
risk_free_returns([start_date, end_date, ...])	创建无风险收益率序列
rolling_compare(df, col1, col2[, window, ...])	计算序列的滚动归一化值
rolling_corr(df, col1, col2[, window, ...])	滚动计算两个序列的相关系数
rolling_daily_performance(df, ret_col[, ...])	计算滚动日收益
rolling_norm(df, col[, window, min_periods, ...])	计算序列的滚动归一化值
rolling_qcut(df, col[, window, min_periods, ...])	计算序列的滚动分位数
rolling_rank(df, col[, window, min_periods, ...])	计算序列的滚动排名
rolling_scale(df, col[, window, ...])	对序列进行滚动归一化
rolling_slope(df, col[, window, ...])	计算序列的滚动斜率
rolling_tanh(df, col[, window, min_periods, ...])	对序列进行滚动 tanh 变换
save_json(data, file)	
set_url_token(token, url)	设置指定 URL 数据接口的凭证码，通常一台机器只需要设置一次即可
show_cointegration(df, col1, col2, **kwargs)	分析两个时间序列协整性，贡献者：珠峰
show_correlation(df[, cols, method])	用 streamlit 展示相关性
show_daily_return(df, **kwargs)	用 streamlit 展示日收益
show_drawdowns(df, ret_col, **kwargs)	展示最大回撤分析
show_event_return(df, factor, **kwargs)	分析事件因子的收益率特征
show_factor_layering(df, x_col[, y_col])	使用 streamlit 绘制因子截面分层收益率图
show_factor_returns(df, x_col, y_col)	使用 streamlit 展示因子收益率
show_monthly_return(df[, ret_col, sub_title])	展示指定列的月度累计收益
show_optuna_study(study, **kwargs)	
show_out_in_compare(df, ret_col, mid_dt, ...)	展示样本内外表现对比
show_psi(df, factor, segment, **kwargs)	PSI 分布稳定性
show_rolling_daily_performance(df, ret_col, ...)	展示滚动统计数据

continues on next page

Table 1 – continued from previous page

show_sectional_ic(df, x_col, y_col[, method])	使用 streamlit 展示截面 IC
show_splited_daily(df, ret_col, **kwargs)	展示分段日收益表现
show_stoploss_by_direction(dfw, **kwargs)	按方向止损分析的展示
show_strategies_dailys(df, **kwargs)	展示多策略多品种日收益率数据：按策略等权日收益
show_strategies_symbol(df, **kwargs)	展示多策略多品种日收益率数据：按品种等权日收益
show_symbol_factor_layering(df, x_col[, y_col])	使用 streamlit 绘制单个标的上的因子分层收益率图
show_ts_rolling_corr(df, col1, col2, **kwargs)	时序上按 rolling 的方式计算相关系数
show_ts_self_corr(df, col, **kwargs)	展示时序上单因子的自相关性分析结果，贡献者：guo
show_weight_backtest(dfw, **kwargs)	展示权重回测结果
show_yearly_stats(df, ret_col, **kwargs)	按年计算日收益表现
stock_holds_performance(dc, dfh, res_path)	计算 A 股日线持仓组合的表现
stoploss_by_direction(dfw[, stoploss])	按持仓方向进行止损
subtract_fee(df[, fee])	依据单品种持仓信号扣除手续费
symbols_bi_infos(symbols, read_bars[, freq, ...])	计算多个标的的笔特征
top_drawdowns(returns[, top])	分析最大回撤，返回最大回撤的波峰、波谷、恢复日期、回撤天数、恢复天数
update_bbars(da[, price_col, numbers])	在给定的 da 数据上计算并添加前面 n 根 bar 的累计收益列
update_nxb(df, **kwargs)	在给定的 df 上计算并添加后面 n 根 bar 的累计收益列
update_tbars(da, event_col)	计算带 Event 方向信息的未来收益
weekly_performance(weekly_returns)	采用单利计算周收益数据的各项指标
welcome()	
x_round(x[, digit])	用去尾法截断小数

cal_trade_price

czsc.**cal_trade_price**(*bars: List[RawBar] | DataFrame*, *decimals=3*, ***kwargs*)

计算给定品种基础周期 K 线数据的交易价格

函数执行逻辑：

1. 首先，根据输入的 bars 参数类型（列表或 DataFrame），将其转换为 DataFrame 格式，并将其存储在变量 df 中。
2. 计算下一根 K 线的开盘价和收盘价，分别存储在新列 next_open 和 next_close 中。同时，将这两个

新列名添加到 price_cols 列表中。

3. 计算 TWAP（时间加权平均价格）和 VWAP（成交量加权平均价格）。为此，函数使用了一个 for 循环，遍历 t_seq 参数（默认值为 (5, 10, 15, 20, 30, 60)）。在每次循环中：
   - 计算 TWAP：使用 rolling(t).mean().shift(-t) 方法计算时间窗口为 t 的滚动平均收盘价。
   - 计算 VWAP：首先计算滚动窗口内的成交量之和（sum_vol_t）和成交量乘以收盘价之和（sum_vcp_t），然后用后者除以前者，并向下移动 t 个单位。
   - 将 TWAP 和 VWAP 的列名添加到 price_cols 列表中。
4. 遍历 price_cols 列表中的每个列，将其中的 NaN 值替换为对应行的收盘价。
5. 从 DataFrame 中选择所需的列（包括基本的 K 线数据列和新计算的交易价格列），并使用 round(decimals) 方法保留指定的小数位数（默认为 3）。
6. 返回处理后的 DataFrame。

Parameters

- **bars** –基础周期 K 线数据，一般是 1 分钟周期的 K 线
- **decimals** –保留小数位数，默认值 3

Returns

交易价格表

calculate_bi_info

czsc.**calculate_bi_info**(*bars: List[RawBar]*, ***kwargs*) → DataFrame

计算笔的特征

Parameters

bars –原始 K 线数据

Returns

笔的特征

check_abnormal_volume

czsc.**check_abnormal_volume**(*df*, ***kwargs*)

检查是否存在异常成交量。

check_freq_and_market

czsc.**check_freq_and_market**(*time_seq: List*, *freq: AnyStr | None = None*)

检查时间序列是否为同一周期，是否为同一市场

函数计算逻辑：

1. 如果 `freq`在特定列表中，函数直接返回 `freq`和”默认”作为市场类型。
2. 如果 `freq`是’1 分钟’，函数会添加额外的时间点到 `time_seq`中。
3. 函数去除 `time_seq`中的重复时间点，并确保其长度至少为 2。
4. 函数遍历 `freq_market_times`字典，寻找与 `time_seq`匹配的项，并返回对应的 `freq_x`和 `market`。
5. 如果没有找到匹配的项，函数返回 None 和”默认”。

Parameters

- **time_seq** –时间序列，如 [‘11:00’，‘15:00’，‘23:00’，‘01:00’，‘02:30’]
- **freq** –时间序列对应的 K 线周期，可选参数，使用该参数可以加快检查速度。可选值：1 分钟、5 分钟、15 分钟、30 分钟、60 分钟、日线、周线、月线、季线、年线

Returns

- freq K 线周期
- market 交易市场

check_high_low

czsc.**check_high_low**(*df*)

检查是否存在 high < low 的情况。

check_price_gap

czsc.**check_price_gap**(*df*, ***kwargs*)

检查是否存在超过阈值的大幅度缺口。

check_signals_acc

czsc.**check_signals_acc**(*bars: List[RawBar]*, *signals_config: List[dict]*, *delta_days: int = 5*, ***kwargs*) → None

输入基础周期 K 线和想要验证的信号，输出信号识别结果的快照

函数执行逻辑：

1. 函数首先获取基础周期 K 线的 base_freq，并检查输入的 K 线数据 bars 是否按时间升序排列。如果 bars 的长度小于 600，函数直接返回。
2. 然后，函数调用 generate_czsc_signals 方法，生成 Czsc 信号，并将结果保存在 df 中。
3. 函数提取出 df 中所有的信号列 s_cols，并打印每一列的值的数量。然后，函数将所有的信号添加到 signals 列表中。
4. 函数将 bars 分为两部分，bars_left 和 bars_right，并获取信号配置 signals_config 中的所有 freqs。
5. 函数创建一个 BarGenerator 对象 bg，并使用 bars_left 中的 K 线数据来初始化它。
6. 函数创建一个 CzscSignals 对象 ct，并将 bg 和信号配置 signals_config 作为参数传入。
7. 函数创建一个字典 last_dt，用于存储每一个信号最后一次出现的时间。
8. 函数遍历 bars_right 中的每一根 K 线，对于每一根 K 线，函数调用 ct.update_signals(bar) 来更新信号。
9. 对于每一个信号，如果当前 K 线的时间与该信号最后一次出现的时间的差值大于 delta_days，并且该信号与当前的信号匹配，函数将创建一个 HTML 文件，保存信号识别结果的快照，并更新该信号最后一次出现的时间。

Parameters

- **bars** –原始 K 线
- **signals_config** –需要验证的信号列表
- **delta_days** –两次相同信号之间的间隔天数

Returns

None

check_zero_volume

czsc.**check_zero_volume**(*df*)

计算零成交量的 K 线占比。

clear_cache

czsc.**clear_cache**(*path: AnyStr | Path = PosixPath('/home/docs/.czsc')*, *subs=None*, *recreate=False*)

清空缓存文件夹

Parameters

- **path** –缓存文件夹路径
- **subs** –需要清空的子文件夹名称，如果为 None，则清空整个文件夹

- **recreate** –是否重新创建文件夹, True 时会重新创建文件夹, False 时不会重新创建文件夹

clear_strategy

czsc.**clear_strategy**(*strategy_name*, *redis_url=None*, *connection_pool=None*, *key_prefix='Weights'*, ***kwargs*)

删除策略所有记录

Parameters

- **strategy_name** –str, 策略名
- **redis_url** –str, redis 连接字符串, 默认为 None, 即从环境变量 RWC_REDIS_URL 中读取
- **connection_pool** –redis.ConnectionPool, redis 连接池
- **key_prefix** –str, redis 中 key 的前缀，默认为 Weights
- **kwargs** –dict, 其他参数

combine_dates_and_pairs

czsc.**combine_dates_and_pairs**(*dates: list*, *pairs: DataFrame*, *results_path*)

结合大盘日期择时和择时策略开平交易进行分析

函数计算逻辑:

1. 将 dates 转换为日期类型，并赋值给变量 dates。
2. 将 pairs 复制到 dfp 变量。
3. 将 dfp 的’开仓时间’列转换为日期类型，并将日期部分提取出来赋值给’开仓日期’列。
4. 从 dfp 中选择开仓日期在 dates 中的数据，赋值给 df_pairs。
5. 从 dfp 中选择开仓时间在 df_pairs 的开仓时间范围内的数据，赋值给 dfp_sub。
6. 使用 dfp_sub 创建 PairsPerformance 对象 tp_old。
7. 使用 df_pairs 创建 PairsPerformance 对象 tp_new。
8. 打印原始交易的基本信息和平仓年度统计。
9. 打印组合过滤后的交易的基本信息和平仓年度统计。
10. 创建结果目录并保存评价结果和交易数据。
11. 返回 tp_old 和 tp_new 对象。

Parameters

- **dates** –大盘日期择时日期数据，数据样例 [‘2020-01-02’, …, ‘2022-01-06’]

- **pairs** –

  择时策略开平交易数据，数据格式如下
  标的代码交易方向最大仓位开仓时间累计开仓平仓时间 0 002698.SZ 多头 1 2015-01-12 13:30:00 24.02790 2015-01-13 09:45:00

  1 300031.SZ 多头 1 2015-01-12 10:30:00 53.87420 2015-01-13 09:45:00 2 300046.SZ 多头 1 2015-01-12 10:15:00 41.35824 2015-01-13 09:45:00 3 300076.SZ 多头 1 2015-01-12 10:30:00 57.84800 2015-01-13 09:45:00 4 300099.SZ 多头 1 2015-01-12 10:15:00 62.57308 2015-01-13 09:45:00

  累计平仓累计换手持仓 K 线数持仓天数盈亏金额交易盈亏盈亏比例

  0 23.38150 2 7 0.843750 -0.64640 -0.0269 -0.0269 1 52.71284 2 13 0.968750 -1.16136 -0.0215 -0.0215 2 40.72068 2 14 0.979167 -0.63756 -0.0154 -0.0154 3 55.45144 2 13 0.968750 -2.39656 -0.0414 -0.0414 4 61.50528 2 14 0.979167 -1.06780 -0.0170 -0.0170

- **results_path** –分析结果目录

Returns

combine_holds_and_pairs

czsc.**combine_holds_and_pairs**(*holds*, *pairs*, *results_path*)

结合股票池和择时策略开平交易进行分析

函数计算逻辑:

1. **将 holds 和 pairs 数据进行处理和准备**。
   - 将 holds 复制到 dfh 变量。
   - 将 dfh 的’成分日期’列转换为日期类型。
   - 将 dfh 的’证券代码’列赋值给’标的代码’列。
   - 将 pairs 复制到 dfp 变量。
   - 将 dfp 的’开仓时间’列转换为日期类型，并将日期部分提取出来赋值给’开仓日期’列。
2. **合并数据并筛选交易对**。
   - 将 dfp 与 dfh 的 [[‘开仓日期’，‘标的代码’，‘持仓权重’]] 列进行左连接，**得到 dfp_**。
   - 从 dfp_ 中选择持仓权重大于 0 的交易对，赋值给 df_pairs。
   - 从 dfp 中选择开仓时间在 df_pairs 的开仓时间范围内的数据，赋值给 dfp_sub。
3. **进行评价和分析**。
   - 使用 dfp_sub 创建 PairsPerformance 对象 tp_old。
   - 使用 df_pairs 创建 PairsPerformance 对象 tp_new。

4. **创建结果目录并保存评价结果和交易数据**。
    - 使用 os.makedirs 创建结果目录。
    - 将 tp_old 的统计结果保存为 Excel 文件，文件名为”原始交易评价.xlsx”。
    - 将 tp_new 的统计结果保存为 Excel 文件，文件名为”组合过滤评价.xlsx”。
    - 将 df_pairs 的数据保存为 Feather 文件，文件名为”组合过滤交易.feather”。
5. 返回 tp_old 和 tp_new 对象。

Parameters

- **holds** –

  组合股票池数据，样例：
  成分日期证券代码 n1b 持仓权重

  0 2020-01-02 000001.SZ 183.758194 0.001232 1 2020-01-02 000002.SZ -156.633896 0.001232 2 2020-01-02 000063.SZ 310.296204 0.001232 3 2020-01-02 000066.SZ -131.824997 0.001232 4 2020-01-02 000069.SZ -38.561699 0.001232

- **pairs** –

  择时策略开平交易数据，数据格式如下
  标的代码交易方向最大仓位开仓时间累计开仓平仓时间 0 002698.SZ 多头 1 2015-01-12 13:30:00 24.02790 2015-01-13 09:45:00

  1 300031.SZ 多头 1 2015-01-12 10:30:00 53.87420 2015-01-13 09:45:00 2 300046.SZ 多头 1 2015-01-12 10:15:00 41.35824 2015-01-13 09:45:00 3 300076.SZ 多头 1 2015-01-12 10:30:00 57.84800 2015-01-13 09:45:00 4 300099.SZ 多头 1 2015-01-12 10:15:00 62.57308 2015-01-13 09:45:00

  累计平仓累计换手持仓 K 线数持仓天数盈亏金额交易盈亏盈亏比例

  0 23.38150 2 7 0.843750 -0.64640 -0.0269 -0.0269 1 52.71284 2 13 0.968750 -1.16136 -0.0215 -0.0215 2 40.72068 2 14 0.979167 -0.63756 -0.0154 -0.0154 3 55.45144 2 13 0.968750 -2.39656 -0.0414 -0.0414 4 61.50528 2 14 0.979167 -1.06780 -0.0170 -0.0170

- **results_path** –分析结果目录

Returns

create_grid_params

czsc.**create_grid_params**(*prefix: str = ''*, *multiply=3*, ***kwargs*) → dict

创建 grid search 参数组合

Parameters

- **prefix** –参数组前缀
- **multiply** –参数组合的位数，如果为 0，则使用 # 分隔参数
- **kwargs** –任意参数的候选序列，参数值推荐使用 iterable

Returns

参数组合字典

Examples

>>>x = create_grid_params(“test”, x=(1, 2), y=(‘a’, ‘b’), detail=True) >>>print(x) Out[0]:

{‘test_x=1_y=a’: {‘x’: 1, ‘y’: ‘a’},
‘test_x=1_y=b’: {‘x’: 1, ‘y’: ‘b’}, ‘test_x=2_y=a’: {‘x’: 2, ‘y’: ‘a’},
‘test_x=2_y=b’: {‘x’: 2, ‘y’: ‘b’}}

单个参数传入单个值也是可以的，但类型必须是 int, float, str 中的任一 >>>x = create_grid_params(“test”, x=2, y=(‘a’, ‘b’), detail=False) >>>print(x) Out[1]:

{‘test001’: {‘x’: 2, ‘y’: ‘a’},
‘test002’: {‘x’: 2, ‘y’: ‘b’}}

cross_sectional_ic

czsc.**cross_sectional_ic**(*df*, *x_col='open'*, *y_col='n1b'*, *method='spearman'*, ***kwargs*)

分析 df 中 x_col 和 y_col 列的截面相关性（IC）

:param df：数据，DateFrame 格式:param x_col：X 列:param y_col：Y 列，一般采用下期收益，也就是 n1b :param method：{‘pearson’, ‘kendall’, ‘spearman’} or callable

- pearson : standard correlation coefficient
- kendall : Kendall Tau correlation coefficient
- spearman : Spearman rank correlation
- callable: callable with input two 1d ndarrays and returning a float

Return：df，res

前者是每日相关系数结果，后者是每日相关系数的统计结果

cross_sectional_ranker

czsc.**cross_sectional_ranker**(*df*, *x_cols*, *y_col*, ***kwargs*)

截面打分排序

Parameters

- **df** –因子数据，必须包含日期、品种、因子值、预测列，且按日期升序排列，样例数据如下：
- **x_cols** –因子列名
- **y_col** –预测列名
- **kwargs** –其他参数
  - model_params: dict, 模型参数，默认 {‘n_estimators’: 40, ‘learning_rate’: 0.01}，可调整，参考 lightgbm 文档
  - n_splits: int, 时间拆分次数，默认 5，即 5 段时间
  - rank_ascending: bool, 打分排序是否升序，默认 False-降序
  - copy: bool, 是否拷贝 df，True-拷贝，False-不拷贝

Returns

df, 包含预测分数和排序列

daily_performance

czsc.**daily_performance**(*daily_returns*)

采用单利计算日收益数据的各项指标

函数计算逻辑：

1. 首先，将传入的日收益率数据转换为 NumPy 数组，并指定数据类型为 float64。
2. 然后，进行一系列判断：如果日收益率数据为空或标准差为零或全部为零，则返回一个字典，其中所有指标的值都为零。
3. 如果日收益率数据满足要求，则进行具体的指标计算：
   - 年化收益率 = 日收益率列表的和 / 日收益率列表的长度 * 252
   - 夏普比率 = 日收益率的均值 / 日收益率的标准差 * 标准差的根号 252
   - 最大回撤 = 累计日收益率的最高累积值 - 累计日收益率
   - 卡玛比率 = 年化收益率 / 最大回撤（如果最大回撤不为零，则除以最大回撤；否则为 10）
   - 日胜率 = 大于零的日收益率的个数 / 日收益率的总个数
   - 年化波动率 = 日收益率的标准差 * 标准差的根号 252

- 非零覆盖 = 非零的日收益率个数 / 日收益率的总个数

4. 将所有指标的值存储在一个字典中，其中键为指标名称，值为相应的计算结果。

Parameters

daily_returns –日收益率数据，样例：[0.01, 0.02, -0.01, 0.03, 0.02, -0.02, 0.01, -0.01, 0.02, 0.01]

Returns

dict

dill_dump

czsc.**dill_dump**(*data*, *file*)

dill_load

czsc.**dill_load**(*file*)

disk_cache

czsc.**disk_cache**(*path: AnyStr | Path = PosixPath('/home/docs/.czsc')*, *suffix: str = 'pkl'*, *ttl: int = -1*)

缓存装饰器，支持多种数据格式

Parameters

- **path** –缓存文件夹父路径，默认为 home_path，每个函数的缓存文件夹为 path/func_name
- **suffix** –缓存文件后缀，支持 pkl, json, txt, csv, xlsx, feather, parquet
- **ttl** –缓存文件有效期，单位：秒

empty_cache_path

czsc.**empty_cache_path**()

feature_adjust

czsc.**feature_adjust**(*df: DataFrame*, *fcol*, *method*, ***kwargs*)

特征调整函数：对特征进行调整，使其符合持仓权重的定义

Parameters

- **df** –pd.DataFrame, 待调整的数据
- **fcol** –str, 因子列名
- **method** –str, 调整方法
  - KEEP: 直接使用原始因子值作为权重
  - V230101: 对因子进行滚动相关系数计算，然后对因子值用 maxabs_scale 进行归一化，最后乘以滚动相关系数的符号
  - V240323: 对因子进行滚动相关系数计算，然后对因子值用 scale + tanh 进行归一化，最后乘以滚动相关系数的符号
- **kwargs** –dict
  - window: int, 滚动窗口大小
  - min_periods: int, 最小计算周期

Returns

pd.DataFrame, 新增 weight 列

feture_cross_layering

czsc.**feture_cross_layering**(*df*, *x_col*, ***kwargs*)

对因子数据在时间截面上进行分层处理

函数计算逻辑：

1. 首先从参数中获取分层数量 n，默认为 10。
2. 确保数据 df 包含 dt、symbol 和指定的因子列 x_col，确保标的数量大于分层数量。
3. 如果因子列的唯一值数量大于分层数量，使用 pd.qcut 函数将因子列进行分层，按照分位数进行分组。
4. 如果因子列的唯一值数量小于等于分层数量，按照因子列的唯一值进行排序，并将每个因子值映射为对应的层级。
5. 将分层结果转换为字符串形式，以表示层级。

Parameters

- **df** –因子数据，数据样例：

dt	symbol	factor01	factor02	factor03
2022-12-19 00:00:00	ZZUR9001	-0.0221211	0.034236	0.0793672
2022-12-20 00:00:00	ZZUR9001	-0.0278691	0.0275818	0.0735083
2022-12-21 00:00:00	ZZUR9001	-0.00617075	0.0512298	0.0990967
2022-12-22 00:00:00	ZZUR9001	-0.0222238	0.0320096	0.0792036
2022-12-23 00:00:00	ZZUR9001	-0.0375133	0.0129455	0.059491

- **x_col** –因子列名
- **kwargs** –
  - n: 分层数量，默认为 10

Returns

df, 添加了 x_col 分层列

find_most_similarity

czsc.**find_most_similarity**(*vector: Series*, *matrix: DataFrame*, *n=10*, *metric='cosine'*, ***kwargs*)

寻找向量在矩阵中最相似的 n 个向量

Parameters

- **vector** –1 维向量, Series 结构
- **matrix** –2 维矩阵, DataFrame 结构, 每一列是一个向量，列名是向量的标记
- **n** –int, 返回最相似的 n 个向量
- **metric** –str, 计算相似度的方法，
  - From scikit-learn: [‘cityblock’, ‘cosine’, ‘euclidean’, ‘l1’, ‘l2’, ’manhattan’]. These metrics support sparse matrix inputs. [‘nan_euclidean’] but it does not yet support sparse matrices.
  - From scipy.spatial.distance: [‘braycurtis’, ‘canberra’, ‘chebyshev’, ’correlation’, ‘dice’, ‘hamming’, ‘jaccard’, ‘kulsinski’, ‘mahalanobis’, ‘minkowski’, ‘rogerstanimoto’, ‘russellrao’, ‘seuclidean’, ‘sokalmichener’, ‘sokalsneath’, ‘sqeuclidean’, ‘yule’] See the documentation for scipy.spatial.distance for details on these metrics. These metrics do not support sparse matrix inputs.
- **kwargs** –其他参数

format_standard_kline

czsc.**format_standard_kline**(*df: DataFrame*, *freq: str*)

格式化标准 K 线数据为 CZSC 标准数据结构 RawBar 列表

Parameters

- **df** –标准 K 线数据，DataFrame 结构

dt	symbol	open	close	high	low	vol	amount
2023-11-17 00:00:00	689009.SH	33.52	33.41	33.69	33.38	1.97575e+0(	6.61661e+07
2023-11-20 00:00:00	689009.SH	33.4	32.91	33.45	32.25	5.15016e+0(	1.68867e+08

- **freq** –K 线级别

Returns

list of RawBar

freq_end_time

czsc.**freq_end_time**(*dt: datetime*, *freq:* Freq | *AnyStr*, *market='A 股'*) → datetime

A 股与期货市场精确的获取 dt 对应的 K 线周期结束时间

Parameters

- **dt** –datetime
- **freq** –Freq

Returns

datetime

freqs_sorted

czsc.**freqs_sorted**(*freqs*)

K 线周期列表排序并去重，第一个元素是基础周期

Parameters

freqs –K 线周期列表

Returns

K 线周期排序列表

generate_czsc_signals

czsc.**generate_czsc_signals**(*bars: List[RawBar]*, *signals_config: List[dict]*, *sdt: AnyStr | datetime = '20170101'*, *init_n: int = 500*, *df=False*, ***kwargs*)

使用 CzscSignals 生成信号

函数执行逻辑：

1. 函数首先从信号配置 signals_config 中获取所有的 freqs。
2. 然后，函数将信号计算开始时间 sdt 转换为 datetime 类型，并将开始时间之前的 K 线数据分配给 bars_left，开始时间之后的 K 线数据分配给 bars_right。
3. 如果 bars_right 为空，即没有开始时间之后的 K 线数据，函数会发出一个警告，并返回一个空的 DataFrame 或空列表。
4. 函数创建一个 BarGenerator 对象 bg，并使用 bars_left 中的 K 线数据来初始化它。
5. 函数创建一个 CzscSignals 对象 cs，并将 bg 和信号配置 signals_config 作为参数传入。
6. 函数遍历 bars_right 中的每一根 K 线，对于每一根 K 线，函数调用 cs.update_signals(bar) 来更新信号，并将更新后的信号添加到 _sigs 列表中。
7. 最后，如果 df 参数为 True，函数将 _sigs 转换为 DataFrame 并返回；否则，直接返回 _sigs。

Parameters

- **bars** –基础周期 K 线序列
- **signals_config** –信号函数配置，格式如下：signals_config = [

  {‘name’: ‘czsc.signals.tas_ma_base_V221101’, ‘freq’: ‘日线’, ‘di’: 1, ‘ma_type’ : ‘SMA’, ‘timeperiod’ : 5}, { ‘name’ : ‘czsc.signals.tas_ma_base_V221101’, ‘freq’ : ‘日线’, ‘di’ : 5, ‘ma_type’ : ‘SMA’, ‘timeperiod’ : 5}, { ‘name’ : ‘czsc.signals.tas_double_ma_V221203’, ‘freq’ : ‘日线’, ‘di’ : 1, ‘ma_seq’ : (5, 20), ‘th’ : 100}, { ‘name’ : ‘czsc.signals.tas_double_ma_V221203’, ‘freq’ : ‘日线’, ‘di’ : 5, ‘ma_seq’ : (5, 20), ‘th’ : 100},

  ]
- **sdt** –信号计算开始时间
- **init_n** –用于 BarGenerator 初始化的基础周期 K 线数量
- **df** –是否返回 df 格式的信号计算结果，默认 False

Returns

信号计算结果

get_dir_size

czsc.**get_dir_size**(*path*)

获取目录大小，单位：Bytes

get_ensemble_weight

czsc.**get_ensemble_weight**(*trader: CzscTrader*, *method: AnyStr | Callable = 'mean'*)

获取 CzscTrader 中所有 positions 按照 method 方法集成之后的权重

函数计算逻辑：

1. 获取 trader 持仓信息并转换为 DataFrame:
   - 遍历交易者的每个持仓位置。
   - 将每个位置的持仓信息转换为 DataFrame，并合并到一个整体的 DataFrame 中。
   - 将持仓列重命名为对应的位置名称。
2. 根据给定的方法计算权重:
   - 如果方法是可调用对象，将持仓信息转换为字典，并传递给该方法进行计算。
   - 如果方法是预定义字符串（" mean"、" max"、" min"、" vote"），根据相应的计算方式计算权重。
3. 返回包含日期、交易标的、权重和价格的 DataFrame:
   - 将计算得到的权重与其他相关列一起组成一个新的 DataFrame。
   - 将交易标的信息添加到新的 DataFrame 中。
   - 返回包含日期、交易标的、权重和价格的 DataFrame 副本。

Parameters

- **trader** –CzscTrader 缠论交易员对象
- **method** –str or callable

  集成方法，可选值包括：'mean'，'max'，'min'，'vote' 也可以传入自定义的函数，函数的输入为 dict，key 为 position.name，value 为 position.pos, 样例输入：

  {'多头策略 A'：1, '多头策略 B'：1, '空头策略 A'：-1}
- **kwargs** –

Returns

pd.DataFrame columns = ['dt', 'symbol', 'weight', 'price']

get_heartbeat_time

czsc.**get_heartbeat_time**(*strategy_name=None*, *redis_url=None*, *connection_pool=None*, *key_prefix='Weights'*, ***kwargs*)

获取策略的最近一次心跳时间

Parameters

- **strategy_name** –str, 策略名，默认为 None, 即获取所有策略的心跳时间
- **redis_url** –str, redis 连接字符串, 默认为 None, 即从环境变量 RWC_REDIS_URL 中读取
- **connection_pool** –redis.ConnectionPool, redis 连接池
- **key_prefix** –str, redis 中 key 的前缀，默认为 Weights
- **kwargs** –dict, 其他参数
  - heartbeat_prefix: str, 心跳 key 的前缀，默认为 heartbeat

Returns

str, 最近一次心跳时间

get_intraday_times

czsc.**get_intraday_times**(*freq='1 分钟'*, *market='A 股'*)

获取指定市场的交易时间段

Parameters

market –市场名称，可选值：A 股、期货、默认

Returns

交易时间段列表

get_py_namespace

czsc.**get_py_namespace**(*file_py: str*, *keys: list = []*) → dict

获取 python 脚本文件中的 namespace

Parameters

- **file_py** –python 脚本文件名
- **keys** –指定需要的对象名称

Returns

namespace

get_signals_config

czsc.**get_signals_config**(*signals_seq: List[str]*, *signals_module: str = 'czsc.signals'*) → List[Dict]

获取信号列表对应的信号函数配置

函数执行逻辑：

1. **首先创建了一个 SignalsParser 类的实例对象 sp，传入了参数 signals_module 进行初始化，**
   初始化工作主要是解析 signals_module 下的信号函数，生成了 sig_pats_map 信号参数模板字典和 sig_name_map 信号列表字典。
2. 然后使用 sp 实例调用 parse 方法，该方法解析 signals_seq 中的信号，并返回信号函数的配置信息。

Parameters

- **signals_seq** –信号列表
- **signals_module** –信号函数所在模块

Returns

信号函数配置

get_signals_freqs

czsc.**get_signals_freqs**(*signals_seq: List*) → List[str]

获取信号列表对应的 K 线周期列表

函数执行逻辑：

1. 然后对于 signals_seq 中的每个信号进行以下操作：
   - 使用正则表达式从信号中提取信号周期，并将其存储在 _freqs 变量中。
   - 如果提取到了信号周期，则将其加入到 freqs 列表中。
2. 最后验证数据是否符合 sorted_freqs 列表规范，并且以 sorted_freqs 列表的排序进行返回。

Parameters

signals_seq –信号列表 / 信号函数配置列表

Returns

K 线周期列表

get_strategy_mates

czsc.**get_strategy_mates**(*redis_url=None*, *connection_pool=None*, *key_pattern='Weights:META:*'*, ***kwargs*)

获取 Redis 中的策略元数据

Parameters

- **redis_url** –str, redis 连接字符串, 默认为 None, 即从环境变量 RWC_REDIS_URL 中读取
- **connection_pool** –redis.ConnectionPool, redis 连接池
- **key_pattern** –str, redis 中 key 的 pattern，默认为 Weights:META:*
- **kwargs** –dict, 其他参数

Returns

pd.DataFrame

get_strategy_weights

czsc.**get_strategy_weights**(*strategy_name*, *redis_url=None*, *connection_pool=None*, *key_prefix='Weights'*, ***kwargs*)

获取策略的持仓权重

Parameters

- **strategy_name** –str, 策略名
- **redis_url** –str, redis 连接字符串, 默认为 None, 即从环境变量 RWC_REDIS_URL 中读取
- **connection_pool** –redis.ConnectionPool, redis 连接池
- **key_prefix** –str, redis 中 key 的前缀，默认为 Weights
- **kwargs** –dict, 其他参数
  - symbols : list, 品种列表, 默认为 None, 即获取所有品种的权重
  - sdt : str, 开始时间, eg: 20210924 10:19:00
  - edt : str, 结束时间, eg: 20220924 10:19:00
  - only_last : boolean, 是否只保留每个品种最近一次权重，默认为 False

Returns

pd.DataFrame

get_sub_elements

czsc.**get_sub_elements**(*elements: List[Any]*, *di: int = 1*, *n: int = 10*) → List[Any]

获取截止到倒数第 di 个元素的前 n 个元素

Parameters

- **elements** –全部元素列表
- **di** –指定结束元素为倒数第 di 个
- **n** –指定需要的元素个数

Returns

部分元素列表

>>>x = [1, 2, 3, 4, 5, 6, 7, 8, 9] >>>y1 = get_sub_elements(x, di=1, n=3) >>>y2 = get_sub_elements(x, di=2, n=3)

get_trading_dates

czsc.**get_trading_dates**(*sdt*, *edt=datetime.datetime(2024, 5, 12, 11, 59, 28, 243956)*)

获取两个日期之间的所有交易日

get_unique_signals

czsc.**get_unique_signals**(*bars: List[*RawBar*]*, *signals_config: List[dict]*, ***kwargs*)

获取信号函数中定义的所有信号列表

函数执行逻辑：

1. 函数首先检查输入的 K 线数据 bars 是否按时间升序排列。如果 bars 的长度小于 600，函数直接返回一个空列表。
2. 然后，函数调用 generate_czsc_signals 方法，生成 CZSC 信号，并将结果保存在 df 中。
3. **函数遍历 df 中的所有列，对于每一列，如果列名包含三个部分，函数提取出该列中的所有唯一值，然后将列名和每一个唯一值组合成一个新的信号，**
   并添加到 _res 列表中。注意，如果唯一值中包含”其他”，则不会被添加到 _res 中。
4. 最后，函数返回 _res，其中包含了所有的唯一信号。

Parameters

- **bars** –基础 K 线数据
- **signals_config** –信号函数配置
- **kwargs** –传递给 generate_czsc_signals 方法的参数

Returns

信号列表

get_url_token

czsc.**get_url_token**(*url*)

获取指定 URL 数据接口的凭证码

holds_concepts_effect

czsc.**holds_concepts_effect**(*holds: DataFrame*, *concepts: dict*, *top_n=20*, *min_n=3*, ***kwargs*)

股票持仓列表的板块效应

原理概述：在选股时，如果股票的概念板块与组合中的其他股票的概念板块有重合，那么这个股票的表现会更好。

函数计算逻辑:

1. 如果 kwargs 中存在’copy’键且对应值为 True，则将 holds 进行复制。
2. 为 holds 添加’概念板块’列，该列的值是 holds 中’symbol’列对应的股票的概念板块列表，如果没有对应的概念板块则填充为空。
3. 添加’概念数量’列，该列的值是每个股票的概念板块数量。
4. 从 holds 中筛选出概念数量大于 0 的行，赋值给 holds。
5. 创建空列表 new_holds 和空字典 dt_key_concepts。

6. 对 holds 按照’dt’进行分组，遍历每个分组，计算板块效应。a. 计算密集出现的概念，选取出现次数最多的前 top_n 个概念，赋值给 key_concepts 列表。b. 将日期 dt 和对应的 key_concepts 存入 dt_key_concepts 字典。c. 计算在密集概念中出现次数超过 min_n 的股票，将符合条件的股票添加到 new_holds 列表中。7. 使用 pd.concat 将 new_holds 中的 DataFrame 进行合并，忽略索引，赋值给 dfh。8. 创建 DataFrame dfk，其中包含日期 (dt) 和对应的强势概念 (key_concepts)。9. 返回 dfh 和 dfk。

Parameters

- **holds** –组合股票池数据，样例：

dt	symbol	weight
2023-05-09 00:00:00	601858.SH	0.00333333
2023-05-09 00:00:00	300502.SZ	0.00333333
2023-05-09 00:00:00	603258.SH	0.00333333
2023-05-09 00:00:00	300499.SZ	0.00333333
2023-05-09 00:00:00	300624.SZ	0.00333333

- **concepts** –股票的概念板块，样例：{

  ’002507.SZ’: [‘电子商务’，‘超级品牌’，‘国企改革’], ‘002508.SZ’: [‘家用电器’，‘杭州亚运会’，‘恒大概念’]

  }

- **top_n** –选取前 n 个密集概念
- **min_n** –单股票至少要有 n 个概念在 top_n 中

Returns

过滤后的选股结果，每个时间点的 top_n 概念

holds_performance

czsc.**holds_performance**(*df*, ***kwargs*)

组合持仓权重表现

Parameters

- **df** –pd.DataFrame, columns=[‘dt’，‘symbol’，‘weight’，‘n1b’] 数据说明，dt: 交易时间，symbol: 标的代码，weight: 权重，n1b: 名义收益率必须是每个时间点都有数据，如果某个时间点没有数据，可以增加一行数据，权重为 0
- **kwargs** –
  - fee: float, 单边费率，BP
  - digits: int, 保留小数位数

Returns

pd.DataFrame, columns=[‘date’，‘change’，‘edge_pre_fee’，‘cost’，‘edge_post_fee’]

import_by_name

czsc.**import_by_name**(*name*)

通过字符串导入模块、类、函数

函数执行逻辑：

1. 检查 name 中是否包含点号（’.’）。如果没有，则直接使用内置的 import 函数来导入整个模块，并返回该模块对象。
2. **如果 name 包含点号，先处理一个相对路径。将 name 拆分为两部分：module_name 和 function_name**。
   使用 Python 内置的 rsplit 方法从右边开始分割，只取一次，这样可以确保我们将最后的一个点号前的部分作为 module_name，点号后面的部分作为 function_name。

3. **使用 import 函数导入指定的 module_name**。
   这里传入三个参数：globals() 和 locals() 分别代表当前全局和局部命名空间；[function_name] 是一个列表，用于指定要导入的子模块或属性名。这样做是为了避免一次性导入整个模块的所有内容，提高效率。
4. 使用 vars 函数获取模块的字典表示形式（即模块内所有的变量和函数），取出 function_name 对应的值，然后返回这个值。

Parameters

name –模块名，如：’czsc.objects.Factor’

Returns

模块对象

index_composition

czsc.**index_composition**(*klines*, *weights=None*, *base_point=1000*, ***kwagrs*)

设置基点，按收益率加权合成指数

输入：

1. 成分标的 K 线行情
2. 每个时刻的成分权重
3. 基点

过程：

1. 计算成分标的每根 K 线的收益
2. 在每个时刻，按成分权重对标的收益加权计算，得到每个时刻的指数涨跌幅
3. 用基点和每个时刻的涨跌幅计算出每个时刻的指数价

输出：指数 K 线行情

Parameters

- **klines** –K 线行情，样例如下：

symbol	dt	open	close	high	low	vol	amount
000001.SI	2021-01-04 09:31:00	3473.82	3469.37	3474.33	3468.86	104101420	13018854400
000001.SI	2021-01-04 09:32:00	3470.18	3467.58	3471.95	3467.58	570367232	7721827328
000001.SI	2021-01-04 09:33:00	3467.11	3466.98	3468.53	3466.94	660060480	8371049984
000001.SI	2021-01-04 09:34:00	3466.83	3463.5	3466.83	3463.4	563931392	7435783168
000001.SI	2021-01-04 09:35:00	3463.23	3460.33	3463.75	3460.33	504271904	6500695552

- **weights** –权重调整记录；索引为 dt，columns 为成分权重，每个时间截面的权重之和可以不为 1，样例如下：

dt	000001.SH	000016.SH	000300.SH	000905.SH
2021-01-04 09:31:00	0.244275	0.236822	0.271415	0.204674
2021-01-04 10:10:00	0.250273	0.140398	0.151955	0.294418
2021-01-04 10:54:00	0.127531	0.123941	0.199969	0.19682

注意：权重调整记录的时间截面必须是 K 线行情的时间截面的子集，且必须包含 K 线行情的第一根 K 线的时间截面

- **base_point** –基点，默认为 1000

Returns

指数 K 线行情，样例如下：

dt	returns	price	vol	amount
2021-01-04 09:31:00	0	1000	2195268112	31725149952
2021-01-04 09:32:00	-0.000230561	999.769	1187524832	18900989440
2021-01-04 09:33:00	-0.000165153	999.604	1330775568	19418287360
2021-01-04 09:34:00	-0.00103582	998.569	1133046300	17488768512
2021-01-04 09:35:00	-0.00067244	997.897	1025222108	15351070080

is_event_feature

czsc.**is_event_feature**(*df*, *col*, ***kwargs*)

事件类因子的判断函数

事件因子的特征：多头事件发生时，因子值为 1；空头事件发生时，因子值为-1；其他情况，因子值为 0。

Parameters

- **df** –DataFrame
- **col** –str, 因子字段名称

is_trading_date

czsc.**is_trading_date**(*date=datetime.datetime(2024, 5, 12, 11, 59, 28, 243952)*)

判断是否是交易日

is_trading_time

czsc.**is_trading_time**(*dt: datetime = datetime.datetime(2024, 5, 12, 11, 59, 22, 586058)*, *market='A 股'*)

判断指定时间是否是交易时间

long_short_equity

czsc.**long_short_equity**(*factors*, *returns*, *hold_period=2*, *rank=5*, ***kwargs*)

根据截面因子值与收益率，回测分析多空对冲组合的收益率

Parameters

- **factors** –

  截面因子，因子值越大，越偏向于做多，因子值越小，越偏向于做空；数据格式如下：
  SFIH9001 SFIF9001 SFIC9001

  dt 2022-08-31 1.403915 1.252826 0.968868 2022-09-01 1.376690 1.253377 0.972276 2022-09-02 1.380867 1.253929 0.974999 2022-09-05 1.370359 1.254482 0.977737 2022-09-06 0.685180 0.633634 0.493986

- **returns** –

  品种收益率矩阵，数据格式如下：
  SFIH9001 SFIF9001 SFIC9001

  dt 2021-01-04 0.007803 0.017228 0.004843 2021-01-05 0.014068 0.008300 0.000598 2021-01-06 0.024520 0.022766 0.004974 2021-01-07 -0.006193 -0.003698 0.005951 2021-01-08 -0.005651 -0.012263 -0.016441

- **hold_period** –持仓周期，dt 时刻的数量，如果是 2，则表示每两个交易时刻调仓一次
- **rank** –排序因子值前几名，或者排名因子值的前百分之几；如果是整数，则表示排名因子值前几名；如果是浮点数，则表示排名因子值的前百分之几。排名靠前，越偏向于做多；排名靠后，越偏向于做空。
- **kwargs** –

Returns

net_value_stats

czsc.**net_value_stats**(*nv: DataFrame*, *exclude_zero: bool = False*, *sub_cost=True*) → dict

统计净值曲线的年化收益、夏普等

Parameters

- **nv** –净值数据，格式如下：

  dt edge cost

  0 2017-01-03 09:30:00 0.0 0.0 1 2017-01-03 10:00:00 0.0 0.0 2 2017-01-03 10:30:00 0.0 0.0 3 2017-01-03 11:00:00 0.0 0.0 4 2017-01-03 13:00:00 0.0 0.0

  列说明：dt: 交易时间 edge: 单利收益，单位：BP cost: 交易成本，单位：BP；可选列，如果没有成本列，则默认为 0
- **exclude_zero** –是否排除收益为 0 的情况，一般认为收益为 0 的情况是没有持仓的
- **sub_cost** –是否扣除成本

Returns

next_trading_date

czsc.**next_trading_date**(*date=datetime.datetime(2024, 5, 12, 11, 59, 28, 243955)*, *n=1*)

获取未来第 N 个交易日

normalize_corr

czsc.**normalize_corr**(*df: DataFrame*, *fcol*, *ycol=None*, ***kwargs*)

标准化因子与收益相关性为正数

方法说明：对因子进行滚动相关系数计算，因子乘以滚动相关系数的符号

注意：

1. simple 模式下，计算过程有一定的未来信息泄露，在回测中使用时需要注意
2. rolling 模式下，计算过程依赖 window 参数，有可能调整后相关性为负数

Parameters

- **df** –pd.DataFrame, 必须包含 dt、symbol、price 列，以及因子列
- **fcol** –str 因子列名
- **kwargs** –dict
  - window: int, 滚动窗口大小
  - min_periods: int, 最小计算周期
  - mode: str, 计算方法, rolling 表示使用滚动调整相关系数，simple 表示使用镜像反转相关系数
  - copy: bool, 是否复制 df

Returns

pd.DataFrame

normalize_feature

czsc.**normalize_feature**(*df*, *x_col*, ***kwargs*)

因子标准化：缩尾，然后标准化

函数计算逻辑：

1. 首先，检查因子列 x_col 是否存在缺失值，如果存在缺失值，则抛出异常，提示缺失值的数量。
2. 从 kwargs 参数中获取缩尾比例 q 的值，默认为 0.05。
3. 对因子列进行缩尾操作，首先根据 dt 分组，然后使用 lambda 函数对每个组内的因子进行缩尾处理，将超过缩尾比例的值截断，并使用 scale 函数进行标准化。
4. 将处理后的因子列重新赋值给原始 DataFrame 对象的对应列。

Parameters

- **df** –pd.DataFrame，数据
- **x_col** –str，因子列名
- **kwargs** –
  - q: float，缩尾比例, 默认 0.05

Returns

pd.DataFrame，处理后的数据

normalize_ts_feature

czsc.**normalize_ts_feature**(*df*, *x_col*, *n=10*, ***kwargs*)

对时间序列数据进行归一化处理

Parameters

- **df** –因子数据，必须包含 dt, x_col 列，其中 dt 为日期，x_col 为因子值
- **x_col** –因子列名
- **n** –分层数量，默认为 10
- **kwargs** –
  - window: 窗口大小，默认为 2000
  - min_periods: 最小样本数量，默认为 300

Returns

df, 添加了 x_col_norm, x_col_qcut, x_col 分层列

optuna_good_params

czsc.**optuna_good_params**(*study: Study*, *keep=0.2*) → DataFrame

获取 optuna 优化结果中的最优参数

Parameters

- **study** –optuna.study.Study
- **keep** –float, 保留最优参数的比例, 默认 0.2 如果 keep 小于 0，则按比例保留；如果 keep 大于 0，则保留 keep 个参数组

Returns

pd.DataFrame, 最优参数组列表

optuna_study

czsc.**optuna_study**(*objective*, *direction='maximize'*, *n_trials=100*, ***kwargs*)

使用 optuna 进行参数优化

overlap

czsc.**overlap**(*df: DataFrame*, *col: str*, ***kwargs*)

给定 df 和 col，计算 col 中相同值的连续出现次数

Parameters

- **df** –pd.DataFrame, 至少包含 dt、symbol 和 col 列
- **col** –str，需要计算连续出现次数的列名
- **kwargs** –dict，其他参数
  - copy: bool, 是否复制 df，默认为 True
  - new_col: str, 计算结果的列名，默认为 f”{col}_overlap”
  - max_overlap: int, 最大允许连续出现次数，默认为 10

prev_trading_date

czsc.**prev_trading_date**(*date=datetime.datetime(2024, 5, 12, 11, 59, 28, 243956)*, *n=1*)

获取过去第 N 个交易日

print_df_sample

czsc.**print_df_sample**(*df*, *n=5*)

psi

czsc.**psi**(*df: DataFrame*, *factor*, *segment*, ***kwargs*)

PSI 群体稳定性指标，反映数据在不同分箱中的分布变化

PSI = ∑(实际占比 - 基准占比) * ln(实际占比 / 基准占比)

参考：https://zhuanlan.zhihu.com/p/79682292 风控模型—群体稳定性指标 (PSI) 深入理解应用

Parameters

- **df** –数据, 必须包含 dt 和 col 列
- **factor** –分组因子
- **segment** –样本分组
- **kwargs** –

Returns

pd.DataFrame

read_json

czsc.**read_json**(*file*)

resample_bars

czsc.**resample_bars**(*df: DataFrame*, *target_freq:* Freq | *AnyStr*, *raw_bars=True*, ***kwargs*)

将给定的 K 线数据重新采样为目标周期的 K 线数据

函数计算逻辑：

1. 确定目标周期 `target_freq`的类型和市场类型。
2. 添加一个新列 `freq_edt`，表示每个数据点对应的目标周期的结束时间。
3. 根据 `freq_edt`对数据进行分组，并对每组数据进行聚合，得到目标周期的 K 线数据。
4. 重置索引，并选择需要的列。
5. 根据 `raw_bars`参数，决定返回的数据类型：如果为 True，转换为 `RawBar`对象；如果为 False，直接返回 DataFrame。
6. 如果 `drop_unfinished`参数为 True，删除最后一根未完成的 K 线。

Parameters

- **df** –

  原始 K 线数据，必须包含以下列：symbol, dt, open, close, high, low, vol, amount。样例如下：
  symbol dt open close high low 0 000001.XSHG 2015-01-05 09:31:00 3258.63 3259.69 3262.85 3258.63

  1 000001.XSHG 2015-01-05 09:32:00 3258.33 3256.19 3259.55 3256.19 2 000001.XSHG 2015-01-05 09:33:00 3256.10 3257.50 3258.42 3256.10 3 000001.XSHG 2015-01-05 09:34:00 3259.33 3261.76 3261.76 3257.98 4 000001.XSHG 2015-01-05 09:35:00 3261.71 3264.88 3265.48 3261.71

  vol amount

  0 1333523100 4.346872e+12 1 511386100 1.665170e+12 2 455375200 1.483385e+12 3 363393800 1.185303e+12 4 402854600 1.315272e+12

- **target_freq** –目标周期
- **raw_bars** –是否将转换后的 K 线序列转换为 RawBar 对象
- **kwargs** –
  - base_freq: 基础周期，如果不指定，则根据 df 中的 dt 列自动推断
  - drop_unfinished: 是否删除最后一根未完成的 K 线

Returns

转换后的 K 线序列

resample_to_daily

czsc.**resample_to_daily**(*df: DataFrame*, *sdt=None*, *edt=None*, *only_trade_date=True*)

将非日线数据转换为日线数据，以便进行日线级别的分析

函数执行逻辑：

1. 首先，函数接收一个数据框 `df`，以及可选的开始日期 `sdt`，结束日期 `edt`，和一个布尔值 `only_trade_date`。
2. 函数将 `df`中的 `dt`列转换为日期时间格式。如果没有提供 `sdt`或 `edt`，则使用 `df`中的最小和最大日期作为开始和结束日期。
3. 创建一个日期序列。如果 `only_trade_date`为真，则只包含交易日期；否则，包含 `sdt`和 `edt`之间的所有日期。
4. 使用 `merge_asof`函数，找到每个日期在原始 `df`中对应的最近一个日期。
5. 创建一个映射，将每个日期映射到原始 `df`中的对应行。
6. 对于日期序列中的每个日期，复制映射中对应的数据行，并将日期设置为当前日期。
7. 最后，将所有复制的数据行合并成一个新的数据框，并返回。

Parameters

- **df** –日线以上周期的数据，必须包含 dt 列
- **sdt** –开始日期
- **edt** –结束日期
- **only_trade_date** –是否只保留交易日数据

Returns

pd.DataFrame

risk_free_returns

czsc.**risk_free_returns**(*start_date='20180101'*, *end_date='20210101'*, *year_returns=0.03*)

创建无风险收益率序列

创建一个 Pandas DataFrame，包含两列：”date”和“returns”。”date”列包含从 trade_dates 获取的所有交易日期，“returns”列包含无风险收益率序列，计算方法是将年化收益率（year_returns）除以 252（一年的交易日数量，假设为每周 5 天）

Parameters

- **start_date** –起始日期
- **end_date** –截止日期
- **year_returns** –年化收益率

Returns

pd.DataFrame

rolling_compare

czsc.**rolling_compare**(*df*, *col1*, *col2*, *window=300*, *min_periods=100*, *new_col=None*, ***kwargs*)

计算序列的滚动归一化值

Parameters

- **df** –pd.DataFrame 待计算的数据
- **col1** –str 第一个列名
- **col2** –str 第二个列名
- **window** –int 滚动窗口大小, 默认为 300
- **new_col** –str 新列名，默认为 None, 表示使用 f’{col}_norm’作为新列名
- **kwargs** –

  min_periods: int
  最小计算周期

rolling_corr

czsc.**rolling_corr**(*df*, *col1*, *col2*, *window=300*, *min_periods=100*, ***kwargs*)

滚动计算两个序列的相关系数

Parameters

- **df** –pd.DataFrame
- **col1** –str
- **col2** –str
- **window** –int, default 300, 滚动窗口大小, None 表示扩展窗口
- **min_periods** –int, default 100, 最小观测数量, 用于计算相关系数的最小观测数量

rolling_daily_performance

czsc.**rolling_daily_performance**(*df: DataFrame*, *ret_col*, *window=252*, *min_periods=100*, ***kwargs*)

计算滚动日收益

Parameters

- **df** –pd.DataFrame, 日收益数据，columns=[‘dt’, ret_col]
- **ret_col** –str, 收益列名
- **window** –int, 滚动窗口, 自然天数
- **min_periods** –int, 最小样本数
- **kwargs** –

rolling_norm

czsc.**rolling_norm**(*df: DataFrame*, *col*, *window=300*, *min_periods=100*, *new_col=None*, ***kwargs*)

计算序列的滚动归一化值

Parameters

- **df** –pd.DataFrame, 待计算的数据
- **col** –str, 待计算的列
- **window** –int, 滚动窗口大小, 默认为 300
- **min_periods** –int, 最小计算周期, 默认为 100
- **new_col** –str, 新列名，默认为 None, 表示使用 f’{col}_norm’作为新列名

rolling_qcut

czsc.**rolling_qcut**(*df: DataFrame*, *col*, *window=300*, *min_periods=100*, *new_col=None*, ***kwargs*)

计算序列的滚动分位数

Parameters

- **df** –pd.DataFrame, 待计算的数据
- **col** –str, 待计算的列
- **window** –int, 滚动窗口大小, 默认为 300
- **min_periods** –int, 最小计算周期, 默认为 100
- **new_col** –str, 新列名，默认为 None, 表示使用 f’{col}_qcut’作为新列名

rolling_rank

czsc.**rolling_rank**(*df: DataFrame*, *col*, *window=300*, *min_periods=100*, *new_col=None*, ***kwargs*)

计算序列的滚动排名

Parameters

- **df** –pd.DataFrame, 待计算的数据
- **col** –str, 待计算的列
- **window** –int, 滚动窗口大小, 默认为 300
- **min_periods** –int, 最小计算周期, 默认为 100
- **new_col** –str, 新列名，默认为 None, 表示使用 f’{col}_rank’作为新列名

rolling_scale

czsc.**rolling_scale**(*df: DataFrame*, *col: str*, *window=300*, *min_periods=100*, *new_col=None*, ***kwargs*)

对序列进行滚动归一化

Parameters

- **df** –pd.DataFrame, 待计算的数据
- **col** –str, 待计算的列
- **window** –int, 滚动窗口大小, 默认为 300
- **min_periods** –int, 最小计算周期, 默认为 100
- **new_col** –str, 新列名，默认为 None, 表示使用 f’{col}_scale’作为新列名

rolling_slope

czsc.**rolling_slope**(*df: DataFrame*, *col: str*, *window=300*, *min_periods=100*, *new_col=None*, ***kwargs*)

计算序列的滚动斜率

大于 0 表示序列的斜率向上，小于 0 表示序列的斜率向下，绝对值越大表示斜率越陡峭

Parameters

- **df** –pd.DataFrame, 待计算的数据
- **col** –str, 待计算的列
- **window** –int, 滚动窗口大小, 默认为 300
- **min_periods** –int, 最小计算周期, 默认为 100
- **new_col** –str, 新列名，默认为 None, 表示使用 f’{col}_slope’作为新列名

- **kwargs** –
  - min_periods: int, 最小计算周期
  - method: str, 计算方法
    - linear: 使用线性回归计算斜率
    - std/mean: 使用序列的 std/mean 计算斜率
    - snr: 使用序列的 snr 计算斜率

rolling_tanh

czsc.**rolling_tanh**(*df: DataFrame*, *col: str*, *window=300*, *min_periods=100*, *new_col=None*, ***kwargs*)

对序列进行滚动 tanh 变换

双曲正切函数：https://baike.baidu.com/item/%E5%8F%8C%E6%9B%B2%E6%AD%A3%E5%88%87%E5%87%BD%E6%95%B0/15469414

Parameters

- **df** –pd.DataFrame, 待计算的数据
- **col** –str, 待计算的列
- **window** –int, 滚动窗口大小, 默认为 300
- **min_periods** –int, 最小计算周期, 默认为 100
- **new_col** –str, 新列名，默认为 None, 表示使用 f’{col}_scale’作为新列名

save_json

czsc.**save_json**(*data*, *file*)

set_url_token

czsc.**set_url_token**(*token*, *url*)

设置指定 URL 数据接口的凭证码，通常一台机器只需要设置一次即可

Parameters

- **token** –凭证码
- **url** –数据接口地址

show_cointegration

czsc.**show_cointegration**(*df*, *col1*, *col2*, ***kwargs*)

分析两个时间序列协整性，贡献者：珠峰

Parameters

- **df** –pd.DataFrame, 必须包含列 dt 和 col1, col2
- **col1** –str, df 中的列名
- **col2** –str, df 中的列名
- **kwargs** –dict, 其他参数
  - sub_header: str, default ‘’, 子标题
  - docs: bool, default False, 是否显示协整检验的原理与使用说明

show_correlation

czsc.**show_correlation**(*df*, *cols=None*, *method='pearson'*, ***kwargs*)

用 streamlit 展示相关性

Parameters

- **df** –pd.DataFrame，数据源
- **cols** –list，分析相关性的字段
- **method** –str，计算相关性的方法，可选 pearson 和 spearman
- **kwargs** –
  - use_st_table: bool，是否使用 st.table 展示相关性，默认为 False
  - use_container_width: bool，是否使用容器宽度，默认为 True

show_daily_return

czsc.**show_daily_return**(*df: DataFrame*, ***kwargs*)

用 streamlit 展示日收益

Parameters

- **df** –pd.DataFrame，数据源
- **kwargs** –
  - sub_title: str，标题
  - stat_hold_days: bool，是否展示持有日绩效指标，默认为 True

- legend_only_cols: list，仅在图例中展示的列名
- use_st_table: bool，是否使用 st.table 展示绩效指标，默认为 False
- plot_cumsum: bool，是否展示日收益累计曲线，默认为 True

show_drawdowns

czsc.**show_drawdowns**(*df: DataFrame*, *ret_col*, ***kwargs*)

展示最大回撤分析

Parameters

- **df** –pd.DataFrame, columns: cells, index: dates
- **ret_col** –str, 回报率列名称
- **kwargs** –
  - sub_title: str, optional, 子标题
  - top: int, optional, 默认 10, 返回最大回撤的数量

show_event_return

czsc.**show_event_return**(*df*, *factor*, ***kwargs*)

分析事件因子的收益率特征

Parameters

- **df** –pd.DataFrame, 数据源
- **factor** –str, 事件因子名称
- **kwargs** –dict, 其他参数
  - sub_title: str, 子标题
  - max_overlap: int, 事件最大重叠次数

show_factor_layering

czsc.**show_factor_layering**(*df*, *x_col*, *y_col='n1b'*, ***kwargs*)

使用 streamlit 绘制因子截面分层收益率图

Parameters

- **df** –因子数据
- **x_col** –因子列名
- **y_col** –收益列名

- **kwargs** –
  - n: 分层数量，默认为 10
  - long: 多头组合，例如“第 10 层”
  - short: 空头组合，例如“第 01 层”

show_factor_returns

czsc.**show_factor_returns**(*df*, *x_col*, *y_col*)

使用 streamlit 展示因子收益率

Parameters

- **df** –pd.DataFrame，数据源
- **x_col** –str，因子列名
- **y_col** –str，收益列名

show_monthly_return

czsc.**show_monthly_return**(*df*, *ret_col='total'*, *sub_title='月度累计收益'*, ***kwargs*)

展示指定列的月度累计收益

Parameters

- **df** –pd.DataFrame，数据源
- **ret_col** –str，收益列名
- **title** –str，标题
- **kwargs** –

show_optuna_study

czsc.**show_optuna_study**(*study: Study*, ***kwargs*)

show_out_in_compare

czsc.**show_out_in_compare**(*df*, *ret_col*, *mid_dt*, ***kwargs*)

展示样本内外表现对比

show_psi

czsc.**show_psi**(*df*, *factor*, *segment*, ***kwargs*)

PSI 分布稳定性

Parameters

- **df** –pd.DataFrame, 数据源
- **factor** –str, 分组因子
- **segment** –str, 分段字段
- **kwargs** –
  - sub_title: str, 子标题

show_rolling_daily_performance

czsc.**show_rolling_daily_performance**(*df*, *ret_col*, ***kwargs*)

展示滚动统计数据

Parameters

- **df** –pd.DataFrame, 日收益数据，columns=[‘dt’, ret_col]
- **ret_col** –str, 收益列名
- **kwargs** –

show_sectional_ic

czsc.**show_sectional_ic**(*df*, *x_col*, *y_col*, *method='pearson'*, ***kwargs*)

使用 streamlit 展示截面 IC

Parameters

- **df** –pd.DataFrame，数据源
- **x_col** –str，因子列名
- **y_col** –str，收益列名
- **method** –str，计算 IC 的方法，可选 pearson 和 spearman

show_splited_daily

czsc.**show_splited_daily**(*df*, *ret_col*, ***kwargs*)

展示分段日收益表现

Parameters

- **df** –pd.DataFrame
- **ret_col** –str, df 中的列名，指定收益列
- **kwargs** –sub_title: str, 子标题

show_stoploss_by_direction

czsc.**show_stoploss_by_direction**(*dfw*, ***kwargs*)

按方向止损分析的展示

Parameters

- **dfw** –pd.DataFrame, 包含权重数据
- **kwargs** –dict, 其他参数
  - stoploss: float, 止损比例
  - show_detail: bool, 是否展示详细信息
  - digits: int, 价格小数位数, 默认 2
  - fee_rate: float, 手续费率, 默认 0.0002

Returns

None

show_strategies_dailys

czsc.**show_strategies_dailys**(*df*, ***kwargs*)

展示多策略多品种日收益率数据：按策略等权日收益

Parameters

- **df** –N 策略 M 品种日收益率数据，columns=[‘dt’，‘strategy’，‘symbol’，‘returns’]，样例如下：

dt	strategy	symbol	returns
2021-01-04 00:00:00	FUT001	SFT9001	-0.00240078
2021-01-05 00:00:00	FUT001	SFT9001	-0.00107012
2021-01-06 00:00:00	FUT001	SFT9001	0.00122168
2021-01-07 00:00:00	FUT001	SFT9001	0.0020896
2021-01-08 00:00:00	FUT001	SFT9001	0.000510725

- **kwargs** –
  - sub_title: str, 子标题

show_strategies_symbol

czsc.**show_strategies_symbol**(*df*, ***kwargs*)

展示多策略多品种日收益率数据：按品种等权日收益

Parameters

- **df** –N 策略 M 品种日收益率数据，columns=[‘dt’，‘strategy’，‘symbol’，‘returns’]，样例如下：

dt	strategy	symbol	returns
2021-01-04 00:00:00	FUT001	SFT9001	-0.00240078
2021-01-05 00:00:00	FUT001	SFT9001	-0.00107012
2021-01-06 00:00:00	FUT001	SFT9001	0.00122168
2021-01-07 00:00:00	FUT001	SFT9001	0.0020896
2021-01-08 00:00:00	FUT001	SFT9001	0.000510725

- **kwargs** –

show_symbol_factor_layering

czsc.**show_symbol_factor_layering**(*df*, *x_col*, *y_col='n1b'*, ***kwargs*)

使用 streamlit 绘制单个标的上的因子分层收益率图

Parameters

- **df** –因子数据，必须包含 dt, x_col, y_col 列，其中 dt 为日期，x_col 为因子值，y_col 为收益率，数据样例：

dt	intercept	n1b
2017-01-03 00:00:00	0	0.00716081
2017-01-04 00:00:00	-0.00154541	0.000250816
2017-01-05 00:00:00	0.000628884	-0.0062695
2017-01-06 00:00:00	-0.00681021	0.00334212
2017-01-09 00:00:00	0.00301077	-0.00182963

- **x_col** –因子列名
- **y_col** –收益列名
- **kwargs** –
  - n: 分层数量，默认为 10

show_ts_rolling_corr

czsc.**show_ts_rolling_corr**(*df*, *col1*, *col2*, ***kwargs*)

时序上按 rolling 的方式计算相关系数

Parameters

- **df** –pd.DataFrame, 必须包含列 dt 和 col1, col2
- **col1** –str, df 中的列名
- **col2** –str, df 中的列名
- **kwargs** –
  - min_periods: int, 最小滑动窗口长度
  - window: int, 滑动窗口长度，0 表示按 expanding 方式滑动
  - corr_method: str, 相关系数计算方法，可选 pearson, kendall, spearman
  - sub_title: str, 子标题

show_ts_self_corr

czsc.**show_ts_self_corr**(*df*, *col*, ***kwargs*)

展示时序上单因子的自相关性分析结果，贡献者：guo

Parameters

- **df** –pd.DataFrame, 必须包含列 dt 和 col
- **col** –str, df 中的列名

show_weight_backtest

czsc.**show_weight_backtest**(*dfw*, ***kwargs*)

展示权重回测结果

Parameters

- **dfw** –回测数据，任何字段都不允许有空值；数据样例：

dt	symbol	weight	price
2019-01-02 09:01:00	DLi9001	0.5	961.695
2019-01-02 09:02:00	DLi9001	0.25	960.72
2019-01-02 09:03:00	DLi9001	0.25	962.669
2019-01-02 09:04:00	DLi9001	0.25	960.72
2019-01-02 09:05:00	DLi9001	0.25	961.695

- **kwargs** –
  - fee: 单边手续费，单位为 BP，默认为 2BP
  - digits: 权重小数位数，默认为 2
  - show_drawdowns: bool，是否展示最大回撤，默认为 False
  - show_daily_detail: bool，是否展示每日收益详情，默认为 False
  - show_backtest_detail: bool，是否展示回测详情，默认为 False
  - show_splited_daily: bool，是否展示分段日收益表现，默认为 False
  - show_yearly_stats: bool，是否展示年度绩效指标，默认为 False
  - show_monthly_return: bool，是否展示月度累计收益，默认为 False

show_yearly_stats

czsc.**show_yearly_stats**(*df*, *ret_col*, ***kwargs*)

按年计算日收益表现

Parameters

- **df** –pd.DataFrame，数据源
- **ret_col** –str，收益列名
- **kwargs** –
  - sub_title: str, 子标题

stock_holds_performance

czsc.**stock_holds_performance**(*dc:* TsDataCache, *dfh*, *res_path*)

计算 A 股日线持仓组合的表现

Parameters

- **dc** –Tushare 数据缓存对象
- **dfh** –

  持仓组合，样例如下，其中【证券代码】要求是 tushare 的格式，成分日期当天状态为持仓
  成分日期证券代码持仓权重 n1b

  0 2020-01-03 300620.SZ 0.008403 141.861099 1 2020-01-03 300677.SZ 0.008403 767.124023 2 2020-01-03 300708.SZ 0.008403 93.029297 3 2020-01-03 002151.SZ 0.008403 -7.465500 4 2020-01-03 002156.SZ 0.008403 350.101715

- **res_path** –结果保存路径

Returns

stoploss_by_direction

czsc.**stoploss_by_direction**(*dfw*, *stoploss=0.03*, ***kwargs*)

按持仓方向进行止损

Parameters

- **dfw** –pd.DataFrame, columns = [‘dt’, ‘symbol’, ‘weight’, ‘price’], 持仓权重数据，其中

  dt 为 K 线结束时间，必须是连续的交易时间序列，不允许有时间断层 symbol 为合约代码，weight 为 K 线结束时间对应的持仓权重，品种之间的权重是独立的，不会互相影响 price 为结束时间对应的交易价格，可以是当前 K 线的收盘价，或者下一根 K 线的开盘价，或者未来 N 根 K 线的 TWAP、VWAP 等

  数据样例如下：========== ======== ======== ======= dt symbol weight price ========== ======== ======== ======= 2019-01-02 09:01:00 DLi9001 0.5 961.695 2019-01-02 09:02:00 DLi9001 0.25 960.72 2019-01-02 09:03:00 DLi9001 0.25 962.669 2019-01-02 09:04:00 DLi9001 0.25 960.72 2019-01-02 09:05:00 DLi9001 0.25 961.695 ========== ======== ======== =======

- **stoploss** –止损比例
- **kwargs** –其他参数

Returns

pd.DataFrame，columns = [‘dt’，‘symbol’，‘weight’，‘raw_weight’，‘price’，‘returns’，’hold_returns’，‘min_hold_returns’，‘order_id’，‘is_stop’]

subtract_fee

czsc.**subtract_fee**(*df*, *fee=1*)

依据单品种持仓信号扣除手续费

函数执行逻辑：

1. 首先，函数对输入的 df 进行检查，确保其包含所需的列：’dt’（日期时间）和’pos’（持仓）。同时，检查’pos’列的值是否符合要求，即只能是 0、1 或-1。
2. 如果 df 中不包含’n1b’（名义收益率）列，函数会根据’price’列计算’n1b’列。
3. 然后，函数为输入的 DataFrame df 添加一个新列’date’，该列包含交易日期（从’dt’列中提取）。
4. 接下来，函数根据持仓（’pos’）和名义收益率（’n1b’）计算’edge_pre_fee’（手续费前收益）和’edge_post_fee’（手续费后收益）两列。
5. **函数根据持仓信号计算开仓和平仓的位置**。
   开仓位置（open_pos）是持仓信号发生变化的位置（即，当前持仓与前一个持仓不同），并且当前持仓不为 0。平仓位置（exit_pos）是持仓信号发生变化的位置（即，当前持仓与前一个持仓不同），并且前一个持仓不为 0。
6. 根据手续费规则，开仓时在第一个持仓 K 线上扣除手续费，平仓时在最后一个持仓 K 线上扣除手续费。函数通过将’edge_post_fee’列的值在开仓和平仓位置上分别减去手续费（fee）来实现这一逻辑。
7. 最后，函数返回修改后的 DataFrame df。

Parameters

- **df** –包含 dt、pos、price、n1b 列的 DataFrame
- **fee** –手续费，单位：BP

Returns

修改后的 DataFrame

symbols_bi_infos

czsc.**symbols_bi_infos**(*symbols*, *read_bars*, *freq='5 分钟'*, *sdt='20130101'*, *edt='20190101'*, ***kwargs*) → DataFrame

计算多个标的的笔特征

Parameters

- **symbols** –品种代码列表
- **read_bars** –读取 K 线数据的函数，要求返回 RawBar 对象列表
- **freq** –K 线周期, defaults to '5 分钟'
- **sdt** –开始时间, defaults to '20130101'
- **edt** –结束时间, defaults to '20190101'

Returns

笔的特征

top_drawdowns

czsc.**top_drawdowns**(*returns: Series*, *top: int = 10*) → DataFrame

分析最大回撤，返回最大回撤的波峰、波谷、恢复日期、回撤天数、恢复天数

Parameters

- **returns** –pd.Series, 日收益率序列，index 为日期
- **top** –int, optional, 返回最大回撤的数量，默认 10

Returns

pd.DataFrame

update_bbars

czsc.**update_bbars**(*da*, *price_col='close'*, *numbers=(1, 2, 5, 10, 20, 30)*) → None

在给定的 da 数据上计算并添加前面 n 根 bar 的累计收益列

函数的逻辑如下：

1. 首先，检查 price_col 是否在输入的 DataFrame（da）的列名中。如果不在，抛出 ValueError。
2. 使用 for 循环遍历 numbers 列表中的每个整数 n，对于每个整数 n，计算 n 根 bar 的累计收益。
3. 返回 None，表示这个函数会直接修改输入的 da，而不返回新的 DataFrame。

Parameters

- **da** –K 线数据，DataFrame 结构

- **price_col** –价格列
- **numbers** –考察的 bar 的数目的列表

Returns

bbars_cols: 后面 n 根 bar 的 bp 值列名

update_nxb

czsc.**update_nxb**(*df: DataFrame*, ***kwargs*) → DataFrame

在给定的 df 上计算并添加后面 n 根 bar 的累计收益列

收益计量单位：BP；1 倍涨幅 = 10000BP

Parameters

- **df** –数据，DataFrame 结构，必须包含 dt, symbol, price 列
- **kwargs** –参数字典
  - nseq: 考察的 bar 的数目的列表，默认为 (1, 2, 3, 5, 8, 10, 13)
  - bp: 是否将收益转换为 BP，默认为 False

Returns

pd.DataFrame

update_tbars

czsc.**update_tbars**(*da: DataFrame*, *event_col: str*) → None

计算带 Event 方向信息的未来收益

函数的逻辑如下：

1. 从输入的 da 的列名中提取所有以‘n’开头，以‘b’结尾的列名，这些列名表示未来 n 根 bar 的累计收益。将这些列名存储在 n_seq 列表中。
2. 使用 for 循环遍历 n_seq 列表中的每个整数 n。
3. **对于每个整数 n，计算带有 Event 方向信息的未来收益**。
   计算方法是：将前面 n 根 bar 的累计收益（列名 f’n{n}b’）与事件信号列（event_col）的值相乘。将计算结果存储在一个新的列中，列名为 f’t{n}b’。
4. 返回 None，表示这个函数会直接修改输入的 da，而不返回新的 DataFrame。

Parameters

- **da** –K 线数据，DataFrame 结构
- **event_col** –事件信号列名，含有 0, 1, -1 三种值，0 表示无事件，1 表示看多事件，-1 表示看空事件

Returns

None

weekly_performance

czsc.**weekly_performance**(*weekly_returns*)

采用单利计算周收益数据的各项指标

Parameters

weekly_returns –周收益率数据，样例：[0.01, 0.02, -0.01, 0.03, 0.02, -0.02, 0.01, -0.01, 0.02, 0.01]

Returns

dict

welcome

czsc.**welcome**()

x_round

czsc.**x_round**(*x: float | int*, *digit: int = 4*) → float | int

用去尾法截断小数

Parameters

- **x** –数字
- **digit** –保留小数位数

Returns

2.2.2 Classes

AliyunOSS(access_key_id, access_key_secret, ...)	
BarGenerator(base_freq, freqs[, max_count, ...])	
CTAResearch(strategy, read_bars, ...)	
CZSC(bars[, get_signals, max_bi_num])	

continues on next page

Table 2 – continued from previous page

CrossSectionalPerformance(dfh, **kwargs)	根据截面持仓信息，计算截面绩效
CzscJsonStrategy(**kwargs)	仅传入 Json 配置的 Positions 就完成策略创建
CzscSignals([bg])	缠中说禅技术分析理论之多级别信号计算
CzscStrategyBase(**kwargs)	择时交易策略的要素：
CzscTrader([bg, positions, ensemble_method])	缠中说禅技术分析理论之多级别联立交易决策类（支持多策略独立执行）
DataClient([token, url, timeout])	
Direction(value)	An enumeration.
DiskCache([path])	
DummyBacktest(strategy, signals_path, ...)	
Event(operate, factors, signals_all, ...)	
EventMatchSensor(events, symbols, read_bars, ...)	
ExitsOptimize(read_bars, **kwargs)	基础策略出场优化流程
Factor(signals_all, signals_any, ...)	
FixedNumberSelector(dfs, k, d, **kwargs)	选择固定数量（等权）的交易品种
Freq(value)	An enumeration.
KlineChart([n_rows])	K 线绘图工具类
NewBar(symbol, id, dt, freq, open, close, ...)	去除包含关系后的 K 线元素
OpensOptimize(read_bars, **kwargs)	基础策略入场优化流程
Operate(value)	An enumeration.
PairsPerformance(df_pairs)	交易对效果评估
Position(symbol, opens[, exits, interval, ...])	
RawBar(symbol, id, dt, freq, open, close, ...)	原始 K 线元素
RedisWeightsClient(strategy_name[, ...])	策略持仓权重收发客户端
Signal([signal, score, k1, k2, k3, v1, v2, v3])	
SignalAnalyzer(*args, **kwargs)	
SignalPerformance(*args, **kwargs)	信号表现分析
SignalsParser([signals_module])	解析一串信号，生成信号函数配置
WeightBacktest(dfw[, digits])	持仓权重回测
WordWriter([file_docx])	用 Word 文档记录信息

continues on next page

Table 2 – continued from previous page

ZS(bis, cache)	中枢对象，主要用于辅助信号函数计算

AliyunOSS

class czsc.**AliyunOSS**(*access_key_id: str*, *access_key_secret: str*, *endpoint: str*, *bucket_name: str*)

Bases: object

Methods Summary

batch_download(oss_keys, local_paths[, ...])	批量从 OSS 下载文件。
batch_upload(filepaths, oss_keys[, replace, ...])	批量上传文件到 OSS。
create_folder(folder_path)	在 OSS 上创建文件夹。
delete_file(oss_key)	从 OSS 删除文件。
download(oss_key, filepath[, replace])	从 OSS 下载文件。
download_folder(oss_folder, local_folder[, ...])	从 OSS 下载指定文件夹。
file_exists(oss_key)	检查文件是否在 OSS 上存在。
get_file_stream(oss_key)	获取 OSS 上文件的数据流。
list_files([prefix, extensions])	列举 OSS 上的文件。
multipart_upload(filepath, oss_key)	分块上传大文件到 OSS。
upload(filepath, oss_key[, replace])	上传文件到 OSS。
upload_folder(local_folder, oss_folder[, ...])	上传本地文件夹到 OSS。

Methods Documentation

batch_download(*oss_keys: List[str]*, *local_paths: List[str]*, *replace: bool = False*, *threads: int = 5*)

批量从 OSS 下载文件。

Parameters

- **oss_keys** –list, 文件在 OSS 上的路径和名称列表。
- **local_paths** –list, 本地存储文件的路径列表。
- **threads** –int, 并行下载的线程数。默认为 5。

batch_upload(*filepaths: List[str]*, *oss_keys: List[str]*, *replace: bool = False*, *threads: int = 5*)

批量上传文件到 OSS。

Parameters

- **filepaths** –list, 本地文件的路径列表。

- **oss_keys** –list, 文件在 OSS 上的路径和名称列表。
- **replace** –boolean, 如果为 True，将覆盖 OSS 上的同名文件。默认为 False。
- **threads** –int, 并行上传的线程数。默认为 5。

create_folder(*folder_path: str*)

在 OSS 上创建文件夹。

Parameters

folder_path –string, 需要创建的文件夹的路径。

delete_file(*oss_key: str*) → bool

从 OSS 删除文件。

Parameters

oss_key –string, 文件在 OSS 上的路径和名称。

Returns

boolean, 如果删除成功，返回 True；否则，返回 False.

download(*oss_key: str*, *filepath: str*, *replace: bool = False*) → bool

从 OSS 下载文件。

Parameters

- **oss_key** –string, 文件在 OSS 上的路径和名称。
- **filepath** –string, 本地存储文件的路径。

Returns

boolean, 如果下载成功，返回 True；否则，返回 False。

download_folder(*oss_folder: str*, *local_folder: str*, *threads: int = 5*)

从 OSS 下载指定文件夹。

Parameters

- **oss_folder** –string, OSS 上的文件夹路径。
- **local_folder** –string, 本地存储文件夹的路径。
- **threads** –int, 并行下载的线程数。默认为 5。

file_exists(*oss_key: str*) → bool

检查文件是否在 OSS 上存在。

Parameters

oss_key –string, 文件在 OSS 上的路径和名称。

Returns

boolean, 如果文件存在，返回 True；否则，返回 False.

get_file_stream(*oss_key: str*) → BytesIO

获取 OSS 上文件的数据流。

Parameters

oss_key –string, 文件在 OSS 上的路径和名称。

Returns

BytesIO, 文件的数据流。

list_files(*prefix=''*, *extensions=None*)

列举 OSS 上的文件。

Parameters

- **prefix** –string, 列举的文件前缀，默认为空。
- **extensions** –list, 需要列举的文件的后缀名，默认为空，表示列举所有文件。

Returns

list, 列举的文件的名称列表。

multipart_upload(*filepath: str*, *oss_key: str*)

分块上传大文件到 OSS。

Parameters

- **filepath** –string, 本地文件的路径。
- **oss_key** –string, 文件在 OSS 上的路径和名称。

upload(*filepath: str*, *oss_key: str*, *replace: bool = False*) → bool

上传文件到 OSS。

Parameters

- **filepath** –string, 本地文件的路径。
- **oss_key** –string, 文件在 OSS 上的路径和名称。
- **replace** –boolean, 如果为 True，将覆盖 OSS 上的同名文件。默认为 False。

Returns

boolean, 如果上传成功，返回 True；否则，返回 False。

upload_folder(*local_folder: str*, *oss_folder: str*, *replace: bool = False*, *threads: int = 5*)

上传本地文件夹到 OSS。

Parameters

- **local_folder** –string, 本地文件夹的路径。
- **oss_folder** –string, OSS 上的目标文件夹路径。
- **replace** –boolean, 如果为 True，将覆盖 OSS 上的同名文件。默认为 False。

- **threads** –int, 并行上传的线程数。默认为 5。

BarGenerator

class czsc.**BarGenerator**(*base_freq: str*, *freqs: List[str]*, *max_count: int = 5000*, *market='默认'*)

Bases: object

Attributes Summary

version

Methods Summary

init_freq_bars(freq, bars)	初始化某个周期的 K 线序列
update(bar)	更新各周期 K 线

Attributes Documentation

version = 'V231008'

Methods Documentation

init_freq_bars(*freq: str*, *bars: List[*RawBar*]*)

初始化某个周期的 K 线序列

函数计算逻辑:

1. 首先，它断言 `freq`必须是 `self.bars`的键之一。如果 `freq`不在 `self.bars`的键中，代码会抛出一个断言错误。
2. 然后，它断言 `self.bars[freq]`必须为空。如果 `self.bars[freq]`不为空，代码会抛出一个断言错误，并显示一条错误消息。
3. 如果以上两个断言都通过，它会将 `bars`赋值给 `self.bars[freq]`，从而初始化指定频率的 K 线序列。
4. 最后，它会将 `bars`列表中的最后一个 `RawBar`对象的 `symbol`属性赋值给 `self.symbol`。

Parameters

- **freq** –周期名称

- **bars** –K 线序列

update(*bar:* RawBar) → None

更新各周期 K 线

函数计算逻辑：

1. 首先，它获取基准频率 `base_freq`，并断言 `bar`的频率值等于 `base_freq`。
2. 然后，它将 `bar`的符号和日期时间设置为 `self.symbol`和 `self.end_dt`。
3. **接下来，它检查是否已经有一个与 `bar`日期时间相同的 K 线存在于 `self.bars[base_freq]`中。**
   如果存在，它会记录一个警告并返回，不进行任何更新。
4. 如果不存在重复的 K 线，它会遍历 `self.bars`的所有键（即所有的频率），并对每个频率调用 `self._update_freq`方法来更新该频率的 K 线。
5. 最后，它会限制在内存中的 K 线数量，确保每个频率的 K 线数量不超过 `self.max_count`。

Parameters

bar –必须是已经结束的 Bar

Returns

None

CTAResearch

class czsc.**CTAResearch**(*strategy*, *read_bars*, *results_path*, ***kwargs*)

Bases: object

Methods Summary

backtest(symbols[, max_workers])	多进程执行 on bar 回测
check_signals(symbol[, sdt, edt])	在单个品种上检查信号
dummy(symbols[, sdt, edt, max_workers])	使用 DummyBacktest 进行 on sig 回测
replay(symbol[, sdt, edt, refresh])	单品种交易回放

Methods Documentation

backtest(*symbols*, *max_workers=3*, ***kwargs*)

多进程执行 on bar 回测

Parameters

- **symbols** –标的代码列表
- **max_workers** –最大进程数

Returns

None

check_signals(*symbol*, *sdt='20200101'*, *edt='20220101'*)

在单个品种上检查信号

Parameters

- **symbol** –标的代码
- **sdt** –开始时间
- **edt** –结束时间

Returns

None

dummy(*symbols*, *sdt='20200101'*, *edt='20220101'*, *max_workers=1*, ***kwargs*)

使用 DummyBacktest 进行 on sig 回测

Parameters

- **symbols** –品种列表
- **sdt** –回测开始时间
- **edt** –回测结束时间
- **max_workers** –最大进程数
- **kwargs** –

Returns

replay(*symbol*, *sdt='20200101'*, *edt='20220101'*, *refresh=True*)

单品种交易回放

Parameters

- **symbol** –标的代码
- **sdt** –开始时间
- **edt** –结束时间

- **refresh** –是否刷新

Returns

None

CZSC

class czsc.**CZSC**(*bars: List[RawBar]*, *get_signals=None*, *max_bi_num=50*)

Bases: object

Attributes Summary

finished_bis	已完成的笔
fx_list	分型列表，包括 bars_ubi 中的分型
last_bi_extend	判断最后一笔是否在延伸中，True 表示延伸中
ubi	Unfinished Bi，未完成的笔
ubi_fxs	bars_ubi 中的分型

Methods Summary

open_in_browser([width, height])	直接在浏览器中打开分析结果
to_echarts([width, height, bs])	绘制 K 线分析图
to_plotly()	使用 plotly 绘制 K 线分析图
update(bar)	更新分析结果

Attributes Documentation

finished_bis

已完成的笔

fx_list

分型列表，包括 bars_ubi 中的分型

last_bi_extend

判断最后一笔是否在延伸中，True 表示延伸中

ubi

Unfinished Bi，未完成的笔

ubi_fxs

bars_ubi 中的分型

Methods Documentation

open_in_browser(*width: str = '1400px'*, *height: str = '580px'*)

直接在浏览器中打开分析结果

Parameters

- **width** –图表宽度
- **height** –图表高度

Returns

to_echarts(*width: str = '1400px'*, *height: str = '580px'*, *bs=[]*)

绘制 K 线分析图

Parameters

- **width** –宽
- **height** –高
- **bs** –交易标记，默认为空

Returns

to_plotly()

使用 plotly 绘制 K 线分析图

update(*bar:* RawBar)

更新分析结果

Parameters

bar –单根 K 线对象

CrossSectionalPerformance

class czsc.**CrossSectionalPerformance**(*dfh: DataFrame*, ***kwargs*)

Bases: object

根据截面持仓信息，计算截面绩效

Methods Summary

cal_turnover()	计算换手率
cross_net_value([by, values])	计算截面等权净值
report(file_docx)	

Methods Documentation

cal_turnover()

计算换手率

cross_net_value(*by='dt'*, *values='edge'*)

计算截面等权净值

Parameters

- **by** –按什么字段计算截面等权净值，默认按交易时间
- **values** –计算截面等权净值时，使用的字段，默认使用 edge，计算策略收益，可选值：edge, n1b 输入 edge，计算策略收益输入 n1b，计算基准收益

Returns

report(*file_docx*)

CzscJsonStrategy

class czsc.**CzscJsonStrategy**(***kwargs*)

Bases: *CzscStrategyBase*

仅传入 Json 配置的 Positions 就完成策略创建

执行逻辑:

1. 定义 CzscJsonStrategy 类，并继承自 CzscStrategyBase。这个类可以通过仅传入 Json 配置的 Positions 来完成策略创建。
2. 类中定义了一个名为 positions 的属性，使用 @property 装饰器将其标记为只读属性。
3. **在 positions 属性的 getter 方法中，执行以下操作：**
   - **从 self.kwargs 字典中获取键为”files_position”的值，并将其赋值给变量 files。**
     这里的 self.kwargs 可能是通过在实例化该类时传入的参数或其他方式设置的一个字典，其中包含了策略配置文件的路径列表。

- 使用 self.kwargs.get 方法获取键为”check_position”的值，并设置默认值为 True，将其赋值给变量 check。这个值用于确定是否对 JSON 持仓策略进行 MD5 校验。
- **调用 self.load_positions(files, check) 方法，并返回其结果。这个方法可能是从父类 CzscStrategyBase 中继承的方法，**
  用于从配置文件中加载持仓策略。将文件列表和校验标志作为参数传递给该方法，并返回加载的持仓策略列表。

必须参数：

files_position: 以 json 文件配置的策略，每个 json 文件对应一个持仓策略配置 check_position: 是否对 json 持仓策略进行 MD5 校验，默认为 True

Attributes Summary

positions	持仓策略列表

Attributes Documentation

positions

CzscSignals

class czsc.**CzscSignals**(*bg:* BarGenerator | *None* = *None*, ***kwargs*)

Bases: object

缠中说禅技术分析理论之多级别信号计算

Methods Summary

get_signals_by_conf()	通过信号参数配置获取信号
open_in_browser([width, height])	直接在浏览器中打开分析结果
take_snapshot([file_html, width, height])	获取快照
update_signals(bar)	输入基础周期已完成 K 线，更新信号，更新仓位

Methods Documentation

get_signals_by_conf()

通过信号参数配置获取信号

函数执行逻辑：

1. 函数首先创建一个空的有序字典 s。
2. 如果 self.signals_config 不存在，函数直接返回空字典 s，否则，函数遍历其中的每一个配置。
3. 对于每一个参数，函数提取出信号名称和 freq，并根据这两个参数获取相应的信号，获取到的信号被添加到字典 s 中。
4. 函数最后返回字典 s，其中包含了所有获取到的信号。

信号参数配置，格式如下：

signals_config = [

{‘name’：‘czsc.signals.tas_ma_base_V221101’，‘freq’：‘日线’，‘di’：1，‘ma_type’：‘SMA’，‘timeperiod’：5}，{‘name’：‘czsc.signals.tas_ma_base_V221101’，‘freq’：‘日线’，‘di’：5，‘ma_type’：‘SMA’，‘timeperiod’：5}，{‘name’：‘czsc.signals.tas_double_ma_V221203’，‘freq’：‘日线’，‘di’：1，‘ma_seq’：(5, 20)，‘th’：100}，{‘name’：‘czsc.signals.tas_double_ma_V221203’，‘freq’：‘日线’，‘di’：5，‘ma_seq’：(5, 20)，‘th’：100}，

]

Returns

信号字典

open_in_browser(*width='1400px'*, *height='580px'*)

直接在浏览器中打开分析结果

函数执行逻辑：

1. 首先创建一个 HTML 文件的路径 file_html，这个文件将被保存在用户的主目录下，文件名为”temp_czsc_advanced_trader.html”。
2. 然后，函数调用 self.take_snapshot 方法，将分析结果保存为一个 HTML 文件。
3. 最后，函数使用 webbrowser.open 方法打开这个 HTML 文件

take_snapshot(*file_html=None*, *width: str = '1400px'*, *height: str = '580px'*)

获取快照

函数执行逻辑：

1. 函数首先创建一个 Tab 对象，用于存储所有的图表和表格。
2. 函数遍历所有的 freq，对于每一个 freq，函数获取相应的 CZSC 对象，并将其转换为一个图表，然后添加到 Tab 对象中。

3. 函数提取出所有的信号，并按照 freq 分组。对于每一个 freq，函数创建一个表格，包含该 freq 下的所有信号，然后添加到 Tab 对象中。
4. 如果还有其他的信号，函数创建一个表格，包含所有的其他信号，然后添加到 Tab 对象中。
5. 最后，如果提供了 file_html 参数，函数将 Tab 对象渲染为一个 HTML 文件并保存；否则，函数返回 Tab 对象。

Parameters

- **file_html** –交易快照保存的 html 文件名
- **width** –图表宽度
- **height** –图表高度

Returns

update_signals(*bar:* RawBar)

输入基础周期已完成 K 线，更新信号，更新仓位

函数执行逻辑：

1. 函数首先调用 self.bg.update(bar)，输入一个已完成的基础周期 K 线 bar，更新各周期 K 线。
2. 然后，函数遍历所有的 K 线 freq 和对应的 K 线数据，对每一个 K 线数据，函数调用 self.kas[freq].update(b[-1])，更新对应的 CZSC 对象。
3. 函数提取出 K 线的标的代码 bar.symbol，并将其赋值给 self.symbol。
4. 函数提取出基础 freq 的最后一根 K 线 last_bar，并从中提取出结束时间 dt，K 线 IDid，以及收盘价 close，并将它们分别赋值给 self.end_dt，self.bid，和 self.latest_price。
5. 函数创建一个空的有序字典 s，并调用 self.get_signals_by_conf() 获取所有的信号，然后将这些信号更新到字典 s 中。
6. 最后，函数将 last_bar 的所有属性更新到字典 s 中。

Parameters

bar –基础周期已完成 K 线

Returns

None

CzscStrategyBase

class czsc.**CzscStrategyBase**(**kwargs)

Bases: ABC

择时交易策略的要素：

1. 交易品种以及该品种对应的参数
2. K 线周期列表
3. 交易信号参数配置
4. 持仓策略列表

Attributes Summary

base_freq	基础 K 线周期
freqs	K 线周期列表
signals_config	交易信号参数配置
sorted_freqs	排好序的 K 线周期列表
symbol	交易标的
unique_signals	所有持仓策略中的交易信号列表

Methods Summary

backtest(bars, **kwargs)	
check(bars, res_path, **kwargs)	检查交易策略中的信号是否正确
dummy(sigs, **kwargs)	使用信号缓存进行策略回测
init_bar_generator(bars, **kwargs)	使用策略定义初始化一个 BarGenerator 对象
init_trader(bars, **kwargs)	使用策略定义初始化一个 CzscTrader 对象
load_positions(files[, check])	从配置文件中加载持仓策略
positions()	持仓策略列表
replay(bars, res_path, **kwargs)	交易策略交易过程回放
save_positions(path)	保存持仓策略配置

Attributes Documentation

base_freq

基础 K 线周期

freqs

K 线周期列表

signals_config

交易信号参数配置

sorted_freqs

排好序的 K 线周期列表

symbol

交易标的

unique_signals

所有持仓策略中的交易信号列表

Methods Documentation

backtest(*bars: List[*RawBar*]*, ***kwargs*) → *CzscTrader*

check(*bars: List[*RawBar*]*, *res_path*, ***kwargs*)

检查交易策略中的信号是否正确

Parameters

- **bars** –基础周期 K 线
- **res_path** –结果目录
- **kwargs** –bg 已经初始化好的 BarGenerator 对象，如果传入了 bg，则忽略 sdt 和 n 参数 sdt 初始化开始日期 n 初始化最小 K 线数量

Returns

dummy(*sigs: List[dict]*, ***kwargs*) → *CzscTrader*

使用信号缓存进行策略回测

Parameters

sigs –信号缓存，一般指 generate_czsc_signals 函数计算的结果缓存

Returns

完成策略回测后的 CzscTrader 对象

init_bar_generator(*bars: List[RawBar]*, ***kwargs*)

使用策略定义初始化一个 BarGenerator 对象

函数执行逻辑：

- 该方法的目的是使用策略定义初始化一个 BarGenerator 对象。BarGenerator 是用于生成 K 线数据的类。
- 参数 bars 表示基础周期的 K 线数据，**kwargs 用于接收额外的关键字参数。
- 首先，方法获取了基础 K 线的频率，并检查了是否已经有一个初始化好的 BarGenerator 对象传入。
- 然后，根据基础频率是否在排序后的频率列表中，确定要使用的频率列表。
- 如果没有传入 BarGenerator 对象，则根据传入的基础 K 线数据和其他参数创建一个新的 BarGenerator 对象，并使用部分 K 线数据初始化它。余下的 K 线数据将用于 trader 的初始化区间。
- 如果传入了 BarGenerator 对象，则会做一些断言检查，确保传入的基础 K 线数据与已有的 BarGenerator 对象的基础周期一致，并且 BarGenerator 的 end_dt 是 datetime 类型。然后，筛选出在 BarGenerator 的 end_dt 之后的 K 线数据。
- 最后，返回 BarGenerator 对象和余下的 K 线数据。

Parameters

- **bars** –基础周期 K 线
- **kwargs** –bg 已经初始化好的 BarGenerator 对象，如果传入了 bg，则忽略 sdt 和 n 参数 sdt 初始化开始日期 n 初始化最小 K 线数量

Returns

init_trader(*bars: List[RawBar]*, ***kwargs*) → *CzscTrader*

使用策略定义初始化一个 CzscTrader 对象

注意：这里会将所有持仓策略在 sdt 之后的交易信号计算出来并缓存在持仓策略实例内部，所以初始化的过程本身也是回测的过程。

函数执行逻辑：

- 首先，它通过调用 init_bar_generator 方法获取已经初始化好的 BarGenerator 对象和余下的 K 线数据。
- 然后，它创建一个 CzscTrader 对象，将 BarGenerator 对象、持仓策略的深拷贝、交易信号配置的深拷贝等参数传递给 CzscTrader 的构造函数。
- 接着，使用余下的 K 线数据对 CzscTrader 对象进行初始化，通过调用 trader.on_bar(bar) 方法处理每一根 K 线数据。
- 最后，返回初始化完成的 CzscTrader 对象。

Parameters

- **bars** –基础周期 K 线
- **kwargs** –bg 已经初始化好的 BarGenerator 对象，如果传入了 bg，则忽略 sdt 和 n 参数 sdt 初始化开始日期 n 初始化最小 K 线数量

Returns

完成策略初始化后的 CzscTrader 对象

load_positions(*files: List*, *check=True*) → List[*Position*]

从配置文件中加载持仓策略

Parameters

- **files** –以 json 格式保存的持仓策略文件列表
- **check** –是否校验 MD5 值，默认为 True

Returns

持仓策略列表

abstract positions() → List[*Position*]

持仓策略列表

replay(*bars: List[*RawBar*]*, *res_path*, ***kwargs*)

交易策略交易过程回放

函数执行逻辑：

- 该方法用于交易策略交易过程的回放。它接受基础周期的 K 线数据、结果目录以及额外的关键字参数作为输入。
- 首先，它检查 refresh 参数，如果为 True，则使用 shutil.rmtree 删除已存在的结果目录。
- 然后，它检查结果目录是否已存在，并且是否允许覆盖。如果目录已存在且不允许覆盖，则记录一条警告信息并返回。
- 通过调用 os.makedirs 创建结果目录，确保目录的存在。
- 接着，调用 init_bar_generator 方法初始化 BarGenerator 对象，并进行相关的初始化操作。
- 创建一个 CzscTrader 对象，并将初始化好的 BarGenerator 对象、持仓策略的深拷贝、交易信号配置的深拷贝等参数传递给 CzscTrader 的构造函数。
- 为每个持仓策略创建相应的目录。
- 遍历 K 线数据，调用 trader.on_bar(bar) 方法处理每一根 K 线数据。
- **在每根 K 线数据处理完成后，检查每个持仓策略是否有操作，并且操作的时间是否与当前 K 线的时间一致**。
  如果有操作，则生成相应的 HTML 文件名，并调用 trader.take_snapshot(file_html) 方法生成交易快照。

- 最后，遍历每个持仓策略，记录其评估信息，包括多空合并表现、多头表现、空头表现等。

Parameters

- **bars** –基础周期 K 线
- **res_path** –结果目录
- **kwargs** –bg 已经初始化好的 BarGenerator 对象，如果传入了 bg，则忽略 sdt 和 n 参数 sdt 初始化开始日期 n 初始化最小 K 线数量 refresh 是否刷新结果目录

Returns

save_positions(*path*)

保存持仓策略配置

Parameters

path –结果路径

Returns

None

CzscTrader

class czsc.**CzscTrader**(*bg: BarGenerator | None = None*, *positions: List[Position] | None = None*, *ensemble_method: AnyStr | Callable = 'mean'*, ***kwargs*)

Bases: *CzscSignals*

缠中说禅技术分析理论之多级别联立交易决策类（支持多策略独立执行）

Attributes Summary

pos_changed	判断仓位是否发生变化

Methods Summary

get_ensemble_pos([method])	获取多个仓位的集成仓位
get_ensemble_weight([method])	获取 CzscTrader 中所有 positions 按照 method 方法集成之后的权重
get_position(name)	获取指定名称的仓位策略对象
on_bar(bar)	输入基础周期已完成 K 线，更新信号，更新仓位
on_sig(sig)	通过信号字典直接交易，用于快速回测场景
take_snapshot([file_html, width, height])	获取快照
update(bar)	输入基础周期已完成 K 线，更新信号，更新仓位
weight_backtest(**kwargs)	执行仓位集成权重的回测

Attributes Documentation

pos_changed

判断仓位是否发生变化

1. 函数首先检查 self.positions 是否为空。如果为空，即没有仓位，函数直接返回 False。
2. **如果 self.positions 不为空，函数遍历所有的仓位，对于每一个仓位，函数检查其 pos_changed 属性**。
   如果任何一个仓位的 pos_changed 属性为 True，即该仓位发生了变化，函数返回 True。

   Returns

   True/False

Methods Documentation

get_ensemble_pos(*method: AnyStr | Callable | None = None*) → float

获取多个仓位的集成仓位

函数执行逻辑：

1. 函数首先检查 self.positions 是否为空。如果为空，即没有仓位，函数直接返回 0。
2. 如果 self.positions 不为空，函数获取集成方法 method。如果没有传入 method 参数，函数使用 self.__ensemble_method 作为集成方法。
3. **如果 method 是一个字符串，函数将其转换为小写，然后获取所有仓位的仓位序列 pos_seq**。
   1. 如果 method 是” mean”，函数计算 pos_seq 的平均值作为集成仓位。
   2. 如果 method 是” vote”，函数计算 pos_seq 的和的符号作为集成仓位。
   3. 如果 method 是” max”，函数获取 pos_seq 的最大值作为集成仓位。

4. 如果 method 不是以上任何一个值，函数抛出一个值错误。

4. 如果 method 不是一个字符串，即它是一个回调函数，函数将所有仓位的名称和仓位组成的字典作为参数传入 method，并将返回值作为集成仓位。

Parameters

method –多个仓位集成一个仓位的方法，可选值 mean, vote, max；也可以传入一个回调函数

假设有三个仓位对象，当前仓位分别是 1, 1, -1 mean - 平均仓位，pos = np.mean([1, 1, -1]) = 0.33 vote - 投票表决，pos = 1 max - 取最大，pos = 1

对于传入回调函数的情况，输入是 self.positions

Returns

pos, 集成仓位

get_ensemble_weight(*method: AnyStr | Callable | None = None*)

获取 CzscTrader 中所有 positions 按照 method 方法集成之后的权重

函数执行逻辑：

1. 函数首先接收一个参数 method，这是集成方法，可以是字符串或者一个回调函数。
2. 函数检查是否提供了 method 参数。如果没有提供，函数使用 self.__ensemble_method 作为集成方法；如果提供了，函数使用提供的 method 作为集成方法。
3. 函数调用 get_ensemble_weight 函数，输入 self 和 method，获取所有仓位按照指定方法集成之后的权重。

Parameters

- **method** –str or callable 集成方法，可选值包括：’mean’，‘max’，‘min’，‘vote’ 也可以传入自定义的函数，函数的输入为 dict，key 为 position.name，value 为 position.pos，样例输入：

  {‘多头策略 A’：1，‘多头策略 B’：1，‘空头策略 A’：-1}

- **kwargs** –

Returns

pd.DataFrame columns = [‘dt’，‘symbol’，‘weight’，‘price’]

get_position(*name: str*) → *Position* | None

获取指定名称的仓位策略对象

函数执行逻辑：

1. 函数首先接收一个参数 name，这是要查找的仓位名称。
2. 函数检查 self.positions 是否为空。如果为空，即没有仓位，函数直接返回 None。

3. 如果 self.positions 不为空，函数遍历所有的仓位，对于每一个仓位，函数检查其名称是否与输入的名称相同。如果相同，函数返回该仓位。
4. 如果遍历所有的仓位都没有找到与输入名称相同的仓位，函数返回 None。

Parameters

name –仓位名称

Returns

Position

on_bar(*bar:* RawBar) → None

输入基础周期已完成 K 线，更新信号，更新仓位

Parameters

bar –基础周期已完成 K 线

Returns

None

on_sig(*sig: dict*) → None

通过信号字典直接交易，用于快速回测场景

函数执行逻辑：

1. 函数首先接收一个参数 sig，这是一个信号字典，赋值给 self.s。
2. **函数从 sig 中提取出标的代码 symbol，结束时间 dt，K 线 ID id，以及收盘价 close，**
   并将它们分别赋值给 self.symbol，self.end_dt，self.bid，和 self.latest_price。
4. 如果 self.positions 不为空，即存在持仓策略，函数遍历所有 position，函数调用 position.update(self.s)，更新该仓位的状态

Parameters

sig –信号字典

Returns

None

take_snapshot(*file_html=None*, *width: str = '1400px'*, *height: str = '580px'*)

获取快照

Parameters

- **file_html** –交易快照保存的 html 文件名
- **width** –图表宽度
- **height** –图表高度

Returns

update(*bar:* RawBar) → None

输入基础周期已完成 K 线，更新信号，更新仓位

函数执行逻辑：

1. 函数首先接收一个参数 bar，这是一个已完成的基础周期 K 线。
2. 函数调用 self.update_signals(bar)，输入这个已完成的基础周期 K 线，更新信号。
3. 如果 self.positions 不为空，即存在仓位，函数遍历所有的仓位，对于每一个仓位，函数调用 position.update(self.s)，更新该仓位的状态。

Parameters

bar –基础周期已完成 K 线

Returns

None

weight_backtest(***kwargs*)

执行仓位集成权重的回测

Parameters

kwargs –

- method: str or callable，集成方法，参考 get_ensemble_weight 方法
- digits: int，权重小数点后保留的位数，例如 2 表示保留两位小数
- fee_rate: float，手续费率，例如 0.0002 表示万二

Returns

回测结果

DataClient

class czsc.**DataClient**(*token=None*, *url='http://api.tushare.pro'*, *timeout=300*, ***kwargs*)

Bases: object

Methods Summary

clear_cache()	清空缓存
post_request(api_name[, fields])	执行 API 数据查询

Methods Documentation

clear_cache()

清空缓存

post_request(*api_name*, *fields=''*, ***kwargs*)

执行 API 数据查询

Parameters

- **api_name** –str, 查询接口名称
- **fields** –str, 查询字段
- **kwargs** –dict, 查询参数
  - ttl: int, 缓存有效期，单位秒，-1 表示不过期

Returns

pd.DataFrame

Direction

class czsc.**Direction**(*value*)

Bases: Enum

An enumeration.

Attributes Summary

Down
Up

Attributes Documentation

Down = '向下'

Up = '向上'

DiskCache

class czsc.**DiskCache**(*path=None*)

Bases: object

Methods Summary

get(k[, suffix])	读取缓存文件
is_found(k[, suffix, ttl])	判断缓存文件是否存在
remove(k[, suffix])	
set(k, v[, suffix])	写入缓存文件

Methods Documentation

get(*k: str*, *suffix: str = 'pkl'*) → Any

读取缓存文件

Parameters

- **k** –缓存文件名
- **suffix** –缓存文件后缀，支持 pkl, json, txt, csv, xlsx, feather, parquet

Returns

缓存文件内容

is_found(*k: str*, *suffix: str = 'pkl'*, *ttl=-1*) → bool

判断缓存文件是否存在

Parameters

- **k** –缓存文件名
- **suffix** –缓存文件后缀，支持 pkl, json, txt, csv, xlsx, feather, parquet
- **ttl** –缓存文件有效期，单位：秒，-1 表示永久有效

Returns

bool

remove(*k: str*, *suffix: str = 'pkl'*)

set(*k: str*, *v: Any*, *suffix: str = 'pkl'*)

写入缓存文件

Parameters

- **k** –缓存文件名
- **v** –缓存文件内容
- **suffix** –缓存文件后缀，支持 pkl, json, txt, csv, xlsx, feather, parquet

DummyBacktest

class czsc.**DummyBacktest**(*strategy*, *signals_path*, *results_path*, *read_bars*, ***kwargs*)

Bases: object

Methods Summary

execute(symbols[, n_jobs])	回测多个品种
one_pos_stats(pos_name)	分析单个持仓策略的表现
one_symbol_dummy(symbol)	回测单个品种
replay(symbol)	回放单个品种的交易

Methods Documentation

execute(*symbols*, *n_jobs=2*, ***kwargs*)

回测多个品种

Parameters

- **symbols** –品种列表
- **n_jobs** –进程数量，默认为 2 需要注意的是：1. 如果进程数过多，可能会导致内存不足 2. 多进程在 pycharm 的 ipython 中无法使用，需要在命令行中运行
- **kwargs** –

Returns

one_pos_stats(*pos_name*)

分析单个持仓策略的表现

one_symbol_dummy(*symbol*)

回测单个品种

replay(*symbol*)

回放单个品种的交易

Event

class czsc.**Event**(*operate: czsc.enum.Operate, factors: List[czsc.objects.Factor], signals_all: List[czsc.objects.Signal] = <factory>, signals_any: List[czsc.objects.Signal] = <factory>, signals_not: List[czsc.objects.Signal] = <factory>, name: str = ''*)

Bases: object

Attributes Summary

name	
unique_signals	获取 Event 的唯一信号列表

Methods Summary

dump()	将 Event 对象转存为 dict
get_signals_config([signals_module])	获取事件的信号配置
is_match(s)	判断 event 是否满足
load(raw)	从 dict 中创建 Event

Attributes Documentation

name: str = ''

unique_signals

获取 Event 的唯一信号列表

Methods Documentation

dump() → dict

将 Event 对象转存为 dict

get_signals_config(*signals_module: str = 'czsc.signals'*) → List[Dict]

获取事件的信号配置

is_match(*s: dict*)

判断 event 是否满足

代码的执行逻辑如下：

1. 首先判断 signals_not 中的信号是否得到满足，如果满足任意一个信号，则直接返回 False，表示事件不满足。
2. 接着判断 signals_all 中的信号是否全部得到满足，如果有任意一个信号不满足，则直接返回 False，表示事件不满足。
3. 然后判断 signals_any 中的信号是否有一个得到满足，如果一个都不满足，则直接返回 False，表示事件不满足。
4. 最后判断因子是否满足，顺序遍历因子列表，找到第一个满足的因子就退出，并返回 True 和该因子的名称，表示事件满足。
5. 如果遍历完所有因子都没有找到满足的因子，则返回 False，表示事件不满足。

classmethod load(*raw: dict*)

从 dict 中创建 Event

Parameters

raw –样例如下 {‘name’：‘单测’，

’operate’：‘开多’，‘factors’：[{‘name’：‘测试’，

’signals_all’：[‘15 分钟 _ 倒 0 笔 _ 长度 _ 大于 5_ 其他 _ 其他 _0’], ‘signals_any’ : [], ‘signals_not’：[]}],

’signals_all’：[‘15 分钟 _ 倒 0 笔 _ 方向 _ 向上 _ 其他 _ 其他 _0’], ‘signals_any’ : [], ‘signals_not’：[]}

Returns

EventMatchSensor

class czsc.**EventMatchSensor**(*events: List[Dict[str, Any] | Event]*, *symbols: List[str]*, *read_bars: Callable*, ***kwargs*)

Bases: object

Methods Summary

get_event_csc(event_name)	获取事件的截面匹配次数

Methods Documentation

get_event_csc(*event_name: str*)

获取事件的截面匹配次数

csc = cross section count，表示截面匹配次数

函数执行逻辑：

1. 创建一个 self.data 的副本 df。
2. 在 df 中筛选出 event_name 列等于 1 的行。
3. **使用 groupby 方法按 symbol 和 dt 对筛选后的数据进行分组，并计算 event_name 列的总和。**
   结果将形成一个新的 DataFrame，其中索引为 (symbol, dt) 组合，只有一个列 event_name，表示每个组合的匹配次数。
4. **再次使用 groupby 方法按 dt 对上一步的结果进行分组，并计算 event_name 列的总和。这次得到的新 DataFrame**
   只有一个列 event_name，表示在每个时间点所有标的的事件匹配总数。

Parameters

event_name –事件名称

Returns

DataFrame

ExitsOptimize

class czsc.**ExitsOptimize**(*read_bars: Callable*, ***kwargs*)

Bases: object

基础策略出场优化流程

Methods Summary

execute([n_jobs])	批量优化策略

Methods Documentation

execute(*n_jobs=1*)

批量优化策略

Parameters

n_jobs –进程数量

Returns

Factor

class czsc.**Factor**(*signals_all: List[czsc.objects.Signal], signals_any: List[czsc.objects.Signal] = <factory>, signals_not: List[czsc.objects.Signal] = <factory>, name: str = ''*)

Bases: object

Attributes Summary

name	
unique_signals	获取 Factor 的唯一信号列表

Methods Summary

dump()	将 Factor 对象转存为 dict
is_match(s)	判断 factor 是否满足
load(raw)	从 dict 中创建 Factor

Attributes Documentation

name: str = ''

unique_signals

获取 Factor 的唯一信号列表

Methods Documentation

dump() → dict

将 Factor 对象转存为 dict

is_match(*s: dict*) → bool

判断 factor 是否满足

classmethod load(*raw: dict*)

从 dict 中创建 Factor

Parameters

raw –样例如下 {‘name’：‘单测’，‘signals_all’: [‘15 分钟 _ 倒 0 笔 _ 方向 _ 向上 _ 其他 _ 其他 _0’，‘15 分钟 _ 倒 0 笔 _ 长度 _ 大于 5_ 其他 _ 其他 _0’], ‘signals_any’: [], ‘signals_not’: []}

Returns

FixedNumberSelector

class czsc.**FixedNumberSelector**(*dfs*, *k*, *d*, ***kwargs*)

Bases: object

选择固定数量（等权）的交易品种

可优化项：1. 传入 res_path, 将分析过程和分析结果保存下来 2. 支持传入大盘择时信号，例如：大盘择时信号为空头时，多头只平不开

Freq

class czsc.**Freq**(*value*)

Bases: Enum

An enumeration.

Attributes Summary

D
F1
F10
F12
F120
F15
F2
F20
F3
F30
F4
F5
F6
F60
M
S
Tick
W
Y

Attributes Documentation

D = '日线'

F1 = '1 分钟'

F10 = '10 分钟'

F12 = '12 分钟'

F120 = '120 分钟'

F15 = '15 分钟'

F2 = '2 分钟'

F20 = '20 分钟'

F3 = '3 分钟'

F30 = '30 分钟'

F4 = '4 分钟'

F5 = '5 分钟'

F6 = '6 分钟'

F60 = '60 分钟'

M = '月线'

S = '季线'

Tick = 'Tick'

W = '周线'

Y = '年线'

KlineChart

class czsc.**KlineChart**(*n_rows=3*, ***kwargs*)

Bases: object

K 线绘图工具类

plotly 参数详解: https://www.jianshu.com/p/4f4daf47cc85

Methods Summary

add_bar_indicator(x, y, name, row[, color])	绘制条形图指标
add_indicator(dt[, scatters, scatter_names, ...])	绘制曲线叠加 bar 型指标
add_kline(kline[, name])	绘制 K 线
add_macd(kline[, row])	绘制 MACD 图
add_marker_indicator(x, y, name, row[, text])	绘制标记类指标
add_scatter_indicator(x, y, name, row[, text])	绘制线性/离散指标
add_sma(kline[, row, ma_seq, visible])	绘制均线图
add_vol(kline[, row])	绘制成交量图
open_in_browser([file_name])	在浏览器中打开

Methods Documentation

add_bar_indicator(*x*, *y*, *name: str*, *row: int*, *color=None*, ***kwargs*)

绘制条形图指标

绘图 API 文档：https://plotly.com/python-api-reference/generated/plotly.graph_objects.Bar.html

函数执行逻辑：

1. 获取自定义参数 hover_template、show_legend、visible 和 base。这些参数分别对应于鼠标悬停时显示的模板、是否显示图例、是否可见和基线（默认为 True）。
2. 如果 color 参数为空，则使用 self.color_red 作为颜色。
3. **使用给定的 x、y 数据创建一个 go.Bar 对象（条形图），并传入以下参数：**
    - x: 指标的 x 轴数据
    - y: 指标的 y 轴数据
    - marker_line_color: 条形边框的颜色
    - marker_color: 条形填充的颜色
    - name: 指标名称
    - showlegend: 是否显示图例
    - hovertemplate: 鼠标悬停时显示的模板
    - visible: 是否可见
    - base: 基线，默认为 True

4. 调用 self.fig.add_trace 方法将创建的 go.Bar 对象添加到指定子图中，并更新所有 traces 的 X 轴属性为“x1”。

Parameters

- **x** –指标的 x 轴
- **y** –指标的 y 轴
- **name** –指标名称
- **row** –放入第几个子图
- **color** –指标的颜色，可以是单个颜色，也可以是一个列表，列表长度和 y 的长度一致，指示每个 y 的颜色比如：color = ‘rgba(249,41,62,0.7)’ 或者 color = [‘rgba(249,41,62,0.7)’, ‘rgba(0,170,59,0.7)’]
- **kwargs** –

Returns

add_indicator(*dt*, *scatters: list | None = None*, *scatter_names: list | None = None*, *bar=None*, *bar_name=''*, *row=4*, ***kwargs*)

绘制曲线叠加 bar 型指标

1. 获取自定义参数 line_width，默认值为 0.6。
2. **如果 scatters（列表）不为空，则遍历 scatters 中的所有散点数据：**
   - **对于每个散点数据，调用 add_scatter_indicator 方法将其绘制为折线图。传递以下参数：**
     - x: 日期时间数据
     - y: 散点数据
     - name: 图例名称，来自 scatter_names 列表
     - row: 指定要添加指标的子图行数，默认值为 4
     - show_legend: 是否显示图例，默认值为 False
     - line_width: 线宽，默认值为 0.6
3. 如果 bar 不为空，则使用 np.where 函数根据 bar 值大于零的情况设置颜色：大于零使用红色（self.color_red），否则使用绿色（self.color_green）。
4. **调用 add_bar_indicator 方法将 bar 绘制为柱状图。传递以下参数：**
   - x: 日期时间数据
   - y: bar 数据
   - name: 图例名称，为传入的 bar_name 参数
   - row: 指定要添加指标的子图行数，默认值为 4

- color: 根据上一步计算的颜色设置
- show_legend: 是否显示图例，默认值为 False

add_kline(*kline: DataFrame*, *name: str = 'K 线'*, ***kwargs*)

绘制 K 线

函数执行逻辑：

1. 检查 kline 数据框是否包含‘text’列。如果没有，则添加一个空字符串列。
2. **使用 go.Candlestick 创建一个 K 线图，并传入以下参数：**
   - x: 日期时间数据
   - open, high, low, close: 开盘价、最高价、最低价和收盘价
   - text: 显示在每个 K 线上的文本标签
   - name: 图例名称
   - showlegend: 是否显示图例
   - increasing_line_color 和 decreasing_line_color: 上涨时的颜色和下跌时的颜色
   - increasing_fillcolor 和 decreasing_fillcolor: 上涨时填充颜色和下跌时填充颜色
   - **kwargs: 可以传递其他自定义参数给 Candlestick 函数。
3. 将创建的烛台图对象添加到 self.fig 中的第一个子图（row=1, col=1）。
4. 使用 fig.update_traces 更新所有 traces 的 xaxis 属性为“x1”。

add_macd(*kline: DataFrame*, *row=3*, ***kwargs*)

绘制 MACD 图

函数执行逻辑：

1. 首先，复制输入的 kline 数据框到 df。
2. 获取自定义参数 fastperiod、slowperiod 和 signalperiod。这些参数分别对应于计算 MACD 时使用的快周期、慢周期和信号周期，默认值分别为 12、26 和 9。
3. 使用 talib 库的 MACD 函数计算 MACD 值（diff, dea, macd）。
4. 创建一个名为 macd_colors 的 numpy 数组，根据 macd 值大于零的情况设置颜色：大于零使用红色（self.color_red），否则使用绿色（self.color_green）。
5. **调用 add_scatter_indicator 方法将 diff 和 dea 绘制为折线图。传递以下参数：**
   - x: 日期时间数据
   - y: diff 或 dea 数据
   - name: 图例名称，分别为“DIFF”和“DEA”
   - row: 指定要添加指标的子图行数，默认值为 3

- line_color: 线的颜色，分别为‘white’和‘yellow’
- show_legend: 是否显示图例，默认值为 False
- line_width: 线宽，默认值为 0.6

6. **调用 add_bar_indicator 方法将 macd 绘制为柱状图。传递以下参数：**
   - x: 日期时间数据
   - y: macd 数据
   - name: 图例名称，为“MACD”
   - row: 指定要添加指标的子图行数，默认值为 3
   - color: 根据 macd_colors 设置颜色
   - show_legend: 是否显示图例，默认值为 False

add_marker_indicator(*x*, *y*, *name: str*, *row: int*, *text=None*, ***kwargs*)

绘制标记类指标

函数执行逻辑：

1. **获取自定义参数 line_color、line_width、hover_template、show_legend 和 visible。**
   这些参数分别对应于折线颜色、宽度、鼠标悬停时显示的模板、是否显示图例和是否可见。
2. **使用给定的 x、y 数据创建一个 go.Scatter 对象（散点图），并传入以下参数：**
   - x: 指标的 x 轴数据
   - y: 指标的 y 轴数据
   - name: 指标名称
   - text: 文本说明
   - line_width: 线宽
   - line_color: 线颜色
   - hovertemplate: 鼠标悬停时显示的模板
   - showlegend: 是否显示图例
   - visible: 是否可见
   - opacity: 透明度
   - mode: 绘制模式，为‘markers’表示只绘制标记
   - marker: 标记的样式，包括大小、颜色和符号
3. 调用 self.fig.add_trace 方法将创建的 go.Scatter 对象添加到指定子图中，并更新所有 traces 的 X 轴属性为“x1”。

Parameters

- **x** –指标的 x 轴
- **y** –指标的 y 轴
- **name** –指标名称
- **row** –放入第几个子图
- **text** –文本说明
- **kwargs** –

Returns

add_scatter_indicator(*x*, *y*, *name: str*, *row: int*, *text=None*, ***kwargs*)

绘制线性/离散指标

绘图 API 文档：https://plotly.com/python-api-reference/generated/plotly.graph_objects.Scatter.html

函数执行逻辑：

1. 获取自定义参数 mode、hover_template、show_legend、opacity 和 visible。这些参数分别对应于绘图模式、鼠标悬停时显示的模板、是否显示图例、透明度和是否可见。
2. **使用给定的 x、y 数据创建一个 go.Scatter 对象（散点图），并传入以下参数：**
   - x: 指标的 x 轴数据
   - y: 指标的 y 轴数据
   - name: 指标名称
   - text: 文本说明
   - mode: 绘制模式，默认为‘text+lines’，表示同时绘制文本和线条
   - hovertemplate: 鼠标悬停时显示的模板
   - showlegend: 是否显示图例
   - visible: 是否可见
   - opacity: 透明度
3. 调用 self.fig.add_trace 方法将创建的 go.Scatter 对象添加到指定子图中，并更新所有 traces 的 X 轴属性为“x1”。

Parameters

- **x** –指标的 x 轴
- **y** –指标的 y 轴
- **name** –指标名称

- **row** –放入第几个子图
- **text** –文本说明
- **kwargs** –

Returns

add_sma(*kline: DataFrame*, *row=1*, *ma_seq=(5, 10, 20)*, *visible=False*, ***kwargs*)

绘制均线图

函数执行逻辑:

1. 复制输入的 kline 数据框到 df。
2. 获取自定义参数 line_width，默认值为 0.6。
3. **遍历 ma_seq 中的所有均线周期:**
   - 对每个周期使用 pandas rolling 方法计算收盘价的移动平均线。
   - **调用 add_scatter_indicator 方法将移动平均线数据绘制为折线图。传递以下参数:**
     - x: 日期时间数据
     - y: 移动平均线数据
     - name: 图例名称，格式为 “MA{ma}”，其中 {ma} 是当前的均线周期。
     - row: 指定要添加指标的子图行数，默认值为 1
     - line_width: 线宽，默认值为 0.6
     - visible: 是否可见，默认值为 False
     - show_legend: 是否显示图例，默认值为 True

add_vol(*kline: DataFrame*, *row=2*, ***kwargs*)

绘制成交量图

函数执行逻辑:

1. 首先，复制输入的 kline 数据框到 df。
2. 使用 np.where 函数根据收盘价（df[‘close’]）和开盘价（df[‘open’]）之间的关系为 df 创建一个新列‘vol_color’。如果收盘价大于开盘价，则使用红色（self.color_red），否则使用绿色（self.color_green）。
3. **调用 add_bar_indicator 方法绘制成交量图。传递以下参数:**
   - x: 日期时间数据
   - y: 成交量数据
   - color: 根据‘vol_color’列的颜色
   - name: 图例名称

- row: 指定要添加指标的子图行数，默认值为 2
- show_legend: 是否显示图例，默认值为 False

open_in_browser(*file_name: str | None = None*, ***kwargs*)

在浏览器中打开

NewBar

class czsc.**NewBar**(*symbol: str*, *id: int*, *dt: ~datetime.datetime*, *freq: ~czsc.enum.Freq*, *open: float*, *close: float*, *high: float*, *low: float*, *vol: float*, *amount: float*, *elements: ~typing.List = <factory>*, *cache: dict = <factory>*)

Bases: object

去除包含关系后的 K 线元素

Attributes Summary

raw_bars

Attributes Documentation

raw_bars

OpensOptimize

class czsc.**OpensOptimize**(*read_bars: Callable*, ***kwargs*)

Bases: object

基础策略入场优化流程

Methods Summary

execute([n_jobs])	批量优化策略

Methods Documentation

execute(*n_jobs=1*)

批量优化策略

Parameters

n_jobs – 进程数量

Returns

Operate

class czsc.**Operate**(*value*)

Bases: Enum

An enumeration.

Attributes Summary

HL
HO
HS
LE
LO
SE
SO

Attributes Documentation

HL = '持多'

HO = '持币'

HS = '持空'

LE = '平多'

LO = '开多'

SE = '平空'

SO = '开空'

PairsPerformance

class czsc.**PairsPerformance**(*df_pairs: DataFrame*)

Bases: object

交易对效果评估

Attributes Summary

basic_info	写入基础信息

Methods Summary

agg_statistics(col)	按列聚合进行交易对评价
agg_to_excel(file_xlsx)	遍历聚合列，保存结果到 Excel 文件中
get_pairs_statistics(df_pairs)	统计一组交易的基本信息

Attributes Documentation

basic_info

写入基础信息

Methods Documentation

agg_statistics(*col: str*)

按列聚合进行交易对评价

agg_to_excel(*file_xlsx*)

遍历聚合列，保存结果到 Excel 文件中

static get_pairs_statistics(*df_pairs: DataFrame*)

统计一组交易的基本信息

Parameters

df_pairs –

Returns

Position

class czsc.**Position**(*symbol: str*, *opens: List[Event]*, *exits: List[Event] = []*, *interval: int = 0*, *timeout: int = 1000*, *stop_loss=1000*, *T0: bool = False*, *name=None*)

Bases: object

Attributes Summary

pairs	开平交易列表
unique_signals	获取所有事件的唯一信号列表

Methods Summary

dump([with_data])	将对象转换为 dict
evaluate([trade_dir])	评估交易表现
evaluate_holds([trade_dir])	按持仓信号评估交易表现
get_signals_config([signals_module])	获取事件的信号配置
load(raw)	从 dict 中创建 Position
update(s)	更新持仓状态

Attributes Documentation

pairs

开平交易列表

返回样例：

[{‘标的代码’：‘000001.SH’，

‘交易方向’：‘多头’，‘开仓时间’：Timestamp(‘2020-04-17 00:00:00’)，‘平仓时间’：Timestamp(‘2020-04-20 00:00:00’)，‘开仓价格’：2838.49，‘平仓价格’：2852.55，‘持仓 K 线数’：1，‘事件序列’：‘开多 @ 站上 SMA5 -> 开多 @ 站上 SMA5’，‘持仓天数’：3.0，‘盈亏比例’：49.53},

{‘标的代码’：‘000001.SH’，

‘交易方向’：‘多头’，‘开仓时间’：Timestamp(‘2020-04-20 00:00:00’)，‘平仓时间’：Timestamp(‘2020-04-24 00:00:00’)，‘开仓价格’：2852.55，‘平仓价格’：2808.53，‘持仓 K 线数’：4，‘事件序列’：‘开多 @ 站上 SMA5 -> 平多 @100BP 止损’，‘持仓天数’：4.0，‘盈亏比例’：-154.32}]

数据说明：

1. 盈亏比例，单位是 BP
2. 持仓天数，单位是自然日
3. 持仓 K 线数，指基础周期 K 线数量

unique_signals

获取所有事件的唯一信号列表

Methods Documentation

dump(*with_data=False*)

将对象转换为 dict

evaluate(*trade_dir: str = '多空'*) → dict

评估交易表现

Parameters

trade_dir –交易方向，可选值 [‘多头’，‘空头’，‘多空’]

Returns

交易表现

evaluate_holds(*trade_dir: str = '多空'*) → dict

按持仓信号评估交易表现

Parameters

trade_dir –交易方向，可选值 [‘多头’，‘空头’，‘多空’]

Returns

交易表现

get_signals_config(*signals_module: str = 'czsc.signals'*) → List[Dict]

获取事件的信号配置

classmethod load(*raw: dict*)

从 dict 中创建 Position

Parameters

raw –样例如下

Returns

update(*s: dict*)

更新持仓状态

函数执行逻辑：

- 首先，检查最新信号的时间是否在上次信号之前，如果是则打印警告信息并返回。
- 初始化一些变量，包括操作类型（op）和操作描述（op_desc）。
- 遍历所有的事件，检查是否与最新信号匹配。如果匹配，则记录操作类型和操作描述，并跳出循环。
- 提取最新信号的相关信息，包括交易对符号、时间、价格和成交量。
- 更新持仓状态的结束时间为最新信号的时间。
- 如果操作类型是开仓（LO 或 SO），更新最后一个事件的信息。
- 定义一个内部函数 __create_operate，用于创建操作记录。
- 根据操作类型更新仓位和操作记录。
  - 如果操作类型是 LO（开多），检查是否满足开仓条件，如果满足则开多仓，否则只平空仓。
  - 如果操作类型是 SO（开空），检查是否满足开仓条件，如果满足则开空仓，否则只平多仓。
  - **如果当前持仓为多仓，进行多头出场的判断：**
    - 如果操作类型是 LE（平多），平多仓。
    - 如果当前价格相对于最后一个事件的价格的收益率小于止损阈值，平多仓。
    - 如果当前成交量相对于最后一个事件的成交量的增加量大于超时阈值，平多仓。
  - **如果当前持仓为空仓，进行空头出场的判断：**

        * 如果操作类型是 SE（平空），平空仓。
        * 如果当前价格相对于最后一个事件的价格的收益率小于止损阈值，平空仓。
        * 如果当前成交量相对于最后一个事件的成交量的增加量大于超时阈值，平空仓。
    - 将当前持仓状态和价格记录到持仓列表中。

    Parameters

    s –最新信号字典

    Returns

RawBar

class czsc.**RawBar**(*symbol: str*, *id: int*, *dt: ~datetime.datetime*, *freq: ~czsc.enum.Freq*, *open: float*, *close: float*, *high: float*, *low: float*, *vol: float*, *amount: float*, *cache: dict = <factory>*)

Bases: object

原始 K 线元素

Attributes Summary

lower	下影
solid	实体
upper	上影

Attributes Documentation

lower

下影

solid

实体

upper

上影

RedisWeightsClient

class czsc.**RedisWeightsClient**(*strategy_name*, *redis_url=None*, *connection_pool=None*, *send_heartbeat=True*, ***kwargs*)

Bases: object

策略持仓权重收发客户端

Attributes Summary

heartbeat_time	获取策略的最近一次心跳时间
metadata	获取策略元数据
version	

Methods Summary

clear_all([with_human])	删除该策略所有记录
get_all_weights([sdt, edt])	获取所有权重数据
get_hist_weights(symbol, sdt, edt)	获取单个品种的持仓权重历史数据
get_keys(pattern)	获取 redis 中指定 pattern 的 keys
get_last_times([symbols])	获取所有品种上策略最近一次发布信号的时间
get_last_weights([symbols, ignore_zero, lua])	获取最近的持仓权重
get_symbols()	获取策略交易的品种列表
publish(symbol, dt, weight[, price, ref, ...])	发布单个策略持仓权重
publish_dataframe(df[, overwrite, batch_size])	批量发布多个策略信号
register_lua_publish(client)	
set_metadata(base_freq, description, author, ...)	设置策略元数据
update_last(**kwargs)	设置策略最近一次更新时间，以及更新参数【可选】

Attributes Documentation

heartbeat_time

获取策略的最近一次心跳时间

metadata

获取策略元数据

version = 'V240303'

Methods Documentation

clear_all(*with_human=True*)

删除该策略所有记录

get_all_weights(*sdt=None*, *edt=None*, ***kwargs*) → DataFrame

获取所有权重数据

Parameters

- **sdt** –str, 开始时间, eg: 20210924 10:19:00
- **edt** –str, 结束时间, eg: 20220924 10:19:00

Returns

pd.DataFrame

get_hist_weights(*symbol*, *sdt*, *edt*) → DataFrame

获取单个品种的持仓权重历史数据

Parameters

- **symbol** –str, 品种代码
- **sdt** –str, 开始时间, eg: 20210924 10:19:00
- **edt** –str, 结束时间, eg: 20220924 10:19:00

Returns

pd.DataFrame

get_keys(*pattern*) → list

获取 redis 中指定 pattern 的 keys

get_last_times(*symbols=None*)

获取所有品种上策略最近一次发布信号的时间

Parameters

symbols –list, 品种列表, 默认为 None, 即获取所有品种

Returns

dict, {symbol: datetime}，如 {‘SFIF9001’: datetime(2021, 9, 24, 15, 19, 0)}

get_last_weights(*symbols=None*, *ignore_zero=True*, *lua=True*)

获取最近的持仓权重

Parameters

- **symbols** –list, 品种列表
- **ignore_zero** –boolean, 是否忽略权重为 0 的品种
- **lua** –boolean, 是否使用 lua 脚本获取，默认为 True 如果要全量获取，推荐使用 lua 脚本，速度更快；如果要获取指定 symbols，不推荐使用 lua 脚本。

Returns

pd.DataFrame

get_symbols()

获取策略交易的品种列表

publish(*symbol*, *dt*, *weight*, *price=0*, *ref=None*, *overwrite=False*)

发布单个策略持仓权重

Parameters

- **symbol** –str, eg; SFIF9001
- **dt** –py_datetime or pandas Timestamp
- **weight** –float, 信号值
- **price** –float, 产生信号时的价格
- **ref** –dict, 自定义数据
- **overwrite** –boolean, 是否覆盖已有记录

Returns

成功发布信号的条数

publish_dataframe(*df*, *overwrite=False*, *batch_size=10000*)

批量发布多个策略信号

Parameters

- **df** –pandas.DataFrame, 必需包含 [‘symbol’, ‘dt’, ‘weight’] 列, 可选 [‘price’, ‘ref’] 列, 如没有 price 则写 0, dtype 同 publish 方法
- **overwrite** –boolean, 是否覆盖已有记录

Returns

成功发布信号的条数

static register_lua_publish(*client*)

set_metadata(*base_freq*, *description*, *author*, *outsample_sdt*, ***kwargs*)

设置策略元数据

update_last(***kwargs*)

设置策略最近一次更新时间，以及更新参数【可选】

Signal

class czsc.**Signal**(*signal: str = ''*, *score: int = 0*, *k1: str = '任意'*, *k2: str = '任意'*, *k3: str = '任意'*, *v1: str = '任意'*, *v2: str = '任意'*, *v3: str = '任意'*)

Bases: object

Attributes Summary

k1	
k2	
k3	
key	获取信号名称
score	
signal	
v1	
v2	
v3	
value	获取信号值

Methods Summary

is_match(s)	判断信号是否与信号列表中的值匹配

Attributes Documentation

k1: str = '任意'

k2: str = '任意'

k3: str = '任意'

key

获取信号名称

score: int = 0

signal: str = ''

v1: str = '任意'

v2: str = '任意'

v3: str = '任意'

value

获取信号值

Methods Documentation

is_match(*s: dict*) → bool

判断信号是否与信号列表中的值匹配

代码的执行逻辑如下：

接收一个字典 s 作为参数，该字典包含了所有信号的信息。从字典 s 中获取名称为 key 的信号的值 v。如果 v 不存在，则抛出异常。从信号的值 v 中解析出 v1、v2、v3 和 score 四个变量。

如果当前信号的得分 score 大于等于目标信号的得分 self.score，则继续执行，否则返回 False。如果当前信号的第一个值 v1 等于目标信号的第一个值 self.v1 或者目标信号的第一个值为“任意”，则继续执行，否则返回 False。如果当前信号的第二个值 v2 等于目标信号的第二个值 self.v2 或者目标信号的第二个值为“任意”，则继续执行，否则返回 False。如果当前信号的第三个值 v3 等于目标信号的第三个值 self.v3 或者目标信号的第三个值为“任意”，则返回 True，否则返回 False。

Parameters

s –所有信号字典

Returns

bool

SignalAnalyzer

class czsc.**SignalAnalyzer**(**args*, ***kwargs*)

Bases: object

Methods Summary

execute([max_workers])	执行信号分析
find_valuable_signals(dfp)	根据信号表现，找出表现好的信号
generate_symbol_signals(symbol)	

Methods Documentation

execute(*max_workers=10*)

执行信号分析

static find_valuable_signals(*dfp*)

根据信号表现，找出表现好的信号

Parameters

dfp –信号表现分析结果

Returns

表现好的信号

generate_symbol_signals(*symbol*)

SignalPerformance

class czsc.**SignalPerformance**(**args*, ***kwargs*)

Bases: object

信号表现分析

Methods Summary

analyze([mode])	分析信号出现前后的收益情况
report([file_xlsx])	

Methods Documentation

analyze(*mode='0b'*) → DataFrame

分析信号出现前后的收益情况

Parameters

mode –分析模式，0b 截面向前看 0n 截面向后看 1b 时序向前看 1n 时序向后看

Returns

report(*file_xlsx=None*)

SignalsParser

class czsc.**SignalsParser**(*signals_module: str = 'czsc.signals'*)

Bases: object

解析一串信号，生成信号函数配置

Methods Summary

config_to_keys(config)	将信号函数配置转换为信号 key 列表
get_function_name(signal)	获取信号对应的信号函数名称
parse(signal_seq)	解析信号序列
parse_params(name, signal)	获取信号函数参数

Methods Documentation

config_to_keys(*config: List[Dict]*)

将信号函数配置转换为信号 key 列表

函数执行逻辑：

1. 首先创建了一个空列表 keys 用于存储信号 key。
2. **对于传入的 config 列表中的每个配置字典 conf 进行以下操作：**

- 获取信号函数的名称。
- 如果该信号函数的名称在 self.sig_pats_map 中存在对应的模板，使用参数填充模板，并将结果添加到 keys 列表中。

Parameters

config –信号函数配置

config = [{‘freq’：‘日线’，‘max_overlap’：‘3’，‘name’：‘czsc.signals.cxt_bi_end_V230222’},
{‘freq1’：‘日线’，‘freq2’：‘60分钟’，‘name’：‘czsc.signals.cxt_zhong_shu_gong_zhen_V221221’}]

Returns

信号 key 列表

get_function_name(*signal: str*)

获取信号对应的信号函数名称

函数执行逻辑：

1. 创建一个 _signal 对象，通过传入的信号字符串进行初始化。
2. 通过遍历 sig_name_map 中的项目，找出那些与 _signal.k3 相匹配的键，并将它们存储在 _k3_match 列表中。
3. 如果只有一个匹配项，则返回该项；否则记录错误日志并返回 None。

Parameters

signal –信号，数据样例：15 分钟 _D1K_ 量柱 V221218_ 低量柱 _6K_ 任意 _0

Returns

信号函数名称

parse(*signal_seq: List[str]*)

解析信号序列

函数执行逻辑：

1. 接受一个 signal_seq 参数。
2. 定义一个空列表 res ，用于存储解析结果。
3. 遍历信号序列 signal_seq 中的每一个信号：
   - 调用 get_function_name 方法，以信号为参数，获取该信号对应的函数名。
   - 进行函数名存在性判断，name 在 sig_pats_map 中存在，调用 parse_params 方法，以函数名和信号为参数，解析参数并返回结果。

Parameters

signal_seq –信号序列, 样例：[‘15 分钟 _D1K_ 量柱 V221218_ 低量柱 _6K_ 任意 _0’，‘日线 _D1K_ 量柱 V221218_ 低量柱 _6K_ 任意 _0’]

Returns

信号函数配置

parse_params(*name*, *signal*)

获取信号函数参数

函数执行逻辑：

1. 首先根据传入的 name 和 signal 参数，通过 Signal(signal).key 获取一个键值。
2. 然后从实例变量 sig_pats_map 中获取与指定名称对应的参数模板，并将其存储在 pats 中。
3. 如果没有找到参数模板，则返回 None。
4. 最后将信号函数的完整名称存储在参数字典中，并返回参数字典。

Parameters

- **name** –信号函数名称, 如：cxt_bi_end_V230222
- **signal** –需要解析的信号, 如：15 分钟 _D1K_ 量柱 V221218_ 低量柱 _6K_ 任意 _0

Returns

WeightBacktest

class czsc.**WeightBacktest**(*dfw*, *digits=2*, ***kwargs*)

Bases: object

持仓权重回测

飞书文档：https://s0cqcxuy3p.feishu.cn/wiki/Pf1fw1woQi4iJikbKJmcYToznxb

Attributes Summary

daily_return	品种等权费后日收益率
stats	回测绩效评价
version	

Methods Summary

backtest([n_jobs])	回测所有合约的收益率
get_symbol_daily(symbol)	获取某个合约的每日收益率
get_symbol_pairs(symbol)	获取某个合约的开平交易记录
process_symbol(symbol)	处理某个合约的回测数据
report(res_path)	回测报告

Attributes Documentation

daily_return

品种等权费后日收益率

stats

回测绩效评价

version = 'V231126'

Methods Documentation

backtest(*n_jobs=1*)

回测所有合约的收益率

函数计算逻辑：

1. 获取数据：遍历所有合约，调用 get_symbol_daily 方法获取每个合约的日收益，调用 get_symbol_pairs 方法获取每个合约的交易流水。
2. **数据处理：将每个合约的日收益合并为一个 DataFrame，使用 pd.pivot_table 方法将数据重塑为以日期为索引、合约为列、**
   收益率为值的表格，并将缺失值填充为 0。计算所有合约收益率的平均值，并将该列添加到 DataFrame 中。将结果存储在 res 字典中，键为合约名，值为包含日行情数据和交易对数据的字典。
3. **绩效评价：计算回测结果的开始日期和结束日期，调用 daily_performance 方法评估总收益率的绩效指标。将每个合约的交易对数据**
   合并为一个 DataFrame，调用 evaluate_pairs 方法评估交易对的绩效指标。将结果存储在 stats 字典中，并更新到绩效评价的字典中。
4. 返回结果：将合约的等权日收益数据和绩效评价结果存储在 res 字典中，并将该字典作为函数的返回结果。

get_symbol_daily(*symbol*)

获取某个合约的每日收益率

函数计算逻辑：

1. 从实例变量 self.dfw 中筛选出交易标的为 symbol 的数据，并复制到新的 DataFrame dfs。
2. 计算每条数据的收益（edge）：权重乘以下一条数据的价格除以当前价格减 1。
3. 计算每条数据的手续费（cost）：当前权重与前一条数据权重之差的绝对值乘以实例变量 self.fee_rate。
4. 计算每条数据扣除手续费后的收益（edge_post_fee）：收益减去手续费。
5. 根据日期进行分组，并对每组进行求和操作，得到每日的总收益、总扣除手续费后的收益和总手续费。
6. 重置索引，并将交易标的符号添加到 DataFrame 中。
7. 重命名列名，将’edge_post_fee’列改为’return’，将’dt’列改为’date’。
8. 选择需要的列，并返回包含日期、交易标的、收益、扣除手续费后的收益和手续费的 DataFrame。

Parameters

symbol –str，合约代码

Returns

pd.DataFrame，品种每日收益率，

columns = [‘date’，‘symbol’，‘edge’，‘return’，‘cost’] 其中

date 为交易日，symbol 为合约代码，edge 为每日收益率，return 为每日收益率减去交易成本后的真实收益，cost 为交易成本

数据样例如下：

date	symbol	edge	return	cost
2019-01-02	DLi9001	0.00230261	0.00195919	0.00085
2019-01-03	DLi9001	0.00425589	0.00310589	0.00115
2019-01-04	DLi9001	-0.0014209	-0.0024709	0.00105
2019-01-07	DLi9001	0.000988305	-0.000111695	0.0011
2019-01-08	DLi9001	-0.0004743	-0.0016243	0.00115

get_symbol_pairs(*symbol*)

获取某个合约的开平交易记录

函数计算逻辑：

1. 从实例变量 self.dfw 中筛选出交易标的为 symbol 的数据，并复制到新的 DataFrame dfs。
2. 将权重乘以 10 的 self.digits 次方，并转换为整数类型，作为 volume 列的值。
3. 生成 bar_id 列，从 1 开始递增，与行数对应。

4. 创建一个空列表 operates，用于存储开平仓交易记录。
5. 定义内部函数 __add_operate，用于向 operates 列表中添加开平仓交易记录。函数接受日期 dt、bar_id、交易量 volume、价格 price 和操作类型 operate 作为参数。函数根据交易量的绝对值循环添加交易记录到 operates 列表中。
6. 将 dfs 转换为字典列表 rows。
7. 处理第一个行记录。- 如果 volume 大于 0，则调用 __add_operate 函数添加”开多”操作的交易记录。- 如果 volume 小于 0，则调用 __add_operate 函数添加”开空”操作的交易记录。
8. 处理后续的行记录。- 使用 zip 函数遍历 rows[:-1] 和 rows[1:]，同时获取当前行 row1 和下一行 row2。- 根据 volume 的正负和变化情况，调用 __add_operate 函数添加对应的开平仓交易记录。
9. 创建空列表 pairs 和 opens，用于存储交易对和开仓记录。
10. 遍历 operates 列表中的交易记录。- 如果操作类型为”开多”或”开空”，将交易记录添加到 opens 列表中，并继续下一次循环。- 如果操作类型为”平多”或”平空”，将对应的开仓记录从 opens 列表中弹出。

    根据开仓和平仓的价格计算盈亏比例，并创建一个交易对字典，将其添加到 pairs 列表中。
11. 将 pairs 列表转换为 DataFrame，并返回包含交易标的的开平仓交易记录的 DataFrame。

process_symbol(*symbol*)

处理某个合约的回测数据

report(*res_path*)

回测报告

WordWriter

class czsc.**WordWriter**(*file_docx=None*)

Bases: object

用 Word 文档记录信息

Methods Summary

add_df_table(df[, style])	添加数据表
add_heading(text[, level])	
add_page_break()	添加分页符
add_paragraph(text[, style, bold, ...])	新增段落
add_picture(file[, width, height, alignment])	写入图片到文档中
add_title(text)	
save([file_docx])	保存结果到文件

Methods Documentation

add_df_table(*df: DataFrame*, *style='Table Grid'*, ***kwargs*)

添加数据表

https://www.jianshu.com/p/93e0df92cf16

Parameters

- **df** –数据表
- **style** –表格样式

Returns

add_heading(*text*, *level=1*)

add_page_break()

添加分页符

add_paragraph(*text*, *style=None*, *bold=False*, *first_line_indent=0.74*)

新增段落

Parameters

- **text** –文本
- **style** –段落样式
- **bold** –是否加粗
- **first_line_indent** –首行缩进，0.74 表示两个空格

Returns

add_picture(*file*, *width=None*, *height=None*, *alignment='center'*) → None

写入图片到文档中

Parameters

- **file** –图片文件路径
- **width** –图片宽度，默认单位 cm
- **height** –图片高度，默认单位 cm
- **alignment** –图片对齐，默认 center

Returns

add_title(*text*)

save(*file_docx=None*)

保存结果到文件

ZS

class czsc.**ZS**(*bis: ~typing.List[~czsc.objects.BI], cache: dict = <factory>*)

Bases: object

中枢对象，主要用于辅助信号函数计算

Attributes Summary

dd	中枢最低点
edir	中枢倒一笔方向，edir 是 end direction 的缩写
edt	中枢结束时间
gg	中枢最高点
is_valid	中枢是否有效
sdir	中枢第一笔方向，sdir 是 start direction 的缩写
sdt	中枢开始时间
zd	中枢下沿
zg	中枢上沿
zz	中枢中轴

Attributes Documentation

dd

中枢最低点

edir

中枢倒一笔方向，edir 是 end direction 的缩写

edt

中枢结束时间

gg

中枢最高点

is_valid

中枢是否有效

sdir

中枢第一笔方向，sdir 是 start direction 的缩写

sdt

中枢开始时间

zd

中枢下沿

zg

中枢上沿

zz

中枢中轴

2.3 czsc.analyze Module

2.3.1 Functions

check_bi(bars[, benchmark])	输入一串无包含关系 K 线，查找其中的一笔
check_fx(k1, k2, k3)	查找分型
check_fxs(bars)	输入一串无包含关系 K 线，查找其中所有分型
kline_pro(kline[, fx, bi, xd, bs, title, ...])	绘制缠中说禅 K 线分析结果
remove_include(k1, k2, k3)	去除包含关系：输入三根 k 线，其中 k1 和 k2 为没有包含关系的 K 线，k3 为原始 K 线

check_bi

czsc.analyze.**check_bi**(*bars: List[NewBar]*, *benchmark=None*)

输入一串无包含关系 K 线，查找其中的一笔

Parameters

- **bars** –无包含关系 K 线列表
- **benchmark** –当下笔能量的比较基准

Returns

check_fx

czsc.analyze.**check_fx**(*k1: NewBar*, *k2: NewBar*, *k3: NewBar*)

查找分型

函数计算逻辑：

1. 如果第二个 `NewBar`对象的最高价和最低价都高于第一个和第三个 `NewBar`对象的对应价格，那么它被认为是顶分型（G）。在这种情况下，函数会创建一个新的 `FX`对象，其标记为 `Mark.G`，并将其赋值给 `fx`。
2. 如果第二个 `NewBar`对象的最高价和最低价都低于第一个和第三个 `NewBar`对象的对应价格，那么它被认为是底分型（D）。在这种情况下，函数会创建一个新的 `FX`对象，其标记为 `Mark.D`，并将其赋值给 `fx`。
3. 函数最后返回 `fx`，如果没有找到分型，*fx*`将为 `*None*。

Parameters

- **k1** –第一个 `NewBar`对象
- **k2** –第二个 `NewBar`对象
- **k3** –第三个 `NewBar`对象

Returns

FX`对象或 `*None*

check_fxs

czsc.analyze.**check_fxs**(*bars: List[NewBar]*) → List[*FX*]

输入一串无包含关系 K 线，查找其中所有分型

函数的主要步骤：

1. 创建一个空列表 `fxs`用于存储找到的分型。

2. 遍历 `bars`列表中的每个元素（除了第一个和最后一个），并对每三个连续的 `NewBar`对象调用 `check_fx`函数。
3. 如果 `check_fx`函数返回一个 `FX`对象，检查它的标记是否与 `fxs`列表中最后一个 `FX`对象的标记相同。如果相同，记录一个错误日志。如果不同，将这个 `FX`对象添加到 `fxs`列表中。
4. 最后返回 `fxs`列表，它包含了 `bars`列表中所有找到的分型。

这个函数的主要目的是找出 `bars`列表中所有的顶分型和底分型，并确保它们是交替出现的。如果发现连续的两个分型标记相同，它会记录一个错误日志。

Parameters

bars –无包含关系 K 线列表

Returns

分型列表

remove_include

czsc.analyze.**remove_include**(*k1:* NewBar, *k2:* NewBar, *k3:* RawBar)

去除包含关系：输入三根 k 线，其中 k1 和 k2 为没有包含关系的 K 线，k3 为原始 K 线

处理逻辑如下：

1. 首先，通过比较 k1 和 k2 的高点 (high) 的大小关系来确定 direction 的值。如果 k1 的高点小于 k2 的高点，则设定 direction 为 Up；如果 k1 的高点大于 k2 的高点，则设定 direction 为 Down；如果 k1 和 k2 的高点相等，则创建一个新的 K 线 k4，与 k3 具有相同的属性，并返回 False 和 k4。
2. **接下来，判断 k2 和 k3 之间是否存在包含关系。如果存在，则根据 direction 的值进行处理。**
   - 如果 direction 为 Up，则选择 k2 和 k3 中的较大高点作为新 K 线 k4 的高点，较大低点作为低点，较大高点所在的时间戳 (dt) 作为 k4 的时间戳。
   - 如果 direction 为 Down，则选择 k2 和 k3 中的较小高点作为新 K 线 k4 的高点，较小低点作为低点，较小低点所在的时间戳 (dt) 作为 k4 的时间戳。
   - 如果 direction 的值不是 Up 也不是 Down，则抛出 ValueError 异常。
3. 根据上述处理得到的高点、低点、开盘价 (open_)、收盘价 (close)，计算新 K 线 k4 的成交量 (vol) 和成交金额 (amount)，并将 k2 中除了与 k3 时间戳相同的元素之外的其他元素与 k3 一起作为 k4 的元素列表 (elements)。
4. 返回一个布尔值和新的 K 线 k4。如果 k2 和 k3 之间存在包含关系，则返回 True 和 k4；否则返回 False 和 k4，其中 k4 与 k3 具有相同的属性。

2.3.2 Classes

BI(symbol, fx_a, fx_b, fxs, direction, bars, ...)	
CZSC(bars[, get_signals, max_bi_num])	
Direction(value)	An enumeration.
FX(symbol, dt, mark, high, low, fx, ...)	
Mark(value)	An enumeration.
NewBar(symbol, id, dt, freq, open, close, ...)	去除包含关系后的 K 线元素
OrderedDict	Dictionary that remembers insertion order
RawBar(symbol, id, dt, freq, open, close, ...)	原始 K 线元素

CZSC

class czsc.analyze.**CZSC**(*bars: List[*RawBar*]*, *get_signals=None*, *max_bi_num=50*)

Bases: object

Attributes Summary

finished_bis	已完成的笔
fx_list	分型列表，包括 bars_ubi 中的分型
last_bi_extend	判断最后一笔是否在延伸中，True 表示延伸中
ubi	Unfinished Bi，未完成的笔
ubi_fxs	bars_ubi 中的分型

Methods Summary

open_in_browser([width, height])	直接在浏览器中打开分析结果
to_echarts([width, height, bs])	绘制 K 线分析图
to_plotly()	使用 plotly 绘制 K 线分析图
update(bar)	更新分析结果

Attributes Documentation

finished_bis

已完成的笔

fx_list

分型列表，包括 bars_ubi 中的分型

last_bi_extend

判断最后一笔是否在延伸中，True 表示延伸中

ubi

Unfinished Bi，未完成的笔

ubi_fxs

bars_ubi 中的分型

Methods Documentation

open_in_browser(*width: str = '1400px'*, *height: str = '580px'*)

直接在浏览器中打开分析结果

Parameters

- **width** –图表宽度
- **height** –图表高度

Returns

to_echarts(*width: str = '1400px'*, *height: str = '580px'*, *bs=[]*)

绘制 K 线分析图

Parameters

- **width** –宽
- **height** –高
- **bs** –交易标记，默认为空

Returns

to_plotly()

使用 plotly 绘制 K 线分析图

update(*bar:* RawBar)

更新分析结果

Parameters

bar –单根 K 线对象

2.4 czsc.signals Package

2.4.1 Functions

adtm_up_dw_line_V230603(c, **kwargs)	ADTM 能量异动，贡献者：琅盎
amv_up_dw_line_V230603(c, **kwargs)	AMV 能量异动，贡献者：琅盎
asi_up_dw_line_V230603(c, **kwargs)	ASI 多空分类，贡献者：琅盎
bar_accelerate_V221110(c, **kwargs)	辨别加速走势
bar_accelerate_V221118(c, **kwargs)	辨别加速走势
bar_accelerate_V240428(c, **kwargs)	辨别加速走势
bar_amount_acc_V230214(c, **kwargs)	N 根 K 线总成交额
bar_big_solid_V230215(c, **kwargs)	窗口内最大实体 K 线的中间价区分多空
bar_bpm_V230227(c, **kwargs)	以 BP 为单位的绝对动量
bar_break_V240428(c, **kwargs)	极值突破
bar_cross_ps_V221112(c, **kwargs)	倒数第 di 根 K 线穿越支撑、压力位的数量【慎用，非常耗时】
bar_dual_thrust_V230403(c, **kwargs)	Dual Thrust 通道突破
bar_eight_V230702(c, **kwargs)	8K 走势分类
bar_end_V221211(c[, freq1])	判断分钟 K 线是否结束
bar_fake_break_V230204(c, **kwargs)	假突破
bar_fang_liang_break_V221216(c, **kwargs)	放量向上突破并回踩指定均线，贡献者：琅盎
bar_limit_down_V230525(c, **kwargs)	跌停后出现无下影线长实体阳线做多
bar_mean_amount_V221112(c, **kwargs)	截取一段时间内的平均成交金额分类信号
bar_operate_span_V221111(c, **kwargs)	日内操作时间区间，c 必须是基础周期的 CZSC 对象
bar_plr_V240427(c, **kwargs)	盈亏比计算
bar_polyfit_V240428(c, **kwargs)	一阶、二阶多项式拟合
bar_r_breaker_V230326(c, **kwargs)	RBreaker 日内回转交易
bar_reversal_V230227(c, **kwargs)	判断最近一根 K 线是否具有反转迹象
bar_section_momentum_V221112(c, **kwargs)	获取某个区间（固定 K 线数量）的动量强弱
bar_shuang_fei_V230507(c, **kwargs)	双飞涨停，贡献者：琅盎
bar_single_V230214(c, **kwargs)	单根 K 线的状态
bar_single_V230506(c, **kwargs)	单 K 趋势因子辅助判断买卖点
bar_time_V230327(c, **kwargs)	K 线日内时间分段信号
bar_tnr_V230629(c, **kwargs)	趋势噪音指标（TNR，Trend to Noise Rate）分层
bar_tnr_V230630(c, **kwargs)	趋势噪音指标（TNR，Trend to Noise Rate）
bar_trend_V240209(c, **kwargs)	趋势跟踪信号
bar_triple_V230506(c, **kwargs)	三 K 加速形态配合成交量变化
bar_vol_bs1_V230224(c, **kwargs)	量价配合的高低点判断
bar_vol_grow_V221112(c, **kwargs)	倒数第 i 根 K 线的成交量相比于前 N 根 K 线放量

continues on next page

Table 3 – continued from previous page

bar_weekday_V230328(c, **kwargs)	K 线周内时间分段信号
bar_window_ps_V230731(c, **kwargs)	指定窗口内支撑压力位分位数计算，贡献者：chenlei
bar_window_ps_V230801(c, **kwargs)	指定窗口内支撑压力位分位数计算
bar_window_std_V230731(c, **kwargs)	指定窗口内波动率的特征
bar_zdf_V221203(c, **kwargs)	单根 K 线的涨跌幅区间
bar_zdt_V230331(c, **kwargs)	计算倒数第 di 根 K 线的涨跌停信息
bar_zt_count_V230504(c, **kwargs)	窗口内涨停计数
bias_up_dw_line_V230618(c, **kwargs)	BIAS 乖离率指标，贡献者：琅盎
byi_bi_end_V230106(c, **kwargs)	白仪分型停顿辅助笔结束判断
byi_bi_end_V230107(c, **kwargs)	白仪验证分型辅助判断笔结束
byi_fx_num_V230628(c, **kwargs)	白仪前面下跌或上涨一笔次级别笔结构数量满足条件；贡献者：谌意勇
byi_second_bs_V230324(c, **kwargs)	白仪二类买卖点辅助 V230324
byi_symmetry_zs_V221107(c, **kwargs)	对称中枢信号
cat_macd_V230518(cat, **kwargs)	freq1 与 freq2 联立信号，freq1 > freq2
cat_macd_V230520(cat, **kwargs)	freq1 与 freq2 联立信号，freq1 > freq2
clv_up_dw_line_V230605(c, **kwargs)	CLV 多空分类，贡献者：琅盎
cmo_up_dw_line_V230605(c, **kwargs)	CMO 能量异动，贡献者：琅盎
coo_cci_V230323(c, **kwargs)	CCI 结合均线的多空信号
coo_kdj_V230322(c, **kwargs)	均线判定方向，KD 决定进场时机
coo_sar_V230325(c, **kwargs)	SAR 和高低点结合判断买卖时机
coo_td_V221110(c, **kwargs)	获取倒数第 i 根 K 线的 TD 信号
coo_td_V221111(c, **kwargs)	获取倒数第 i 根 K 线的 TD 信号
cvolp_up_dw_line_V230612(c, **kwargs)	CVOLP 动量变化率指标，贡献者：琅盎
cxt_bi_base_V230228(c, **kwargs)	BI 基础信号
cxt_bi_end_V230104(c, **kwargs)	单均线辅助判断笔结束
cxt_bi_end_V230105(c, **kwargs)	K 线形态 + 均线辅助判断笔结束
cxt_bi_end_V230222(c, **kwargs)	当前是最后笔的第几次新低底分型或新高顶分型，用于笔结束辅助
cxt_bi_end_V230224(c, **kwargs)	量价配合的笔结束辅助
cxt_bi_end_V230312(c, **kwargs)	MACD 辅助判断笔结束信号
cxt_bi_end_V230320(c, **kwargs)	100 以内质数时序窗口辅助笔结束判断
cxt_bi_end_V230322(c, **kwargs)	分型配合均线辅助判断笔的结束
cxt_bi_end_V230324(c, **kwargs)	笔结束分型的均线突破判断笔的结束
cxt_bi_end_V230618(c, **kwargs)	笔结束辅助判断
cxt_bi_end_V230815(c, **kwargs)	一两根 K 线快速突破反向笔
cxt_bi_status_V230101(c, **kwargs)	笔的表里关系
cxt_bi_status_V230102(c, **kwargs)	笔的表里关系
cxt_bi_stop_V230815(c, **kwargs)	定位笔的止损距离大小

continues on next page

Table 3 – continued from previous page

cxt_bi_trend_V230824(c, **kwargs)	判断 N 笔形态，贡献者：chenglei
cxt_bi_trend_V230913(c, **kwargs)	辅助判断股票通道信号，贡献者：马鸣
cxt_bi_zdf_V230601(c, **kwargs)	BI 涨跌幅的分层判断
cxt_double_zs_V230311(c, **kwargs)	两个中枢组合辅助判断 BS1，贡献者：韩知辰
cxt_eleven_bi_V230622(c, **kwargs)	十一笔形态分类
cxt_first_buy_V221126(c, **kwargs)	一买信号
cxt_first_sell_V221126(c, **kwargs)	一卖信号
cxt_five_bi_V230619(c, **kwargs)	五笔形态分类
cxt_fx_power_V221107(c, **kwargs)	倒数第 di 个分型的强弱
cxt_intraday_V230701(cat, **kwargs)	每日走势分类
cxt_nine_bi_V230621(c, **kwargs)	九笔形态分类
cxt_range_oscillation_V230620(c, **kwargs)	判断区间震荡
cxt_second_bs_V230320(c, **kwargs)	均线辅助识别第二类买卖点
cxt_seven_bi_V230620(c, **kwargs)	七笔形态分类
cxt_third_bs_V230318(c, **kwargs)	均线辅助识别第三类买卖点
cxt_third_bs_V230319(c, **kwargs)	均线辅助识别第三类买卖点，增加均线形态
cxt_third_buy_V230228(c, **kwargs)	笔三买辅助
cxt_three_bi_V230618(c, **kwargs)	三笔形态分类
cxt_ubi_end_V230816(c, **kwargs)	当前是未完成笔的第几次新低或新高，用于笔结束辅助
cxt_zhong_shu_gong_zhen_V221221(cat[, ...])	大小级别中枢共振，类二买共振；贡献者：琅盎
dema_up_dw_line_V230605(c, **kwargs)	DEMA 短线趋势指标，贡献者：琅盎
demakder_up_dw_line_V230605(c, **kwargs)	DEMAKER 价格趋势指标，贡献者：琅盎
emv_up_dw_line_V230605(c, **kwargs)	EMV 简易波动指标，贡献者：琅盎
er_up_dw_line_V230604(c, **kwargs)	ER 价格动量指标，贡献者：琅盎
jcc_ci_tou_V221101(c, **kwargs)	刺透形态
jcc_fan_ji_xian_V221121(c, **kwargs)	反击线；贡献者：lynxluu
jcc_fen_shou_xian_V20221113(c, **kwargs)	分手线：分手形态是一个中继形态；贡献者：琅盎
jcc_gap_yin_yang_V221121(c, **kwargs)	跳空与并列阴阳形态贡献者：平凡
jcc_ping_tou_V221113(c, **kwargs)	平头形态，贡献者：平凡
jcc_san_fa_V20221115(c, **kwargs)	上升 & 下降三法；贡献者：琅盎
jcc_san_fa_V20221118(c, **kwargs)	上升 & 下降三法
jcc_san_szx_V221122(c, **kwargs)	三星形态
jcc_san_xing_xian_V221023(c, **kwargs)	伞形线
jcc_shan_chun_V221121(c, **kwargs)	山川形态，表示三山形态和三川形态
jcc_szx_V221111(c, **kwargs)	十字线
jcc_ta_xing_V221124(c, **kwargs)	塔形顶底
jcc_ten_mo_V221028(c, **kwargs)	吞没形态；贡献者：琅盎

continues on next page

Table 3 – continued from previous page

jcc_three_crow_V221108(c, **kwargs)	三只乌鸦，贡献者：马鸣
jcc_two_crow_V221108(c, **kwargs)	两只乌鸦
jcc_wu_yun_gai_ding_V221101(c, **kwargs)	乌云盖顶，贡献者：魏永超
jcc_xing_xian_V221118(c, **kwargs)	星形态
jcc_yun_xian_V221118(c, **kwargs)	孕线形态
jcc_zhu_huo_xian_V221027(c, **kwargs)	烛火线，贡献者：琅盎
jcc_zhuo_yao_dai_xian_v221113(c, **kwargs)	捉腰带线，贡献者：平凡
kcatr_up_dw_line_V230823(c, **kwargs)	用 ATR 波幅构造上下轨，收盘价突破判断多空贡献者：琅盎
ntmdk_V230824(c, **kwargs)	NTMDK 多空指标，贡献者：琅盎
obv_up_dw_line_V230719(c, **kwargs)	OBV 能量指标，贡献者：琅盎
obvm_line_V230610(c, **kwargs)	OBV 能量指标，贡献者：琅盎
pos_bar_stop_V230524(cat, **kwargs)	按照开仓点附近的 N 根 K 线极值止损
pos_fix_exit_V230624(cat, **kwargs)	固定比例止损，止盈
pos_fx_stop_V230414(cat, **kwargs)	按照开仓点附近的分型止损
pos_holds_V230414(cat, **kwargs)	开仓后 N 根 K 线涨幅小于 M%%，则平仓
pos_holds_V230807(cat, **kwargs)	开仓后 N 根 K 线收益小于 M%%，且当前收益大于 T%%，平仓保本
pos_holds_V240428(cat, **kwargs)	保本单：开仓后最大盈利超过 H 个 BP，且当前收益低于最大盈利的 T%，平仓保本
pos_ma_V230414(cat, **kwargs)	判断开仓后是否升破 MA 均线或跌破 MA 均线
pos_profit_loss_V230624(cat, **kwargs)	开仓后盈亏比达到一定比值，才允许平仓贡献者：谌意勇
pos_status_V230808(cat, **kwargs)	Position 策略的持仓状态
pos_stop_V240428(cat, **kwargs)	止损单，持有 N 根 K 线后，多头跌破前低或空头升破前高，平仓
pos_take_V240428(cat, **kwargs)	止盈单，持有 N 根 K 线后，多头持仓期间出现 T 根倍量阳线或空头持仓期间出现 T 根倍量阴线，平仓
pressure_support_V240222(c, **kwargs)	支撑压力线辅助 V240222
pressure_support_V240402(c, **kwargs)	支撑压力线辅助 V240402
pressure_support_V240406(c, **kwargs)	支撑压力线辅助 V240406
skdj_up_dw_line_V230611(c, **kwargs)	SKDJ 随机波动指标，贡献者：琅盎
tas_accelerate_V230531(c, **kwargs)	BOLL 辅助判断加速行情
tas_angle_V230802(c, **kwargs)	笔的角度比较贡献者：谌意勇
tas_atr_V230630(c, **kwargs)	ATR 波动强弱
tas_atr_break_V230424(c, **kwargs)	ATR 突破
tas_boll_bc_V221118(c, **kwargs)	BOLL 背驰辅助
tas_boll_cc_V230312(c, **kwargs)	多空进出场信号，贡献者：琅盎

continues on next page

Table 3 – continued from previous page

tas_boll_power_V221112(c, **kwargs)	BOLL 指标强弱
tas_boll_vt_V230212(c, **kwargs)	以 BOLL 通道为依据的多空进出场信号
tas_cci_base_V230402(c, **kwargs)	CCI 基础信号
tas_cross_status_V230619(c, **kwargs)	0 轴上下金死叉次数计算信号函数贡献者：谌意勇
tas_cross_status_V230624(c, **kwargs)	指定金死叉数值信号函数, 以此来确定 MACD 交易区间贡献者：谌意勇
tas_cross_status_V230625(c, **kwargs)	指定金死叉数值信号函数, 以此来确定 MACD 交易区间贡献者：谌意勇
tas_double_ma_V221203(c, **kwargs)	双均线多空和强弱信号
tas_double_ma_V230511(c, **kwargs)	双均线金叉死叉后的反向信号
tas_double_ma_V240208(c, **kwargs)	双均线多空信号，辅助 V240208
tas_first_bs_V230217(c, **kwargs)	均线结合 K 线形态的一买一卖辅助判断
tas_hlma_V230301(c, **kwargs)	HMA 多空信号，贡献者：琅盎
tas_kdj_base_V221101(c, **kwargs)	KDJ 金叉死叉信号
tas_kdj_evc_V221201(c, **kwargs)	KDJ 极值计数信号, evc 是 extreme value counts 的首字母缩写
tas_kdj_evc_V230401(c, **kwargs)	KDJ 极值计数信号, evc 是 extreme value counts 的首字母缩写
tas_low_trend_V230627(c, **kwargs)	阴跌趋势、小阳趋势
tas_ma_base_V221101(c, **kwargs)	MA 多空和方向信号
tas_ma_base_V221203(c, **kwargs)	MA 多空和方向信号，加距离限制
tas_ma_base_V230313(c, **kwargs)	单均线多空和方向辅助开平仓信号
tas_ma_round_V221206(c, **kwargs)	笔端点在均线附近，贡献者：谌意勇
tas_ma_system_V230513(c, **kwargs)	均线系统多空排列
tas_macd_base_V221028(c, **kwargs)	MACD|DIF|DEA 多空和方向信号
tas_macd_base_V230320(c, **kwargs)	MACD|DIF|DEA 多空和方向信号，支持 max_overlap 参数
tas_macd_bc_V221201(c, **kwargs)	MACD 背驰辅助
tas_macd_bc_V230803(c, **kwargs)	MACD 辅助背驰判断
tas_macd_bc_V230804(c, **kwargs)	MACD 黄白线辅助背驰判断
tas_macd_bc_V240307(c, **kwargs)	MACD 柱子辅助背驰判断
tas_macd_bc_ubi_V230804(c, **kwargs)	未完成笔 MACD 黄白线辅助背驰判断
tas_macd_bs1_V230312(c, **kwargs)	MACD 辅助一买一卖信号
tas_macd_bs1_V230313(c, **kwargs)	MACD 红绿柱判断第一买卖点，贡献者：琅盎
tas_macd_bs1_V230411(c, **kwargs)	基于 MACD DIF 的笔背驰判断信号
tas_macd_bs1_V230412(c, **kwargs)	基于 MACD DIF 的笔背驰判断信号
tas_macd_change_V221105(c, **kwargs)	MACD 颜色变化；贡献者：马鸣
tas_macd_direct_V221106(c, **kwargs)	MACD 方向；贡献者：马鸣
tas_macd_dist_V230408(c, **kwargs)	DIF/DEA/MACD 分层信号辅助判断买卖点

continues on next page

Table 3 – continued from previous page

tas_macd_dist_V230409(c, **kwargs)	DIF/DEA/MACD 远离零轴辅助判断买卖点
tas_macd_dist_V230410(c, **kwargs)	DIF/DEA/MACD 分层信号辅助判断买卖点
tas_macd_first_bs_V221201(c, **kwargs)	MACD 金叉死叉判断第一买卖点
tas_macd_first_bs_V221216(c, **kwargs)	MACD 金叉死叉判断第一买卖点
tas_macd_power_V221108(c, **kwargs)	MACD 强弱
tas_macd_second_bs_V221201(c, **kwargs)	MACD 金叉死叉判断第二买卖点
tas_macd_xt_V221208(c, **kwargs)	MACD 形态信号
tas_rsi_base_V230227(c, **kwargs)	RSI 超买超卖信号
tas_rumi_V230704(c, **kwargs)	对均线偏离度平滑处理, 通过平滑处理的方式降低 DIFF 的敏感度来解决均线缠绕的问题贡献者：谌意勇
tas_sar_base_V230425(c, **kwargs)	SAR 基础信号
tas_second_bs_V230228(c, **kwargs)	均线结合 K 线形态的第二买卖点辅助判断
tas_second_bs_V230303(c, **kwargs)	利用笔和均线辅助二买信号生成
tas_slope_V231019(c, **kwargs)	DIF 趋势线斜率判断多空
update_atr_cache(c, **kwargs)	更新 ATR 缓存
update_boll_cache(c, **kwargs)	更新 K 线的 BOLL 缓存
update_cci_cache(c, **kwargs)	更新 CCI 缓存
update_kdj_cache(c, **kwargs)	更新 KDJ 缓存
update_ma_cache(c, **kwargs)	更新均线缓存
update_macd_cache(c, **kwargs)	更新 MACD 缓存
update_rsi_cache(c, **kwargs)	更新 RSI 缓存
update_sar_cache(c, **kwargs)	更新 SAR 缓存
vol_double_ma_V230214(c, **kwargs)	成交量双均线信号
vol_gao_di_V221218(c, **kwargs)	高量柱 & 低量柱 & 高量黄金柱，贡献者：琅盎
vol_single_ma_V230214(c, **kwargs)	均线辅助识别第三类买卖点，增加均线形态
vol_ti_suo_V221216(c, **kwargs)	梯量/缩量柱：顺势与逆势工具，贡献者：琅盎
vol_window_V230731(c, **kwargs)	指定窗口内成交量的特征
vol_window_V230801(c, **kwargs)	指定窗口内成交量的特征
xl_bar_basis_V240411(c, **kwargs)	看涨吞没和看跌吞没形态
xl_bar_basis_V240412(c, **kwargs)	长蜡烛形态
xl_bar_position_V240328(c, **kwargs)	相对位置信号; 贡献者：谢磊
xl_bar_trend_V240329(c, **kwargs)	底部反转形态信号; 贡献者：谢磊
xl_bar_trend_V240330(c, **kwargs)	完全分类，均线金叉过滤信号; 贡献者：谢磊
xl_bar_trend_V240331(c, **kwargs)	突破信号; 贡献者：谢磊
zdy_bi_end_V230406(c, **kwargs)	分型停顿判断 K 线结束
zdy_bi_end_V230407(c, **kwargs)	分型停顿判断 K 线结束
zdy_dif_V230527(c, **kwargs)	DIF 远离零轴辅助判断买卖点
zdy_dif_V230528(c, **kwargs)	DIF 远离零轴辅助判断买卖点

continues on next page

Table 3 – continued from previous page

zdy_macd_V230518(c, **kwargs)	MACD 交叉次数
zdy_macd_V230519(c, **kwargs)	MACD 连续缩柱
zdy_macd_V230527(c, **kwargs)	DIF/DEA/MACD 远离零轴辅助判断买卖点
zdy_macd_bc_V230422(c, **kwargs)	MACD 面积背驰
zdy_macd_bs1_V230422(c, **kwargs)	MACD 辅助判断第一类买卖点
zdy_macd_dif_V230516(c, **kwargs)	MACD 柱子与 DIF 的关系
zdy_macd_dif_V230517(c, **kwargs)	MACD 三次开仓条件
zdy_macd_dif_iqr_V230521(c, **kwargs)	MACD 柱子与 DIF 的关系
zdy_stop_loss_V230406(cat, **kwargs)	笔操作止损逻辑
zdy_take_profit_V230406(cat, **kwargs)	笔操作止盈逻辑
zdy_take_profit_V230407(cat, **kwargs)	笔操作止盈逻辑
zdy_vibrate_V230406(cat, freq1, freq2, **kwargs)	中枢震荡短差操作
zdy_zs_V230423(c, **kwargs)	约束中枢的形态和高度
zdy_zs_space_V230421(c, **kwargs)	中枢空间形态约束

adtm_up_dw_line_V230603

czsc.signals.**adtm_up_dw_line_V230603**(*c:* CZSC, ***kwargs*) → OrderedDict

ADTM 能量异动，贡献者：琅盎

参数模板："{freq}_D{di}N{n}M{m}TH{th}_ADTMV230603"

信号逻辑：

1. **如果今天的开盘价大于昨天的开盘价，取最高价 - 开盘价、开盘价 - 昨天的开盘价这二者中最大值，**
   再将取出的最大值求和；反之取 0，形成 up_sum
2. **如果今天的开盘价小于昨天的开盘价，取开盘价 - 最低价、昨天的开盘价 -开盘价这二者中最大值，**
   再将取出的最大值求和；么之取 0，形成 dw_sum
3. 当 up_sum > dw_sum 或最大值的差值之商小于 TH 看多，反之看空

信号列表：

- Signal（'日线 _D1N30M20TH5_ADTMV230603_ 看空 _ 任意 _ 任意 _0'）
- Signal（'日线 _D1N30M20TH5_ADTMV230603_ 看多 _ 任意 _ 任意 _0'）

Parameters

- **c** –CZSC 对象

- **kwargs** –参数字典 - :param di: 信号计算截止倒数第 i 根 K 线 - :param n: 获取 K 线的根数，默认为 30 - :param m: 获取 K 线的根数，默认为 20 - :param th: adtm 阈值，默认为 5，代表 5 / 10 = 0.5

Returns

信号识别结果

amv_up_dw_line_V230603

czsc.signals.**amv_up_dw_line_V230603**(*c:* CZSC, ***kwargs*) → OrderedDict

AMV 能量异动，贡献者：琅盎

参数模板："{freq}_D{di}N{n}M{m}_AMV 能量 V230603"

信号逻辑：

用成交量作为权重对开盘价和收盘价的均值进行加权移动平均。成交量越大的价格对移动平均结果的影响越大，AMV 指标减小了成交量小的价格波动的影响。当短期 AMV 线上穿/下穿长期 AMV 线时，产生买入/卖出信号。

信号列表：

- Signal('日线 _D1N30M120_AMV 能量 V230603_ 看多 _ 任意 _ 任意 _0')
- Signal('日线 _D1N30M120_AMV 能量 V230603_ 看空 _ 任意 _ 任意 _0')

Parameters

- **c** –CZSC 对象
- **kwargs** –参数字典 - :param di: 信号计算截止倒数第 i 根 K 线 - :param n: 获取 K 线的根数，默认为 30 - :param m: 获取 K 线的根数，默认为 20

Returns

信号识别结果

asi_up_dw_line_V230603

czsc.signals.**asi_up_dw_line_V230603**(*c:* CZSC, ***kwargs*) → OrderedDict

ASI 多空分类，贡献者：琅盎

参数模板："{freq}_D{di}N{n}P{p}_ASI 多空 V230603"

信号逻辑：

由于 SI 的波动性比较大，所以我们一般对 SI 累计求和得到 ASI 并捕捉 ASI 的变化趋势。一般我们不会直接看 ASI 的数值（对 SI 累计求和的求和起点不同会导致求出 ASI 的值不同），而是会观察 ASI 的变化方向。我们利用 ASI 与其均线的交叉来产生交易信号，上穿/下穿均线时买入/卖出

信号列表：

- Signal(‘日线 _D1N30P120_ASI 多空 V230603_ 看多 _ 任意 _ 任意 _0’)
- Signal(‘日线 _D1N30P120_ASI 多空 V230603_ 看空 _ 任意 _ 任意 _0’)

Parameters

- **c** –CZSC 对象
- **kwargs** –参数字典 - :param di: 信号计算截止倒数第 i 根 K 线 - :param n: 获取 K 线的根数，默认为 30 - :param p: 获取 K 线的根数，默认为 20

Returns

信号识别结果

bar_accelerate_V221110

czsc.signals.**bar_accelerate_V221110**(*c:* CZSC, ***kwargs*) → OrderedDict

辨别加速走势

参数模板：”{freq}_D{di}W{window}_ 加速 V221110”

信号逻辑：

- 上涨加速：窗口内最后一根 K 线的收盘在窗口区间的 80% 以上；且窗口内阳线数量占比超过 80%
- 下跌加速：窗口内最后一根 K 线的收盘在窗口区间的 20% 以下；且窗口内阴线数量占比超过 80%

信号列表：

- Signal(‘60 分钟 _D1W13_ 加速 V221110_ 上涨 _ 任意 _ 任意 _0’)
- Signal(‘60 分钟 _D1W13_ 加速 V221110_ 下跌 _ 任意 _ 任意 _0’)

Parameters

- **c** –CZSC 对象
- **kwargs** –参数字典 - di: 区间结束 K 线位置，倒数 - window: 取截止 di 的近 window 根 K 线

Returns

信号识别结果

bar_accelerate_V221118

czsc.signals.**bar_accelerate_V221118**(*c:* CZSC, ***kwargs*) → OrderedDict

辨别加速走势

参数模板："{freq}_D{di}W{window}#{ma_type}#{timeperiod}_ 加速 V221118"

信号逻辑：

上涨加速指窗口内 K 线收盘价全部大于 ma1，且 close 与 ma1 的距离不断正向放大；反之为下跌加速。

信号列表：

- Signal('日线 _D1W13#SMA#10_ 加速 V221118_ 上涨 _ 任意 _ 任意 _0')
- Signal('日线 _D1W13#SMA#10_ 加速 V221118_ 下跌 _ 任意 _ 任意 _0')

Parameters

- **c** –CZSC 对象
- **di** –取近 n 根 K 线为截止
- **ma_type** –MA 类型，支持 SMA、EMA、WMA、DEMA、TEMA、TRIMA、KAMA、MAMA、T3
- **timeperiod** –MA 的周期
- **window** –识别加速走势的窗口大小

Returns

信号识别结果

bar_accelerate_V240428

czsc.signals.**bar_accelerate_V240428**(*c:* CZSC, ***kwargs*) → OrderedDict

辨别加速走势

参数模板："{freq}_D{di}W{w}T{t}_ 加速 V240428"

信号逻辑：

以上涨加速为例，计算过程如下：1. 给定窗口大小 w，rolling 计算 w 个周期的 diff 绝对值；2. 如果当前 diff 绝对值大于最近 300 个周期的 diff 绝对值的 75% 分位数，且当前 diff 大于 0，判定为上涨加速；3. 窗口内至少要有 T 根倍量上涨的 K 线。

信号列表：

- Signal('日线 _D1W21T2_ 加速 V240428_ 上涨 _ 任意 _ 任意 _0')
- Signal('日线 _D1W21T2_ 加速 V240428_ 下跌 _ 任意 _ 任意 _0')

Parameters

c –CZSC 对象

Returns

信号识别结果

bar_amount_acc_V230214

czsc.signals.**bar_amount_acc_V230214**(*c:* CZSC, ***kwargs*) → OrderedDict

N 根 K 线总成交额

参数模板："{freq}_D{di}N{n}_ 累计超 {t} 千万"

信号描述：

1. 获取截至倒数第 di 根 K 线的前 n 根 K 线，计算总成交额，如果大于 t 千万，则为是，否则为否

信号列表：

- Signal（'日线 _D2N1_ 累计超 10 千万 _ 是 _ 任意 _ 任意 _0'）

Parameters

- **c** –CZSC 对象
- **di** –倒数第几根 K 线
- **n** –前几根 K 线
- **kwargs** –t: 总成交额阈值

Returns

信号识别结果

bar_big_solid_V230215

czsc.signals.**bar_big_solid_V230215**(*c:* CZSC, ***kwargs*)

窗口内最大实体 K 线的中间价区分多空

参数模板："{freq}_D{di}N{n}_MID"

信号逻辑：

1. 找到窗口内最大实体 K 线, 据其中间位置区分多空

信号列表：

- Signal（'日线 _D1N10_MID_ 看空 _ 大阳 _ 任意 _0'）
- Signal（'日线 _D1N10_MID_ 看空 _ 大阴 _ 任意 _0'）
- Signal（'日线 _D1N10_MID_ 看多 _ 大阴 _ 任意 _0'）

- Signal(‘日线 _D1N10_MID_ 看多 _ 大阳 _ 任意 _0’)

Parameters

- **c** –CZSC 对象
- **di** –倒数第 i 根 K 线
- **n** –窗口大小

Returns

信号字典

bar_bpm_V230227

czsc.signals.**bar_bpm_V230227**(*c:* CZSC, ***kwargs*) → OrderedDict

以 BP 为单位的绝对动量

参数模板：”{freq}_D{di}N{n}T{th}_ 绝对动量 V230227”

信号逻辑：

1. 以 BP 为单位的绝对动量，计算最近 n 根 K 线的涨幅，如果大于 th，则为超强，否则为强势；
2. 反之，如果小于-th，则为超弱，否则为弱势

信号列表：

- Signal(‘15 分钟 _D2N5T300_ 绝对动量 V230227_ 弱势 _ 任意 _ 任意 _0’)
- Signal(‘15 分钟 _D2N5T300_ 绝对动量 V230227_ 强势 _ 任意 _ 任意 _0’)
- Signal(‘15 分钟 _D2N5T300_ 绝对动量 V230227_ 超强 _ 任意 _ 任意 _0’)
- Signal(‘15 分钟 _D2N5T300_ 绝对动量 V230227_ 超弱 _ 任意 _ 任意 _0’)

Parameters

- **c** –CZSC 对象
- **kwargs** –
  - di: 倒数第几根 K 线
  - n: 连续多少根 K 线
  - th: 超过多少 bp

Returns

信号识别结果

bar_break_V240428

czsc.signals.**bar_break_V240428**(*c:* CZSC, ***kwargs*) → OrderedDict

极值突破

参考资料：1. [后浪“拍死”前浪的日内波动极值策略](https://zhuanlan.zhihu.com/p/390025811) 2. 罗军，广发证券，2011，《基于日内波动极值的股指期货趋势跟随系统》

参数模板：”{freq}_D{di}W{w}_ 事件 V240428”

信号逻辑：

以 60 分钟级别为例，计算过程如下：1. 获取 w 根 K 线的最高价和最低价，分别记为 H 和 L；2. 当前 K 线的收盘价，记为 C；大于 H 时，触发收盘新高信号；小于 L 时，触发收盘新低信号。

信号列表：

- Signal(‘60 分钟 _D1W20_ 事件 V240428_ 收盘新低 _ 任意 _ 任意 _0’)
- Signal(‘60 分钟 _D1W20_ 事件 V240428_ 收盘新高 _ 任意 _ 任意 _0’)

Parameters

- **c** –CZSC 对象
- **kwargs** –
  - di: int, default 1, 周期偏移量
  - w: int, default 60, 计算多项式拟合的 K 线数量

Returns

信号识别结果

bar_cross_ps_V221112

czsc.signals.**bar_cross_ps_V221112**(*c:* CZSC, ***kwargs*) → OrderedDict

倒数第 di 根 K 线穿越支撑、压力位的数量【慎用，非常耗时】

参数模板：”{freq}_D{di}K_N{num}”

信号逻辑：

1. 计算最近 600 根 K 线的支撑、压力位列表；
2. 如果 dik 是阳性，切上穿 num 个以上的压力位，择时多头；反之，空头。

信号列表：

- Signal(‘15 分钟 _D2K_N3_ 空头 _ 任意 _ 任意 _0’)
- Signal(‘15 分钟 _D2K_N3_ 多头 _ 任意 _ 任意 _0’)

Parameters

- **c** –CZSC 对象
- **di** –信号计算在倒数第 di 根
- **num** –阈值

Returns

s

bar_dual_thrust_V230403

czsc.signals.**bar_dual_thrust_V230403**(*c:* CZSC, ***kwargs*)

Dual Thrust 通道突破

参数模板："{freq}_D{di} 通道突破 #{N}#{K1}#{K2}_BS 辅助 V230403"

信号逻辑：

参见：https://www.myquant.cn/docs/python_strategyies/424

其核心思想是定义一个区间，区间的上界和下界分别为支撑线和阻力线。当价格超过上界时，看多，跌破下界，看空。

信号列表：

- Signal('日线 _D1 通道突破 #5#20#20_BS 辅助 V230403_ 看空 _ 任意 _ 任意 _0')
- Signal('日线 _D1 通道突破 #5#20#20_BS 辅助 V230403_ 看多 _ 任意 _ 任意 _0')

Parameters

- **c** –基础周期的 CZSC 对象
- **kwargs** –其他参数 - di: 倒数第 di 根 K 线 - N: 前 N 天的数据 - K1: 参数，根据经验优化 - K2: 参数，根据经验优化

Returns

信号字典

bar_eight_V230702

czsc.signals.**bar_eight_V230702**(*c:* CZSC, ***kwargs*) → OrderedDict

8K 走势分类

参数模板："{freq}_D{di}#8K_ 走势分类 V230702"

信号逻辑：

参见博客：https://blog.sina.com.cn/s/blog_486e105c010009uy.html 这篇博客给出了 8K 走势分类的逻辑。

信号列表：

- Signal(‘30 分钟 _D1#8K_ 走势分类 V230702_ 弱平衡市 _ 任意 _ 任意 _0’)
- Signal(‘30 分钟 _D1#8K_ 走势分类 V230702_ 双中枢下跌 _ 任意 _ 任意 _0’)
- Signal(‘30 分钟 _D1#8K_ 走势分类 V230702_ 转折平衡市 _ 任意 _ 任意 _0’)
- Signal(‘30 分钟 _D1#8K_ 走势分类 V230702_ 强平衡市 _ 任意 _ 任意 _0’)
- Signal(‘30 分钟 _D1#8K_ 走势分类 V230702_ 双中枢上涨 _ 任意 _ 任意 _0’)
- Signal(‘30 分钟 _D1#8K_ 走势分类 V230702_ 无中枢上涨 _ 任意 _ 任意 _0’)
- Signal(‘30 分钟 _D1#8K_ 走势分类 V230702_ 无中枢下跌 _ 任意 _ 任意 _0’)

Parameters

c –CZSC 对象

Returns

信号识别结果

bar_end_V221211

czsc.signals.**bar_end_V221211**(*c:* CZSC, *freq1='60 分钟'*, ***kwargs*) → OrderedDict

判断分钟 K 线是否结束

参数模板：”{freq}_{freq1} 结束 _BS 辅助 221211”

信号逻辑：

以 freq 为基础周期，freq1 为大周期，判断 freq1 K 线是否结束。如果结束，返回信号值为“闭合”，否则返回“未闭 x”，x 为未闭合的次数。

信号列表：

- Signal(‘15 分钟 _60 分钟结束 _BS 辅助 221211_ 未闭 1_ 任意 _ 任意 _0’)
- Signal(‘15 分钟 _60 分钟结束 _BS 辅助 221211_ 未闭 2_ 任意 _ 任意 _0’)
- Signal(‘15 分钟 _60 分钟结束 _BS 辅助 221211_ 未闭 3_ 任意 _ 任意 _0’)
- Signal(‘15 分钟 _60 分钟结束 _BS 辅助 221211_ 闭合 _ 任意 _ 任意 _0’)

Parameters

- **c** –基础周期的 CZSC 对象
- **freq1** –分钟周期名称

Returns

s

bar_fake_break_V230204

czsc.signals.**bar_fake_break_V230204**(*c:* CZSC, ***kwargs*) → OrderedDict

假突破

参数模板："{freq}_D{di}N{n}M{m}_ 假突破"

信号描述：

1. 向下假突破，最近 N 根 K 线的滑动 M 窗口出现过大幅下跌破 K 线重叠中枢，随后几根 K 线快速拉回，看多；
2. 反之，向上假突破，看空。

信号列表：

- Signal('15 分钟 _D1N20M5_ 假突破 _ 看空 _ 任意 _ 任意 _0')
- Signal('15 分钟 _D1N20M5_ 假突破 _ 看多 _ 任意 _ 任意 _0')

Parameters

- **c** –CZSC 对象
- **di** –从最新的第几个 bar 开始计算

Returns

信号字典

bar_fang_liang_break_V221216

czsc.signals.**bar_fang_liang_break_V221216**(*c:* CZSC, ***kwargs*) → OrderedDict

放量向上突破并回踩指定均线，贡献者：琅盎

参数模板："{freq}_D{di}TH{th}#{ma_type}#{timeperiod}_ 突破 V221216" **信号逻辑：**

1. 放量突破
2. 缩量回踩，最近一根 K 线的成交量小于前面一段时间的均量

信号列表：

- Signal('15 分钟 _D1TH300#SMA#233_ 突破 V221216_ 放量突破 _ 缩量回踩 _ 任意 _0')

Parameters

- **c** –CZSC 对象
- **di** –信号计算截止倒数第 i 根 K 线
- **ma_type** –指定均线的类型，默认为 SMA
- **timeperiod** –指定均线的周期，默认为 233

- **th** –当前最低价同指定均线的距离阈值，单位 BP

Returns

信号识别结果

bar_limit_down_V230525

czsc.signals.**bar_limit_down_V230525**(*c:* CZSC, ***kwargs*) → OrderedDict

跌停后出现无下影线长实体阳线做多

参数模板："{freq}_ 跌停后无下影线长实体阳线 _ 短线 V230525"

信号逻辑：

1. 跌停后出现无下影线长实体阳线做多

信号列表：

- Signal('日线 _ 跌停后无下影线长实体阳线 _ 短线 V230525_ 满足 _ 任意 _ 任意 _0')

Parameters

- **c** –CZSC 对象
- **kwargs** –参数字典

Returns

返回信号结果

bar_mean_amount_V221112

czsc.signals.**bar_mean_amount_V221112**(*c:* CZSC, ***kwargs*) → OrderedDict

截取一段时间内的平均成交金额分类信号

参数模板："{freq}_D{di}K{n}B 均额 _{th1} 至 {th2} 千万"

**信号逻辑: **

倒数第 i 根 K 线向前 n 根 K 线的成交金额均值在 th1 和 th2 之间

信号列表：

- Signal('15 分钟 _D2K20B 均额 _1 至 4 千万 _ 否 _ 任意 _ 任意 _0')
- Signal('15 分钟 _D2K20B 均额 _1 至 4 千万 _ 是 _ 任意 _ 任意 _0')

Parameters

- **c** –CZSC 对象
- **di** –信号计算截止的倒数第 i 根

- **n** –向前看 n 根
- **th1** –成交金额下限，单位：千万
- **th2** –成交金额上限，单位：千万

Returns

s

bar_operate_span_V221111

czsc.signals.**bar_operate_span_V221111**(*c:* CZSC, ***kwargs*) → OrderedDict

日内操作时间区间，c 必须是基础周期的 CZSC 对象

参数模板："{freq}_T{t1}#{t2}_ 时间区间"

信号列表：

- Signal('15 分钟 _T0935#1450_ 时间区间 _ 是 _ 任意 _ 任意 _0')
- Signal('15 分钟 _T0935#1450_ 时间区间 _ 否 _ 任意 _ 任意 _0')

Parameters

c –基础周期的 CZSC 对象

Returns

s

bar_plr_V240427

czsc.signals.**bar_plr_V240427**(*c:* CZSC, ***kwargs*) → OrderedDict

盈亏比计算

plr 是 Profit Loss Ratio 的缩写，即盈亏比。盈亏比是一个用于衡量投资者在投资活动中获得的盈利与损失之间的比率的指标。

参数模板："{freq}_D{di}W{w}T{t}M{m}_ 盈亏比 V240427"

信号逻辑：

以多头的阴亏比为例，计算过程如下：1. 找到最近的一个最低点，记为 L；2. 在最低点 L 之前的 K 线中，找到第一个最高点，记为 H；3. 计算 H 到 L 之间的盈亏比，记为 plr；

信号列表：

- Signal('60 分钟 _D1W60T20M 多头 _ 盈亏比 V240427_ 不满足 _ 任意 _ 任意 _0')
- Signal('60 分钟 _D1W60T20M 多头 _ 盈亏比 V240427_ 满足 _ 任意 _ 任意 _0')
- Signal('60 分钟 _D1W60T20M 空头 _ 盈亏比 V240427_ 不满足 _ 任意 _ 任意 _0')
- Signal('60 分钟 _D1W60T20M 空头 _ 盈亏比 V240427_ 满足 _ 任意 _ 任意 _0')

Parameters

- **c** –CZSC 对象
- **kwargs** –
  - di: int, default 1, 周期偏移量
  - w: int, default 60, 计算盈亏比的 K 线数量
  - t: int, default 20, 盈亏比阈值，plr > t / 10 时满足信号条件
  - m: str, default “多头”, 信号方向,”多头”或“空头”

Returns

信号识别结果

bar_polyfit_V240428

czsc.signals.**bar_polyfit_V240428**(*c: CZSC*, ***kwargs*) → OrderedDict

一阶、二阶多项式拟合

参考资料：1. [基于低阶多项式拟合的日内趋势策略](https://zhuanlan.zhihu.com/p/391605615) 2. 罗军，广发证券，2011，《基于低阶多项式拟合的股指期货趋势交易 (LPTT) 策略》

参数模板：”{freq}_D{di}W{w}_ 分类 V240428”

信号逻辑：

若对一阶线性函数求一阶导数，也就是平常所说的斜率，若导数 dy/dt>0，说明价格正处于上升趋势；若导数 dy/dt<0，则为下跌趋势。若对二阶线性函数求二阶导数，若二阶导数 d2y/dt2>0，价格曲线为凹（开口向上）；若二阶导数 d2y/dt2<0，价格曲线为凸（开口向下）。

做个类比，价格曲线就像汽车行走的距离轨迹，对距离（位移）求一阶导数就是速度，速度大于 0 说明朝正方向开，小于 0 说明朝反方向开。求二阶导数就是加速度，假设此时汽车为正向行驶，加速度大于 0 说明在汽车速度还在增加的状态当中，在不断加速，反之则是处在减速的过程当中。

信号列表：

- Signal(‘60 分钟 _D1W20_ 分类 V240428_ 加速上涨 _ 任意 _ 任意 _0’)
- Signal(‘60 分钟 _D1W20_ 分类 V240428_ 减速上涨 _ 任意 _ 任意 _0’)
- Signal(‘60 分钟 _D1W20_ 分类 V240428_ 加速下跌 _ 任意 _ 任意 _0’)
- Signal(‘60 分钟 _D1W20_ 分类 V240428_ 减速下跌 _ 任意 _ 任意 _0’)

Parameters

- **c** –CZSC 对象
- **kwargs** –
  - di: int, default 1, 周期偏移量

- w: int, default 60, 计算多项式拟合的 K 线数量

Returns

信号识别结果

bar_r_breaker_V230326

czsc.signals.**bar_r_breaker_V230326**(*c:* CZSC, ***kwargs*)

RBreaker 日内回转交易

参数模板："{freq}_RBreaker_BS 辅助 V230326"

信号逻辑：

参见：https://www.myquant.cn/docs/python_strategyies/425

空仓时：突破策略空仓时，当盘中价格 > 突破买入价，则认为上涨的趋势还会继续，开仓做多；空仓时，当盘中价格 < 突破卖出价，则认为下跌的趋势还会继续，开仓做空。

持仓时：反转策略持多单时：当日内最高价 > 观察卖出价后，盘中价格回落，跌破反转卖出价构成的支撑线时，采取反转策略，即做空；持空单时：当日内最低价 < 观察买入价后，盘中价格反弹，超过反转买入价构成的阻力线时，采取反转策略，即做多。

信号列表：

- Signal('日线 _RBreaker_BS 辅助 V230326_ 做多 _ 反转 _ 任意 _0')
- Signal('日线 _RBreaker_BS 辅助 V230326_ 做空 _ 趋势 _ 任意 _0')
- Signal('日线 _RBreaker_BS 辅助 V230326_ 做多 _ 趋势 _ 任意 _0')
- Signal('日线 _RBreaker_BS 辅助 V230326_ 做空 _ 反转 _ 任意 _0')

Returns

信号字典

bar_reversal_V230227

czsc.signals.**bar_reversal_V230227**(*c:* CZSC, ***kwargs*) → OrderedDict

判断最近一根 K 线是否具有反转迹象

参数模板："{freq}_D{di}A{avg_bp}_ 反转 V230227"

信号逻辑：

- 看多：当前 K 线为阴线，或阳线长上影; 且截止前一根 K 线，连续 3 / 5 / 8 根 K 线累计涨幅超过 avg_bp * n，或连续 13 根 K 线都是阳线
- 反之，看空

信号列表：

- Signal(‘15 分钟 _D1A300_ 反转 V230227_ 看多 _ 任意 _ 任意 _0’)
- Signal(‘15 分钟 _D1A300_ 反转 V230227_ 看空 _ 任意 _ 任意 _0’)

Parameters

- **c** –CZSC 对象
- **di** –倒数第几根 K 线
- **avg_bp** –平均单根 K 线的涨跌幅，用于判断是否是反转

Returns

bar_section_momentum_V221112

czsc.signals.**bar_section_momentum_V221112**(*c: CZSC*, ***kwargs*) → OrderedDict

获取某个区间（固定 K 线数量）的动量强弱

参数模板：”{freq}_D{di}K{n}B_ 阈值 {th}BP”

信号列表：

- Signal(‘15 分钟 _D2K10B_ 阈值 100BP_ 下跌 _ 强势 _ 低波动 _0’)
- Signal(‘15 分钟 _D2K10B_ 阈值 100BP_ 下跌 _ 弱势 _ 低波动 _0’)
- Signal(‘15 分钟 _D2K10B_ 阈值 100BP_ 下跌 _ 弱势 _ 高波动 _0’)
- Signal(‘15 分钟 _D2K10B_ 阈值 100BP_ 上涨 _ 弱势 _ 低波动 _0’)
- Signal(‘15 分钟 _D2K10B_ 阈值 100BP_ 上涨 _ 弱势 _ 高波动 _0’)
- Signal(‘15 分钟 _D2K10B_ 阈值 100BP_ 上涨 _ 强势 _ 低波动 _0’)
- Signal(‘15 分钟 _D2K10B_ 阈值 100BP_ 上涨 _ 强势 _ 高波动 _0’)
- Signal(‘15 分钟 _D2K10B_ 阈值 100BP_ 下跌 _ 强势 _ 高波动 _0’)

Parameters

- **c** –CZSC 对象
- **di** –区间结束 K 线位置，倒数
- **n** –取近 n 根 K 线
- **th** –动量强弱划分的阈值，单位 BP

Returns

s

bar_shuang_fei_V230507

czsc.signals.**bar_shuang_fei_V230507**(*c:* CZSC, ***kwargs*) → OrderedDict

双飞涨停，贡献者：琅盎

参数模板："{freq}_D{di} 双飞 _ 短线 V230507"

信号逻辑：

1. 今天涨停;
2. 昨天收阴，且跌幅大于 5%
3. 前天涨停

信号列表：

- Signal('日线 _D1 双飞 _ 短线 V230507_ 看多 _ 任意 _ 任意 _0')

Parameters

- **c** –CZSC 对象
- **kwargs** –参数字典
  - di: 信号计算截止倒数第 i 根 K 线

Returns

信号识别结果

bar_single_V230214

czsc.signals.**bar_single_V230214**(*c:* CZSC, ***kwargs*) → OrderedDict

单根 K 线的状态

参数模板："{freq}_D{di}T{t}_ 状态"

信号描述：

1. 上涨阳线，下跌阴线；
2. 长实体，长上影，长下影，其他；

信号列表：

- Signal('日线 _D2T10_ 状态 _ 阴线 _ 长实体 _ 任意 _0')
- Signal('日线 _D2T10_ 状态 _ 阳线 _ 长实体 _ 任意 _0')
- Signal('日线 _D2T10_ 状态 _ 阴线 _ 长上影 _ 任意 _0')
- Signal('日线 _D2T10_ 状态 _ 阳线 _ 长上影 _ 任意 _0')
- Signal('日线 _D2T10_ 状态 _ 阴线 _ 长下影 _ 任意 _0')

- Signal(‘日线 _D2T10_ 状态 _ 阳线 _ 长下影 _ 任意 _0’)

Parameters

- **c** –CZSC 对象
- **di** –倒数第几根 K 线
- **kwargs** –t: 长实体、长上影、长下影的阈值，默认为 1.0

Returns

信号识别结果

bar_single_V230506

czsc.signals.**bar_single_V230506**(*c:* CZSC, ***kwargs*) → OrderedDict

单 K 趋势因子辅助判断买卖点

参数模板：”{freq}_D{di} 单 K 趋势 N{n}_BS 辅助 V230506”

信号逻辑：

1. 定义趋势因子：(收盘价 / 开盘价 -1) / 成交量
2. 选取最近 100 根 K 线，计算趋势因子，分成 n 层

信号列表：

- Signal(‘15 分钟 _D1 单 K 趋势 N5_BS 辅助 V230506_ 第 3 层 _ 任意 _ 任意 _0’)
- Signal(‘15 分钟 _D1 单 K 趋势 N5_BS 辅助 V230506_ 第 4 层 _ 任意 _ 任意 _0’)
- Signal(‘15 分钟 _D1 单 K 趋势 N5_BS 辅助 V230506_ 第 2 层 _ 任意 _ 任意 _0’)
- Signal(‘15 分钟 _D1 单 K 趋势 N5_BS 辅助 V230506_ 第 1 层 _ 任意 _ 任意 _0’)
- Signal(‘15 分钟 _D1 单 K 趋势 N5_BS 辅助 V230506_ 第 5 层 _ 任意 _ 任意 _0’)

Parameters

- **c** –CZSC 对象
- **kwargs** –参数字典:return: 返回信号结果

bar_time_V230327

czsc.signals.**bar_time_V230327**(*c:* CZSC, ***kwargs*)

K 线日内时间分段信号

参数模板："{freq}_ 日内时间 _ 分段 V230327"

信号逻辑：

- 60 分钟或 30 分钟 K 线，按日内出现顺序分段

信号列表：

- Signal（'60 分钟 _ 日内时间 _ 分段 V230327_ 第 1 段 _ 任意 _ 任意 _0'）
- Signal（'60 分钟 _ 日内时间 _ 分段 V230327_ 第 2 段 _ 任意 _ 任意 _0'）
- Signal（'60 分钟 _ 日内时间 _ 分段 V230327_ 第 3 段 _ 任意 _ 任意 _0'）
- Signal（'60 分钟 _ 日内时间 _ 分段 V230327_ 第 4 段 _ 任意 _ 任意 _0'）

Parameters

- **c** –CZSC 对象
- **kwargs** –

Returns

信号识别结果

bar_tnr_V230629

czsc.signals.**bar_tnr_V230629**(*c:* CZSC, ***kwargs*) → OrderedDict

趋势噪音指标（TNR，Trend to Noise Rate）分层

参数模板："{freq}_D{di}TNR{timeperiod}_ 趋势 V230629"

信号逻辑：

TNR 计算公式：取 N 根 K 线，首尾两个 close 的绝对差值除以相邻两个 close 的绝对差值累计。

取最近 100 个 bar 的 TNR 进行分层。

信号列表：

- Signal（'15 分钟 _D1TNR14_ 趋势 V230629_ 第 7 层 _ 任意 _ 任意 _0'）
- Signal（'15 分钟 _D1TNR14_ 趋势 V230629_ 第 6 层 _ 任意 _ 任意 _0'）
- Signal（'15 分钟 _D1TNR14_ 趋势 V230629_ 第 8 层 _ 任意 _ 任意 _0'）
- Signal（'15 分钟 _D1TNR14_ 趋势 V230629_ 第 9 层 _ 任意 _ 任意 _0'）
- Signal（'15 分钟 _D1TNR14_ 趋势 V230629_ 第 10 层 _ 任意 _ 任意 _0'）

- Signal(‘15 分钟 _D1TNR14_ 趋势 V230629_ 第 5 层 _ 任意 _ 任意 _0’)
- Signal(‘15 分钟 _D1TNR14_ 趋势 V230629_ 第 2 层 _ 任意 _ 任意 _0’)
- Signal(‘15 分钟 _D1TNR14_ 趋势 V230629_ 第 1 层 _ 任意 _ 任意 _0’)
- Signal(‘15 分钟 _D1TNR14_ 趋势 V230629_ 第 3 层 _ 任意 _ 任意 _0’)
- Signal(‘15 分钟 _D1TNR14_ 趋势 V230629_ 第 4 层 _ 任意 _ 任意 _0’)

Parameters

- **c** –czsc 对象
- **kwargs** –
  - di: 倒数第 i 根 K 线
  - timeperiod: TNR 指标的参数

Returns

信号字典

bar_tnr_V230630

czsc.signals.**bar_tnr_V230630**(*c:* CZSC, ***kwargs*) → OrderedDict

趋势噪音指标（TNR，Trend to Noise Rate）

参数模板：”{freq}_D{di}TNR{timeperiod}K{k}_ 趋势 V230630”

信号逻辑：

TNR 计算公式：取 N 根 K 线，首尾两个 close 的绝对差值除以相邻两个 close 的绝对差值累计。

噪音变化判断，如果 t 时刻的 TNR > 过去 k 个 TNR 的均值，则说明噪音在减少，此时趋势较强；反之，噪音在增加，此时趋势较弱。

信号列表：

- Signal(‘15 分钟 _D1TNR14K3_ 趋势 V230630_ 噪音减少 _ 任意 _ 任意 _0’)
- Signal(‘15 分钟 _D1TNR14K3_ 趋势 V230630_ 噪音增加 _ 任意 _ 任意 _0’)

Parameters

- **c** –czsc 对象
- **kwargs** –
  - di: 倒数第 i 根 K 线
  - timeperiod: TNR 指标的参数
  - k: 过去 k 个 TNR 的均值

Returns

信号字典

bar_trend_V240209

czsc.signals.**bar_trend_V240209**(*c:* CZSC, ***kwargs*) → OrderedDict

趋势跟踪信号

参数模板："{freq}_D{di}N{N} 趋势跟踪 _BS 辅助 V240209"

信号逻辑：

以多头为例：1. 低点出现在高点之后，且低点右侧的高点到当前 K 线之间的 K 线数量在 5-30 之间；2. 低点右侧的 K 线的 DIF 值小于前 N 根 K 线的 DIF 值的标准差的一半；3. 低点右侧的 K 线的最低价大于低点的最低价；4. 低点右侧的 K 线的 MACD 值小于前 N 根 K 线的 MACD 值的标准差的一半。

信号列表：

- Signal('60 分钟 _D1N60 趋势跟踪 _BS 辅助 V240209_ 多头 _ 任意 _ 任意 _0')
- Signal('60 分钟 _D1N60 趋势跟踪 _BS 辅助 V240209_ 空头 _ 任意 _ 任意 _0')

Parameters

- **c** –CZSC 对象
- **kwargs** –参数设置
  - di: int, default 1, 倒数第几根 K 线
  - N: int, default 20, 窗口大小

Returns

信号识别结果

bar_triple_V230506

czsc.signals.**bar_triple_V230506**(*c:* CZSC, ***kwargs*) → OrderedDict

三 K 加速形态配合成交量变化

参数模板："{freq}_D{di} 三 K 加速 _ 裸 K 形态 V230506"

信号逻辑：

1. 连续三根阳线，【三连涨】，如果高低点不断创新高，【新高涨】
2. 连续三根阴线，【三连跌】，如果高低点不断创新低，【新低跌】
3. 加入成交量变化的判断，成交量逐渐放大或成交量逐渐缩小

信号列表：

- Signal(‘15 分钟 _D1 三 K 加速 _ 裸 K 形态 V230506_ 三连涨 _ 量柱无序 _ 任意 _0’)
- Signal(‘15 分钟 _D1 三 K 加速 _ 裸 K 形态 V230506_ 三连跌 _ 量柱无序 _ 任意 _0’)
- Signal(‘15 分钟 _D1 三 K 加速 _ 裸 K 形态 V230506_ 新高涨 _ 依次放量 _ 任意 _0’)
- Signal(‘15 分钟 _D1 三 K 加速 _ 裸 K 形态 V230506_ 新低跌 _ 依次缩量 _ 任意 _0’)
- Signal(‘15 分钟 _D1 三 K 加速 _ 裸 K 形态 V230506_ 新低跌 _ 量柱无序 _ 任意 _0’)
- Signal(‘15 分钟 _D1 三 K 加速 _ 裸 K 形态 V230506_ 三连涨 _ 依次放量 _ 任意 _0’)
- Signal(‘15 分钟 _D1 三 K 加速 _ 裸 K 形态 V230506_ 三连跌 _ 依次放量 _ 任意 _0’)
- Signal(‘15 分钟 _D1 三 K 加速 _ 裸 K 形态 V230506_ 新低跌 _ 依次放量 _ 任意 _0’)
- Signal(‘15 分钟 _D1 三 K 加速 _ 裸 K 形态 V230506_ 三连跌 _ 依次缩量 _ 任意 _0’)
- Signal(‘15 分钟 _D1 三 K 加速 _ 裸 K 形态 V230506_ 新高涨 _ 依次缩量 _ 任意 _0’)
- Signal(‘15 分钟 _D1 三 K 加速 _ 裸 K 形态 V230506_ 新高涨 _ 量柱无序 _ 任意 _0’)
- Signal(‘15 分钟 _D1 三 K 加速 _ 裸 K 形态 V230506_ 三连涨 _ 依次缩量 _ 任意 _0’)

Parameters

- **c** –CZSC 对象
- **kwargs** –参数字典:return: 返回信号结果

bar_vol_bs1_V230224

czsc.signals.**bar_vol_bs1_V230224**(*c:* CZSC, ***kwargs*)

量价配合的高低点判断

参数模板:”{freq}_D{di}N{n} 量价 _BS1 辅助”

信号逻辑:

1. 高点看空：窗口内最近一根 K 线上影大于下影的两倍，同时最高价和成交量同时创新高
2. 反之，低点看多

信号列表:

- Signal(‘15 分钟 _D2N34 量价 _BS1 辅助 _ 看多 _ 任意 _ 任意 _0’)
- Signal(‘15 分钟 _D2N34 量价 _BS1 辅助 _ 看空 _ 任意 _ 任意 _0’)

Parameters

- **c** –CZSC 对象
- **di** –倒数第 i 根 K 线
- **n** –窗口大小

Returns

信号字典

bar_vol_grow_V221112

czsc.signals.**bar_vol_grow_V221112**(*c:* CZSC, ***kwargs*) → OrderedDict

倒数第 i 根 K 线的成交量相比于前 N 根 K 线放量

参数模板："{freq}_D{di}K{n}B_ 放量 V221112"

**信号逻辑: **

放量的定义为，倒数第 i 根 K 线的量能 / 过去 N 根的平均量能，在 2-4 倍之间。

信号列表：

- Signal('15 分钟 _D1K5B_ 放量 V221112_ 否 _ 任意 _ 任意 _0')
- Signal('15 分钟 _D1K5B_ 放量 V221112_ 是 _ 任意 _ 任意 _0')

Parameters

- **c** –CZSC 对象
- **kwargs** –参数字典
  - di: 倒数第 i 根 K 线
  - n: 过去 N 根 K 线

Returns

信号识别结果

bar_weekday_V230328

czsc.signals.**bar_weekday_V230328**(*c:* CZSC, ***kwargs*)

K 线周内时间分段信号

参数模板："{freq}_ 周内时间 _ 分段 V230328"

信号逻辑：

- 按周内日线出现顺序分段

信号列表：

- Signal('60 分钟 _ 周内时间 _ 分段 V230328_ 周一 _ 任意 _ 任意 _0')
- Signal('60 分钟 _ 周内时间 _ 分段 V230328_ 周二 _ 任意 _ 任意 _0')
- Signal('60 分钟 _ 周内时间 _ 分段 V230328_ 周三 _ 任意 _ 任意 _0')
- Signal('60 分钟 _ 周内时间 _ 分段 V230328_ 周四 _ 任意 _ 任意 _0')

- Signal(‘60 分钟 _ 周内时间 _ 分段 V230328_ 周五 _ 任意 _ 任意 _0’)

Parameters

- **c** –CZSC 对象
- **kwargs** –

Returns

信号识别结果

bar_window_ps_V230731

czsc.signals.**bar_window_ps_V230731**(*c:* CZSC, ***kwargs*) → OrderedDict

指定窗口内支撑压力位分位数计算，贡献者：chenlei

参数模板：”{freq}_W{w}M{m}N{n}L{l}_ 支撑压力位 V230731”

信号逻辑：

1. 计算最近 N 笔的最高价 NH 和最低价 NL，这个可以近似理解成价格的支撑和压力位
2. 计算并缓存最新 K 线的收盘价格 C 处于 NH、NL 之间的位置，计算方法为 P = （C - NL）/ (NH - NL)
3. 取最近 M 个 P 值序列，按分位数分层，分层数量为 L，分层的最大值为最近的压力，最小值为最近的支撑，当前值为最近的价格位置

信号列表：

- Signal(‘15 分钟 _W5M40N8L5_ 支撑压力位 V230731_ 压力 N5_ 支撑 N5_ 当前 N5_0’)
- Signal(‘15 分钟 _W5M40N8L5_ 支撑压力位 V230731_ 压力 N5_ 支撑 N4_ 当前 N5_0’)
- Signal(‘15 分钟 _W5M40N8L5_ 支撑压力位 V230731_ 压力 N5_ 支撑 N4_ 当前 N4_0’)
- Signal(‘15 分钟 _W5M40N8L5_ 支撑压力位 V230731_ 压力 N5_ 支撑 N3_ 当前 N5_0’)
- Signal(‘15 分钟 _W5M40N8L5_ 支撑压力位 V230731_ 压力 N5_ 支撑 N2_ 当前 N2_0’)
- Signal(‘15 分钟 _W5M40N8L5_ 支撑压力位 V230731_ 压力 N5_ 支撑 N1_ 当前 N2_0’)

Parameters

- **c** –CZSC 对象
- **kwargs** –参数字典
  - **param w**
    评价分位数分布用的窗口大小
  - **param m**
    计算分位数所需取 K 线的数量。

- **param n**
  最近 N 笔
- **param l**
  分层的数量。

Returns
信号识别结果

bar_window_ps_V230801

czsc.signals.**bar_window_ps_V230801**(*c:* CZSC, ***kwargs*) → OrderedDict

指定窗口内支撑压力位分位数计算

参数模板："{freq}_N{n}W{w}_ 支撑压力位 V230801"

信号逻辑：

1. 计算最近 N 笔的最高价 NH 和最低价 NL，这个可以近似理解成价格的支撑和压力位
2. 计算并缓存最新 K 线的收盘价格 C 处于 NH、NL 之间的位置，计算方法为 P =（C - NL）/ (NH - NL)
3. 取最近 M 个 P 值序列，四舍五入精确到小数点后 1 位，作为当前 K 线的分位数

信号列表：

- Signal('60 分钟 _N8W5_ 支撑压力位 V230801_ 最大 N7_ 最小 N4_ 当前 N5_0')
- Signal('60 分钟 _N8W5_ 支撑压力位 V230801_ 最大 N8_ 最小 N4_ 当前 N4_0')
- Signal('60 分钟 _N8W5_ 支撑压力位 V230801_ 最大 N6_ 最小 N2_ 当前 N6_0')
- Signal('60 分钟 _N8W5_ 支撑压力位 V230801_ 最大 N6_ 最小 N2_ 当前 N5_0')
- Signal('60 分钟 _N8W5_ 支撑压力位 V230801_ 最大 N6_ 最小 N2_ 当前 N3_0')
- Signal('60 分钟 _N8W5_ 支撑压力位 V230801_ 最大 N4_ 最小 N0_ 当前 N3_0')
- Signal('60 分钟 _N8W5_ 支撑压力位 V230801_ 最大 N4_ 最小 N0_ 当前 N2_0')
- Signal('60 分钟 _N8W5_ 支撑压力位 V230801_ 最大 N4_ 最小 N0_ 当前 N1_0')
- Signal('60 分钟 _N8W5_ 支撑压力位 V230801_ 最大 N7_ 最小 N3_ 当前 N6_0')
- Signal('60 分钟 _N8W5_ 支撑压力位 V230801_ 最大 N9_ 最小 N4_ 当前 N9_0')
- Signal('60 分钟 _N8W5_ 支撑压力位 V230801_ 最大 N4_ 最小 N0_ 当前 N4_0')

Parameters

- **c** –CZSC 对象
- **kwargs** –参数字典

- **param w**
  评价分位数分布用的窗口大小
- **param n**
  最近 N 笔

Returns
信号识别结果

bar_window_std_V230731

czsc.signals.**bar_window_std_V230731**(*c:* CZSC, ***kwargs*) → OrderedDict

指定窗口内波动率的特征

参数模板："{freq}_D{di}W{window}M{m}N{n}_ 窗口波动 V230731"

信号逻辑：

滚动计算最近 m 根 K 线的波动率，分成 n 层，最大值为 n，最小值为 1；最近 window 根 K 线的最大值为 max_layer，最小值为 min_layer。以这两个值作为窗口内的波动率特征。

信号列表：

- Signal('60 分钟 _D2W3M100N10_ 窗口波动 V230731_ 高波 N7_ 低波 N6_ 任意 _0')
- Signal('60 分钟 _D2W3M100N10_ 窗口波动 V230731_ 高波 N6_ 低波 N6_ 任意 _0')
- Signal('60 分钟 _D2W3M100N10_ 窗口波动 V230731_ 高波 N8_ 低波 N6_ 任意 _0')
- Signal('60 分钟 _D2W3M100N10_ 窗口波动 V230731_ 高波 N9_ 低波 N6_ 任意 _0')
- Signal('60 分钟 _D2W3M100N10_ 窗口波动 V230731_ 高波 N9_ 低波 N9_ 任意 _0')
- Signal('60 分钟 _D2W3M100N10_ 窗口波动 V230731_ 高波 N9_ 低波 N8_ 任意 _0')
- Signal('60 分钟 _D2W3M100N10_ 窗口波动 V230731_ 高波 N8_ 低波 N8_ 任意 _0')
- Signal('60 分钟 _D2W3M100N10_ 窗口波动 V230731_ 高波 N8_ 低波 N7_ 任意 _0')
- Signal('60 分钟 _D2W3M100N10_ 窗口波动 V230731_ 高波 N7_ 低波 N7_ 任意 _0')
- Signal('60 分钟 _D2W3M100N10_ 窗口波动 V230731_ 高波 N7_ 低波 N5_ 任意 _0')
- Signal('60 分钟 _D2W3M100N10_ 窗口波动 V230731_ 高波 N6_ 低波 N5_ 任意 _0')
- Signal('60 分钟 _D2W3M100N10_ 窗口波动 V230731_ 高波 N5_ 低波 N4_ 任意 _0')
- Signal('60 分钟 _D2W3M100N10_ 窗口波动 V230731_ 高波 N5_ 低波 N3_ 任意 _0')
- Signal('60 分钟 _D2W3M100N10_ 窗口波动 V230731_ 高波 N4_ 低波 N3_ 任意 _0')

Parameters

- **c** –CZSC 对象

- **kwargs** –参数字典
  - **param di**
    信号计算截止倒数第 i 根 K 线
  - **param w**
    观察的窗口大小。
  - **param m**
    计算分位数所需取 K 线的数量。
  - **param n**
    分层的数量。

Returns
信号识别结果

bar_zdf_V221203

czsc.signals.**bar_zdf_V221203**(*c:* CZSC, ***kwargs*) → OrderedDict

单根 K 线的涨跌幅区间

参数模板:”{freq}_D{di}{mode}_{t1} 至 {t2}”

信号列表:

- Signal(‘日线 _D1ZF_300 至 600_ 满足 _ 任意 _ 任意 _0’)
- Signal(‘日线 _D1DF_300 至 600_ 满足 _ 任意 _ 任意 _0’)

Parameters

- **c** –CZSC 对象
- **di** –信号计算截止倒数第 i 根 K 线
- **mode** –模式，ZF 表示涨幅，DF 表示跌幅
- **span** –区间大小

Returns
信号识别结果

bar_zdt_V230331

czsc.signals.**bar_zdt_V230331**(*c:* CZSC, ***kwargs*) → OrderedDict

计算倒数第 di 根 K 线的涨跌停信息

参数模板："{freq}_D{di}_ 涨跌停 V230331"

信号逻辑：

- close 等于 high 大于等于前一根 K 线的 close，近似认为是涨停；反之，跌停。

信号列表：

- Signal('15 分钟 _D1_ 涨跌停 V230331_ 涨停 _ 任意 _ 任意 _0')
- Signal('15 分钟 _D1_ 涨跌停 V230331_ 跌停 _ 任意 _ 任意 _0')

Parameters

- **c** –基础周期的 CZSC 对象
- **kwargs** –
  - di: 倒数第 di 根 K 线

Returns

信号识别结果

bar_zt_count_V230504

czsc.signals.**bar_zt_count_V230504**(*c:* CZSC, ***kwargs*) → OrderedDict

窗口内涨停计数

参数模板："{freq}_D{di}W{window} 涨停计数 _ 裸 K 形态 V230504"

信号逻辑：

1. 连续三根阳线，且高低点不断创新高，看多
2. 连续三根阴线，且高低点不断创新低，看空

信号列表：

- Signal('日线 _D1W5 涨停计数 _ 裸 K 形态 V230504_1 次 _ 连续 0 次 _ 任意 _0')
- Signal('日线 _D1W5 涨停计数 _ 裸 K 形态 V230504_2 次 _ 连续 1 次 _ 任意 _0')
- Signal('日线 _D1W5 涨停计数 _ 裸 K 形态 V230504_3 次 _ 连续 2 次 _ 任意 _0')

Parameters

- **c** –CZSC 对象

- **kwargs** –参数字典

Returns

返回信号结果

bias_up_dw_line_V230618

czsc.signals.**bias_up_dw_line_V230618**(*c:* CZSC, ***kwargs*) → OrderedDict

BIAS 乖离率指标，贡献者：琅盎

参数模板："{freq}_D{di}N{n}M{m}P{p}TH1{th1}TH2{th2}TH3{th3}_BIAS 乖离率 V230618"

信号逻辑：

乖离率 BIAS 用来衡量收盘价与移动平均线之间的差距。当 BIAS6 大于 3 且 BIAS12 大于 5 且 BIAS24 大于 8，三个乖离率均进入股价强势上涨区间，产生买入信号；当 BIAS6 小于-3 且 BIAS12 小于-5 且 BIAS24 小于-8 时，三种乖离率均进入股价强势下跌区间，产生卖出信号

信号列表：

- Signal('日线 _D1N6M12P24TH11TH23TH35_BIAS 乖离率 V230618_ 看空 _ 任意 _ 任意 _0')
- Signal('日线 _D1N6M12P24TH11TH23TH35_BIAS 乖离率 V230618_ 看多 _ 任意 _ 任意 _0')

Parameters

- **c** –CZSC 对象
- **kwargs** –参数字典
  - **param di**
    信号计算截止倒数第 i 根 K 线
  - **param n**
    获取 K 线的根数，默认为 30
  - **param m**
    获取 K 线的根数，默认为 20

Returns

信号识别结果

byi_bi_end_V230106

czsc.signals.**byi_bi_end_V230106**(*c:* CZSC, ***kwargs*) → OrderedDict

白仪分型停顿辅助笔结束判断

参数模板："{freq}_D0 停顿分型 _BE 辅助 V230106"

信号逻辑：

分型停顿图解：https://pic1.zhimg.com/80/v2-3c5c3f264bffdf14c5ac6ae83bc5d5f0_720w.webp

1. 白仪底分型停顿，认为是向下笔结束；反之，向上笔结束
2. 底分型停顿：底分型后一根大阳线收盘在底分型的高点上方；反之，顶分型停顿

信号列表：

- Signal('15 分钟 _D0 停顿分型 _BE 辅助 V230106_ 看多 _ 强 _ 任意 _0')
- Signal('15 分钟 _D0 停顿分型 _BE 辅助 V230106_ 看空 _ 强 _ 任意 _0')
- Signal('15 分钟 _D0 停顿分型 _BE 辅助 V230106_ 看空 _ 弱 _ 任意 _0')
- Signal('15 分钟 _D0 停顿分型 _BE 辅助 V230106_ 看多 _ 弱 _ 任意 _0')

Notes：

1. BE 是 Bi End 的缩写

Parameters

c –CZSC 对象

Returns

信号识别结果

byi_bi_end_V230107

czsc.signals.**byi_bi_end_V230107**(*c:* CZSC, ***kwargs*) → OrderedDict

白仪验证分型辅助判断笔结束

参数模板："{freq}_D0 验证分型 _BE 辅助 V230107"

信号逻辑：

验证分型图解：https://pic1.zhimg.com/80/v2-80ac88269286707db98a5560107da4ec_720w.webp

1. 白仪验证底分型，认为是向下笔结束；反之，向上笔结束

信号列表：

- Signal('15 分钟 _D0 验证分型 _BE 辅助 V230107_ 看空 _ 强 _ 任意 _0')
- Signal('15 分钟 _D0 验证分型 _BE 辅助 V230107_ 看空 _ 弱 _ 任意 _0')

- Signal(‘15 分钟 _D0 验证分型 _BE 辅助 V230107_ 看多 _ 弱 _ 任意 _0’)
- Signal(‘15 分钟 _D0 验证分型 _BE 辅助 V230107_ 看多 _ 强 _ 任意 _0’)

Parameters

c –CZSC 对象

Returns

信号识别结果

byi_fx_num_V230628

czsc.signals.**byi_fx_num_V230628**(*c:* CZSC, ***kwargs*) → OrderedDict

白仪前面下跌或上涨一笔次级别笔结构数量满足条件；贡献者：谌意勇

参数模板:”{freq}_D{di} 笔分型数大于 {num}_BE 辅助 V230628”

信号逻辑:

对于采用分型停顿或者分型验证开开仓，前一笔内部次级别笔结构尽量带结构，此信号函数为当分型笔数量判断大于 num 为满足条件

信号列表:

- Signal(‘15 分钟 _D1 笔分型数大于 4_BE 辅助 V230628_ 向下 _ 满足 _ 任意 _0’)
- Signal(‘15 分钟 _D1 笔分型数大于 4_BE 辅助 V230628_ 向上 _ 满足 _ 任意 _0’)

Parameters

- **c** –CZSC 对象
- **di** –从倒数第几笔开始检查
- **num** –前笔内部次级别笔数量

Returns

信号识别结果

byi_second_bs_V230324

czsc.signals.**byi_second_bs_V230324**(*c:* CZSC, ***kwargs*) → OrderedDict

白仪二类买卖点辅助 V230324

参数模板:”{freq}_D{di}MACD{fastperiod}#{slowperiod}#{signalperiod} 回抽零轴 _BS2 辅助 V230324”

参考资料：https://zhuanlan.zhihu.com/p/550719065 由于文字描述的比较模糊，笔的算法也有差异，这里的实现和原文有一定出入

参数模板:”{freq}_D{di}MACD{fastperiod}#{slowperiod}#{signalperiod} 回抽零轴 _BS2 辅助 V230324”

信号逻辑：

1. **二买定义：**
    a. 1,3,5 笔的 dif 值都小于 0，且 1,3,5 笔的 dif 值中最大值小于-2 倍标准差，且 8 笔的 dif 值大于 0，且 9 笔的 dif 值小于 0.3 倍标准差
    b. 第 9 笔向下
2. **二卖定义：**
    a. 1,3,5 笔的 dif 值都大于 0，且 1,3,5 笔的 dif 值中最小值大于 2 倍标准差，且 8 笔的 dif 值小于 0，且 9 笔的 dif 值大于-0.3 倍标准差
    b. 第 9 笔向上

信号列表：

- Signal(‘15 分钟 _D1MACD12#26#9 回抽零轴 _BS2 辅助 V230324_ 看空 _ 任意 _ 任意 _0’)
- Signal(‘15 分钟 _D1MACD12#26#9 回抽零轴 _BS2 辅助 V230324_ 看多 _ 任意 _ 任意 _0’)

Parameters

- **c** –CZSC 对象
- **di** –从倒数第几笔开始检查

Returns

信号识别结果

byi_symmetry_zs_V221107

czsc.signals.**byi_symmetry_zs_V221107**(*c:* CZSC, ***kwargs*)

对称中枢信号

参数模板：”{freq}_D{di}B_ 对称中枢”

信号逻辑：

从 di 笔往前数 7/5/3 笔，如果构成中枢，且所有笔的力度序列标准差小于均值的一定比例，则认为是对称中枢

信号列表：

- Signal(‘15 分钟 _D1B_ 对称中枢 _ 否 _ 向下 _ 任意 _0’)
- Signal(‘15 分钟 _D1B_ 对称中枢 _ 是 _ 向上 _7 笔 _0’)
- Signal(‘15 分钟 _D1B_ 对称中枢 _ 否 _ 向上 _ 任意 _0’)
- Signal(‘15 分钟 _D1B_ 对称中枢 _ 是 _ 向下 _3 笔 _0’)
- Signal(‘15 分钟 _D1B_ 对称中枢 _ 是 _ 向上 _3 笔 _0’)

- Signal(‘15 分钟 _D1B_ 对称中枢 _ 是 _ 向下 _5 笔 _0’)
- Signal(‘15 分钟 _D1B_ 对称中枢 _ 是 _ 向上 _5 笔 _0’)
- Signal(‘15 分钟 _D1B_ 对称中枢 _ 是 _ 向下 _7 笔 _0’)

Parameters

c –CZSC 对象

Returns

信号识别结果

cat_macd_V230518

czsc.signals.**cat_macd_V230518**(*cat:* CzscSignals, ***kwargs*) → OrderedDict

freq1 与 freq2 联立信号，freq1 > freq2

参数模板:”{freq1}#{freq2}_MACD 交叉 _ 联立 V230518”

信号逻辑:

1. 看多：freq1 MACD 金叉后，freq2 MACD 首次金叉
2. 看空：freq1 MACD 死叉后，freq2 MACD 首次死叉

信号列表:

- Signal(‘日线 #60 分钟 _MACD 交叉 _ 联立 V230518_ 看空 _ 任意 _ 任意 _0’)
- Signal(‘日线 #60 分钟 _MACD 交叉 _ 联立 V230518_ 看多 _ 任意 _ 任意 _0’)

Parameters

- **cat** –CzscSignals 对象
- **kwargs** –参数字典:return: 返回信号结果

cat_macd_V230520

czsc.signals.**cat_macd_V230520**(*cat:* CzscSignals, ***kwargs*) → OrderedDict

freq1 与 freq2 联立信号，freq1 > freq2

参数模板:”{freq1}#{freq2}_MACD 交叉 _ 联立 V230520”

信号逻辑:

1. 看多：freq1 MACD 多头缩柱后，freq2 MACD 首次金叉
2. 看空：freq1 MACD 空头缩柱后，freq2 MACD 首次死叉

信号列表:

- Signal(‘日线 #60 分钟 _MACD 交叉 _ 联立 V230520_ 看空 _ 零轴下方 _ 任意 _0’)
- Signal(‘日线 #60 分钟 _MACD 交叉 _ 联立 V230520_ 看多 _ 零轴上方 _ 任意 _0’)
- Signal(‘日线 #60 分钟 _MACD 交叉 _ 联立 V230520_ 看空 _ 零轴上方 _ 任意 _0’)
- Signal(‘日线 #60 分钟 _MACD 交叉 _ 联立 V230520_ 看多 _ 零轴下方 _ 任意 _0’)

Parameters

- **cat** –CzscSignals 对象
- **kwargs** –参数字典

Returns

返回信号结果

clv_up_dw_line_V230605

czsc.signals.**clv_up_dw_line_V230605**(*c:* CZSC, ***kwargs*) → OrderedDict

CLV 多空分类，贡献者：琅盎

参数模板：”{freq}_D{di}N{n}_CLV 多空 V230605”

信号逻辑：

CLV 用来衡量收盘价在最低价和最高价之间的位置。当 CLOSE=HIGH 时，CLV=1; 当 CLOSE=LOW 时，CLV=-1; 当 CLOSE 位于 HIGH 和 LOW 的中点时，CLV=0。CLV>0（<0），说明收盘价离最高（低）价更近。我们用 CLVMA 上穿/下穿 0 来产生买入/卖出信号

信号列表：

- Signal(‘日线 _D1N70_CLV 多空 V230605_ 看多 _ 任意 _ 任意 _0’)
- Signal(‘日线 _D1N70_CLV 多空 V230605_ 看空 _ 任意 _ 任意 _0’)

Parameters

- **c** –CZSC 对象
- **kwargs** –参数字典 - :param di: 信号计算截止倒数第 i 根 K 线 - :param n: 获取 K 线的根数，默认为 60

Returns

信号识别结果

cmo_up_dw_line_V230605

czsc.signals.**cmo_up_dw_line_V230605**(*c:* CZSC, ***kwargs*) → OrderedDict

CMO 能量异动，贡献者：琅盎

参数模板："{freq}_D{di}N{n}M{m}_CMO 能量 V230605"

信号逻辑：**

CMO 指标用过去 N 天的价格上涨量和价格下跌量得到，CMO>(<)0 表示当前处于上涨（下跌）趋势，CMO 越大（小）则当前上涨（下跌）趋势越强。我们用 CMO 上穿 30/下穿-30 来产生买入/卖出信号。

信号列表：

- Signal('30 分钟 _D1N70M30_CMO 能量 V230605_ 看空 _ 任意 _ 任意 _0')
- Signal('30 分钟 _D1N70M30_CMO 能量 V230605_ 看多 _ 任意 _ 任意 _0')

Parameters

- **c** – CZSC 对象
- **kwargs** – 参数字典 - :param di: 信号计算截止倒数第 i 根 K 线 - :param n: 获取 K 线的根数，默认为 60 - :param m: 信号预警轴，默认为 30

Returns

信号识别结果

coo_cci_V230323

czsc.signals.**coo_cci_V230323**(*c:* CZSC, ***kwargs*) → OrderedDict

CCI 结合均线的多空信号

参数模板："{freq}_D{di}CCI{n}#{ma_type}#{m}_BS 辅助 V230323"

信号逻辑：

1. CCI 大于 100，且向上突破均线，看多；
2. CCI 小于-100，且向下突破均线，看空；

信号列表：

- Signal('15 分钟 _D1CCI20#SMA#5_BS 辅助 V230323_ 空头 _ 向下 _ 任意 _0')
- Signal('15 分钟 _D1CCI20#SMA#5_BS 辅助 V230323_ 空头 _ 向上 _ 任意 _0')
- Signal('15 分钟 _D1CCI20#SMA#5_BS 辅助 V230323_ 多头 _ 向上 _ 任意 _0')
- Signal('15 分钟 _D1CCI20#SMA#5_BS 辅助 V230323_ 多头 _ 向下 _ 任意 _0')

Parameters

- **c** –CZSC 对象
- **kwargs** –参数字典 - :param di: 信号计算截止倒数第 i 根 K 线 - :param n: CCI 的计算周期 - :param m: 乘以 N 表示均线的计算周期

Returns

返回信号结果

coo_kdj_V230322

czsc.signals.**coo_kdj_V230322**(*c:* CZSC, ***kwargs*) → OrderedDict

均线判定方向，KD 决定进场时机

参数模板：”{freq}_D{di}KDJ{fastk_period}#{slowk_period}#{slowd_period}#{ma_type}#{n}_BS 辅助 V230322”

信号逻辑：

1. K 线向上突破均线，且 K < D，看多；
2. K 线向下突破均线，且 K > D，看空；

信号列表：

- Signal(‘15 分钟 _D1KDJ9#3#3#EMA#20_BS 辅助 V230322_ 空头 _ 任意 _ 任意 _0’)
- Signal(‘15 分钟 _D1KDJ9#3#3#EMA#20_BS 辅助 V230322_ 多头 _ 任意 _ 任意 _0’)

Parameters

- **c** –CZSC 对象
- **kwargs** –

  参数字典

  - **param di**
    信号计算截止倒数第 i 根 K 线
  - **param n**
    均线计算周期
  - **param fastk_period**
    kdj 参数
  - **param slowk_period**
    kdj 参数
  - **param slowd_period**
    kdj 参数

return
返回信号结果

coo_sar_V230325

czsc.signals.**coo_sar_V230325**(*c:* CZSC, ***kwargs*) → OrderedDict

SAR 和高低点结合判断买卖时机

参数模板："{freq}_D{di}N{n}SAR_BS 辅助 V230325"

信号逻辑：

1. 最高价大于 N 个周期收盘价的最高价，收盘价在 SAR 上方，看多；
2. 最低价小于 N 个周期收盘价的最低价，收盘价在 SAR 下方，看空；

信号列表：

- Signal('15 分钟 _D1N20SAR_BS 辅助 V230325_ 空头 _ 任意 _ 任意 _0')
- Signal('15 分钟 _D1N20SAR_BS 辅助 V230325_ 多头 _ 任意 _ 任意 _0')

Parameters

- **c** –CZSC 对象
- **kwargs** –

  参数字典

  - **param di**
    信号计算截止倒数第 i 根 K 线
  - **param n**
    信号计算的 K 线数量

return
返回信号结果

coo_td_V221110

czsc.signals.**coo_td_V221110**(*c:* CZSC, ***kwargs*) → OrderedDict

获取倒数第 i 根 K 线的 TD 信号

参数模板："{freq}_D{di}K_TD"

信号列表：

- Signal('60 分钟 _D2K_TD_ 延续 _ 非底 _ 任意 _0')
- Signal('60 分钟 _D2K_TD_ 延续 _ 非顶 _ 任意 _0')
- Signal('60 分钟 _D2K_TD_ 延续 _TD 顶 _ 任意 _0')
- Signal('60 分钟 _D2K_TD_ 看空 _ 非底 _ 任意 _0')

- Signal(‘60 分钟 _D2K_TD_ 延续 _TD 底 _ 任意 _0’)
- Signal(‘60 分钟 _D2K_TD_ 看多 _ 非顶 _ 任意 _0’)

Parameters

- **c** –CZSC 对象
- **di** –倒数第 di 根 K 线

Returns

s

coo_td_V221111

czsc.signals.**coo_td_V221111**(*c:* CZSC, ***kwargs*) → OrderedDict

获取倒数第 i 根 K 线的 TD 信号

参数模板:”{freq}_D{di}TD_BS 辅助 V221111”

信号逻辑:

神奇九转指标

信号列表:

- Signal(‘15 分钟 _D1TD_BS 辅助 V221111_ 延续 _TD 底 _ 任意 _0’)
- Signal(‘15 分钟 _D1TD_BS 辅助 V221111_ 看多 _ 非顶 _ 任意 _0’)
- Signal(‘15 分钟 _D1TD_BS 辅助 V221111_ 延续 _ 非顶 _ 任意 _0’)
- Signal(‘15 分钟 _D1TD_BS 辅助 V221111_ 延续 _ 非底 _ 任意 _0’)
- Signal(‘15 分钟 _D1TD_BS 辅助 V221111_ 延续 _TD 顶 _ 任意 _0’)
- Signal(‘15 分钟 _D1TD_BS 辅助 V221111_ 看空 _ 非底 _ 任意 _0’)

Parameters

- **c** –CZSC 对象
- **di** –倒数第 di 根 K 线

Returns

s

cvolp_up_dw_line_V230612

czsc.signals.**cvolp_up_dw_line_V230612**(*c:* CZSC, ***kwargs*) → OrderedDict

CVOLP 动量变化率指标，贡献者：琅盎

参数模板:”{freq}_D{di}N{n}M{m}UP{up}DW{dw}_CVOLP 动量变化率 V230612”

信号逻辑:

成交量移动平均平滑变化率。先计算了成交量的 N 周期指数移动平均线，然后计算了 EMAP 的 M 周期前的值，最后计算了 CVOLP 的值。CVOLP 上穿 up 买入，下穿 dw 卖出。

信号列表：

- Signal（‘日线 _D1N13M21UP5DW5_CVOLP 动量变化率 V230612_ 看多 _ 任意 _ 任意 _0’）
- Signal（‘日线 _D1N13M21UP5DW5_CVOLP 动量变化率 V230612_ 看空 _ 任意 _ 任意 _0’）

Parameters

- **c** –CZSC 对象
- **kwargs** –参数字典 - :param di: 信号计算截止倒数第 i 根 K 线 - :param n: 取 K 线数量 - :param m: 信号预警值 - :param up: 看多信号预警值 - :param dw: 看空信号预警值

Returns

信号识别结果

cxt_bi_base_V230228

czsc.signals.**cxt_bi_base_V230228**(*c:* CZSC, ***kwargs*) → OrderedDict

BI 基础信号

参数模板：”{freq}_D0BL{bi_init_length}_V230228”

信号逻辑：

1. 取最后一个笔，最后一笔向下，则当前笔向上，最后一笔向上，则当前笔向下；
2. 根据延伸 K 线数量判断当前笔的状态，中继或转折。

信号列表：

- Signal（‘15 分钟 _D0BL9_V230228_ 向下 _ 中继 _ 任意 _0’）
- Signal（‘15 分钟 _D0BL9_V230228_ 向上 _ 转折 _ 任意 _0’）
- Signal（‘15 分钟 _D0BL9_V230228_ 向下 _ 转折 _ 任意 _0’）
- Signal（‘15 分钟 _D0BL9_V230228_ 向上 _ 中继 _ 任意 _0’）

Parameters

- **c** –CZSC 对象
- **kwargs** –

Returns

信号识别结果

cxt_bi_end_V230104

czsc.signals.**cxt_bi_end_V230104**(*c:* CZSC, ***kwargs*) → OrderedDict

单均线辅助判断笔结束

参数模板："{freq}_D0{ma_type}#{timeperiod}T{th}_BE 辅助 V230104"

信号逻辑：

1. 向下笔底分型，连续三根阳线跨越 SMA5 超过一定阈值，且最后一根阳线收盘价在 SMA5 上方，向下笔结束；
2. 向上笔顶分型，连续三根阴线跨越 SMA5 超过一定阈值，且最后一根阴线收盘价在 SMA5 下方，向上笔结束。

信号列表：

- Signal('15 分钟 _D0SMA#5T50_BE 辅助 V230104_ 看多 _ 任意 _ 任意 _0')
- Signal('15 分钟 _D0SMA#5T50_BE 辅助 V230104_ 看空 _ 任意 _ 任意 _0')

Notes：

1. BE 是 Bi End 的缩写

Parameters

- **c** –CZSC 对象
- **kwargs** –ma_type: 均线类型，timeperiod: 均线周期，th: 距离 SMA5 均线的阈值，单位：BP

Returns

信号识别结果

cxt_bi_end_V230105

czsc.signals.**cxt_bi_end_V230105**(*c:* CZSC, ***kwargs*) → OrderedDict

K 线形态 + 均线辅助判断笔结束

参数模板："{freq}_D0{ma_type}#{timeperiod}T{th}_BE 辅助 V230105"

信号逻辑：

1. 向下笔底分型右侧两根 K 线，第一根阴线，第二根 K 线阳线，且收盘价超过均线一定阈值，向下笔结束。
2. 反之，向上笔结束。

信号列表：

- Signal('15 分钟 _D0SMA#5T50_BE 辅助 V230105_ 看多 _ 任意 _ 任意 _0')
- Signal('15 分钟 _D0SMA#5T50_BE 辅助 V230105_ 看空 _ 任意 _ 任意 _0')

Notes：

1. BE 是 Bi End 的缩写

Parameters

- **c** –CZSC 对象
- **kwargs** –ma_type: 均线类型，timeperiod: 均线周期，th: 距离 SMA5 均线的阈值，单位：BP

Returns

信号识别结果

cxt_bi_end_V230222

czsc.signals.**cxt_bi_end_V230222**(*c:* CZSC, ***kwargs*) → OrderedDict

当前是最后笔的第几次新低底分型或新高顶分型，用于笔结束辅助

参数模板："{freq}_D1MO{max_overlap}_BE 辅助 V230222"

信号逻辑：

1. 取最后笔及未成笔的分型，

2. 当前如果是顶分型，则看当前顶分型是否新高，是第几个新高 2. 当前如果是底分型，则看当前底分型是否新低，是第几个新低

信号列表：

- Signal('日线 _D1MO3_BE 辅助 V230222_ 新低 _ 第 2 次 _ 任意 _0')
- Signal('日线 _D1MO3_BE 辅助 V230222_ 新高 _ 第 2 次 _ 任意 _0')
- Signal('日线 _D1MO3_BE 辅助 V230222_ 新低 _ 第 3 次 _ 任意 _0')
- Signal('日线 _D1MO3_BE 辅助 V230222_ 新低 _ 第 4 次 _ 任意 _0')
- Signal('日线 _D1MO3_BE 辅助 V230222_ 新高 _ 第 3 次 _ 任意 _0')
- Signal('日线 _D1MO3_BE 辅助 V230222_ 新高 _ 第 4 次 _ 任意 _0')
- Signal('日线 _D1MO3_BE 辅助 V230222_ 新高 _ 第 5 次 _ 任意 _0')
- Signal('日线 _D1MO3_BE 辅助 V230222_ 新低 _ 第 1 次 _ 任意 _0')
- Signal('日线 _D1MO3_BE 辅助 V230222_ 新高 _ 第 1 次 _ 任意 _0')
- Signal('日线 _D1MO3_BE 辅助 V230222_ 新低 _ 第 5 次 _ 任意 _0')

Parameters

- **c** –CZSC 对象
- **kwargs** –

Returns

信号识别结果

cxt_bi_end_V230224

czsc.signals.**cxt_bi_end_V230224**(*c:* CZSC, ***kwargs*)

量价配合的笔结束辅助

参数模板："{freq}_D1_BE 辅助 V230224"

信号逻辑：

1. 向下笔结束：fx_b 内最低的那根 K 线下影大于上影的两倍，同时 fx_b 内的平均成交量小于当前笔的平均成交量的 0.618
2. 向上笔结束：fx_b 内最高的那根 K 线上影大于下影的两倍，同时 fx_b 内的平均成交量大于当前笔的平均成交量的 2 倍

信号列表：

- Signal('15 分钟 _D1_BE 辅助 V230224_ 看多 _ 任意 _ 任意 _0')
- Signal('15 分钟 _D1_BE 辅助 V230224_ 看空 _ 任意 _ 任意 _0')

Parameters

c –CZSC 对象

Returns

信号字典

cxt_bi_end_V230312

czsc.signals.**cxt_bi_end_V230312**(*c:* CZSC, ***kwargs*)

MACD 辅助判断笔结束信号

参数模板："{freq}_D0MACD{fastperiod}#{slowperiod}#{signalperiod}_BE 辅助 V230312"

信号逻辑：

1. 看多，当下笔向下，笔的最后一个分型 MACD 向上
2. 反之，看空，当下笔向上，笔的最后一个分型 MACD 向下

信号列表：

- Signal('15 分钟 _D0MACD12#26#9_BE 辅助 V230312_ 看多 _ 任意 _ 任意 _0')
- Signal('15 分钟 _D0MACD12#26#9_BE 辅助 V230312_ 看空 _ 任意 _ 任意 _0')

Notes：

1. BE 是 Bi End 的缩写

Parameters

c –CZSC 对象

Returns

信号识别结果

cxt_bi_end_V230320

czsc.signals.**cxt_bi_end_V230320**(*c:* CZSC, ***kwargs*) → OrderedDict

100 以内质数时序窗口辅助笔结束判断

参数模板："{freq}_D0 质数窗口 MO{max_overlap}_BE 辅助 V230320"

信号逻辑：

1. 未完成笔延伸长度等于某个质数，且最后 3 根 K 线创新高，或者新低，笔结束

信号列表：

- Signal('15 分钟 _D0 质数窗口 MO3_BE 辅助 V230320_ 看多 _17K_ 任意 _0')
- Signal('15 分钟 _D0 质数窗口 MO3_BE 辅助 V230320_ 看多 _23K_ 任意 _0')
- Signal('15 分钟 _D0 质数窗口 MO3_BE 辅助 V230320_ 看多 _29K_ 任意 _0')
- Signal('15 分钟 _D0 质数窗口 MO3_BE 辅助 V230320_ 看多 _11K_ 任意 _0')
- Signal('15 分钟 _D0 质数窗口 MO3_BE 辅助 V230320_ 看多 _13K_ 任意 _0')
- Signal('15 分钟 _D0 质数窗口 MO3_BE 辅助 V230320_ 看多 _19K_ 任意 _0')
- Signal('15 分钟 _D0 质数窗口 MO3_BE 辅助 V230320_ 看多 _37K_ 任意 _0')
- Signal('15 分钟 _D0 质数窗口 MO3_BE 辅助 V230320_ 看多 _41K_ 任意 _0')
- Signal('15 分钟 _D0 质数窗口 MO3_BE 辅助 V230320_ 看空 _13K_ 任意 _0')
- Signal('15 分钟 _D0 质数窗口 MO3_BE 辅助 V230320_ 看空 _11K_ 任意 _0')
- Signal('15 分钟 _D0 质数窗口 MO3_BE 辅助 V230320_ 看空 _17K_ 任意 _0')
- Signal('15 分钟 _D0 质数窗口 MO3_BE 辅助 V230320_ 看空 _19K_ 任意 _0')
- Signal('15 分钟 _D0 质数窗口 MO3_BE 辅助 V230320_ 看空 _23K_ 任意 _0')
- Signal('15 分钟 _D0 质数窗口 MO3_BE 辅助 V230320_ 看空 _37K_ 任意 _0')
- Signal('15 分钟 _D0 质数窗口 MO3_BE 辅助 V230320_ 看多 _31K_ 任意 _0')
- Signal('15 分钟 _D0 质数窗口 MO3_BE 辅助 V230320_ 看空 _29K_ 任意 _0')
- Signal('15 分钟 _D0 质数窗口 MO3_BE 辅助 V230320_ 看空 _31K_ 任意 _0')
- Signal('15 分钟 _D0 质数窗口 MO3_BE 辅助 V230320_ 看空 _41K_ 任意 _0')
- Signal('15 分钟 _D0 质数窗口 MO3_BE 辅助 V230320_ 看空 _43K_ 任意 _0')
- Signal('15 分钟 _D0 质数窗口 MO3_BE 辅助 V230320_ 看空 _47K_ 任意 _0')
- Signal('15 分钟 _D0 质数窗口 MO3_BE 辅助 V230320_ 看多 _43K_ 任意 _0')
- Signal('15 分钟 _D0 质数窗口 MO3_BE 辅助 V230320_ 看空 _53K_ 任意 _0')
- Signal('15 分钟 _D0 质数窗口 MO3_BE 辅助 V230320_ 看空 _59K_ 任意 _0')
- Signal('15 分钟 _D0 质数窗口 MO3_BE 辅助 V230320_ 看空 _61K_ 任意 _0')

Parameters

c –CZSC 对象

Returns

信号识别结果

cxt_bi_end_V230322

czsc.signals.**cxt_bi_end_V230322**(*c:* CZSC, ***kwargs*) → OrderedDict

分型配合均线辅助判断笔的结束

参数模板："{freq}_D0 分型配合 {ma_type}#{timeperiod}_BE 辅助 V230322"

信号逻辑：

1. 对于顶分型，如果最右边的 k 线的 MA 最小，这是向下笔正常延伸的情况
2. 对于底分型，如果最右边的 k 线的 MA 最大，这是向上笔正常延伸的情况

信号列表：

- Signal('15 分钟 _D0 分型配合 SMA#5_BE 辅助 V230322_ 看多 _ 同向分型 _ 任意 _0')
- Signal('15 分钟 _D0 分型配合 SMA#5_BE 辅助 V230322_ 看空 _ 反向分型 _ 任意 _0')
- Signal('15 分钟 _D0 分型配合 SMA#5_BE 辅助 V230322_ 看多 _ 反向分型 _ 任意 _0')
- Signal('15 分钟 _D0 分型配合 SMA#5_BE 辅助 V230322_ 看空 _ 同向分型 _ 任意 _0')

Parameters

c –CZSC 对象

Returns

信号识别结果

cxt_bi_end_V230324

czsc.signals.**cxt_bi_end_V230324**(*c:* CZSC, ***kwargs*) → OrderedDict

笔结束分型的均线突破判断笔的结束

参数模板："{freq}_D0{ma_type}#{timeperiod} 均线突破 _BE 辅助 V230324"

信号逻辑：

1. 向上笔最后一个顶分型左边两个 k 线的 MA 最小值被收盘价突破，向上笔结束
2. 向下笔最后一个底分型左边两个 k 线的 MA 最大值被收盘价突破，向下笔结束

信号列表：

- Signal('15 分钟 _D0SMA#5 均线突破 _BE 辅助 V230324_ 看空 _ 任意 _ 任意 _0')
- Signal('15 分钟 _D0SMA#5 均线突破 _BE 辅助 V230324_ 看多 _ 任意 _ 任意 _0')

Parameters

c –CZSC 对象

Returns

信号识别结果

cxt_bi_end_V230618

czsc.signals.**cxt_bi_end_V230618**(*c:* CZSC, ***kwargs*) → OrderedDict

笔结束辅助判断

参数模板："{freq}_D{di}MO{max_overlap}_BE 辅助 V230618"

信号逻辑：

以向下笔为例，判断笔内是否有小级别中枢，如果有则看多：

1. 笔内任意两根 k 线的重叠使该价格位的计数加 1，计算从笔.high 到笔.low 之间各价格位的重叠次数
2. **通过各价格位的重叠可以得到横轴价格，纵轴重叠次数的图，通过计算途中波峰的个数来得到近似的小中枢个数**
   例子：横轴从小到大对应的重叠次数为 111223321111213333444553321，则可以通过计算从 n 变为 1 的次数来得到波峰个数这里 2-1，2-1，2-1，得到波峰数为 3

信号列表：

- Signal('日线 _D1MO1_BE 辅助 V230618_ 看多 _1 小中枢 _ 任意 _0')
- Signal('日线 _D1MO1_BE 辅助 V230618_ 看空 _3 小中枢 _ 任意 _0')
- Signal('日线 _D1MO1_BE 辅助 V230618_ 看空 _2 小中枢 _ 任意 _0')
- Signal('日线 _D1MO1_BE 辅助 V230618_ 看空 _1 小中枢 _ 任意 _0')
- Signal('日线 _D1MO1_BE 辅助 V230618_ 看多 _2 小中枢 _ 任意 _0')
- Signal('日线 _D1MO1_BE 辅助 V230618_ 看空 _5 小中枢 _ 任意 _0')
- Signal('日线 _D1MO1_BE 辅助 V230618_ 看空 _4 小中枢 _ 任意 _0')
- Signal('日线 _D1MO1_BE 辅助 V230618_ 看多 _3 小中枢 _ 任意 _0')

信号说明：

类似 cxt_third_bs_V230318 信号，但增加了笔内有无小级别中枢的判断。用 k 线重叠来近似小级别中枢的判断

Parameters

- **c** –CZSC 对象
- **kwargs** –
  - di: int, 默认 1，表示取倒数第几笔
  - max_overlap: int, 默认 3，表示笔内最多允许有几个信号重叠

Returns

信号识别结果

cxt_bi_end_V230815

czsc.signals.**cxt_bi_end_V230815**(*c:* CZSC, ***kwargs*) → OrderedDict

一两根 K 线快速突破反向笔

参数模板："{freq}_ 快速突破 _BE 辅助 V230815"

信号逻辑：

以向上笔为例：右侧分型完成后第一根 K 线的最低价低于该笔的最低价，认为向上笔结束，反向向下笔开始。

信号列表：

- Signal('15 分钟 _ 快速突破 _BE 辅助 V230815_ 向下 _ 任意 _ 任意 _0')
- Signal('15 分钟 _ 快速突破 _BE 辅助 V230815_ 向上 _ 任意 _ 任意 _0')

Parameters

- **c** –CZSC 对象
- **kwargs** –

Returns

信号识别结果

cxt_bi_status_V230101

czsc.signals.**cxt_bi_status_V230101**(*c:* CZSC, ***kwargs*) → OrderedDict

笔的表里关系

参数模板："{freq}_D1_ 表里关系 V230101"

表里关系的定义参考：http://blog.sina.com.cn/s/blog_486e105c01007wc1.html

信号逻辑：

1. 最后一笔向下，且未完成笔的长度大于 7 根 K 线，表里关系为向上，否则为向下；
2. 最后一笔向上，且未完成笔的长度大于 7 根 K 线，表里关系为向下，否则为向上；
3. 向下的笔遇到底分型，表里关系为底分；向上笔的遇到底分型为延伸；
4. 向上的笔遇到顶分型，表里关系为顶分；向下笔的遇到顶分型为延伸。

信号列表：

- Signal('15 分钟 _D1_ 表里关系 V230101_ 向下 _ 延伸 _ 任意 _0')
- Signal('15 分钟 _D1_ 表里关系 V230101_ 向下 _ 底分 _ 任意 _0')
- Signal('15 分钟 _D1_ 表里关系 V230101_ 向上 _ 顶分 _ 任意 _0')
- Signal('15 分钟 _D1_ 表里关系 V230101_ 向上 _ 延伸 _ 任意 _0')

Parameters

c –CZSC 对象

Returns

信号字典

cxt_bi_status_V230102

czsc.signals.**cxt_bi_status_V230102**(*c:* CZSC, ***kwargs*) → OrderedDict

笔的表里关系

参数模板："{freq}_D1_ 表里关系 V230102"

表里关系的定义参考：http://blog.sina.com.cn/s/blog_486e105c01007wc1.html

信号逻辑：

1. 最后一笔向下，且未完成笔的长度大于 7 根 K 线，表里关系为向上，否则为向下；
2. 最后一笔向上，且未完成笔的长度大于 7 根 K 线，表里关系为向下，否则为向上；
3. 向下的笔遇到底分型，表里关系为底分；向上笔的遇到底分型为延伸；
4. 向上的笔遇到顶分型，表里关系为顶分；向下笔的遇到顶分型为延伸。

信号列表：

- Signal('15 分钟 _D1_ 表里关系 V230102_ 向下 _ 底分 _ 任意 _0')
- Signal('15 分钟 _D1_ 表里关系 V230102_ 向下 _ 延伸 _ 任意 _0')
- Signal('15 分钟 _D1_ 表里关系 V230102_ 向上 _ 顶分 _ 任意 _0')
- Signal('15 分钟 _D1_ 表里关系 V230102_ 向上 _ 延伸 _ 任意 _0')

注意：与 cxt_bi_status_V230101 的区别在于，该信号只在分型成立的最后一根 K 线触发，而不是每根 K 线都会触发。

Parameters

c –CZSC 对象

Returns

信号字典

cxt_bi_stop_V230815

czsc.signals.**cxt_bi_stop_V230815**(*c:* CZSC, ***kwargs*) → OrderedDict

定位笔的止损距离大小

参数模板："{freq}_ 距离 {th}BP_ 止损 V230815"

信号逻辑：

以向上笔为例：如果当前 K 线的收盘价高于该笔的最高价的 1 - 0.5%，则认为在止损阈值内，否则认为在止损阈值外。

信号列表：

- Signal('15 分钟 _ 距离 50BP_ 止损 V230815_ 向下 _ 阈值外 _ 任意 _0')
- Signal('15 分钟 _ 距离 50BP_ 止损 V230815_ 向上 _ 阈值内 _ 任意 _0')
- Signal('15 分钟 _ 距离 50BP_ 止损 V230815_ 向下 _ 阈值内 _ 任意 _0')

- Signal(‘15 分钟 _ 距离 50BP_ 止损 V230815_ 向上 _ 阈值外 _ 任意 _0’)

  Parameters

  - **c** –CZSC 对象
  - **kwargs** –
    - th: 止损距离阈值，单位为 BP, 默认为 50BP, 即 0.5%

  Returns

  信号识别结果

cxt_bi_trend_V230824

czsc.signals.**cxt_bi_trend_V230824**(*c:* CZSC, ***kwargs*) → OrderedDict

判断 N 笔形态，贡献者：chenglei

参数模板：”{freq}_D{di}N{n}TH{th}_ 形态 V230824”

信号逻辑：

1. 通过对最近 N 笔的中心点的均值和-n 笔的中心点的位置关系来判断当前 N 比是上涨形态还是下跌，横盘震荡形态
2. 给定阈值 th，判断上涨下跌横盘按照所有笔中心点/第-n 笔中心点与正负 th 区间的相对位置来判断。
3. 当在区间上时为上涨，区间内为横盘，区间下为下跌

信号列表：

- Signal(‘日线 _D1N4TH5_ 形态 V230824_ 横盘 _ 任意 _ 任意 _0’)
- Signal(‘日线 _D1N4TH5_ 形态 V230824_ 向上 _ 任意 _ 任意 _0’)
- Signal(‘日线 _D1N4TH5_ 形态 V230824_ 向下 _ 任意 _ 任意 _0’)

  Parameters

  - **c** –CZSC 对象
  - **kwargs** –
    - di: 倒数第几笔
    - n： 检查范围
    - th: 振幅阈值，2 表示 2%，即 2% 以内的振幅都认为是震荡

  Returns

  信号识别结果

cxt_bi_trend_V230913

czsc.signals.**cxt_bi_trend_V230913**(*c:* CZSC, ***kwargs*) → OrderedDict

辅助判断股票通道信号，贡献者：马鸣

参数模板："{freq}_D{di}N{n} 笔趋势 _ 高低点辅助判断 V230913"

信号逻辑：

1. 倒数 di 笔之间的高低点形成趋势线，根据股价的当前位置，推断趋势强弱

信号列表：

- Signal('日线 _D3N1 笔趋势 _ 高低点辅助判断 V230913_ 下降趋势 _ 超强 _ 任意 _0')
- Signal('日线 _D3N1 笔趋势 _ 高低点辅助判断 V230913_ 观望 _ 末笔延伸 _ 任意 _0')
- Signal('日线 _D3N1 笔趋势 _ 高低点辅助判断 V230913_ 上升趋势 _ 强 _ 任意 _0')
- Signal('日线 _D3N1 笔趋势 _ 高低点辅助判断 V230913_ 观望 _ 趋势线交叉 _ 任意 _0')
- Signal('日线 _D3N1 笔趋势 _ 高低点辅助判断 V230913_ 下降趋势 _ 强 _ 任意 _0')
- Signal('日线 _D3N1 笔趋势 _ 高低点辅助判断 V230913_ 上升趋势 _ 超强 _ 任意 _0')

Parameters

- **c** –CZSC 对象
- **kwargs** –参数字典

  -:param di: 倒数 di 笔 -:param n: 倒数第 n 根 K 线

Returns

信号识别结果

cxt_bi_zdf_V230601

czsc.signals.**cxt_bi_zdf_V230601**(*c:* CZSC, ***kwargs*) → OrderedDict

BI 涨跌幅的分层判断

参数模板："{freq}_D{di}N{n}_ 分层 V230601"

信号逻辑：

取最近 50 个缠论笔，计算涨跌幅，分 N 层判断。

信号列表：

- Signal('60 分钟 _D1N5_ 分层 V230601_ 向下 _ 第 5 层 _ 任意 _0')
- Signal('60 分钟 _D1N5_ 分层 V230601_ 向上 _ 第 5 层 _ 任意 _0')
- Signal('60 分钟 _D1N5_ 分层 V230601_ 向下 _ 第 3 层 _ 任意 _0')
- Signal('60 分钟 _D1N5_ 分层 V230601_ 向上 _ 第 2 层 _ 任意 _0')
- Signal('60 分钟 _D1N5_ 分层 V230601_ 向上 _ 第 4 层 _ 任意 _0')
- Signal('60 分钟 _D1N5_ 分层 V230601_ 向下 _ 第 2 层 _ 任意 _0')
- Signal('60 分钟 _D1N5_ 分层 V230601_ 向上 _ 第 1 层 _ 任意 _0')
- Signal('60 分钟 _D1N5_ 分层 V230601_ 向下 _ 第 1 层 _ 任意 _0')
- Signal('60 分钟 _D1N5_ 分层 V230601_ 向上 _ 第 3 层 _ 任意 _0')

- Signal(‘60 分钟 _D1N5_ 分层 V230601_ 向下 _ 第 4 层 _ 任意 _0’)

Parameters

- **c** –CZSC 对象
- **kwargs** –参数字典 - di: 倒数第几根 K 线 - n: 取截止 dik 的前 n 根 K 线

Returns

返回信号结果

cxt_double_zs_V230311

czsc.signals.**cxt_double_zs_V230311**(*c:* CZSC, ***kwargs*)

两个中枢组合辅助判断 BS1，贡献者：韩知辰

参数模板：”{freq}_D{di} 双中枢 _BS1 辅助 V230311”

信号逻辑：

1. 最后一笔向下，最近两个中枢依次向下，最后一个中枢的倒数第一笔的 K 线长度大于倒数第二笔的 K 线长度，看多；
2. 最后一笔向上，最近两个中枢依次向上，最后一个中枢的倒数第一笔的 K 线长度大于倒数第二笔的 K 线长度，看空；

信号列表：

- Signal(‘15 分钟 _D1 双中枢 _BS1 辅助 V230311_ 看多 _ 任意 _ 任意 _0’)
- Signal(‘15 分钟 _D1 双中枢 _BS1 辅助 V230311_ 看空 _ 任意 _ 任意 _0’)

Parameters

- **c** –CZSC 对象
- **di** –倒数第 di 笔

Returns

s

cxt_eleven_bi_V230622

czsc.signals.**cxt_eleven_bi_V230622**(*c:* CZSC, ***kwargs*) → OrderedDict

十一笔形态分类

参数模板：”{freq}_D{di} 十一笔 _ 形态 V230622”

信号逻辑：

十一笔的形态分类

信号列表：

- Signal(‘60 分钟 _D1 十一笔 _ 形态 V230622_ 类三买 _ 任意 _ 任意 _0’)
- Signal(‘60 分钟 _D1 十一笔 _ 形态 V230622_A3B3C5 式类一卖 _ 任意 _ 任意 _0’)

- Signal(‘60 分钟 _D1 十一笔 _ 形态 V230622_ 类二买 _ 任意 _ 任意 _0’)
- Signal(‘60 分钟 _D1 十一笔 _ 形态 V230622_A5B3C3 式类一卖 _ 任意 _ 任意 _0’)

Parameters

- **c** –CZSC 对象
- **kwargs** –
  - di: 倒数第几笔

Returns

信号识别结果

cxt_first_buy_V221126

czsc.signals.**cxt_first_buy_V221126**(*c:* CZSC, ***kwargs*) → OrderedDict

一买信号

参数模板:”{freq}_D{di}B_BUY1”

信号列表:

- Signal(‘15 分钟 _D1B_BUY1_ 一买 _5 笔 _ 任意 _0’)
- Signal(‘15 分钟 _D1B_BUY1_ 一买 _11 笔 _ 任意 _0’)
- Signal(‘15 分钟 _D1B_BUY1_ 一买 _7 笔 _ 任意 _0’)
- Signal(‘15 分钟 _D1B_BUY1_ 一买 _21 笔 _ 任意 _0’)
- Signal(‘15 分钟 _D1B_BUY1_ 一买 _17 笔 _ 任意 _0’)
- Signal(‘15 分钟 _D1B_BUY1_ 一买 _19 笔 _ 任意 _0’)
- Signal(‘15 分钟 _D1B_BUY1_ 一买 _9 笔 _ 任意 _0’)
- Signal(‘15 分钟 _D1B_BUY1_ 一买 _15 笔 _ 任意 _0’)
- Signal(‘15 分钟 _D1B_BUY1_ 一买 _13 笔 _ 任意 _0’)

Parameters

- **c** –CZSC 对象
- **kwargs** –
  - di: 倒数第 di 个笔

Returns

信号字典

cxt_first_sell_V221126

czsc.signals.**cxt_first_sell_V221126**(*c:* CZSC, ***kwargs*) → OrderedDict

一卖信号

参数模板："{freq}_D{di}B_SELL1"

信号列表：

- Signal('15 分钟 _D1B_SELL1_ 一卖 _17 笔 _ 任意 _0')
- Signal('15 分钟 _D1B_SELL1_ 一卖 _15 笔 _ 任意 _0')
- Signal('15 分钟 _D1B_SELL1_ 一卖 _5 笔 _ 任意 _0')
- Signal('15 分钟 _D1B_SELL1_ 一卖 _7 笔 _ 任意 _0')
- Signal('15 分钟 _D1B_SELL1_ 一卖 _9 笔 _ 任意 _0')
- Signal('15 分钟 _D1B_SELL1_ 一卖 _19 笔 _ 任意 _0')
- Signal('15 分钟 _D1B_SELL1_ 一卖 _21 笔 _ 任意 _0')
- Signal('15 分钟 _D1B_SELL1_ 一卖 _13 笔 _ 任意 _0')
- Signal('15 分钟 _D1B_SELL1_ 一卖 _11 笔 _ 任意 _0')

Parameters

- **c** –CZSC 对象
- **di** –CZSC 对象

Returns

信号字典

cxt_five_bi_V230619

czsc.signals.**cxt_five_bi_V230619**(*c:* CZSC, ***kwargs*) → OrderedDict

五笔形态分类

参数模板："{freq}_D{di} 五笔 _ 形态 V230619"

信号逻辑：

五笔的形态分类

信号列表：

- Signal('60 分钟 _D1 五笔 _ 形态 V230619_ 上颈线突破 _ 任意 _ 任意 _0')
- Signal('60 分钟 _D1 五笔 _ 形态 V230619_ 类三卖 _ 任意 _ 任意 _0')
- Signal('60 分钟 _D1 五笔 _ 形态 V230619_ 类趋势底背驰 _ 任意 _ 任意 _0')
- Signal('60 分钟 _D1 五笔 _ 形态 V230619_ 类趋势顶背驰 _ 任意 _ 任意 _0')
- Signal('60 分钟 _D1 五笔 _ 形态 V230619_ 下颈线突破 _ 任意 _ 任意 _0')
- Signal('60 分钟 _D1 五笔 _ 形态 V230619_ 类三买 _ 任意 _ 任意 _0')
- Signal('60 分钟 _D1 五笔 _ 形态 V230619_aAb 式顶背驰 _ 任意 _ 任意 _0')
- Signal('60 分钟 _D1 五笔 _ 形态 V230619_aAb 式底背驰 _ 任意 _ 任意 _0')

Parameters

- **c** –CZSC 对象
- **kwargs** –
  - di: 倒数第几笔

Returns

信号识别结果

cxt_fx_power_V221107

czsc.signals.**cxt_fx_power_V221107**(*c:* CZSC, ***kwargs*) → OrderedDict

倒数第 di 个分型的强弱

参数模板："{freq}_D{di}F_ 分型强弱"

信号列表：

- Signal('15 分钟 _D1F_ 分型强弱 _ 中顶 _ 有中枢 _ 任意 _0')
- Signal('15 分钟 _D1F_ 分型强弱 _ 弱底 _ 有中枢 _ 任意 _0')
- Signal('15 分钟 _D1F_ 分型强弱 _ 强顶 _ 有中枢 _ 任意 _0')
- Signal('15 分钟 _D1F_ 分型强弱 _ 弱顶 _ 有中枢 _ 任意 _0')
- Signal('15 分钟 _D1F_ 分型强弱 _ 强底 _ 有中枢 _ 任意 _0')
- Signal('15 分钟 _D1F_ 分型强弱 _ 中底 _ 有中枢 _ 任意 _0')
- Signal('15 分钟 _D1F_ 分型强弱 _ 强顶 _ 无中枢 _ 任意 _0')
- Signal('15 分钟 _D1F_ 分型强弱 _ 中顶 _ 无中枢 _ 任意 _0')
- Signal('15 分钟 _D1F_ 分型强弱 _ 弱底 _ 无中枢 _ 任意 _0')
- Signal('15 分钟 _D1F_ 分型强弱 _ 中底 _ 无中枢 _ 任意 _0')
- Signal('15 分钟 _D1F_ 分型强弱 _ 弱顶 _ 无中枢 _ 任意 _0')
- Signal('15 分钟 _D1F_ 分型强弱 _ 强底 _ 无中枢 _ 任意 _0')

Parameters

- **c** –CZSC 对象
- **di** –倒数第 di 个分型

Returns

cxt_intraday_V230701

czsc.signals.**cxt_intraday_V230701**(*cat:* CzscSignals, ***kwargs*) → OrderedDict

每日走势分类

参数模板："{freq1}#{freq2}_D{di} 日 _ 走势分类 V230701"

信号逻辑：

参见博客：https://blog.sina.com.cn/s/blog_486e105c010009uy.html

信号列表：

- Signal('30 分钟 # 日线 _D2 日 _ 走势分类 V230701_ 强平衡市 _ 任意 _ 任意 _0')
- Signal('30 分钟 # 日线 _D2 日 _ 走势分类 V230701_ 弱平衡市 _ 任意 _ 任意 _0')
- Signal('30 分钟 # 日线 _D2 日 _ 走势分类 V230701_ 双中枢下跌 _ 任意 _ 任意 _0')
- Signal('30 分钟 # 日线 _D2 日 _ 走势分类 V230701_ 转折平衡市 _ 任意 _ 任意 _0')
- Signal('30 分钟 # 日线 _D2 日 _ 走势分类 V230701_ 双中枢上涨 _ 任意 _ 任意 _0')

Parameters

c –CZSC 对象

Returns

信号识别结果

cxt_nine_bi_V230621

czsc.signals.**cxt_nine_bi_V230621**(*c:* CZSC, ***kwargs*) → OrderedDict

九笔形态分类

参数模板："{freq}_D{di} 九笔 _ 形态 V230621"

信号逻辑：

九笔的形态分类

信号列表：

- Signal('60 分钟 _D1 九笔 _ 形态 V230621_ 类三买 A_ 任意 _ 任意 _0')
- Signal('60 分钟 _D1 九笔 _ 形态 V230621_aAb 式类一卖 _ 任意 _ 任意 _0')
- Signal('60 分钟 _D1 九笔 _ 形态 V230621_ 类三卖 A_ 任意 _ 任意 _0')
- Signal('60 分钟 _D1 九笔 _ 形态 V230621_aAbcd 式类一买 _ 任意 _ 任意 _0')
- Signal('60 分钟 _D1 九笔 _ 形态 V230621_ABC 式类一卖 _ 任意 _ 任意 _0')
- Signal('60 分钟 _D1 九笔 _ 形态 V230621_aAbBc 式类一买 _ 任意 _ 任意 _0')
- Signal('60 分钟 _D1 九笔 _ 形态 V230621_aAbcd 式类一卖 _ 任意 _ 任意 _0')
- Signal('60 分钟 _D1 九笔 _ 形态 V230621_ZD 三卖 _ 任意 _ 任意 _0')
- Signal('60 分钟 _D1 九笔 _ 形态 V230621_aAbBc 式类一卖 _ 任意 _ 任意 _0')
- Signal('60 分钟 _D1 九笔 _ 形态 V230621_ABC 式类一买 _ 任意 _ 任意 _0')

Parameters

- **c** –CZSC 对象
- **kwargs** –
  - di: 倒数第几笔

Returns

信号识别结果

cxt_range_oscillation_V230620

czsc.signals.**cxt_range_oscillation_V230620**(*c:* CZSC, ***kwargs*) → OrderedDict

判断区间震荡

参数模板："{freq}_D{di}TH{th}_ 区间震荡 V230620"

信号逻辑：

1. 在区间震荡中，无论振幅大小，各笔的中心应改在相近的价格区间内平移，当各笔的中心的振幅大于一定数值时就认为这个窗口内没有固定区间的中枢震荡
2. 给定阈值 th，当各笔的中心的振幅大于 th 时，认为这个窗口内没有固定区间的中枢震荡

信号列表：

- Signal('日线 _D1TH5_ 区间震荡 V230620_2 笔震荡 _ 向下 _ 任意 _0')
- Signal('日线 _D1TH5_ 区间震荡 V230620_3 笔震荡 _ 向上 _ 任意 _0')
- Signal('日线 _D1TH5_ 区间震荡 V230620_4 笔震荡 _ 向下 _ 任意 _0')
- Signal('日线 _D1TH5_ 区间震荡 V230620_5 笔震荡 _ 向上 _ 任意 _0')
- Signal('日线 _D1TH5_ 区间震荡 V230620_6 笔震荡 _ 向下 _ 任意 _0')
- Signal('日线 _D1TH5_ 区间震荡 V230620_5 笔震荡 _ 向下 _ 任意 _0')
- Signal('日线 _D1TH5_ 区间震荡 V230620_2 笔震荡 _ 向上 _ 任意 _0')
- Signal('日线 _D1TH5_ 区间震荡 V230620_3 笔震荡 _ 向下 _ 任意 _0')
- Signal('日线 _D1TH5_ 区间震荡 V230620_4 笔震荡 _ 向上 _ 任意 _0')

Parameters

- **c** –CZSC 对象
- **kwargs** –
  - di: 倒数第几笔
  - th: 振幅阈值，2 表示 2%，即 2% 以内的振幅都认为是震荡

Returns

信号识别结果

cxt_second_bs_V230320

czsc.signals.**cxt_second_bs_V230320**(*c:* CZSC, ***kwargs*) → OrderedDict

均线辅助识别第二类买卖点

参数模板："{freq}_D{di}#{ma_type}#{timeperiod}_BS2 辅助 V230320"

信号逻辑：

1. 二买：1）123 笔序列向下，其中 1,3 笔的低点都在均线下方；2）5 的 fx_a 的均线值小于 fx_b 均线值
2. 二卖：1）123 笔序列向上，其中 1,3 笔的高点都在均线上方；2）5 的 fx_a 的均线值大于 fx_b 均线值

信号列表：

- Signal('15 分钟 _D1#SMA#21_BS2 辅助 V230320_ 二买 _ 任意 _ 任意 _0')
- Signal('15 分钟 _D1#SMA#21_BS2 辅助 V230320_ 二卖 _ 任意 _ 任意 _0')

Parameters

- **c** –CZSC 对象
- **di** –从最后一个笔的第几个开始识别
- **kwargs** –ma_type: 均线类型，timeperiod: 均线周期

Returns

信号识别结果

cxt_seven_bi_V230620

czsc.signals.**cxt_seven_bi_V230620**(*c:* CZSC, ***kwargs*) → OrderedDict

七笔形态分类

参数模板："{freq}_D{di} 七笔 _ 形态 V230620"

信号逻辑：

七笔的形态分类

信号列表：

- Signal('60 分钟 _D1 七笔 _ 形态 V230620_ 类三卖 _ 任意 _ 任意 _0')
- Signal('60 分钟 _D1 七笔 _ 形态 V230620_ 向上中枢完成 _ 任意 _ 任意 _0')
- Signal('60 分钟 _D1 七笔 _ 形态 V230620_aAbcd 式顶背驰 _ 任意 _ 任意 _0')
- Signal('60 分钟 _D1 七笔 _ 形态 V230620_ 类三买 _ 任意 _ 任意 _0')
- Signal('60 分钟 _D1 七笔 _ 形态 V230620_ 向下中枢完成 _ 任意 _ 任意 _0')
- Signal('60 分钟 _D1 七笔 _ 形态 V230620_aAb 式底背驰 _ 任意 _ 任意 _0')
- Signal('60 分钟 _D1 七笔 _ 形态 V230620_abcAd 式顶背驰 _ 任意 _ 任意 _0')
- Signal('60 分钟 _D1 七笔 _ 形态 V230620_abcAd 式底背驰 _ 任意 _ 任意 _0')
- Signal('60 分钟 _D1 七笔 _ 形态 V230620_aAb 式顶背驰 _ 任意 _ 任意 _0')

- Signal(‘60 分钟 _D1 七笔 _ 形态 V230620_ 类趋势顶背驰 _ 任意 _ 任意 _0’)
- Signal(‘60 分钟 _D1 七笔 _ 形态 V230620_aAbcd 式底背驰 _ 任意 _ 任意 _0’)

Parameters

- **c** –CZSC 对象
- **kwargs** –
  - di: 倒数第几笔

Returns

信号识别结果

cxt_third_bs_V230318

czsc.signals.**cxt_third_bs_V230318**(*c:* CZSC, ***kwargs*) → OrderedDict

均线辅助识别第三类买卖点

参数模板:”{freq}_D{di}#{ma_type}#{timeperiod}_BS3 辅助 V230318”

信号逻辑:

1. 三买：1）123 构成中枢，4 离开，5 回落不回中枢；2）均线新高
2. 三卖：1）123 构成中枢，4 离开，5 回升不回中枢；2）均线新低

信号列表:

- Signal(‘15 分钟 _D1#SMA#34_BS3 辅助 V230318_ 三卖 _ 任意 _ 任意 _0’)
- Signal(‘15 分钟 _D1#SMA#34_BS3 辅助 V230318_ 三买 _ 任意 _ 任意 _0’)

Parameters

- **c** –CZSC 对象
- **di** –从最后一个笔的第几个开始识别
- **kwargs** –ma_type: 均线类型，timeperiod: 均线周期

Returns

信号识别结果

cxt_third_bs_V230319

czsc.signals.**cxt_third_bs_V230319**(*c:* CZSC, ***kwargs*) → OrderedDict

均线辅助识别第三类买卖点，增加均线形态

参数模板:”{freq}_D{di}#{ma_type}#{timeperiod}_BS3 辅助 V230319”

信号逻辑:

1. 三买：1）123 构成中枢，4 离开，5 回落不回中枢；2）均线新高或均线底分
2. 三卖：1）123 构成中枢，4 离开，5 回升不回中枢；2）均线新低或均线顶分

信号列表:

- Signal(‘15 分钟 _D1#SMA#34_BS3 辅助 V230319_ 三卖 _ 均线新低 _ 任意 _0’)
- Signal(‘15 分钟 _D1#SMA#34_BS3 辅助 V230319_ 三买 _ 均线底分 _ 任意 _0’)
- Signal(‘15 分钟 _D1#SMA#34_BS3 辅助 V230319_ 三买 _ 均线新高 _ 任意 _0’)
- Signal(‘15 分钟 _D1#SMA#34_BS3 辅助 V230319_ 三买 _ 均线新低 _ 任意 _0’)

信号说明：

类似 cxt_third_bs_V230318 信号，但增加了均线形态。

Parameters

- **c** –CZSC 对象
- **di** –从最后一个笔的第几个开始识别
- **kwargs** –ma_type: 均线类型，timeperiod: 均线周期

Returns

信号识别结果

cxt_third_buy_V230228

czsc.signals.**cxt_third_buy_V230228**(*c:* CZSC, ***kwargs*) → OrderedDict

笔三买辅助

参数模板：”{freq}_D{di}_ 三买辅助 V230228”

信号逻辑：

1. 定义大于前一个向上笔的高点的笔为向上突破笔，最近所有向上突破笔有价格重叠
2. 当下笔的最低点在任一向上突破笔的高点上，且当下笔的最低点离笔序列最低点的距离不超过向上突破笔列表均值的 1.618 倍

信号列表：

- Signal(‘15 分钟 _D1_ 三买辅助 V230228_ 三买 _14 笔 _ 任意 _0’)
- Signal(‘15 分钟 _D1_ 三买辅助 V230228_ 三买 _10 笔 _ 任意 _0’)
- Signal(‘15 分钟 _D1_ 三买辅助 V230228_ 三买 _6 笔 _ 任意 _0’)
- Signal(‘15 分钟 _D1_ 三买辅助 V230228_ 三买 _8 笔 _ 任意 _0’)
- Signal(‘15 分钟 _D1_ 三买辅助 V230228_ 三买 _12 笔 _ 任意 _0’)

Parameters

- **c** –CZSC 对象
- **kwargs** –
  - di: 倒数第几笔

Returns

信号识别结果

cxt_three_bi_V230618

czsc.signals.**cxt_three_bi_V230618**(*c:* CZSC, ***kwargs*) → OrderedDict

三笔形态分类

参数模板："{freq}_D{di} 三笔 _ 形态 V230618"

信号逻辑：

三笔的形态分类

信号列表：

- Signal('日线 _D1 三笔 _ 形态 V230618_ 向下盘背 _ 任意 _ 任意 _0')
- Signal('日线 _D1 三笔 _ 形态 V230618_ 向上奔走型 _ 任意 _ 任意 _0')
- Signal('日线 _D1 三笔 _ 形态 V230618_ 向上扩张 _ 任意 _ 任意 _0')
- Signal('日线 _D1 三笔 _ 形态 V230618_ 向下奔走型 _ 任意 _ 任意 _0')
- Signal('日线 _D1 三笔 _ 形态 V230618_ 向上收敛 _ 任意 _ 任意 _0')
- Signal('日线 _D1 三笔 _ 形态 V230618_ 向下无背 _ 任意 _ 任意 _0')
- Signal('日线 _D1 三笔 _ 形态 V230618_ 向上不重合 _ 任意 _ 任意 _0')
- Signal('日线 _D1 三笔 _ 形态 V230618_ 向下收敛 _ 任意 _ 任意 _0')
- Signal('日线 _D1 三笔 _ 形态 V230618_ 向下扩张 _ 任意 _ 任意 _0')
- Signal('日线 _D1 三笔 _ 形态 V230618_ 向下不重合 _ 任意 _ 任意 _0')
- Signal('日线 _D1 三笔 _ 形态 V230618_ 向上盘背 _ 任意 _ 任意 _0')
- Signal('日线 _D1 三笔 _ 形态 V230618_ 向上无背 _ 任意 _ 任意 _0')

Parameters

- **c** –CZSC 对象
- **kwargs** –
  - di: 倒数第几笔

Returns

信号识别结果

cxt_ubi_end_V230816

czsc.signals.**cxt_ubi_end_V230816**(*c:* CZSC, ***kwargs*) → OrderedDict

当前是未完成笔的第几次新低或新高，用于笔结束辅助

参数模板："{freq}_UBI_BE 辅助 V230816"

信号逻辑：

以向上未完成笔为例：取所有顶分型，计算创新高的底分型数量 N，如果当前 K 线创新高，则新高次数为 N+1

信号列表：

- Signal(‘日线 _UBI_BE 辅助 V230816_ 新低 _ 第 4 次 _ 任意 _0’)
- Signal(‘日线 _UBI_BE 辅助 V230816_ 新低 _ 第 5 次 _ 任意 _0’)
- Signal(‘日线 _UBI_BE 辅助 V230816_ 新低 _ 第 6 次 _ 任意 _0’)
- Signal(‘日线 _UBI_BE 辅助 V230816_ 新高 _ 第 2 次 _ 任意 _0’)
- Signal(‘日线 _UBI_BE 辅助 V230816_ 新高 _ 第 3 次 _ 任意 _0’)
- Signal(‘日线 _UBI_BE 辅助 V230816_ 新高 _ 第 4 次 _ 任意 _0’)
- Signal(‘日线 _UBI_BE 辅助 V230816_ 新高 _ 第 5 次 _ 任意 _0’)
- Signal(‘日线 _UBI_BE 辅助 V230816_ 新高 _ 第 6 次 _ 任意 _0’)
- Signal(‘日线 _UBI_BE 辅助 V230816_ 新高 _ 第 7 次 _ 任意 _0’)
- Signal(‘日线 _UBI_BE 辅助 V230816_ 新低 _ 第 2 次 _ 任意 _0’)
- Signal(‘日线 _UBI_BE 辅助 V230816_ 新低 _ 第 3 次 _ 任意 _0’)

Parameters

- **c** –CZSC 对象
- **kwargs** –

Returns

信号识别结果

cxt_zhong_shu_gong_zhen_V221221

czsc.signals.**cxt_zhong_shu_gong_zhen_V221221**(*cat:* CzscSignals, *freq1='日线'*, *freq2='60 分钟'*, ***kwargs*) → OrderedDict

大小级别中枢共振，类二买共振；贡献者：琅盎

参数模板：”{freq1}_{freq2}_ 中枢共振 V221221”

信号逻辑：

1. 不区分上涨或下跌中枢
2. 次级别中枢 DD 大于本级别中枢中轴
3. 次级别向下笔出底分型开多；反之看空

信号列表：

- Signal(‘日线 _60 分钟 _ 中枢共振 V221221_ 看空 _ 任意 _ 任意 _0’)
- Signal(‘日线 _60 分钟 _ 中枢共振 V221221_ 看多 _ 任意 _ 任意 _0’)

Parameters

- **cat** –
- **freq1** –大级别周期
- **freq2** –小级别周期

Returns

信号识别结果

dema_up_dw_line_V230605

czsc.signals.**dema_up_dw_line_V230605**(*c:* CZSC, ***kwargs*) → OrderedDict

DEMA 短线趋势指标，贡献者：琅盎

参数模板："{freq}_D{di}N{n}_DEMA 短线趋势 V230605"

信号逻辑：

DEMA 指标是一种趋势指标，用于衡量价格趋势的方向和强度。与其他移动平均线指标相比，DEMA 指标更加灵敏，能够更快地反应价格趋势的变化，因此在短期交易中具有一定的优势。当收盘价大于 DEMA 看多，当收盘价小于 DEMA 看空

信号列表：

- Signal('日线 _D1N5_DEMA 短线趋势 V230605_ 看多 _ 任意 _ 任意 _0')
- Signal('日线 _D1N5_DEMA 短线趋势 V230605_ 看空 _ 任意 _ 任意 _0')

Parameters

- **c** –CZSC 对象
- **kwargs** –参数字典 - :param di: 信号计算截止倒数第 i 根 K 线 - :param n: 获取 K 线的根数，默认为 5

Returns

信号识别结果

demakder_up_dw_line_V230605

czsc.signals.**demakder_up_dw_line_V230605**(*c:* CZSC, ***kwargs*) → OrderedDict

DEMAKER 价格趋势指标，贡献者：琅盎

参数模板："{freq}_D{di}N{n}TH{th}TL{tl}_DEMAKER 价格趋势 V230605"

信号逻辑：

DEMAKER 指标的作用是用于判断价格的趋势和力度当 demaker>0.6 时上升趋势强烈，当 demaker<0.4 时下跌趋势强烈。当 demaker 上穿 0.6/下穿 0.4 时产生买入/卖出信号。

信号列表：

- Signal('日线 _D1N105TH5TL5_DEMAKER 价格趋势 V230605_ 看多 _ 任意 _ 任意 _0')
- Signal('日线 _D1N105TH5TL5_DEMAKER 价格趋势 V230605_ 看空 _ 任意 _ 任意 _0')

Parameters

- **c** –CZSC 对象
- **kwargs** –参数字典
  - **param di**
    信号计算截止倒数第 i 根 K 线

- **param n**
  获取 K 线的根数，默认为 105
- **param th**
  开多阈值，默认为 6
- **param tl**
  开空阈值，默认为 4

Returns

信号识别结果

emv_up_dw_line_V230605

czsc.signals.**emv_up_dw_line_V230605**(*c:* CZSC, ***kwargs*) → OrderedDict

EMV 简易波动指标，贡献者：琅盎

参数模板："{freq}_D{di}_EMV 简易波动 V230605"

信号逻辑：

emv 综合考虑了成交量和价格（中间价）的变化。emv>0 则多头处于优势，emv 上升说明买方力量在增大；emv<0 则空头处于优势，emv 下降说明卖方力量在增大。如果 emv 上穿 0，则产生买入信号；如果 emv 下穿 0，则产生卖出信号。

信号列表：

- Signal('日线 _D1_EMV 简易波动 V230605_ 看多 _ 任意 _ 任意 _0')
- Signal('日线 _D1_EMV 简易波动 V230605_ 看空 _ 任意 _ 任意 _0')

Parameters

- **c** –CZSC 对象
- **kwargs** –参数字典
  - **param di**
    信号计算截止倒数第 i 根 K 线

Returns

信号识别结果

er_up_dw_line_V230604

czsc.signals.**er_up_dw_line_V230604**(*c:* CZSC, ***kwargs*) → OrderedDict

ER 价格动量指标，贡献者：琅盎

参数模板："{freq}_D{di}W{w}N{n}_ER 价格动量 V230604"

信号逻辑：

er 为动量指标。用来衡量市场的多空力量对比。在多头市场，人们会更贪婪地在接近高价的地方买入，BullPower 越高则当前多头力量越强；而在空头市场，人们可能因为恐惧而在接近低价的地方卖出。BearPower 越低则当前空头力量越强。当两者都大于 0 时，反映当前多头力量占据主导地位；两者都小于 0 则反映空头力量占据主导地位。如果 BearPower 上穿 0，则产生买入信号；如果 BullPower 下穿 0，则产生卖出信号。

信号列表：

- Signal('日线 _D1W21N10_ER 价格动量 V230604_ 均线下方 _ 第 10 层 _ 任意 _0')
- Signal('日线 _D1W21N10_ER 价格动量 V230604_ 均线下方 _ 第 9 层 _ 任意 _0')
- Signal('日线 _D1W21N10_ER 价格动量 V230604_ 均线下方 _ 第 8 层 _ 任意 _0')
- Signal('日线 _D1W21N10_ER 价格动量 V230604_ 均线上方 _ 第 5 层 _ 任意 _0')
- Signal('日线 _D1W21N10_ER 价格动量 V230604_ 均线上方 _ 第 1 层 _ 任意 _0')
- Signal('日线 _D1W21N10_ER 价格动量 V230604_ 均线上方 _ 第 10 层 _ 任意 _0')
- Signal('日线 _D1W21N10_ER 价格动量 V230604_ 均线上方 _ 第 2 层 _ 任意 _0')
- Signal('日线 _D1W21N10_ER 价格动量 V230604_ 均线上方 _ 第 6 层 _ 任意 _0')
- Signal('日线 _D1W21N10_ER 价格动量 V230604_ 均线上方 _ 第 7 层 _ 任意 _0')
- Signal('日线 _D1W21N10_ER 价格动量 V230604_ 均线上方 _ 第 8 层 _ 任意 _0')
- Signal('日线 _D1W21N10_ER 价格动量 V230604_ 均线上方 _ 第 9 层 _ 任意 _0')
- Signal('日线 _D1W21N10_ER 价格动量 V230604_ 均线上方 _ 第 4 层 _ 任意 _0')
- Signal('日线 _D1W21N10_ER 价格动量 V230604_ 均线下方 _ 第 5 层 _ 任意 _0')
- Signal('日线 _D1W21N10_ER 价格动量 V230604_ 均线下方 _ 第 7 层 _ 任意 _0')
- Signal('日线 _D1W21N10_ER 价格动量 V230604_ 均线下方 _ 第 3 层 _ 任意 _0')
- Signal('日线 _D1W21N10_ER 价格动量 V230604_ 均线下方 _ 第 2 层 _ 任意 _0')
- Signal('日线 _D1W21N10_ER 价格动量 V230604_ 均线下方 _ 第 6 层 _ 任意 _0')
- Signal('日线 _D1W21N10_ER 价格动量 V230604_ 均线下方 _ 第 1 层 _ 任意 _0')
- Signal('日线 _D1W21N10_ER 价格动量 V230604_ 均线下方 _ 第 4 层 _ 任意 _0')
- Signal('日线 _D1W21N10_ER 价格动量 V230604_ 均线上方 _ 第 3 层 _ 任意 _0')

Parameters

- **c** –CZSC 对象
- **kwargs** –参数字典 - :param di: 信号计算截止倒数第 i 根 K 线 - :param n: 获取 K 线的根数，默认为 105

Returns

信号识别结果

jcc_ci_tou_V221101

czsc.signals.**jcc_ci_tou_V221101**(*c:* CZSC, ***kwargs*) → OrderedDict

刺透形态

参数模板："{freq}_D{di}Z{z}TH{th}_ 刺透形态"

信号列表：

- Signal('15 分钟 _D1Z100TH50_ 刺透形态 _ 满足 _ 任意 _ 任意 _0')

Parameters

- **c** –CZSC 对象
- **di** –倒数第 di 跟 K 线
- **z** –可调阈值，前一天下跌 K 线实体的最低跌幅（收盘价-开盘价）/开盘价 *10000，500 表示至少跌 5%
- **th** –可调阈值，当天收盘价涨超前一天实体高度的百分比

Returns

刺绣形态识别结果

jcc_fan_ji_xian_V221121

czsc.signals.**jcc_fan_ji_xian_V221121**(*c:* CZSC, ***kwargs*) → OrderedDict

反击线；贡献者：lynxluu

参数模板："{freq}_D{di}_ 反击线"

信号逻辑：

1. 反击线分两种，看涨反击线和看跌反击线，共同特点是两根 K 线收盘价接近;
2. 看涨反击线，下降趋势，先阴线，后大幅低开收阳线;
3. 看跌反击线，上升趋势，先阳线，后大幅高开收阴线;

信号列表：

- Signal('15 分钟 _D1_ 反击线 _ 满足 _ 看涨反击线 _ 任意 _0')
- Signal('15 分钟 _D1_ 反击线 _ 满足 _ 看跌反击线 _ 任意 _0')

Parameters

- **c** –CZSC 对象
- **di** –倒数第 di 根 K 线取倒数三根 k 线

Returns

反击线识别结果

jcc_fen_shou_xian_V20221113

czsc.signals.**jcc_fen_shou_xian_V20221113**(*c:* CZSC, ***kwargs*) → OrderedDict

分手线：分手形态是一个中继形态；贡献者：琅盎

参数模板："{freq}_D{di}K_ 分手线"

**分手线形态，有三条判断标准 **

1. 分手线是由二根开盘价相等、运动方向相反的 K 线组成，因此也称分离线。2. 上升分手线出现在上升途中，由一阴一阳两根开盘价相等的 K 线组成，属于上涨持续形态；如果下跌趋势发展了较长时间之后出现上涨分手线，后市可能上涨应极积关注。3. 下跌分手线出现在下跌途中，由一阳一阴两根开盘价相等的 K 线组成，属于下跌持续形态；如果上涨趋势发展了较长时间之后出现下跌分手线，后市可能下跌应及时出场。

有效信号列表：

- Signal('60 分钟 _D1K_ 分手线 _ 满足 _ 上升分手 _ 任意 _0')
- Signal('60 分钟 _D1K_ 分手线 _ 满足 _ 下跌分手 _ 任意 _0')

Parameters

- **c** –CZSC 对象
- **di** –倒数第 di 根 K 线，加上这个参数就可以不用借助缓存就可以回溯
- **zdf** –可调阈值，涨跌幅，单位 BP

Returns

分离形态识别结果

jcc_gap_yin_yang_V221121

czsc.signals.**jcc_gap_yin_yang_V221121**(*c:* CZSC, ***kwargs*) → OrderedDict

跳空与并列阴阳形态贡献者：平凡

参数模板："{freq}_D{di}K_ 并列阴阳"

向上跳空并列阴阳（向下反之）：

1. 其中一根白色蜡烛线和一根黑色蜡烛线共同形成了一个向上的窗口。
2. 这根黑色蜡烛线的开市价位于前一个白色实体之内，收市价位于前一个白色实体之下。
3. 在这样的情况下，这根黑色蜡烛线的收市价，需要在窗口之上。
4. 黑白两根 K 线的实体相差不大

有效信号列表：

- Signal('15 分钟 _D1K_ 并列阴阳 _ 向上跳空 _ 任意 _ 任意 _0')
- Signal('15 分钟 _D1K_ 并列阴阳 _ 向下跳空 _ 任意 _ 任意 _0')

Parameters

- **c** –CZSC 对象

- **di** –倒数第 di 跟 K 线

Returns

识别结果

jcc_ping_tou_V221113

czsc.signals.**jcc_ping_tou_V221113**(*c:* CZSC, ***kwargs*) → OrderedDict

平头形态，贡献者：平凡

参数模板："{freq}_D{di}TH{th}_ 平头形态"

平头形态，判断标准：

1. 平头形态是由几乎具有相同水平的最高点的两根蜡烛线组成的，或者是由几乎具有相同的最低点的两根蜡烛线组成的。
2. 在理想情况下，平头形态应当由前一根长实体蜡烛线与后一根小实体蜡烛线组合而成

有效信号列表：

- Signal('15 分钟 _D2TH20_ 平头形态 _ 满足 _ 平头顶部 _ 任意 _0')
- Signal('15 分钟 _D2TH20_ 平头形态 _ 满足 _ 平头底部 _ 任意 _0')

Parameters

- **c** –CZSC 对象
- **di** –倒数第 di 根 K 线
- **th** –百分比，右侧 K 线的高/低点与当前 K 线的高/低点之间的差距比例，单位 BP

Returns

平头形态识别结果

jcc_san_fa_V20221115

czsc.signals.**jcc_san_fa_V20221115**(*c:* CZSC, ***kwargs*) → OrderedDict

上升 & 下降三法；贡献者：琅盎

参数模板："{freq}_D{di}K_ 三法"

上升三法形态由以下几个方面组成：

1. 首先出现的是一根长长的阳线。2. 在这根长长的阳线之后，紧跟着一群依次下降的或者横向延伸的小实体蜡烛线。这群小实体蜡烛线的理想数目是 3 根，但是 2 根或者 3 根以上也是可以接受的，条件是：只要这群小实体蜡烛线基本上都局限在前面长长的白色蜡烛线的高点到低点的价格范围之内。小蜡烛线既可以是白色的，也可以是黑色的，不过，黑色蜡烛线最理想。3. 最后一天应当是一根坚挺的白色实体蜡烛线，并且它的收盘价高于前一天的收盘价，同时其开盘价应当高于前一天的收盘价。

下降三法形态由以下几个方面组成：

1. 下降三法形态与上升三法形态完全是对等的，只不过方向相反。这类形态的形成过程如下：2. 市场应当处在下降趋势中，首先出场的是一根长长的黑色蜡烛线。在这根黑色蜡烛线之后，跟随着大约 3 根依次上升的小蜡烛线，并且这群蜡烛线的实体都局限在第一根蜡烛线的范围之内（包括其上、下影线）。3. 最后一天，开盘价应低于前一天的收盘价，并且收盘价应低于第一根黑色蜡烛线的收盘价。本形态与看跌旗形或看跌三角旗形形态相似。本形态的理想情形是，在第一根长实体之后，小实体的颜色与长实体相反。

信号列表：

- Signal（‘60 分钟 _D1K_ 三法 _ 满足 _ 上升三法 _ 任意 _0’）
- Signal（‘60 分钟 _D1K_ 三法 _ 满足 _ 下降三法 _ 任意 _0’）

Parameters

- **c** –CZSC 对象
- **di** –倒数第 di 跟 K 线
- **zdf** –倒 1 和倒数第 5 根 K 线涨跌幅，单位 BP

Returns

上升及下降三法形态识别结果

jcc_san_fa_V20221118

czsc.signals.**jcc_san_fa_V20221118**(*c:* CZSC, ***kwargs*) → OrderedDict

上升 & 下降三法

参数模板：”{freq}_D{di}K_ 三法 A”

上升三法形态由以下几个方面组成：

1. 首先出现的是一根长长的阳线。2. 在这根长长的阳线之后，紧跟着一群依次下降的或者横向延伸的小实体蜡烛线。这群小实体蜡烛线的理想数目是 3 根，但是 2 根或者 3 根以上也是可以接受的，条件是：只要这群小实体蜡烛线基本上都局限在前面长长的白色蜡烛线的高点到低点的价格范围之内。小蜡烛线既可以是白色的，也可以是黑色的，不过，黑色蜡烛线最理想。3. 最后一天应当是一根坚挺的白色实体蜡烛线，并且它的收盘价高于前一天的收盘价，同时其开盘价应当高于前一天的收盘价。

下降三法形态由以下几个方面组成：

1. 下降三法形态与上升三法形态完全是对等的，只不过方向相反。这类形态的形成过程如下：2. 市场应当处在下降趋势中，首先出场的是一根长长的黑色蜡烛线。在这根黑色蜡烛线之后，跟随着大约 3 根依次上升的小蜡烛线，并且这群蜡烛线的实体都局限在第一根蜡烛线的范围之内（包括其上、下影线）。3. 最后一天，开盘价应低于前一天的收盘价，并且收盘价应低于第一根黑色蜡烛线的收盘价。本形态与看跌旗形或看跌三角旗形形态相似。本形态的理想情形是，在第一根长实体之后，小实体的颜色与长实体相反。

信号列表：

- Signal（‘60 分钟 _D1K_ 三法 A_ 上升三法 _8K_ 任意 _0’）
- Signal（‘60 分钟 _D1K_ 三法 A_ 上升三法 _6K_ 任意 _0’）

- Signal(‘60 分钟 _D1K_ 三法 A_ 上升三法 _5K_ 任意 _0’)
- Signal(‘60 分钟 _D1K_ 三法 A_ 下降三法 _5K_ 任意 _0’)
- Signal(‘60 分钟 _D1K_ 三法 A_ 下降三法 _6K_ 任意 _0’)
- Signal(‘60 分钟 _D1K_ 三法 A_ 上升三法 _7K_ 任意 _0’)
- Signal(‘60 分钟 _D1K_ 三法 A_ 下降三法 _7K_ 任意 _0’)
- Signal(‘60 分钟 _D1K_ 三法 A_ 下降三法 _8K_ 任意 _0’)

Parameters

- **c** –CZSC 对象
- **di** –倒数第 di 跟 K 线

Returns

上升及下降三法形态识别结果

jcc_san_szx_V221122

czsc.signals.**jcc_san_szx_V221122**(*c:* CZSC, ***kwargs*) → OrderedDict

三星形态

参数模板:”{freq}_D{di}T{th}_ 三星”

信号逻辑:

1. 最近五根 K 线中出现三个十字星

信号列表:

- Signal(‘15 分钟 _D1T10_ 三星 _ 满足 _ 任意 _ 任意 _0’)

Parameters

- **c** –CZSC 对象
- **di** –倒数第 di 跟 K 线
- **th** –可调阈值，(h -l) / (c - o) 的绝对值大于 th, 判定为十字线

Returns

识别结果

jcc_san_xing_xian_V221023

czsc.signals.**jcc_san_xing_xian_V221023**(*c:* CZSC, ***kwargs*) → OrderedDict

伞形线

参数模板:”{freq}_D{di}TH{th}_ 伞形线”

有效信号列表:

- Signal(‘15 分钟 _D5TH200_ 伞形线 _ 满足 _ 上吊 _ 任意 _0’)
- Signal(‘15 分钟 _D5TH200_ 伞形线 _ 满足 _ 锤子 _ 任意 _0’)

Parameters

- **c** –CZSC 对象
- **di** –倒数第 di 跟 K 线
- **th** –可调阈值，下影线超过实体的倍数，保留两位小数

Returns

伞形线识别结果

jcc_shan_chun_V221121

czsc.signals.**jcc_shan_chun_V221121**(*c:* CZSC, ***kwargs*) → OrderedDict

山川形态，表示三山形态和三川形态

参数模板："{freq}_D{di}B_ 山川形态"

信号逻辑：

1. 三山顶部形态一般认为，本形态构成了一种主要顶部反转过程。如果市场先后三次均从某一个高价位上回落下来，或者市场对某一个高价位向上进行了三次尝试，但都失败了，那么三山顶部形态就形成了。2. 三川底部形态恰巧是三山顶部形态的反面。在市场先后三度向下试探某个底部水平后，就形成了这类形态。市场必须向上突破这个底部形态的最高水平，才能证实底部过程已经完成。

信号列表：

- Signal('15 分钟 _D1B_ 山川形态 _ 三山 _ 任意 _ 任意 _0')
- Signal('15 分钟 _D1B_ 山川形态 _ 三川 _ 任意 _ 任意 _0')

Parameters

- **c** –CZSC 对象
- **di** –截止倒数第 di 笔

Returns

识别结果

jcc_szx_V221111

czsc.signals.**jcc_szx_V221111**(*c:* CZSC, ***kwargs*) → OrderedDict

十字线

参数模板："{freq}_D{di}TH{th}_ 十字线"

信号逻辑：

1，十字线定义，(h -l) / (c - o) 的绝对值大于 th，或 c == o 2. 长腿十字线，上下影线都很长；墓碑十字线，上影线很长；蜻蜓十字线，下影线很长；

信号列表：

- Signal(‘60 分钟 _D1TH10_ 十字线 _ 蜻蜓十字线 _ 北方 _ 任意 _0’)
- Signal(‘60 分钟 _D1TH10_ 十字线 _ 十字线 _ 任意 _ 任意 _0’)
- Signal(‘60 分钟 _D1TH10_ 十字线 _ 蜻蜓十字线 _ 任意 _ 任意 _0’)
- Signal(‘60 分钟 _D1TH10_ 十字线 _ 墓碑十字线 _ 任意 _ 任意 _0’)
- Signal(‘60 分钟 _D1TH10_ 十字线 _ 长腿十字线 _ 任意 _ 任意 _0’)
- Signal(‘60 分钟 _D1TH10_ 十字线 _ 十字线 _ 北方 _ 任意 _0’)
- Signal(‘60 分钟 _D1TH10_ 十字线 _ 墓碑十字线 _ 北方 _ 任意 _0’)
- Signal(‘60 分钟 _D1TH10_ 十字线 _ 长腿十字线 _ 北方 _ 任意 _0’)

Parameters

- **c** –CZSC 对象
- **kwargs** –
  - di: 倒数第 di 跟 K 线
  - th: 可调阈值，(h -l) / (c - o) 的绝对值大于 th, 判定为十字线

Returns

十字线识别结果

jcc_ta_xing_V221124

czsc.signals.**jcc_ta_xing_V221124**(*c:* CZSC, ***kwargs*) → OrderedDict

塔形顶底

参数模板：”{freq}_D{di}K_ 塔形”

信号逻辑：

1. 首尾两根 K 线的实体最大
2. 首 k 上涨，尾 K 下跌，且中间高点相近，且低点大于首尾低点的较大者，塔形顶部；反之，底部。

信号列表：

- Signal(‘15 分钟 _D1K_ 塔形 _ 顶部 _6K_ 任意 _0’)
- Signal(‘15 分钟 _D1K_ 塔形 _ 顶部 _9K_ 任意 _0’)
- Signal(‘15 分钟 _D1K_ 塔形 _ 底部 _7K_ 任意 _0’)
- Signal(‘15 分钟 _D1K_ 塔形 _ 顶部 _5K_ 任意 _0’)
- Signal(‘15 分钟 _D1K_ 塔形 _ 底部 _5K_ 任意 _0’)
- Signal(‘15 分钟 _D1K_ 塔形 _ 底部 _8K_ 任意 _0’)
- Signal(‘15 分钟 _D1K_ 塔形 _ 底部 _6K_ 任意 _0’)
- Signal(‘15 分钟 _D1K_ 塔形 _ 顶部 _7K_ 任意 _0’)
- Signal(‘15 分钟 _D1K_ 塔形 _ 顶部 _8K_ 任意 _0’)
- Signal(‘15 分钟 _D1K_ 塔形 _ 底部 _9K_ 任意 _0’)

Parameters

- **c** –CZSC 对象

- **di** –倒数第 di 跟 K 线

Returns

识别结果

jcc_ten_mo_V221028

czsc.signals.**jcc_ten_mo_V221028**(*c:* CZSC, ***kwargs*) → OrderedDict

吞没形态；贡献者：琅盎

参数模板："{freq}_D{di}_ 吞没形态"

吞没形态，有三条判别标准：

1. 在吞没形态之前，市场必须处在明确的上升趋势（看跌吞没形态）或下降趋势（看涨吞没形态）中，哪怕这个趋势只是短期的。
2. 吞没形态由两条蜡烛线组成。其中第二根蜡烛线的实体必须覆盖第一根蜡烛线的实体（但是不一定需要吞没前者的上下影线）。
3. 吞没形态的第二个实体应与第一个实体的颜色相反。

有效信号列表：

- Signal('15 分钟 _D1_ 吞没形态 _ 满足 _ 看跌吞没 _ 任意 _0')
- Signal('15 分钟 _D1_ 吞没形态 _ 满足 _ 看涨吞没 _ 任意 _0')

Parameters

- **c** –CZSC 对象
- **di** –倒数第 di 跟 K 线

Returns

吞没形态识别结果

jcc_three_crow_V221108

czsc.signals.**jcc_three_crow_V221108**(*c:* CZSC, ***kwargs*)

三只乌鸦，贡献者：马鸣

参数模板："{freq}_D{di}_ 三只乌鸦"

信号逻辑：

1、连续出现了三根依次下降的黑色蜡烛线，每根黑色蜡烛线的开市价处于前一个实体的范围之内，则构成了所谓的常规三只乌鸦形态；2、三只乌鸦形态中的第二根和第三根蜡烛线都开市于之前的实体之下构成更加疲软的三只乌鸦形态；3、允许有上影线，但是不能出现包含关系；4、允许有下影线，但是最低价接近收盘价；5、该信号出现在阶段性顶部。

信号列表：

- Signal('30 分钟 _D1_ 三只乌鸦 _ 满足 _ 常规 _ 任意 _0')
- Signal('30 分钟 _D1_ 三只乌鸦 _ 满足 _ 加强 _ 任意 _0')

- Signal(‘30 分钟 _D1_ 三只乌鸦 _ 满足 _ 半加强 _ 任意 _0’)

  Parameters

  - **c** –CZSC 对象
  - **di** –连续倒 di 根 K 线

  Returns

jcc_two_crow_V221108

czsc.signals.**jcc_two_crow_V221108**(*c:* CZSC, ***kwargs*)

两只乌鸦

参数模板:”{freq}_D{di}K_ 两只乌鸦”

信号逻辑:

1. 市场本来处于上升趋势中;
2. 第一根 K 线为长阳;
3. 第二根 K 线跳空高开低走，收于前收盘价上方;
4. 第三根 K 线同样高开低走，包含第二根 K 线，收于第一根 K 线收盘价上方。

信号列表:

- Signal(‘60 分钟 _D1K_ 两只乌鸦 _ 看空 _ 任意 _ 任意 _0’)

  Parameters

  - **c** –CZSC 对象
  - **di** –连续倒 di 根 K 线

  Returns

jcc_wu_yun_gai_ding_V221101

czsc.signals.**jcc_wu_yun_gai_ding_V221101**(*c:* CZSC, ***kwargs*) → OrderedDict

乌云盖顶，贡献者：魏永超

参数模板:”{freq}_D{di}Z{z}TH{th}_ 乌云盖顶”

信号逻辑:

1. 当前的走势属于上升趋势，或者水平调整区间的顶部。
2. 前一天是坚挺的白色实体，也就是大阳线。
3. 当天跳空高开，开盘价高于前一天的最高价。
4. 当天收盘在最低价附近，且明显向下扎入前一天的 K 线实体内。一般要求当天收盘价低于前一天阳线实体的 50%。

信号列表:

- Signal(‘日线 _D5Z500TH50_ 乌云盖顶 _ 满足 _ 任意 _ 任意 _0’)

Parameters

- **c** –CZSC 对象
- **di** –倒数第 di 跟 K 线
- **z** –可调阈值，前一天上涨 K 线实体的最低涨幅（收盘价-开盘价）/开盘价 *10000，500 表示至少涨 5%
- **th** –可调阈值，当天收盘价跌入前一天实体高度的百分比

Returns

乌云盖顶识别结果

jcc_xing_xian_V221118

czsc.signals.**jcc_xing_xian_V221118**(*c:* CZSC, ***kwargs*) → OrderedDict

星形态

参数模板："{freq}_D{di}TH{th}_ 星形线"

星形态，判断标准：

1. 启明星：

   蜡烛线 1。一根长长的黑色实体，形象地表明空头占据主宰地位蜡烛线 2。一根小小的实体，并且它与前一根实体之间不相接触（这两条蜡烛线组成了基本的星线形态）。小实体意味着卖方丧失了驱动市场走低的能力蜡烛线 3。一根白色实体，它明显地向上推进到了第一个时段的黑色实体之内，标志着启明星形态的完成。这表明多头已经夺回了主导权

   在理想的启明星形态中，第二根蜡烛线（即星线）的实体，与第三根蜡烛线的实体之间有价格跳空。根据我的经验，即使没有这个价格跳空，似乎也不会削减启明星形态的技术效力。其决定性因素是，第二根蜡烛线应为纺锤线，同时第三根蜡烛线应显著深入到第一根黑色蜡烛线内部

2. 黄昏星：
   a. 如果第一根与第二根蜡烛线，第二根与第三根蜡烛线的实体之间不存在重叠。
   b. 如果第三根蜡烛线的收市价向下深深扎入第一根蜡烛线的实体内部。
   c. 如果第一根蜡烛线的交易量较小，而第三根蜡烛线的交易量较大。这表明之前趋势的驱动力正在减弱，新趋势方向的驱动力正在加强
3. 十字黄昏星

   在常规的黄昏星形态中，第二根蜡烛线具有较小的实体，如果不是较小的实体，而是一个十字线，则称为十字黄昏星形态
4. 十字启明星

   在启明星形态中，如果其星线（即三根蜡烛线中的第二根蜡烛线）是一个十字线，则成为十字启明星形态

信号列表：

- Signal（'60 分钟 _D1TH2_ 星形线 _ 黄昏星 _ 任意 _ 任意 _0'）
- Signal（'60 分钟 _D1TH2_ 星形线 _ 启明星 _ 任意 _ 任意 _0'）

- Signal(‘60 分钟 _D1TH2_ 星形线 _ 启明星 _ 中间十字 _ 任意 _0’)
- Signal(‘60 分钟 _D1TH2_ 星形线 _ 黄昏星 _ 中间十字 _ 任意 _0’)

Parameters

- **c** –CZSC 对象
- **di** –倒数第 di 跟 K 线
- **th** –左侧实体是当前实体的多少倍

Returns

星形线识别结果

jcc_yun_xian_V221118

czsc.signals.**jcc_yun_xian_V221118**(*c:* CZSC, ***kwargs*) → OrderedDict

孕线形态

参数模板:”{freq}_D{di}_ 孕线”

** 信号逻辑: ** 二日 K 线模式，分多头孕线与空头孕线，两者相反，以多头孕线为例，在下跌趋势中，第一日 K 线长阴，第二日开盘和收盘价都在第一日价格振幅之内，为阳线，预示趋势反转，股价上升

有效信号列表:

- Signal(‘60 分钟 _D1_ 孕线 _ 看空 _ 任意 _ 任意 _0’)
- Signal(‘60 分钟 _D1_ 孕线 _ 看多 _ 任意 _ 任意 _0’)

Parameters

- **c** –CZSC 对象
- **di** –倒数第 di 跟 K 线

Returns

孕线识别结果

jcc_zhu_huo_xian_V221027

czsc.signals.**jcc_zhu_huo_xian_V221027**(*c:* CZSC, ***kwargs*) → OrderedDict

烛火线，贡献者: 琅盎

参数模板:”{freq}_D{di}T{th}F{zf}_ 烛火线”

**信号列表: **

- Signal(‘60 分钟 _D1T200F500_ 烛火线 _ 满足 _ 风中烛 _ 任意 _0’)
- Signal(‘60 分钟 _D1T200F500_ 烛火线 _ 满足 _ 箭在弦 _ 任意 _0’)

Parameters

- **c** –CZSC 对象

- **di** –倒数第 di 跟 K 线
- **th** –可调阈值，下影线超过实体的倍数，保留两位小数
- **zf** –可调阈值，震荡幅度大小，单位 BP

Returns

烛火线识别结果

jcc_zhuo_yao_dai_xian_v221113

czsc.signals.**jcc_zhuo_yao_dai_xian_v221113**(*c:* CZSC, ***kwargs*) → OrderedDict

捉腰带线，贡献者：平凡

参数模板："{freq}_D{di}L{left}_ 捉腰带线"

捉腰带线判别标准：

捉腰带形态是由单独一根蜡烛线构成的。看涨捉腰带形态是一根坚挺的白色蜡烛线，其开市价位于时段的最低点（或者，这根蜡烛线只有极短的下影线），然后市场一路上扬，收市价位于或接近本时段的最高

有效信号列表：

- Signal('60 分钟 _D1L20_ 捉腰带线 _ 看跌 _ 光头阴线 _ 任意 _0')
- Signal('60 分钟 _D1L20_ 捉腰带线 _ 看多 _ 光脚阳线 _ 任意 _0')

Parameters

- **c** –CZSC 对象
- **di** –倒数第 di 跟 K 线
- **left** –从 di 向左数 left 根 K 线

Returns

捉腰带线识别结果

kcatr_up_dw_line_V230823

czsc.signals.**kcatr_up_dw_line_V230823**(*c:* CZSC, ***kwargs*) → OrderedDict

用 ATR 波幅构造上下轨，收盘价突破判断多空贡献者：琅盎

参数模板："{freq}_D{di}N{n}M{m}T{th}_KCATR 多空 V230823"

信号逻辑：

与布林带类似，都是用价格的移动平均构造中轨，不同的是表示波幅的方法，这里用 atr 来作为波幅构造上下轨。价格突破上轨，可看成新的上升趋势，买入；价格突破下轨，

信号列表：

- Signal('日线 _D1N30M16T2_KCATR 多空 V230823_ 看多 _ 任意 _ 任意 _0')

- Signal(‘日线 _D1N30M16T2_KCATR 多空 V230823_ 看空 _ 任意 _ 任意 _0’)

Parameters

- **c** –CZSC 对象
- **kwargs** –参数字典
  - **param di**
    信号计算截止倒数第 i 根 K 线
  - **param n**
    获取 K 线的根数进行 ATR 计算，默认为 30
  - **param m**
    获取 K 线的根数进行均价计算，默认为 16
  - **param th**
    突破 ATR 的倍数，默认为 2

Returns

信号识别结果

ntmdk_V230824

czsc.signals.**ntmdk_V230824**(*c:* CZSC, ***kwargs*) → OrderedDict

NTMDK 多空指标，贡献者：琅盎

参数模板：”{freq}_D{di}M{m}_NTMDK 多空 V230824”

信号逻辑：

此信号函数的逻辑非常简单，流传于股市中有一句话：日日新高日日持股，那么此信号函数利用的是收盘价和 M 日前的收盘价进行比较，如果差值为正即多头成立，反之空头成立。

信号列表：

- Signal(‘日线 _D1M10_NTMDK 多空 V230824_ 看空 _ 任意 _ 任意 _0’)
- Signal(‘日线 _D1M10_NTMDK 多空 V230824_ 看多 _ 任意 _ 任意 _0’)

Parameters

- **c** –CZSC 对象
- **kwargs** –参数字典
  - **param di**
    信号计算截止倒数第 i 根 K 线
  - **param m**
    m 天前的价格

Returns

信号识别结果

obv_up_dw_line_V230719

czsc.signals.**obv_up_dw_line_V230719**(*c:* CZSC, ***kwargs*) → OrderedDict

OBV 能量指标，贡献者：琅盎

参数模板："{freq}_D{di}N{n}M{m}MO{max_overlap}_OBV 能量 V230719"

信号逻辑：

OBV 指标把成交量分为正的成交量（价格上升时的成交量）和负的成交量（价格下降时）的成交量。OBV 就是分了正负之后的成交量的累计和。

1. 先定义 OBVM，OBVM 是 OBV 7 天的指数平均。
2. 再定义一条信号线 Signal line，这条线是 OBVM 10 天的指数平均。

其中的「7 天」和「10 天」都是参数，根据你的交易时间级别设置，设置的越小，OBVM 对成交量的变化越敏感，产生的交易信号也就越多。

开多仓的规则：当 OBVM 上穿 Signal line，开多仓；当 OBVM 下穿 Signal line，平仓。这个规则只捕捉上涨趋势。

信号列表：

- Signal('日线 _D1N7M10MO3_OBV 能量 V230719_ 看多 _ 任意 _ 任意 _0')
- Signal('日线 _D1N7M10MO3_OBV 能量 V230719_ 看空 _ 任意 _ 任意 _0')

Parameters

- **c** –CZSC 对象
- **kwargs** –参数字典
  - **param di**
    信号计算截止倒数第 i 根 K 线
  - **param n**
    short 窗口大小。
  - **param m**
    long 窗口大小。
  - **param max_overlap**
    信号计算时，最大重叠 K 线数量。

Returns

信号识别结果

obvm_line_V230610

czsc.signals.**obvm_line_V230610**(*c:* CZSC, ***kwargs*) → OrderedDict

OBV 能量指标，贡献者：琅盎

参数模板："{freq}_D{di}N{n}M{m}_OBV 能量 V230610"

信号逻辑：

OBV 指标把成交量分为正的成交量（价格上升时的成交量）和负的成交量（价格下降时）的成交量。OBV 就是分了正负之后的成交量的累计和。

首先，根据传入的参数 di、n 和 m，从 CZSC 对象中获取对应的 K 线数据，然后计算 OBV 序列。接着，使用 talib 库中的 EMA 函数计算 OBV 序列的短期和长期指数移动平均线，最后根据两条移动平均线的大小关系判断看多或看空信号。

飞书文档：https://s0cqcxuy3p.feishu.cn/wiki/CEMLwa46Ii1sJZkT3IVcJKAwntc

信号列表：

- Signal('日线 _D1N10M30_OBV 能量 V230610_ 看空 _ 任意 _ 任意 _0')
- Signal('日线 _D1N10M30_OBV 能量 V230610_ 看多 _ 任意 _ 任意 _0')

Parameters

- **c** –CZSC 对象
- **kwargs** –参数字典
  - **param di**
    信号计算截止倒数第 i 根 K 线
  - **param n**
    short 窗口大小。
  - **param m**
    long 窗口大小。

Returns

信号识别结果

pos_bar_stop_V230524

czsc.signals.**pos_bar_stop_V230524**(*cat:* CzscTrader, ***kwargs*) → OrderedDict

按照开仓点附近的 N 根 K 线极值止损

参数模板："{pos_name}_{freq1}N{n}K_ 止损 V2305224"

信号逻辑：

多头止损逻辑如下，反之为空头止损逻辑：

1. 从多头开仓点开始，在给定的 K 线周期 freq1 上向前找 N 个 K 线，记为 F1

2. 将这 N 个 K 线的最低点，记为 L1，如果最新价跌破 L1，则止损

信号列表：

- Signal(‘日线三买多头 _ 日线 N3K_ 止损 V2305224_ 多头止损 _ 任意 _ 任意 _0’)
- Signal(‘日线三买多头 _ 日线 N3K_ 止损 V2305224_ 空头止损 _ 任意 _ 任意 _0’)

Parameters

- **cat** –CzscTrader 对象
- **kwargs** –参数字典
  - pos_name: str，开仓信号的名称
  - freq1: str，给定的 K 线周期
  - n: int，向前找的 K 线个数，默认为 3

Returns

OrderedDict

pos_fix_exit_V230624

czsc.signals.**pos_fix_exit_V230624**(*cat:* CzscTrader, ***kwargs*) → OrderedDict

固定比例止损，止盈

参数模板：”{pos_name}_ 固定 {th}BP 止盈止损 _ 出场 V230624”

信号逻辑：

以多头为例，如果持有收益超过 th 个 BP，则止盈；如果亏损超过 th 个 BP，则止损。

信号列表：

- Signal(‘日线三买多头 _ 固定 100BP 止盈止损 _ 出场 V230624_ 多头止损 _ 任意 _ 任意 _0’)
- Signal(‘日线三买多头 _ 固定 100BP 止盈止损 _ 出场 V230624_ 空头止损 _ 任意 _ 任意 _0’)

Parameters

- **cat** –CzscTrader 对象
- **kwargs** –参数字典 - pos_name: str，开仓信号的名称 - freq1: str，给定的 K 线周期 - n: int，向前找的 K 线个数，默认为 3

Returns

pos_fx_stop_V230414

czsc.signals.**pos_fx_stop_V230414**(*cat:* CzscTrader, ***kwargs*) → OrderedDict

按照开仓点附近的分型止损

参数模板："{freq1}_{pos_name}N{n}_ 止损 V230414"

信号逻辑：

多头止损逻辑如下，反之为空头止损逻辑：

1. 从多头开仓点开始，在给定对的 K 线周期 freq1 上向前找 N 个底分型，记为 F1
2. 将这 N 个底分型的最低点，记为 L1，如果最新价低于 L1，则止损

信号列表：

- Signal（'日线 _ 日线三买多头 N1_ 止损 V230414_ 多头止损 _ 任意 _ 任意 _0'）
- Signal（'日线 _ 日线三买多头 N1_ 止损 V230414_ 空头止损 _ 任意 _ 任意 _0'）

Parameters

- **cat** –CzscTrader 对象
- **kwargs** –参数字典 - pos_name: str，开仓信号的名称 - freq1: str，给定的 K 线周期 - n: int，向前找的分型个数，默认为 3

Returns

pos_holds_V230414

czsc.signals.**pos_holds_V230414**(*cat:* CzscTrader, ***kwargs*) → OrderedDict

开仓后 N 根 K 线涨幅小于 M%%，则平仓

参数模板："{pos_name}_{freq1}N{n}M{m}_ 趋势判断 V230414"

信号逻辑：

1. 找出开仓后的 N 根 K 线，计算涨幅，如果涨幅小于 M%%，则平仓
2. 这里面的逻辑是，如果开仓后的 N 根 K 线涨幅小于 M%%，则说明趋势不明朗，平仓等待

信号列表：

- Signal（'日线三买多头 N1_60 分钟 N5M100_ 趋势判断 V230414_ 多头存疑 _ 任意 _ 任意 _0'）
- Signal（'日线三买多头 N1_60 分钟 N5M100_ 趋势判断 V230414_ 多头良好 _ 任意 _ 任意 _0'）

Parameters

- **cat** –CzscTrader 对象
- **kwargs** –参数字典 - pos_name: str，开仓信号的名称 - freq1: str，给定的 K 线周期 - n: int，最少持有 K 线数量，默认为 5，表示 5 根 K 线之后开始判断趋势 - m: int，涨幅阈值，默认为 100，表示涨幅小于 100BP 时，平仓

Returns

pos_holds_V230807

czsc.signals.**pos_holds_V230807**(*cat:* CzscTrader, ***kwargs*) → OrderedDict

开仓后 N 根 K 线收益小于 M%%，且当前收益大于 T%%，平仓保本

参数模板："{pos_name}_{freq1}N{n}M{m}T{t}_BS 辅助 V230807"

信号逻辑：

1. 针对某个开仓点，如果 N 根 K 线之后，收益低于 M，则认为开仓失误；
2. 开仓的失误发生后，如果市场给了逃命的机会，不贪心，等待收益大于 T 时，平仓保本；
3. 保本有两种场景：开仓后先亏损后反弹；开仓后先盈利后回落。

信号列表：

- Signal('日线三买多头 N1_60 分钟 N5M50T10_BS 辅助 V230807_ 多头保本 _ 任意 _ 任意 _0')
- Signal('日线三买多头 N1_60 分钟 N5M50T10_BS 辅助 V230807_ 空头保本 _ 任意 _ 任意 _0')

Parameters

- **cat** –CzscTrader 对象
- **kwargs** –参数字典
  - pos_name: str，开仓信号的名称
  - freq1: str，给定的 K 线周期
  - n: int，最少持有 K 线数量，默认为 5，表示 5 根 K 线之后开始判断
  - m: int，收益最小阈值，默认为 100，表示收益小于 100BP 时，需要开始判断保本单
  - t: int，保本收益阈值，默认为 10，表示收益大于 10BP 时，可以平仓保本

Returns

pos_holds_V240428

czsc.signals.**pos_holds_V240428**(*cat:* CzscTrader, ***kwargs*) → OrderedDict

保本单：开仓后最大盈利超过 H 个 BP，且当前收益低于最大盈利的 T%，平仓保本

参数模板："{pos_name}_{freq1}H{h}T{t}N{n}_ 保本 V240428"

信号逻辑：

以多头保本单为例，计算过程如下：

1. 从多头开仓点开始，在给定的 K 线周期 freq1 上计算开仓后的最大盈利，记为 Y1；
2. 计算当前收益，记为 Y2；
3. 如果 Y1 大于 H，且 Y2 < Y1 * T / 100，则平仓保本。

信号列表：

- Signal('日线三买多头 N1_60 分钟 H100T20N5_ 保本 V240428_ 空头保本 _ 任意 _ 任意 _0')
- Signal('日线三买多头 N1_60 分钟 H100T20N5_ 保本 V240428_ 多头保本 _ 任意 _ 任意 _0')

Parameters

- **cat** –CzscTrader 对象
- **kwargs** –参数字典
  - pos_name: str，开仓信号的名称
  - freq1: str，给定的 K 线周期
  - h: int，最大盈利，单位 BP，默认为 100
  - t: int，最大盈利的 T%，默认为 20
  - n: int，最少持有 K 线数量，默认为 5，表示 5 根 K 线之后开始判断

Returns

OrderedDict

pos_ma_V230414

czsc.signals.**pos_ma_V230414**(*cat:* CzscTrader, ***kwargs*) → OrderedDict

判断开仓后是否升破 MA 均线或跌破 MA 均线

参数模板："{pos_name}_{freq1}#{ma_type}#{timeperiod}_ 持有状态 V230414"

信号逻辑：

多头止损逻辑如下，反之为空头止损逻辑：

1. 从多头开仓点开始，在给定对的 K 线周期 freq1 上向前找 N 个底分型，记为 F1
2. 将这 N 个底分型的最低点，记为 L1，如果 L1 的价格低于开仓点的价格，则止损

信号列表：

- Signal('日线三买多头 N1_60 分钟 #SMA#5_ 持有状态 V230414_ 多头 _ 升破均线 _ 任意 _0')
- Signal('日线三买多头 N1_60 分钟 #SMA#5_ 持有状态 V230414_ 空头 _ 跌破均线 _ 任意 _0')

Parameters

- **cat** –CzscTrader 对象
- **kwargs** –参数字典 - pos_name: str，开仓信号的名称 - freq1: str，给定的 K 线周期 - n: int，向前找的分型个数，默认为 3

Returns

pos_profit_loss_V230624

czsc.signals.**pos_profit_loss_V230624**(*cat:* CzscTrader, ***kwargs*) → OrderedDict

开仓后盈亏比达到一定比值，才允许平仓贡献者：谌意勇

参数模板："{pos_name}_{freq1}YKB{ykb}N{n}_ 盈亏比判断 V230624"

信号逻辑：

1. 通过公式计算盈亏比 =abs(现价-开仓价）/abs(开仓价-止损价）* 10, 当比值大于一定阀值时才允许平仓

信号列表：

- Signal('日线通道突破 _60 分钟 YKB20N3_ 盈亏比判断 V230624_ 空头止损 _ 任意 _ 任意 _0')
- Signal('日线通道突破 _60 分钟 YKB20N3_ 盈亏比判断 V230624_ 多头止损 _ 任意 _ 任意 _0')
- Signal('日线通道突破 _60 分钟 YKB20N3_ 盈亏比判断 V230624_ 多头达标 _ 任意 _ 任意 _0')
- Signal('日线通道突破 _60 分钟 YKB20N3_ 盈亏比判断 V230624_ 空头达标 _ 任意 _ 任意 _0')

Parameters

- **cat** –CzscTrader 对象
- **kwargs** –参数字典
  - pos_name: str，开仓信号的名称
  - freq1: str，给定的 K 线周期
  - ykb: int，默认为 20, 表示 2 倍盈亏比，计算盈亏比 =abs(现价-开仓价）/abs(开仓价-止损价）
  - n: int 默认为 3 止损取最近 n 个分型的最低点或最高点

Returns

pos_status_V230808

czsc.signals.**pos_status_V230808**(*cat:* CzscTrader, ***kwargs*) → OrderedDict

Position 策略的持仓状态

参数模板："{pos_name}_ 持仓状态 _BS 辅助 V230808"

信号逻辑：

对指定的持仓策略，有三种状态：持多、持空、持币。

信号列表：

- Signal('日线三买多头 N1_ 持仓状态 _BS 辅助 V230808_ 持多 _ 任意 _ 任意 _0')
- Signal('日线三买多头 N1_ 持仓状态 _BS 辅助 V230808_ 持空 _ 任意 _ 任意 _0')
- Signal('日线三买多头 N1_ 持仓状态 _BS 辅助 V230808_ 持币 _ 任意 _ 任意 _0')

Parameters

- **cat** –CzscTrader 对象
- **kwargs** –参数字典 - pos_name: str，开仓信号的名称

Returns

pos_stop_V240428

czsc.signals.**pos_stop_V240428**(*cat:* CzscTrader, ***kwargs*) → OrderedDict

止损单，持有 N 根 K 线后，多头跌破前低或空头升破前高，平仓

参数模板："{pos_name}_{freq1}T{t}N{n}_ 止损 V240428"

信号逻辑：

以多头止损为例，计算过程如下：

1. 从多头开仓点开始，在给定的 K 线周期 freq1 上计算开仓 N 根 K 线后的最新价 close；
2. 计算开仓前的 unique_price 列表，获取低于开仓价的列表，降序排列后的第 t 个价位作为止损价 Y；
3. 如果 close < Y，则止损平仓。

信号列表：

- Signal（'日线三买多头 N1_60 分钟 T5N5_ 止损 V240428_ 空头止损 _ 任意 _ 任意 _0'）
- Signal（'日线三买多头 N1_60 分钟 T5N5_ 止损 V240428_ 多头止损 _ 任意 _ 任意 _0'）

Parameters

- **cat** –CzscTrader 对象
- **kwargs** –参数字典
  - pos_name: str，开仓信号的名称
  - freq1: str，给定的 K 线周期
  - t: int，止损多少跳，默认为 20
  - n: int，最少持有 K 线数量，默认为 5，表示 5 根 K 线之后开始判断

Returns

OrderedDict

pos_take_V240428

czsc.signals.**pos_take_V240428**(*cat:* CzscTrader, ***kwargs*) → OrderedDict

止盈单，持有 N 根 K 线后，多头持仓期间出现 T 根倍量阳线或空头持仓期间出现 T 根倍量阴线，平仓

参数模板："{pos_name}_{freq1}T{t}N{n}_ 止盈 V240428"

信号逻辑：

以多头为例，计算过程如下：

1. 从多头开仓点后 N 根 K 线开始，寻找倍量阳线，计算数量为 C；
2. 如果 C >= T，则止盈平仓。

信号列表：

- Signal（‘日线三买多头 N1_60 分钟 T5N5_ 止盈 V240428_ 空头止盈 _ 任意 _ 任意 _0’）
- Signal（‘日线三买多头 N1_60 分钟 T5N5_ 止盈 V240428_ 多头止盈 _ 任意 _ 任意 _0’）

Parameters

- **cat** –CzscTrader 对象
- **kwargs** –参数字典
  - pos_name: str，开仓信号的名称
  - freq1: str，给定的 K 线周期
  - t: int，倍量 K 线数量止盈，默认为 3
  - n: int，最少持有 K 线数量，默认为 5，表示 5 根 K 线之后开始判断

Returns

OrderedDict

pressure_support_V240222

czsc.signals.**pressure_support_V240222**(*c:* CZSC, ***kwargs*) → OrderedDict

支撑压力线辅助 V240222

参数模板：”{freq}_D{di}W{w} 高低点验证 _ 支撑压力 V240222”

信号逻辑：

给定窗口内，当前价格与前高前低的关系，判断当前价格的压力和支撑。以高点验证压力位为例：

1. 当前高点与前高的差值在 x 个标准差以内
2. 当前高点与前高分别在窗口的两端
3. 中间的最低价与高点的差值在 y 个标准差以外

信号列表：

- Signal（‘60 分钟 _D1W20 高低点验证 _ 支撑压力 V240222_ 支撑位 _ 任意 _ 任意 _0’）
- Signal（‘60 分钟 _D1W20 高低点验证 _ 支撑压力 V240222_ 压力位 _ 任意 _ 任意 _0’）

Parameters

- **c** –CZSC 对象
- **kwargs** –无

Returns

信号识别结果

pressure_support_V240402

czsc.signals.**pressure_support_V240402**(*c:* CZSC, ***kwargs*) → OrderedDict

支撑压力线辅助 V240402

参数模板："{freq}_D{di}W{w}_ 支撑压力 V240402"

信号逻辑：

对于给定 K 线，判断是否存在支撑压力线，判断逻辑如下：

1. 当前收盘价落在 5 个以上的分型高低点区间内；
2. 当前收盘价在最近 20 根 K 线的最高价附近，认为是压力位；反之，认为是支撑位

信号列表：

- Signal('60 分钟 _D1W60_ 支撑压力 V240402_ 压力位 _ 任意 _ 任意 _0')
- Signal('60 分钟 _D1W60_ 支撑压力 V240402_ 支撑位 _ 任意 _ 任意 _0')

Parameters

- **c** –CZSC 对象
- **kwargs** –无

Returns

信号识别结果

pressure_support_V240406

czsc.signals.**pressure_support_V240406**(*c:* CZSC, ***kwargs*) → OrderedDict

支撑压力线辅助 V240406

参数模板："{freq}_D{di}W{w}_ 支撑压力 V240406"

信号逻辑：

对于给定 K 线，判断是否存在支撑压力线，判断逻辑如下：

1. 当前收盘价在最近 20 根 K 线的最高价附近，认为是压力位；反之，认为是支撑位
2. 辅助判断：窗口内最高价附近有 3 个以上的分型

信号列表：

- Signal('60 分钟 _D1W60_ 支撑压力 V240406_ 压力位 _ 任意 _ 任意 _0')
- Signal('60 分钟 _D1W60_ 支撑压力 V240406_ 支撑位 _ 任意 _ 任意 _0')

Parameters

- **c** –CZSC 对象
- **kwargs** –无

Returns

信号识别结果

skdj_up_dw_line_V230611

czsc.signals.**skdj_up_dw_line_V230611**(*c:* CZSC, ***kwargs*) → OrderedDict

SKDJ 随机波动指标，贡献者：琅盎

参数模板："{freq}_D{di}N{n}M{m}UP{up}DW{dw}_SKDJ 随机波动 V230611"

信号逻辑：

SKDJ 为慢速随机波动（即慢速 KDJ）。SKDJ 中的 K 即 KDJ 中的 D，SKJ 中的 D 即 KDJ 中的 D 取移动平均。其用法与 KDJ 相同。当 D<40(处于超卖状态) 且 K 上穿 D 时买入，当 D>60（处于超买状态）K 下穿 D 时卖出。

信号列表：

- Signal('日线 _D1N233M145UP60DW40_SKDJ 随机波动 V230611_ 看多 _ 任意 _ 任意 _0')
- Signal('日线 _D1N233M145UP60DW40_SKDJ 随机波动 V230611_ 看空 _ 任意 _ 任意 _0')

Parameters

- **c** –CZSC 对象
- **kwargs** –参数字典 - :param di: 信号计算截止倒数第 i 根 K 线 - :param n: 取 K 线数量 n 必需大于 m*2 - :param m: 计算均值需要的参数 - :param up: 信号预警值 - :param dw: 信号预警值

Returns

信号识别结果

tas_accelerate_V230531

czsc.signals.**tas_accelerate_V230531**(*c:* CZSC, ***kwargs*) → OrderedDict

BOLL 辅助判断加速行情

参数模板："{freq}_D{di}N{n}T{t}_BOLL 加速 V230531"

信号逻辑：

最近 N 根 K 线全部在中轨上方，上轨累计涨幅大于中轨 2 倍以上，多头加速，反之空头加速。

信号列表：

- Signal('60 分钟 _D1N20T20_BOLL 加速 V230531_ 空头加速 _ 未破下轨 _ 任意 _0')
- Signal('60 分钟 _D1N20T20_BOLL 加速 V230531_ 空头加速 _ 跌破下轨 _ 任意 _0')
- Signal('60 分钟 _D1N20T20_BOLL 加速 V230531_ 多头加速 _ 升破上轨 _ 任意 _0')
- Signal('60 分钟 _D1N20T20_BOLL 加速 V230531_ 多头加速 _ 未破上轨 _ 任意 _0')

Parameters

- **c** –CZSC 对象
- **kwargs** –参数字典
  - di: 倒数第几根 K 线

- n: 取截止 dik 的前 n 根 K 线
- t: 阈值，上下轨变化率超过中轨变化率的 t / 10 倍，默认 20，即上下轨变化率超过中轨变化率的 2 倍

Returns

返回信号结果

tas_angle_V230802

czsc.signals.**tas_angle_V230802**(*c:* CZSC, ***kwargs*) → OrderedDict

笔的角度比较贡献者：谌意勇

参数模板："{freq}_D{di}N{n}T{th}_ 笔角度 V230802"

信号逻辑：

笔的角度，走过的笔的空间最高价和最低价的空间与走过的时间（原始 K 的数量）形成比值。如果当前笔的角度小于前面 9 笔的平均角度的 50%，当前笔向上认为是空头笔，否则是多头笔。

信号列表：

- Signal('60 分钟 _D1N9T50_ 笔角度 V230802_ 空头 _ 任意 _ 任意 _0')
- Signal('60 分钟 _D1N9T50_ 笔角度 V230802_ 多头 _ 任意 _ 任意 _0')

Parameters

- **c** –CZSC 对象
- **kwargs** –n：统计笔的数量 -di：取第几笔

Returns

信号识别结果

tas_atr_V230630

czsc.signals.**tas_atr_V230630**(*c:* CZSC, ***kwargs*) → OrderedDict

ATR 波动强弱

参数模板："{freq}_D{di}ATR{timeperiod}_ 波动 V230630"

信号逻辑：

ATR 与收盘价的比值衡量了价格振幅比率的大小，对这个值进行分层。

信号列表：

- Signal('日线 _D1ATR14_ 波动 V230630_ 第 7 层 _ 任意 _ 任意 _0')
- Signal('日线 _D1ATR14_ 波动 V230630_ 第 6 层 _ 任意 _ 任意 _0')
- Signal('日线 _D1ATR14_ 波动 V230630_ 第 8 层 _ 任意 _ 任意 _0')
- Signal('日线 _D1ATR14_ 波动 V230630_ 第 9 层 _ 任意 _ 任意 _0')
- Signal('日线 _D1ATR14_ 波动 V230630_ 第 10 层 _ 任意 _ 任意 _0')

- Signal(‘日线 _D1ATR14_ 波动 V230630_ 第 5 层 _ 任意 _ 任意 _0’)
- Signal(‘日线 _D1ATR14_ 波动 V230630_ 第 4 层 _ 任意 _ 任意 _0’)
- Signal(‘日线 _D1ATR14_ 波动 V230630_ 第 3 层 _ 任意 _ 任意 _0’)
- Signal(‘日线 _D1ATR14_ 波动 V230630_ 第 2 层 _ 任意 _ 任意 _0’)
- Signal(‘日线 _D1ATR14_ 波动 V230630_ 第 1 层 _ 任意 _ 任意 _0’)

Parameters

- **c** –czsc 对象
- **kwargs** –
  - di: 倒数第 i 根 K 线
  - timeperiod: ATR 指标的参数

Returns

信号字典

tas_atr_break_V230424

czsc.signals.**tas_atr_break_V230424**(*c:* CZSC, ***kwargs*)

ATR 突破

参数模板:”{freq}_D{di}ATR{timeperiod}T{th} 突破 _BS 辅助 V230424”

信号逻辑:

1. 以 ATR 为基础的通道突破;
2. close 向上突破 LL + th * ATR, 看多;
3. close 向下突破 HH - th * ATR，看空

信号列表:

- Signal(‘日线 _D1ATR5T30 突破 _BS 辅助 V230424_ 看空 _ 任意 _ 任意 _0’)
- Signal(‘日线 _D1ATR5T30 突破 _BS 辅助 V230424_ 看多 _ 任意 _ 任意 _0’)

Parameters

- **c** –基础周期的 CZSC 对象
- **kwargs** –其他参数 - di: 倒数第 di 根 K 线 - timeperiod: ATR 的计算周期 - th: ATR 突破的倍数，根据经验优化

Returns

信号字典

tas_boll_bc_V221118

czsc.signals.**tas_boll_bc_V221118**(*c:* CZSC, ***kwargs*)

BOLL 背驰辅助

参数模板："{freq}_D{di}N{n}M{m}L{line}#BOLL{timeperiod}_ 背驰 V221118"

信号逻辑：

近 n 个最低价创近 m 个周期新低，近 m 个周期跌破下轨，近 n 个周期不破下轨，这是 BOLL 一买（底部背驰）信号，顶部背驰反之。

信号列表：

- Signal('15 分钟 _D1N3M10L3#BOLL20_ 背驰 V221118_ 一卖 _ 任意 _ 任意 _0')
- Signal('15 分钟 _D1N3M10L3#BOLL20_ 背驰 V221118_ 一买 _ 任意 _ 任意 _0')

Parameters

- **c** –CZSC 对象
- **di** –倒数第 di 根 K 线
- **n** –近 n 个周期
- **m** –近 m 个周期
- **line** –选第几个上下轨

Returns

tas_boll_cc_V230312

czsc.signals.**tas_boll_cc_V230312**(*c:* CZSC, ***kwargs*) → OrderedDict

多空进出场信号，贡献者：琅盎

参数模板："{freq}_D{di}BOLL{timeperiod}S{nbdev}SP{sp}_BS 辅助 V230312"

信号逻辑：

1. 收盘在布林线上轨下方，且跌破中线一定距离，看空；反之，看多。

信号列表：

- Signal('15 分钟 _D1BOLL20S20SP400_BS 辅助 V230312_ 看多 _ 任意 _ 任意 _0')
- Signal('15 分钟 _D1BOLL20S20SP400_BS 辅助 V230312_ 看空 _ 任意 _ 任意 _0')

Parameters

- **c** –CZSC 对象
- **di** –信号计算截止倒数第 i 根 K 线
- **sp** –停盈点数，单位：BP；1BP = 0.01%

Returns

tas_boll_power_V221112

czsc.signals.**tas_boll_power_V221112**(*c:* CZSC, ***kwargs*)

BOLL 指标强弱

参数模板："{freq}_D{di}BOLL{timeperiod}_ 强弱 V221112"

信号逻辑：

1. close 大于中线，多头；反之，空头
2. close 超过轨 3，超强，以此类推

信号列表：

- Signal('15 分钟 _D1BOLL20_ 强弱 V221112_ 空头 _ 强势 _ 任意 _0')
- Signal('15 分钟 _D1BOLL20_ 强弱 V221112_ 空头 _ 弱势 _ 任意 _0')
- Signal('15 分钟 _D1BOLL20_ 强弱 V221112_ 空头 _ 超强 _ 任意 _0')
- Signal('15 分钟 _D1BOLL20_ 强弱 V221112_ 多头 _ 弱势 _ 任意 _0')
- Signal('15 分钟 _D1BOLL20_ 强弱 V221112_ 空头 _ 极强 _ 任意 _0')
- Signal('15 分钟 _D1BOLL20_ 强弱 V221112_ 多头 _ 强势 _ 任意 _0')
- Signal('15 分钟 _D1BOLL20_ 强弱 V221112_ 多头 _ 超强 _ 任意 _0')
- Signal('15 分钟 _D1BOLL20_ 强弱 V221112_ 多头 _ 极强 _ 任意 _0')

Parameters

- **c** –CZSC 对象
- **kwargs** –其他参数 - di: 信号计算截止倒数第 i 根 K 线 - timeperiod: BOLL 指标计算周期

Returns

s

tas_boll_vt_V230212

czsc.signals.**tas_boll_vt_V230212**(*c:* CZSC, ***kwargs*) → OrderedDict

以 BOLL 通道为依据的多空进出场信号

参数模板："{freq}_D{di}BOLL{timeperiod}S{nbdev}MO{max_overlap}_BS 辅助 V230212"

信号逻辑：

1. 看多，当日收盘价在上轨上方，且最近 max_overlap 根 K 线中至少有一个收盘价都在上轨下方；
2. 看空，当日收盘价在下轨下方，且最近 max_overlap 根 K 线中至少有一个收盘价都在下轨上方；

信号列表：

- Signal('15 分钟 _D1BOLL20S20MO5_BS 辅助 V230212_ 看空 _ 任意 _ 任意 _0')
- Signal('15 分钟 _D1BOLL20S20MO5_BS 辅助 V230212_ 看多 _ 任意 _ 任意 _0')

Parameters

- **c** –CZSC 对象

- **di** –信号计算截止倒数第 i 根 K 线

Returns

tas_cci_base_V230402

czsc.signals.**tas_cci_base_V230402**(*c:* CZSC, ***kwargs*) → OrderedDict

CCI 基础信号

参数模板："{freq}_D{di}CCI{timeperiod}#{min_count}#{max_count}_BS 辅助 V230402"

信号逻辑：

1. CCI 连续大于 100 的次数大于 min_count, 小于 max_count，看空；反之，看多。

信号列表：

- Signal('60 分钟 _D1CCI14#3#6_BS 辅助 V230402_ 空头 _ 任意 _ 任意 _0')
- Signal('60 分钟 _D1CCI14#3#6_BS 辅助 V230402_ 多头 _ 任意 _ 任意 _0')

Parameters

- **c** –CZSC 对象
- **kwargs** –参数字典 - di: int, 默认 1，倒数第几根 K 线 - timeperiod: int, 默认 14，计算 CCI 的周期 - min_count: int, 默认 3，CCI 连续大于 100 的次数 - max_count: int, 默认 min_count+3，CCI 连续大于 100 的次数

Returns

信号识别结果

tas_cross_status_V230619

czsc.signals.**tas_cross_status_V230619**(*c:* CZSC, ***kwargs*) → OrderedDict

0 轴上下金死叉次数计算信号函数贡献者：谌意勇

参数模板："{freq}_D{di}MACD{fastperiod}#{slowperiod}#{signalperiod}_ 金死叉 V230619"

信号逻辑：

精确确立 MACD 指标中 0 轴以上或以下位置第几次金叉和死叉，作为开仓的辅助买点：

信号列表：

- Signal('日线 _D1MACD12#26#9_ 金死叉 V230619_0 轴上死叉第 2 次 _ 任意 _ 任意 _0')
- Signal('日线 _D1MACD12#26#9_ 金死叉 V230619_0 轴下金叉第 1 次 _ 任意 _ 任意 _0')
- Signal('日线 _D1MACD12#26#9_ 金死叉 V230619_0 轴下死叉第 1 次 _ 任意 _ 任意 _0')
- Signal('日线 _D1MACD12#26#9_ 金死叉 V230619_0 轴下金叉第 2 次 _ 任意 _ 任意 _0')
- Signal('日线 _D1MACD12#26#9_ 金死叉 V230619_0 轴下死叉第 2 次 _ 任意 _ 任意 _0')
- Signal('日线 _D1MACD12#26#9_ 金死叉 V230619_0 轴下金叉第 3 次 _ 任意 _ 任意 _0')
- Signal('日线 _D1MACD12#26#9_ 金死叉 V230619_0 轴上死叉第 1 次 _ 任意 _ 任意 _0')
- Signal('日线 _D1MACD12#26#9_ 金死叉 V230619_0 轴上金叉第 1 次 _ 任意 _ 任意 _0')

- Signal(‘日线 _D1MACD12#26#9_ 金死叉 V230619_0 轴下死叉第 3 次 _ 任意 _ 任意 _0’)
- Signal(‘日线 _D1MACD12#26#9_ 金死叉 V230619_0 轴下金叉第 4 次 _ 任意 _ 任意 _0’)

Parameters

- **c** –CZSC 对象
- **kwargs** –参数字典
  - **param di**
    信号计算截止倒数第 i 根 K 线
  - **param fastperiod**
    MACD 快线周期
  - **param slowperiod**
    MACD 慢线周期
  - **param signalperiod**
    MACD 信号线周期

Returns

信号识别结果

tas_cross_status_V230624

czsc.signals.**tas_cross_status_V230624**(*c:* CZSC, ***kwargs*) → OrderedDict

指定金死叉数值信号函数, 以此来确定 MACD 交易区间贡献者：谌意勇

参数模板：”{freq}_D{di}N{n}MD{md}_MACD 交叉数量 V230624”

信号逻辑：

1、通过指定 0 轴上下金死叉数量，来选择自己想要的指标形态，通过配合其他信号函数出信号 2、金叉数量和死叉数量要注意连续对应。0 轴上一定是第一次先死叉，再金叉，死叉的数值同
金叉数值相比永远是相等或者大 1，不能出现 >=2 的情况，0 轴下则反之。

信号列表：

- Signal(‘日线 _D1N100MD1_MACD 交叉数量 V230624_0 轴上金叉第 1 次 _0 轴上死叉第 1 次 _ 任意 _0’)
- Signal(‘日线 _D1N100MD1_MACD 交叉数量 V230624_0 轴上金叉第 1 次 _0 轴上死叉第 2 次 _ 任意 _0’)
- Signal(‘日线 _D1N100MD1_MACD 交叉数量 V230624_0 轴下金叉第 0 次 _0 轴下死叉第 0 次 _ 任意 _0’)
- Signal(‘日线 _D1N100MD1_MACD 交叉数量 V230624_0 轴下金叉第 1 次 _0 轴下死叉第 0 次 _ 任意 _0’)
- Signal(‘日线 _D1N100MD1_MACD 交叉数量 V230624_0 轴下金叉第 1 次 _0 轴下死叉第 1 次 _ 任意 _0’)

- Signal(‘日线 _D1N100MD1_MACD 交叉数量 V230624_0 轴下金叉第 2 次 _0 轴下死叉第 1 次 _ 任意 _0’)
- Signal(‘日线 _D1N100MD1_MACD 交叉数量 V230624_0 轴下金叉第 2 次 _0 轴下死叉第 2 次 _ 任意 _0’)
- Signal(‘日线 _D1N100MD1_MACD 交叉数量 V230624_0 轴下金叉第 3 次 _0 轴下死叉第 2 次 _ 任意 _0’)
- Signal(‘日线 _D1N100MD1_MACD 交叉数量 V230624_0 轴上金叉第 0 次 _0 轴上死叉第 0 次 _ 任意 _0’)
- Signal(‘日线 _D1N100MD1_MACD 交叉数量 V230624_0 轴上金叉第 0 次 _0 轴上死叉第 1 次 _ 任意 _0’)
- Signal(‘日线 _D1N100MD1_MACD 交叉数量 V230624_0 轴下金叉第 3 次 _0 轴下死叉第 3 次 _ 任意 _0’)
- Signal(‘日线 _D1N100MD1_MACD 交叉数量 V230624_0 轴下金叉第 4 次 _0 轴下死叉第 3 次 _ 任意 _0’)

Parameters

- **c** –czsc 对象
- **kwargs** –
  - di: 倒数第 i 根 K 线
  - n: 从 dik 往前数 n 根 k 线（此数值不需要精确，函数会自动截取最后上下 0 轴以后的数据）
  - **md: 抖动过滤参数, 金死叉之间格距离小于此数值，将被忽略（去除一些杂波扰动因素, 最小值不小于 1）**

0 轴上下金死叉状态信息，与其他信号加以辅助操作。

Returns

信号字典

tas_cross_status_V230625

czsc.signals.**tas_cross_status_V230625**(*c:* CZSC, ***kwargs*) → OrderedDict

指定金死叉数值信号函数, 以此来确定 MACD 交易区间贡献者：谌意勇

参数模板：”{freq}_D{di}N{n}MD{md}J{j}S{s}_MACD 交叉数量 V230625”

信号逻辑：

1、通过指定 jc 或者 sc 数值来确定为哪第几次金叉或死叉之后的信号。两者最少要指定一个，并且指定其中一个时，另外一个需为 0.

信号列表：

- Signal(‘15 分钟 _D1N100MD1J3S0_MACD 交叉数量 V230625_0 轴下第 3 次金叉以后 _ 任意 _ 任意 _0’)

- Signal(‘15 分钟 _D1N100MD1J3S0_MACD 交叉数量 V230625_0 轴上第 3 次金叉以后 _ 任意 _ 任意 _0’)

  Parameters

  - **c** –czsc 对象
  - **kwargs** –
    - di: 倒数第 i 根 K 线
    - j: 金叉数值
    - s: 死叉数值
    - n: 从 dik 往前数 n 根 k 线（此数值不需要精确，函数会自动截取最后上下 0 轴以后的数据）
    - **md: 抖动过滤参数, 金死叉之间格距离小于此数值，将被忽略（去除一些杂波扰动因素, 最小值不小于 1**
      0 轴上下金死叉状态信息，与其他信号加以辅助操作。

  Returns

  信号字典

tas_double_ma_V221203

czsc.signals.**tas_double_ma_V221203**(*c:* CZSC, ***kwargs*) → OrderedDict

双均线多空和强弱信号

参数模板:”{freq}_D{di}T{th}#{ma_type}#{timeperiod1}#{timeperiod2}_JX 辅助 V221203”

信号逻辑:

1. ma1 > ma2，多头；反之，空头
2. ma1 离开 ma2 的距离大于 th，强势；反之，弱势

信号列表:

- Signal(‘15 分钟 _D1T100#SMA#5#10_JX 辅助 V221203_ 空头 _ 弱势 _ 任意 _0’)
- Signal(‘15 分钟 _D1T100#SMA#5#10_JX 辅助 V221203_ 多头 _ 弱势 _ 任意 _0’)
- Signal(‘15 分钟 _D1T100#SMA#5#10_JX 辅助 V221203_ 多头 _ 强势 _ 任意 _0’)
- Signal(‘15 分钟 _D1T100#SMA#5#10_JX 辅助 V221203_ 空头 _ 强势 _ 任意 _0’)

  Parameters

  - **c** –CZSC 对象
  - **di** –信号计算截止倒数第 i 根 K 线
  - **ma_type** –均线类型，必须是 *ma_type_map* 中的 key
  - **ma_seq** –快慢均线计算周期，快线在前
  - **th** –ma1 相比 ma2 的距离阈值，单位 BP

Returns

信号识别结果

tas_double_ma_V230511

czsc.signals.**tas_double_ma_V230511**(*c:* CZSC, ***kwargs*)

双均线金叉死叉后的反向信号

参数模板："{freq}_D{di}#{ma_type}#{t1}#{t2}_BS 辅助 V230511"

信号逻辑：

1. t1 周期均线上穿 t2 周期均线，且当前 K 线为大实体阴线，看多信号；
2. t1 周期均线下穿 t2 周期均线，且当前 K 线为大实体阳线，看空信号；

信号列表：

- Signal('日线 _D2#SMA#5#20_BS 辅助 V230511_ 看空 _ 任意 _ 任意 _0')
- Signal('日线 _D2#SMA#5#20_BS 辅助 V230511_ 看多 _ 第一个 _ 任意 _0')
- Signal('日线 _D2#SMA#5#20_BS 辅助 V230511_ 看多 _ 任意 _ 任意 _0')
- Signal('日线 _D2#SMA#5#20_BS 辅助 V230511_ 看空 _ 第一个 _ 任意 _0')

Parameters

- **c** –基础周期的 CZSC 对象
- **kwargs** –其他参数 - di: 倒数第 di 根 K 线 - t1: 均线 1 周期 - t2: 均线 2 周期 - ma_type: 均线类型，支持：MA, EMA, WMA, DEMA, TEMA, TRIMA, KAMA, MAMA, T3

Returns

信号字典

tas_double_ma_V240208

czsc.signals.**tas_double_ma_V240208**(*c:* CZSC, ***kwargs*) → OrderedDict

双均线多空信号，辅助 V240208

参数模板："{freq}_D{di}N{N}M{M} 双均线 _BS 辅助 V240208"

信号逻辑：

1. 找出最近 3 个均线交叉点，时间上由远到近，分别为 X1，X2，X3
2. 以多头为例：X3 和 X1 为金叉，且 X2 的价格最高

信号列表：

- Signal('60 分钟 _D1N5M21 双均线 _BS 辅助 V240208_ 多头 _ 任意 _ 任意 _0')
- Signal('60 分钟 _D1N5M21 双均线 _BS 辅助 V240208_ 空头 _ 任意 _ 任意 _0')

Parameters

- **c** –CZSC 对象

- **kwargs** –参数设置
  - di: int, default 1, 倒数第几根 K 线
  - N: int, default 20, 快线周期
  - M: int, default 60, 慢线周期

Returns

信号识别结果

tas_first_bs_V230217

czsc.signals.**tas_first_bs_V230217**(*c:* CZSC, ***kwargs*) → OrderedDict

均线结合 K 线形态的一买一卖辅助判断

参数模板："{freq}_D{di}N{n}#{ma_type}#{timeperiod}_BS1 辅助 V230217"

信号逻辑：

1. 窗口 N 内的 K 线的最低点全部小于 SMA5，且阴线数量占比超过 60%，且最近三根 K 线创新低，最后一根 K 线收在 SMA5 上方，看多；
2. 反之，看空。

信号列表：

- Signal('15 分钟 _D1N10#SMA#5_BS1 辅助 V230217_ 一买 _ 任意 _ 任意 _0')
- Signal('15 分钟 _D1N10#SMA#5_BS1 辅助 V230217_ 一卖 _ 任意 _ 任意 _0')

Parameters

- **c** –CZSC 对象
- **kwargs** –
  - di: 倒数第几根 K 线，1 表示最后一根 K 线
  - n: 窗口大小
  - ma_type: 均线类型
  - timeperiod: 均线周期

Returns

信号识别结果

tas_hlma_V230301

czsc.signals.**tas_hlma_V230301**(*c:* CZSC, ***kwargs*) → OrderedDict

HMA 多空信号，贡献者：琅盎

参数模板："{freq}_D{di}#{ma_type}#{timeperiod}HLMA_BS 辅助 V230301"

信号逻辑：

1. 收盘价大于 HMA and 上一根 K 线的收盘价小于均线
2. 收盘价小于 LMA and 上一根 K 线的收盘价大于均线

信号列表：

- Signal('15 分钟 _D1#SMA#3HLMA_BS 辅助 V230301_ 看多 _ 任意 _ 任意 _0')
- Signal('15 分钟 _D1#SMA#3HLMA_BS 辅助 V230301_ 看空 _ 任意 _ 任意 _0')

Parameters

- **c** –CZSC 对象
- **kwargs** –其他参数 - di: 信号计算截止倒数第 i 根 K 线 - ma_type: 均线类型，必须是 ma_type_map 中的 key - timeperiod: 均线周期

Returns

信号识别结果

tas_kdj_base_V221101

czsc.signals.**tas_kdj_base_V221101**(*c:* CZSC, ***kwargs*) → OrderedDict

KDJ 金叉死叉信号

参数模板："{freq}_D{di}K#KDJ{fastk_period}#{slowk_period}#{slowd_period}_KDJ 辅助 V221101"

信号逻辑：

1. J > K > D，多头；反之，空头
2. J 值定方向

信号列表：

- Signal('15 分钟 _D1K#KDJ9#3#3_KDJ 辅助 V221101_ 空头 _ 向下 _ 任意 _0')
- Signal('15 分钟 _D1K#KDJ9#3#3_KDJ 辅助 V221101_ 多头 _ 向上 _ 任意 _0')
- Signal('15 分钟 _D1K#KDJ9#3#3_KDJ 辅助 V221101_ 多头 _ 向下 _ 任意 _0')
- Signal('15 分钟 _D1K#KDJ9#3#3_KDJ 辅助 V221101_ 空头 _ 向上 _ 任意 _0')

Parameters

- **c** –CZSC 对象
- **di** –信号计算截止倒数第 i 根 K 线

Returns

tas_kdj_evc_V221201

czsc.signals.**tas_kdj_evc_V221201**(*c:* CZSC, ***kwargs*) → OrderedDict

KDJ 极值计数信号, evc 是 extreme value counts 的首字母缩写

参数模板: "{freq}_D{di}T{th}KDJ{fastk_period}#{slowk_period}#{slowd_period}#{key} 值突破 {c1}#{c2}_KDJ 极值 V221201"

信号逻辑:

1. K < th，记录一次多头信号，连续出现信号次数在 count_range 范围，则认为是有效多头信号；
2. K > 100 - th, 记录一次空头信号，连续出现信号次数在 count_range 范围，则认为是有效空头信号

信号列表:

- Signal('15 分钟 _D1T10KDJ9#3#3#K 值突破 5#8_KDJ 极值 V221201_ 多头 _C5_ 任意 _0')
- Signal('15 分钟 _D1T10KDJ9#3#3#K 值突破 5#8_KDJ 极值 V221201_ 多头 _C6_ 任意 _0')
- Signal('15 分钟 _D1T10KDJ9#3#3#K 值突破 5#8_KDJ 极值 V221201_ 空头 _C5_ 任意 _0')
- Signal('15 分钟 _D1T10KDJ9#3#3#K 值突破 5#8_KDJ 极值 V221201_ 空头 _C6_ 任意 _0')
- Signal('15 分钟 _D1T10KDJ9#3#3#K 值突破 5#8_KDJ 极值 V221201_ 空头 _C7_ 任意 _0')
- Signal('15 分钟 _D1T10KDJ9#3#3#K 值突破 5#8_KDJ 极值 V221201_ 多头 _C7_ 任意 _0')

Parameters

- **c** –CZSC 对象
- **di** –信号计算截止倒数第 i 根 K 线
- **key** –KDJ 值的名称，可以是 K，D，J
- **th** –信号计算截止倒数第 i 根 K 线
- **count_range** –信号计数范围

Returns

tas_kdj_evc_V230401

czsc.signals.**tas_kdj_evc_V230401**(*c:* CZSC, ***kwargs*) → OrderedDict

KDJ 极值计数信号, evc 是 extreme value counts 的首字母缩写

参数模板: "{freq}_D{di}T{th}KDJ{fastk_period}#{slowk_period}#{slowd_period}#{key} 值突破 {min_count}#{max_count}_BS 辅助 V230401"

信号逻辑:

1. K < th，记录一次多头信号，连续出现信号次数在 count_range 范围，则认为是有效多头信号；
2. K > 100 - th, 记录一次空头信号，连续出现信号次数在 count_range 范围，则认为是有效空头信号

信号列表:

- Signal('60 分钟 _D1T10KDJ9#3#3#K 值突破 5#8_BS 辅助 V230401_ 空头 _ 任意 _ 任意 _0')
- Signal('60 分钟 _D1T10KDJ9#3#3#K 值突破 5#8_BS 辅助 V230401_ 多头 _ 任意 _ 任意 _0')

Parameters

- **c** –CZSC 对象
- **kwargs** –参数字典 - di: 信号计算截止倒数第 i 根 K 线 - key: KDJ 值的名称，可以是 K，D，J - th: 信号计算截止倒数第 i 根 K 线 - min_count: 连续出现信号次数的最小值 - max_count: 连续出现信号次数的最大值

Returns

tas_low_trend_V230627

czsc.signals.**tas_low_trend_V230627**(*c:* CZSC, ***kwargs*) → OrderedDict

阴跌趋势、小阳趋势

参数模板："{freq}_D{di}N{n}TH{th}_ 趋势 230627"

信号逻辑：

1、阴跌趋势：在连续 N 根 K 线上 rolling 计数，如果当前最低价小于 rolling min close，min_count + 1 ，当 min_count > 0.8 * n 且 N 根 K 线中振幅超过 TH 的 K 线数量小于 0.2 * N，则为阴跌趋势；

2. **小阳趋势：在连续 N 根 K 线上 rolling 计数，如果当前最高价大于 rolling max close，max_count + 1** ，当 max_count > 0.8 * n 且 N 根 K 线中振幅超过 TH 的 K 线数量小于 0.2 * N，则为小阳趋势；

信号列表：

- Signal('15 分钟 _D1N13TH500_ 趋势 230627_ 阴跌趋势 _ 任意 _ 任意 _0')
- Signal('15 分钟 _D1N13TH500_ 趋势 230627_ 小阳趋势 _ 任意 _ 任意 _0')

Parameters

- **c** –czsc 对象
- **kwargs** –
  - di: 倒数第 i 根 K 线
  - n: 从 dik 往前数 n 根 k 线（此数值不需要精确，函数会自动截取最后上下 0 轴以后的数据）
  - th: 实体振幅阈值，单位为 BP

Returns

信号字典

tas_ma_base_V221101

czsc.signals.**tas_ma_base_V221101**(*c:* CZSC, ***kwargs*) → OrderedDict

MA 多空和方向信号

参数模板："{freq}_D{di}{ma_type}#{timeperiod}_ 分类 V221101"

信号逻辑：

1. close > ma，多头；反之，空头
2. ma[-1] > ma[-2]，向上；反之，向下

信号列表：

- Signal('15 分钟 _D1SMA#5_ 分类 V221101_ 空头 _ 向下 _ 任意 _0')
- Signal('15 分钟 _D1SMA#5_ 分类 V221101_ 多头 _ 向下 _ 任意 _0')
- Signal('15 分钟 _D1SMA#5_ 分类 V221101_ 多头 _ 向上 _ 任意 _0')
- Signal('15 分钟 _D1SMA#5_ 分类 V221101_ 空头 _ 向上 _ 任意 _0')

Parameters

- **c** –CZSC 对象
- **kwargs** –
  - di: 信号计算截止倒数第 i 根 K 线
  - ma_type: 均线类型，必须是 *ma_type_map* 中的 key
  - timeperiod: 均线计算周期

Returns

tas_ma_base_V221203

czsc.signals.**tas_ma_base_V221203**(*c:* CZSC, ***kwargs*) → OrderedDict

MA 多空和方向信号，加距离限制

参数模板："{freq}_D{di}{ma_type}#{timeperiod}T{th}_ 分类 V221203"

信号逻辑：

1. close > ma，多头；反之，空头
2. ma[-1] > ma[-2]，向上；反之，向下
3. close 与 ma 的距离超过 th

信号列表：

- Signal('15 分钟 _D1SMA#5T100_ 分类 V221203_ 空头 _ 向下 _ 靠近 _0')
- Signal('15 分钟 _D1SMA#5T100_ 分类 V221203_ 多头 _ 向下 _ 靠近 _0')
- Signal('15 分钟 _D1SMA#5T100_ 分类 V221203_ 多头 _ 向上 _ 靠近 _0')
- Signal('15 分钟 _D1SMA#5T100_ 分类 V221203_ 空头 _ 向上 _ 靠近 _0')
- Signal('15 分钟 _D1SMA#5T100_ 分类 V221203_ 空头 _ 向下 _ 远离 _0')
- Signal('15 分钟 _D1SMA#5T100_ 分类 V221203_ 多头 _ 向上 _ 远离 _0')

- Signal(‘15 分钟 _D1SMA#5T100_ 分类 V221203_ 多头 _ 向下 _ 远离 _0’)
- Signal(‘15 分钟 _D1SMA#5T100_ 分类 V221203_ 空头 _ 向上 _ 远离 _0’)

Parameters

- **c** –CZSC 对象
- **kwargs** –
  - di: 信号计算截止倒数第 i 根 K 线
  - ma_type: 均线类型，必须是 *ma_type_map* 中的 key
  - timeperiod: 均线计算周期
  - th: 距离阈值，单位 BP

Returns

信号识别结果

tas_ma_base_V230313

czsc.signals.**tas_ma_base_V230313**(*c:* CZSC, ***kwargs*) → OrderedDict

单均线多空和方向辅助开平仓信号

参数模板：”{freq}_D{di}#{ma_type}#{timeperiod}MO{max_overlap}_BS 辅助 V230313”

信号逻辑：

1. close > ma，多头；反之，空头
2. ma[-1] > ma[-2]，向上；反之，向下
3. 加入 max_overlap 参数控制相同信号最大重叠次数

信号列表：

- Signal(‘15 分钟 _D1#SMA#5MO5_BS 辅助 V230313_ 看空 _ 向下 _ 任意 _0’)
- Signal(‘15 分钟 _D1#SMA#5MO5_BS 辅助 V230313_ 看多 _ 向下 _ 任意 _0’)
- Signal(‘15 分钟 _D1#SMA#5MO5_BS 辅助 V230313_ 看多 _ 向上 _ 任意 _0’)
- Signal(‘15 分钟 _D1#SMA#5MO5_BS 辅助 V230313_ 看空 _ 向上 _ 任意 _0’)

Parameters

- **c** –CZSC 对象
- **kwargs** –其他参数 - ma_type: 均线类型，必须是 *ma_type_map* 中的 key - timeperiod: 均线计算周期 - di: 信号计算截止倒数第 i 根 K 线 - max_overlap: 相同信号最大重叠次数

Returns

信号识别结果

tas_ma_round_V221206

czsc.signals.**tas_ma_round_V221206**(*c:* CZSC, ***kwargs*) → OrderedDict

笔端点在均线附近，贡献者：谌意勇

参数模板："{freq}_D{di}TH{th}# 碰 {ma_type}#{timeperiod}_BE 辅助 V221206"

信号逻辑：

倒数第 i 笔的端点到均线的绝对价差 / 笔的价差 < th / 100 表示笔端点在均线附近

信号列表：

- Signal('15 分钟 _D1TH10# 碰 SMA#60_BE 辅助 V221206_ 上碰 _ 任意 _ 任意 _0')
- Signal('15 分钟 _D1TH10# 碰 SMA#60_BE 辅助 V221206_ 下碰 _ 任意 _ 任意 _0')

Parameters

- **c** –CZSC 对象
- **di** –指定倒数第几笔
- **ma_type** –均线类型，必须是 *ma_type_map* 中的 key
- **timeperiod** –均线计算周期
- **th** –笔的端点到均线的绝对价差 / 笔的价差 < th / 100 表示笔端点在均线附近

Returns

信号识别结果

tas_ma_system_V230513

czsc.signals.**tas_ma_system_V230513**(*c:* CZSC, ***kwargs*) → OrderedDict

均线系统多空排列

参数模板："{freq}_D{di}SMA{ma_seq}_ 均线系统 V230513"

信号逻辑：

1. 5 日均线 > 10 日均线 > 20 日均线，多头排列，以此类推；
2. 5 日均线 < 10 日均线 < 20 日均线，空头排列，以此类推；

信号列表：

- Signal('60 分钟 _D1SMA5#10#20_ 均线系统 V230513_ 多头排列 _ 任意 _ 任意 _0')
- Signal('60 分钟 _D1SMA5#10#20_ 均线系统 V230513_ 空头排列 _ 任意 _ 任意 _0')

Parameters

- **c** –CZSC 对象
- **kwargs** –参数字典:return: 返回信号结果

tas_macd_base_V221028

czsc.signals.**tas_macd_base_V221028**(*c:* CZSC, ***kwargs*) → OrderedDict

MACD|DIF|DEA 多空和方向信号

参数模板："{freq}_D{di}MACD{fastperiod}#{slowperiod}#{signalperiod}#{key}_BS 辅助 V221028"

信号逻辑：

1. dik 对应的 MACD 值大于 0，多头；反之，空头
2. dik 的 MACD 值大于上一个值，向上；反之，向下

信号列表：

- Signal('15 分钟 _D1MACD12#26#9#MACD_BS 辅助 V221028_ 空头 _ 向下 _ 任意 _0')
- Signal('15 分钟 _D1MACD12#26#9#MACD_BS 辅助 V221028_ 空头 _ 向上 _ 任意 _0')
- Signal('15 分钟 _D1MACD12#26#9#MACD_BS 辅助 V221028_ 多头 _ 向上 _ 任意 _0')
- Signal('15 分钟 _D1MACD12#26#9#MACD_BS 辅助 V221028_ 多头 _ 向下 _ 任意 _0')

参数列表：

Parameters

- **c** –CZSC 对象
- **di** –倒数第 i 根 K 线
- **key** –指定使用哪个 Key 来计算，可选值 [macd, dif, dea]

Returns

tas_macd_base_V230320

czsc.signals.**tas_macd_base_V230320**(*c:* CZSC, ***kwargs*) → OrderedDict

MACD|DIF|DEA 多空和方向信号，支持 max_overlap 参数

参数模板："{freq}_D{di}MACD{fastperiod}#{slowperiod}#{signalperiod}MO{max_overlap}#{key}_BS 辅助 V230320"

信号逻辑：

1. dik 对应的 MACD 值大于 0，多头；反之，空头；最大允许重叠 max_overlap 个 K 线
2. dik 的 MACD 值大于上一个值，向上；反之，向下

信号列表：

- Signal('15 分钟 _D1MACD12#26#9MO3#MACD_BS 辅助 V230320_ 多头 _ 向上 _ 任意 _0')
- Signal('15 分钟 _D1MACD12#26#9MO3#MACD_BS 辅助 V230320_ 多头 _ 向下 _ 任意 _0')
- Signal('15 分钟 _D1MACD12#26#9MO3#MACD_BS 辅助 V230320_ 空头 _ 向下 _ 任意 _0')
- Signal('15 分钟 _D1MACD12#26#9MO3#MACD_BS 辅助 V230320_ 空头 _ 向上 _ 任意 _0')

Parameters

- **c** –CZSC 对象

- **kwargs** –其他参数 - max_overlap: 最大允许重叠的 K 线数 - di: 倒数第 i 根 K 线 - key: 指定使用哪个 Key 来计算，可选值 [macd, dif, dea]

Returns

信号识别结果

tas_macd_bc_V221201

czsc.signals.**tas_macd_bc_V221201**(*c:* CZSC, ***kwargs*)

MACD 背驰辅助

参数模板："{freq}_D{di}N{n}M{m}#MACD{fastperiod}#{slowperiod}#{signalperiod}_BCV221201"

信号逻辑：

1. 近 n 个最低价创近 m 个周期新低（以收盘价为准），macd 柱子不创新低，这是底部背驰信号
2. 若底背驰信号出现时 macd 为红柱，相当于进一步确认
3. 顶部背驰反之

信号列表：

- Signal('15 分钟 _D1N3M50#MACD12#26#9_BCV221201_ 底部 _ 绿柱 _ 任意 _0')
- Signal('15 分钟 _D1N3M50#MACD12#26#9_BCV221201_ 底部 _ 红柱 _ 任意 _0')
- Signal('15 分钟 _D1N3M50#MACD12#26#9_BCV221201_ 顶部 _ 红柱 _ 任意 _0')
- Signal('15 分钟 _D1N3M50#MACD12#26#9_BCV221201_ 顶部 _ 绿柱 _ 任意 _0')

Parameters

- **c** –CZSC 对象
- **di** –倒数第 i 根 K 线
- **n** –近期窗口大小
- **m** –远期窗口大小

Returns

信号识别结果

tas_macd_bc_V230803

czsc.signals.**tas_macd_bc_V230803**(*c:* CZSC, ***kwargs*) → OrderedDict

MACD 辅助背驰判断

参数模板："{freq}_MACD 双分型背驰 _BS 辅助 V230803"

信号逻辑：

以向上笔为例，当出现两个顶分型时，当后一个顶分型中间 K 线的 MACD 柱子与前一个相比更低，认为是顶背驰。

信号列表：

- Signal(‘60 分钟 _MACD 双分型背驰 _BS 辅助 V230803_ 多头 _ 任意 _ 任意 _0’)
- Signal(‘60 分钟 _MACD 双分型背驰 _BS 辅助 V230803_ 空头 _ 任意 _ 任意 _0’)

Parameters

- **c** –CZSC 对象
- **kwargs** –无

Returns

信号识别结果

tas_macd_bc_V230804

czsc.signals.**tas_macd_bc_V230804**(*c:* CZSC, ***kwargs*) → OrderedDict

MACD 黄白线辅助背驰判断

参数模板:”{freq}_D{di}MACD 背驰 _BS 辅助 V230804”

信号逻辑:

以向上笔为例，当前笔在中枢中轴上方，且 MACD 黄白线不是最高，认为是背驰，做空；反之，做多。

信号列表:

- Signal(‘60 分钟 _D1MACD 背驰 _BS 辅助 V230804_ 空头 _ 任意 _ 任意 _0’)
- Signal(‘60 分钟 _D1MACD 背驰 _BS 辅助 V230804_ 多头 _ 任意 _ 任意 _0’)

Parameters

- **c** –CZSC 对象
- **kwargs** –无

Returns

信号识别结果

tas_macd_bc_V240307

czsc.signals.**tas_macd_bc_V240307**(*c:* CZSC, ***kwargs*) → OrderedDict

MACD 柱子辅助背驰判断

参数模板:”{freq}_D{di}N{n} 柱子背驰 _BS 辅助 V240307”

信号逻辑:

以顶背驰为例，最近 N 根 K 线的 MACD 柱子都大于 0，且最近一个柱子高点小于前面的柱子高点，认为是顶背驰，做空；反之，做多。

信号列表:

- Signal(‘60 分钟 _D1N20 柱子背驰 _BS 辅助 V240307_ 底背驰 _ 第 1 次 _ 任意 _0’)
- Signal(‘60 分钟 _D1N20 柱子背驰 _BS 辅助 V240307_ 底背驰 _ 第 2 次 _ 任意 _0’)

- Signal(‘60 分钟 _D1N20 柱子背驰 _BS 辅助 V240307_ 底背驰 _ 第 3 次 _ 任意 _0’)
- Signal(‘60 分钟 _D1N20 柱子背驰 _BS 辅助 V240307_ 顶背驰 _ 第 1 次 _ 任意 _0’)
- Signal(‘60 分钟 _D1N20 柱子背驰 _BS 辅助 V240307_ 顶背驰 _ 第 2 次 _ 任意 _0’)
- Signal(‘60 分钟 _D1N20 柱子背驰 _BS 辅助 V240307_ 顶背驰 _ 第 3 次 _ 任意 _0’)

Parameters

- **c** –CZSC 对象
- **kwargs** –无

Returns

信号识别结果

tas_macd_bc_ubi_V230804

czsc.signals.**tas_macd_bc_ubi_V230804**(*c:* CZSC, ***kwargs*) → OrderedDict

未完成笔 MACD 黄白线辅助背驰判断

参数模板:”{freq}_MACD 背驰 _UBI 观察 V230804”

信号逻辑:

以向上未完成笔为例，当前笔在中枢中轴上方，且 MACD 黄白线不是最高，认为是背驰，做空；反之，做多。

信号列表:

- Signal(‘60 分钟 _MACD 背驰 _UBI 观察 V230804_ 多头 _ 任意 _ 任意 _0’)
- Signal(‘60 分钟 _MACD 背驰 _UBI 观察 V230804_ 空头 _ 任意 _ 任意 _0’)

Parameters

- **c** –CZSC 对象
- **kwargs** –无

Returns

信号识别结果

tas_macd_bs1_V230312

czsc.signals.**tas_macd_bs1_V230312**(*c:* CZSC, ***kwargs*)

MACD 辅助一买一卖信号

参数模板:”{freq}_D{di}MACD{fastperiod}#{slowperiod}#{signalperiod}_BS1 辅助 V230312”

信号逻辑:

1. 看多，最近一笔新低，且该笔的高点对应一个三卖，MACD 向上；
2. 反之，看空，最近一笔新高，且该笔的低点对应一个三买，MACD 向下。

信号列表:

- Signal(‘15 分钟 _D1MACD12#26#9_BS1 辅助 V230312_ 看多 _ 任意 _ 任意 _0’)
- Signal(‘15 分钟 _D1MACD12#26#9_BS1 辅助 V230312_ 看空 _ 任意 _ 任意 _0’)

Parameters

- **c** –CZSC 对象
- **di** –倒数第 i 笔

Returns

信号识别结果

tas_macd_bs1_V230313

czsc.signals.**tas_macd_bs1_V230313**(*c:* CZSC, ***kwargs*)

MACD 红绿柱判断第一买卖点，贡献者：琅盎

参数模板：”{freq}_D{di}MACD{fastperiod}#{slowperiod}#{signalperiod}_BS1 辅助 V230313”

信号逻辑：

1. 最近一次交叉为死叉，且前面两次死叉都在零轴下方，价格创新低，那么一买即将出现；一卖反之。
2. 或最近一次交叉为金叉，且前面三次死叉都在零轴下方，价格创新低，那么一买即将出现；一卖反之。

信号列表：

- Signal(‘15 分钟 _D1MACD12#26#9_BS1 辅助 V230313_ 一买 _ 死叉 _ 任意 _0’)
- Signal(‘15 分钟 _D1MACD12#26#9_BS1 辅助 V230313_ 一买 _ 金叉 _ 任意 _0’)
- Signal(‘15 分钟 _D1MACD12#26#9_BS1 辅助 V230313_ 一卖 _ 金叉 _ 任意 _0’)
- Signal(‘15 分钟 _D1MACD12#26#9_BS1 辅助 V230313_ 一卖 _ 死叉 _ 任意 _0’)

Parameters

- **c** –CZSC 对象
- **di** –倒数第 i 根 K 线

Returns

信号识别结果

tas_macd_bs1_V230411

czsc.signals.**tas_macd_bs1_V230411**(*c:* CZSC, ***kwargs*) → OrderedDict

基于 MACD DIF 的笔背驰判断信号

参数模板：”{freq}_D{di}T{tha}#{thb}#{thc}_BS1 辅助 V230411”

信号逻辑：

tha, thb, thc 分别为阈值，取值范围为 0 ~ 10000

取 5 笔，从远到近分别记为 1、2、3、4、5，如果满足以下条件，则判断为顶背驰，反之为底背驰：
1. 5 向上，1~3 的累计涨幅超过阈值 tha，且 3 顶部的 dif 值大于 1 顶部 dif 值；
2. 5 的顶部相比于 3 的顶部的涨幅超过阈值-thb，且相应 dif 值的变化率小于阈值-thc；

信号列表：

- Signal(‘15 分钟 _D1T100#10#30_BS1 辅助 V230411_ 底背驰 _ 任意 _ 任意 _0’)
- Signal(‘15 分钟 _D1T100#10#30_BS1 辅助 V230411_ 顶背驰 _ 任意 _ 任意 _0’)

Parameters

- **c** –CZSC 对象
- **kwargs** –参数字典
  - di: 信号计算截止倒数第 i 根 K 线
  - tha: 前三笔的累计涨跌超过阈值 tha，单位：BP，表示万分之一
  - thb: 第 3 笔相比第 1 笔的顶的涨跌幅阈值 thb，单位：BP，表示万分之一
  - thc: 第 5 笔相比第 3 笔的 DIF 值的变化率阈值 thc，单位：BP，表示万分之一

Returns

返回信号结果

tas_macd_bs1_V230412

czsc.signals.**tas_macd_bs1_V230412**(*c:* CZSC, ***kwargs*) → OrderedDict

基于 MACD DIF 的笔背驰判断信号

参数模板：”{freq}_D{di}T{tha}#{thb}_BS1 辅助 V230412”

信号逻辑：

取 5 笔，从远到近分别记为 1、2、3、4、5，如果满足以下条件，则判断为顶背驰，反之为底背驰：
1. 5 向上，1~3 的累计涨幅超过阈值 tha，且 3 顶部的 dif 值大于 1 顶部 dif 值；
2. 5 的顶部相比于 3 的顶部的涨幅超过阈值 thb，且 5 顶部的 dif 值小于 3 顶部 dif 值；

信号列表：

- Signal(‘15 分钟 _D1T100#10_BS1 辅助 V230412_ 底背驰 _ 任意 _ 任意 _0’)
- Signal(‘15 分钟 _D1T100#10_BS1 辅助 V230412_ 顶背驰 _ 任意 _ 任意 _0’)

Parameters

- **c** –CZSC 对象
- **kwargs** –参数字典
  - di: 信号计算截止倒数第 i 根 K 线
  - tha: 前三笔的累计涨跌超过阈值 tha
  - thb: 第三笔相比第二个向上笔的顶的涨幅超过阈值 thb

Returns

tas_macd_change_V221105

czsc.signals.**tas_macd_change_V221105**(*c:* CZSC, ***kwargs*) → OrderedDict

MACD 颜色变化；贡献者：马鸣

参数模板：”{freq}_D{di}K{n}#MACD{fastperiod}#{slowperiod}#{signalperiod} 变色次数 _BS 辅助 V221105”

信号逻辑：

从 dik 往前数 n 根 k 线对应的 macd 红绿柱子变换次数

信号列表：

- Signal(‘15 分钟 _D1K55#MACD12#26#9 变色次数 _BS 辅助 V221105_1 次 _ 任意 _ 任意 _0’)
- Signal(‘15 分钟 _D1K55#MACD12#26#9 变色次数 _BS 辅助 V221105_2 次 _ 任意 _ 任意 _0’)
- Signal(‘15 分钟 _D1K55#MACD12#26#9 变色次数 _BS 辅助 V221105_3 次 _ 任意 _ 任意 _0’)
- Signal(‘15 分钟 _D1K55#MACD12#26#9 变色次数 _BS 辅助 V221105_4 次 _ 任意 _ 任意 _0’)
- Signal(‘15 分钟 _D1K55#MACD12#26#9 变色次数 _BS 辅助 V221105_5 次 _ 任意 _ 任意 _0’)
- Signal(‘15 分钟 _D1K55#MACD12#26#9 变色次数 _BS 辅助 V221105_6 次 _ 任意 _ 任意 _0’)
- Signal(‘15 分钟 _D1K55#MACD12#26#9 变色次数 _BS 辅助 V221105_7 次 _ 任意 _ 任意 _0’)
- Signal(‘15 分钟 _D1K55#MACD12#26#9 变色次数 _BS 辅助 V221105_8 次 _ 任意 _ 任意 _0’)
- Signal(‘15 分钟 _D1K55#MACD12#26#9 变色次数 _BS 辅助 V221105_9 次 _ 任意 _ 任意 _0’)

Parameters

- **c** –czsc 对象
- **di** –倒数第 i 根 K 线
- **n** –从 dik 往前数 n 根 k 线

Returns

tas_macd_direct_V221106

czsc.signals.**tas_macd_direct_V221106**(*c:* CZSC, ***kwargs*) → OrderedDict

MACD 方向；贡献者：马鸣

参数模板：”{freq}_D{di}K#MACD{fastperiod}#{slowperiod}#{signalperiod} 方向 _BS 辅助 V221106”

信号逻辑：连续三根 macd 柱子值依次增大，向上；反之，向下

信号列表：

- Signal(‘15 分钟 _D1K#MACD12#26#9 方向 _BS 辅助 V221106_ 向下 _ 任意 _ 任意 _0’)
- Signal(‘15 分钟 _D1K#MACD12#26#9 方向 _BS 辅助 V221106_ 模糊 _ 任意 _ 任意 _0’)
- Signal(‘15 分钟 _D1K#MACD12#26#9 方向 _BS 辅助 V221106_ 向上 _ 任意 _ 任意 _0’)

参数列表：

Parameters

- **c** –CZSC 对象
- **di** –连续倒 3 根 K 线

Returns

tas_macd_dist_V230408

czsc.signals.**tas_macd_dist_V230408**(*c:* CZSC, ***kwargs*) → OrderedDict

DIF/DEA/MACD 分层信号辅助判断买卖点

参数模板："{freq}_{key} 分层 W{w}N{n}_BS 辅助 V230408"

信号逻辑：

1. 获取最近 w 根 K 线，计算 DIF/DEA/MACD，分成 n 层

信号列表：

- Signal('15 分钟 _DIF 分层 W100N10_BS 辅助 V230408_ 第 7 层 _ 任意 _ 任意 _0')
- Signal('15 分钟 _DIF 分层 W100N10_BS 辅助 V230408_ 第 6 层 _ 任意 _ 任意 _0')
- Signal('15 分钟 _DIF 分层 W100N10_BS 辅助 V230408_ 第 5 层 _ 任意 _ 任意 _0')
- Signal('15 分钟 _DIF 分层 W100N10_BS 辅助 V230408_ 第 4 层 _ 任意 _ 任意 _0')
- Signal('15 分钟 _DIF 分层 W100N10_BS 辅助 V230408_ 第 3 层 _ 任意 _ 任意 _0')
- Signal('15 分钟 _DIF 分层 W100N10_BS 辅助 V230408_ 第 2 层 _ 任意 _ 任意 _0')
- Signal('15 分钟 _DIF 分层 W100N10_BS 辅助 V230408_ 第 1 层 _ 任意 _ 任意 _0')
- Signal('15 分钟 _DIF 分层 W100N10_BS 辅助 V230408_ 第 8 层 _ 任意 _ 任意 _0')
- Signal('15 分钟 _DIF 分层 W100N10_BS 辅助 V230408_ 第 9 层 _ 任意 _ 任意 _0')
- Signal('15 分钟 _DIF 分层 W100N10_BS 辅助 V230408_ 第 10 层 _ 任意 _ 任意 _0')

Parameters

c –CZSC 对象

Returns

信号识别结果

tas_macd_dist_V230409

czsc.signals.**tas_macd_dist_V230409**(*c:* CZSC, ***kwargs*) → OrderedDict

DIF/DEA/MACD 远离零轴辅助判断买卖点

参数模板："{freq}_{key} 远离 W{w}N{n}T{t}_BS 辅助 V230409"

信号逻辑：

1. 获取最近 w 根 K 线，计算 DIF/DEA/MACD，计算绝对值的平均值
2. 如果最新的值大于平均值 * t / 10，则认为远离零轴

信号列表：

- Signal(‘60 分钟 _DIF 远离 W100N10T20_BS 辅助 V230409_ 空头远离 _ 任意 _ 任意 _0’)
- Signal(‘60 分钟 _DIF 远离 W100N10T20_BS 辅助 V230409_ 多头远离 _ 任意 _ 任意 _0’)

Parameters

c –CZSC 对象

Returns

信号识别结果

tas_macd_dist_V230410

czsc.signals.**tas_macd_dist_V230410**(*c:* CZSC, ***kwargs*) → OrderedDict

DIF/DEA/MACD 分层信号辅助判断买卖点

参数模板:”{freq}_{key} 多空分层 W{w}N{n}_BS 辅助 V230410”

信号逻辑:

1. 获取最近 w 根 K 线，计算 DIF/DEA/MACD
2. 最近一根 K 线的值，如果大于 0，则认为多头，小于 0，则认为空头
3. 多头情况下，只对 DIF/DEA/MACD 的值大于 0 的分层进行判断，空头情况下，只对 DIF/DEA/MACD 的值小于 0 的分层进行判断

信号列表:

- Signal(‘60 分钟 _DIF 多空分层 W200N5_BS 辅助 V230410_ 多头 _ 第 1 层 _ 任意 _0’)
- Signal(‘60 分钟 _DIF 多空分层 W200N5_BS 辅助 V230410_ 多头 _ 第 2 层 _ 任意 _0’)
- Signal(‘60 分钟 _DIF 多空分层 W200N5_BS 辅助 V230410_ 空头 _ 第 5 层 _ 任意 _0’)
- Signal(‘60 分钟 _DIF 多空分层 W200N5_BS 辅助 V230410_ 空头 _ 第 4 层 _ 任意 _0’)
- Signal(‘60 分钟 _DIF 多空分层 W200N5_BS 辅助 V230410_ 空头 _ 第 3 层 _ 任意 _0’)
- Signal(‘60 分钟 _DIF 多空分层 W200N5_BS 辅助 V230410_ 空头 _ 第 2 层 _ 任意 _0’)
- Signal(‘60 分钟 _DIF 多空分层 W200N5_BS 辅助 V230410_ 空头 _ 第 1 层 _ 任意 _0’)
- Signal(‘60 分钟 _DIF 多空分层 W200N5_BS 辅助 V230410_ 多头 _ 第 3 层 _ 任意 _0’)
- Signal(‘60 分钟 _DIF 多空分层 W200N5_BS 辅助 V230410_ 多头 _ 第 4 层 _ 任意 _0’)
- Signal(‘60 分钟 _DIF 多空分层 W200N5_BS 辅助 V230410_ 多头 _ 第 5 层 _ 任意 _0’)

Parameters

c –CZSC 对象

Returns

信号识别结果

tas_macd_first_bs_V221201

czsc.signals.**tas_macd_first_bs_V221201**(*c:* CZSC, ***kwargs*)

MACD 金叉死叉判断第一买卖点

参数模板："{freq}_D{di}MACD{fastperiod}#{slowperiod}#{signalperiod}_BS1 辅助 V221201"

信号逻辑：

1. 最近一次交叉为死叉，且前面两次死叉都在零轴下方，那么一买即将出现；一卖反之。

信号列表：

- Signal('15 分钟 _D1MACD12#26#9_BS1 辅助 V221201_ 一买 _ 任意 _ 任意 _0')
- Signal('15 分钟 _D1MACD12#26#9_BS1 辅助 V221201_ 一卖 _ 任意 _ 任意 _0')

Parameters

- **c** –CZSC 对象
- **di** –倒数第 i 根 K 线

Returns

信号识别结果

tas_macd_first_bs_V221216

czsc.signals.**tas_macd_first_bs_V221216**(*c:* CZSC, ***kwargs*)

MACD 金叉死叉判断第一买卖点

参数模板："{freq}_D{di}MACD{fastperiod}#{slowperiod}#{signalperiod}_BS1 辅助 V221216"

信号逻辑：

1. 最近一次交叉为死叉，且前面两次死叉都在零轴下方，价格创新低，那么一买即将出现；一卖反之。
2. 或最近一次交叉为金叉，且前面三次死叉都在零轴下方，价格创新低，那么一买即将出现；一卖反之。

信号列表：

- Signal('15 分钟 _D1MACD12#26#9_BS1 辅助 V221216_ 一买 _ 死叉 _ 任意 _0')
- Signal('15 分钟 _D1MACD12#26#9_BS1 辅助 V221216_ 一买 _ 金叉 _ 任意 _0')
- Signal('15 分钟 _D1MACD12#26#9_BS1 辅助 V221216_ 一卖 _ 金叉 _ 任意 _0')
- Signal('15 分钟 _D1MACD12#26#9_BS1 辅助 V221216_ 一卖 _ 死叉 _ 任意 _0')

Parameters

- **c** –CZSC 对象
- **di** –倒数第 i 根 K 线

Returns

信号识别结果

tas_macd_power_V221108

czsc.signals.**tas_macd_power_V221108**(*c:* CZSC, ***kwargs*) → OrderedDict

MACD 强弱

参数模板："{freq}_D{di}K#MACD{fastperiod}#{slowperiod}#{signalperiod} 强弱 _BS 辅助 V221108"

信号逻辑：

1. 指标超强满足条件：DIF > DEA > 0；释义：指标超强表示市场价格处于中长期多头趋势中，可能形成凌厉的逼空行情
2. 指标强势满足条件：DIF-DEA > 0（MACD 柱线 > 0）释义：指标强势表示市场价格处于中短期多头趋势中，价格涨多跌少，通常是反弹行情
3. 指标弱势满足条件：DIF-DEA < 0（MACD 柱线 < 0）释义：指标弱势表示市场价格处于中短期空头趋势中，价格跌多涨少，通常是回调行情
4. 指标超弱满足条件：DIF < DEA < 0 释义：指标超弱表示市场价格处于中长期空头趋势中，可能形成杀多行情

信号列表：

- Signal('15 分钟 _D1K#MACD12#26#9 强弱 _BS 辅助 V221108_ 弱势 _ 任意 _ 任意 _0')
- Signal('15 分钟 _D1K#MACD12#26#9 强弱 _BS 辅助 V221108_ 超弱 _ 任意 _ 任意 _0')
- Signal('15 分钟 _D1K#MACD12#26#9 强弱 _BS 辅助 V221108_ 强势 _ 任意 _ 任意 _0')
- Signal('15 分钟 _D1K#MACD12#26#9 强弱 _BS 辅助 V221108_ 超强 _ 任意 _ 任意 _0')

Parameters

- **c** –CZSC 对象
- **di** –信号产生在倒数第 di 根 K 线

Returns

信号识别结果

tas_macd_second_bs_V221201

czsc.signals.**tas_macd_second_bs_V221201**(*c:* CZSC, ***kwargs*)

MACD 金叉死叉判断第二买卖点

参数模板："{freq}_D{di}MACD{fastperiod}#{slowperiod}#{signalperiod}_BS2 辅助 V221201"

信号逻辑：

1. 最近一次交叉为死叉，DEA 大于 0，且前面两次死叉都在零轴下方，那么二买即将出现；二卖反之。
2. 或最近一次交叉为金叉，且前面三次死叉中前两次都在零轴下方，后一次在零轴上方，那么二买即将出现；二卖反之。

信号列表：

- Signal('15 分钟 _D1MACD12#26#9_BS2 辅助 V221201_ 二买 _ 死叉 _ 任意 _0')
- Signal('15 分钟 _D1MACD12#26#9_BS2 辅助 V221201_ 二买 _ 金叉 _ 任意 _0')

- Signal(‘15 分钟 _D1MACD12#26#9_BS2 辅助 V221201_ 二卖 _ 死叉 _ 任意 _0’)
- Signal(‘15 分钟 _D1MACD12#26#9_BS2 辅助 V221201_ 二卖 _ 金叉 _ 任意 _0’)

Parameters

- **c** –CZSC 对象
- **di** –倒数第 i 根 K 线

Returns

信号识别结果

tas_macd_xt_V221208

czsc.signals.**tas_macd_xt_V221208**(*c:* CZSC, ***kwargs*)

MACD 形态信号

参数模板:”{freq}_D{di}K#MACD{fastperiod}#{slowperiod}#{signalperiod} 形态 _BS 辅助 V221208”

信号逻辑:

1. MACD 柱子的形态分类，具体见代码定义

信号列表:

- Signal(‘15 分钟 _D1K#MACD12#26#9 形态 _BS 辅助 V221208_ 绿抽脚 _ 任意 _ 任意 _0’)
- Signal(‘15 分钟 _D1K#MACD12#26#9 形态 _BS 辅助 V221208_ 杀多棒 _ 任意 _ 任意 _0’)
- Signal(‘15 分钟 _D1K#MACD12#26#9 形态 _BS 辅助 V221208_ 空翻多 _ 任意 _ 任意 _0’)
- Signal(‘15 分钟 _D1K#MACD12#26#9 形态 _BS 辅助 V221208_ 红缩头 _ 任意 _ 任意 _0’)
- Signal(‘15 分钟 _D1K#MACD12#26#9 形态 _BS 辅助 V221208_ 逼空棒 _ 任意 _ 任意 _0’)
- Signal(‘15 分钟 _D1K#MACD12#26#9 形态 _BS 辅助 V221208_ 多翻空 _ 任意 _ 任意 _0’)

Parameters

- **c** –CZSC 对象
- **di** –倒数第 i 根 K 线

Returns

tas_rsi_base_V230227

czsc.signals.**tas_rsi_base_V230227**(*c:* CZSC, ***kwargs*) → OrderedDict

RSI 超买超卖信号

参数模板:”{freq}_D{di}T{th}RSI{timeperiod}_RSI 辅助 V230227”

信号逻辑:

在正常情况下，RSI 指标都会在 30-70 的区间内波动。当 6 日 RSI 超过 80 时，表示市场已经处于超买区间。6 日 RSI 达到 90 以上时，表示市场已经严重超买，股价极有可能已经达到阶段顶点。这时投资者

应该果断卖出。当 6 日 RSI 下降到 20 时，表示市场已经处于超卖区间。6 日 RSI 一旦下降到 10 以下，则表示市场已经严重超卖，股价极有可能会止跌回升，是很好的买入信号。

信号列表：

- Signal（‘15 分钟 _D1T20RSI6_RSI 辅助 V230227_ 超卖 _ 向下 _ 任意 _0’）
- Signal（‘15 分钟 _D1T20RSI6_RSI 辅助 V230227_ 超卖 _ 向上 _ 任意 _0’）
- Signal（‘15 分钟 _D1T20RSI6_RSI 辅助 V230227_ 超买 _ 向上 _ 任意 _0’）
- Signal（‘15 分钟 _D1T20RSI6_RSI 辅助 V230227_ 超买 _ 向下 _ 任意 _0’）

Parameters

- **c** –CZSC 对象
- **di** –倒数第几根 K 线
- **n** –RSI 的计算周期
- **th** –RSI 阈值

Returns

信号识别结果

tas_rumi_V230704

czsc.signals.**tas_rumi_V230704**(*c:* CZSC, ***kwargs*) → OrderedDict

对均线偏离度平滑处理, 通过平滑处理的方式降低 DIFF 的敏感度来解决均线缠绕的问题贡献者：谌意勇

参数模板：”{freq}_D{di}F{timeperiod1}S{timeperiod2}R{rumi_window}_BS 辅助 V230704”

信号逻辑：

RUMI 计算参考：1. https://zhuanlan.zhihu.com/p/610377004 2. https://zhuanlan.zhihu.com/p/618394552

多空规则：1. RUMI 上穿 0 轴，买入做多 2. RUMI 下穿 0 轴，卖出做空。

信号列表：

- Signal（‘60 分钟 _D1F3S50R30_BS 辅助 V230704_ 空头 _ 任意 _ 任意 _0’）
- Signal（‘60 分钟 _D1F3S50R30_BS 辅助 V230704_ 多头 _ 任意 _ 任意 _0’）

Parameters

- **c** –CZSC 对象
- **kwargs** –
  - di: 信号计算截止倒数第 i 根 K 线
  - timeperiod1: 均线 1 的周期
  - timeperiod2: 均线 2 的周期
  - rumi_window: rumi 的周期

Returns

信号识别结果

tas_sar_base_V230425

czsc.signals.**tas_sar_base_V230425**(*c:* CZSC, ***kwargs*)

SAR 基础信号

参数模板："{freq}_D{di}MO{max_overlap}SAR_BS 辅助 V230425"

信号逻辑：

1. 收盘价升破 SAR，且前面 MO 根 K 中有任意一根 K 线的收盘价都低于 SAR，看多信号
2. 收盘价跌破 SAR，且前面 MO 根 K 中有任意一根 K 线的收盘价都高于 SAR，看空信号

信号列表：

- Signal('日线 _D1MO5SAR_BS 辅助 V230425_ 看空 _ 任意 _ 任意 _0')
- Signal('日线 _D1MO5SAR_BS 辅助 V230425_ 看多 _ 任意 _ 任意 _0')

Parameters

- **c** –基础周期的 CZSC 对象
- **kwargs** –其他参数 - di: 倒数第 di 根 K 线 - max_overlap: 信号最大重叠 K 线数

Returns

信号字典

tas_second_bs_V230228

czsc.signals.**tas_second_bs_V230228**(*c:* CZSC, ***kwargs*) → OrderedDict

均线结合 K 线形态的第二买卖点辅助判断

参数模板："{freq}_D{di}N{n}#{ma_type}#{timeperiod}_BS2 辅助 V230228"

信号逻辑：

1. 二买辅助：1）MA20 创新高且向上；2）近三根 K 线最低价跌破 MA20，且当前收盘价在 MA20 上
2. 反之，二卖辅助。

信号列表：

- Signal('日线 _D2N21#SMA#20_BS2 辅助 V230228_ 二卖 _ 任意 _ 任意 _0')
- Signal('日线 _D2N21#SMA#20_BS2 辅助 V230228_ 二买 _ 任意 _ 任意 _0')

Parameters

- **c** –CZSC 对象
- **di** –倒数第几根 K 线，1 表示最后一根 K 线
- **n** –窗口大小
- **kwargs** –

Returns

信号识别结果

tas_second_bs_V230303

czsc.signals.**tas_second_bs_V230303**(*c:* CZSC, ***kwargs*)

利用笔和均线辅助二买信号生成

参数模板："{freq}_D{di}{ma_type}#{timeperiod}_BS2 辅助 V230303"

信号逻辑：

1. 最近 5 笔创新低，且最近一向下笔最低点跌破中期均线，且中期均线向上，二买信号；
2. 反之，二卖信号。

信号列表

- Signal('15 分钟 _D1SMA#30_BS2 辅助 V230303_ 二卖 _ 任意 _ 任意 _0')
- Signal('15 分钟 _D1SMA#30_BS2 辅助 V230303_ 二买 _ 任意 _ 任意 _0')

Parameters

- **c** –CZSC 对象
- **di** –指定倒数第几笔
- **ma_type** –均线类型，必须是 *ma_type_map* 中的 key
- **timeperiod** –均线计算周期

Returns

信号识别结果

tas_slope_V231019

czsc.signals.**tas_slope_V231019**(*c:* CZSC, ***kwargs*) → OrderedDict

DIF 趋势线斜率判断多空

参数模板："{freq}_D{di}DIF{n} 斜率 T{th}_BS 辅助 V231019"

信号逻辑：

取最近 N 根 K 线的 DIF 值计算斜率，然后取 N * 10 根 K 线的斜率值，计算斜率值的分位数，如果分位数大于 th，则看多，小于 1-th，则看空。

信号列表：

- Signal('60 分钟 _D1DIF10 斜率 T80_BS 辅助 V231019_ 看多 _ 任意 _ 任意 _0')
- Signal('60 分钟 _D1DIF10 斜率 T80_BS 辅助 V231019_ 看空 _ 任意 _ 任意 _0')

Parameters

- **cat** –CzscSignals 对象

- **kwargs –参数字典:return: 返回信号结果

update_atr_cache

czsc.signals.**update_atr_cache**(*c:* CZSC, ***kwargs*)

更新 ATR 缓存

平均真实波幅（ATR）的计算方法：

1、当前交易日的最高价与最低价间的波幅 2、前一交易日收盘价与当个交易日最高价间的波幅 3、前一交易日收盘价与当个交易日最低价间的波幅

今日振幅、今日最高与昨收差价，今日最低与昨收差价中的最大值，为真实波幅，在有了真实波幅后，就可以利用一段时间的平均值计算 ATR 了。

Parameters

c –CZSC 对象

Returns

update_boll_cache

czsc.signals.**update_boll_cache**(*c:* CZSC, ***kwargs*)

更新 K 线的 BOLL 缓存

Parameters

c –交易对象

Returns

update_cci_cache

czsc.signals.**update_cci_cache**(*c:* CZSC, ***kwargs*)

更新 CCI 缓存

CCI = (TP - MA) / MD / 0.015; 其中，

- TP=(最高价 + 最低价 + 收盘价)÷3;
- MA= 最近 N 日收盘价的累计之和 ÷N;
- MD= 最近 N 日（MA - 收盘价）的累计之和 ÷N;
- 0.015 为计算系数，N 为计算周期

Parameters

c –CZSC 对象

Returns

update_kdj_cache

czsc.signals.**update_kdj_cache**(*c:* CZSC, ***kwargs*)

更新 KDJ 缓存

Parameters

c –CZSC 对象

Returns

update_ma_cache

czsc.signals.**update_ma_cache**(*c:* CZSC, ***kwargs*)

更新均线缓存

Parameters

- **c** –CZSC 对象
- **kwargs** –
  - ma_type: 均线类型，可选值：SMA, EMA, WMA, KAMA, TEMA, DEMA, MAMA, TRIMA
  - timeperiod: 计算周期

Returns

cache_key

update_macd_cache

czsc.signals.**update_macd_cache**(*c:* CZSC, ***kwargs*)

更新 MACD 缓存

Parameters

c –CZSC 对象

Returns

update_rsi_cache

czsc.signals.**update_rsi_cache**(*c:* CZSC, ***kwargs*)

更新 RSI 缓存

相对强弱指数（RSI）是通过比较一段时期内的平均收盘涨数和平均收盘跌数来分析市场买沽盘的意向和实力，从而判断未来市场的走势。RSI 在 1978 年 6 月由 WellsWider 创制。

RSI = 100 × RS / (1 + RS) 或者 RSI=100 - 100÷(1+RS) RS = X 天的平均上涨点数 / X 天的平均下跌点数

Parameters

c –CZSC 对象

Returns

update_sar_cache

czsc.signals.**update_sar_cache**(*c:* CZSC, ***kwargs*)

更新 SAR 缓存

SAR 是止损转向操作点指标的简称，英文名称为“Stop and Reverse”，缩写为 SAR，一般称为抛物线指标。该指标是由美国技术分析大师威尔斯·威尔德所创造出来的。

详细介绍：

- https://zhuanlan.zhihu.com/p/210169446
- https://www.investopedia.com/terms/p/parabolicindicator.asp

Parameters

c –CZSC 对象

Returns

vol_double_ma_V230214

czsc.signals.**vol_double_ma_V230214**(*c:* CZSC, ***kwargs*) → OrderedDict

成交量双均线信号

参数模板：”{freq}_D{di}VOL 双均线 {ma_type}#{t1}#{t2}_BS 辅助 V230214”

信号逻辑：

1. 短均线在长均线上方，看多；反之，看空

信号列表：

- Signal(‘15 分钟 _D1VOL 双均线 SMA#5#20_BS 辅助 V230214_ 看空 _ 任意 _ 任意 _0’)
- Signal(‘15 分钟 _D1VOL 双均线 SMA#5#20_BS 辅助 V230214_ 看多 _ 任意 _ 任意 _0’)

信号说明：

Parameters

- **c** –CZSC 对象
- **kwargs** –t1: 短线均线，t2: 长线均线, ma_type: 均线类型, di: 信号计算截止倒数第 i 根 K 线

Returns

信号识别结果

vol_gao_di_V221218

czsc.signals.**vol_gao_di_V221218**(*c:* CZSC, ***kwargs*) → OrderedDict

高量柱 & 低量柱 & 高量黄金柱，贡献者：琅盎

参数模板："{freq}_D{di}K_ 量柱 V221218"

**高/低量柱判断标准 **

1. 高量柱是在一个阶段内量柱的对比（3 天以上），在这一阶段出现的的最高量就是高量柱。2. 高量柱是在对应的这一价位成交火爆的标志，出现高量柱后的走势多数向上少数向下。3. 高量柱出现后，不跌或横盘应看涨。4. 高量柱 + 缩量柱＝黄金柱是最具价值的高量柱组合 5. 低量柱判断标准正好和高量柱相反 6. 低量柱出现后，不跌或横盘应看跌。7. 操作中还需要根据所处位置作判断

信号列表：

- Signal（'15 分钟 _D1K_ 量柱 V221218_ 低量柱 _7K_ 任意 _0'）
- Signal（'15 分钟 _D1K_ 量柱 V221218_ 低量柱 _10K_ 任意 _0'）
- Signal（'15 分钟 _D1K_ 量柱 V221218_ 高量柱 _10K_ 任意 _0'）
- Signal（'15 分钟 _D1K_ 量柱 V221218_ 高量黄金柱 _10K_ 任意 _0'）
- Signal（'15 分钟 _D1K_ 量柱 V221218_ 高量柱 _6K_ 任意 _0'）
- Signal（'15 分钟 _D1K_ 量柱 V221218_ 低量柱 _9K_ 任意 _0'）
- Signal（'15 分钟 _D1K_ 量柱 V221218_ 高量黄金柱 _7K_ 任意 _0'）
- Signal（'15 分钟 _D1K_ 量柱 V221218_ 高量柱 _7K_ 任意 _0'）
- Signal（'15 分钟 _D1K_ 量柱 V221218_ 低量柱 _8K_ 任意 _0'）
- Signal（'15 分钟 _D1K_ 量柱 V221218_ 高量柱 _8K_ 任意 _0'）
- Signal（'15 分钟 _D1K_ 量柱 V221218_ 低量柱 _6K_ 任意 _0'）
- Signal（'15 分钟 _D1K_ 量柱 V221218_ 高量柱 _9K_ 任意 _0'）
- Signal（'15 分钟 _D1K_ 量柱 V221218_ 高量黄金柱 _8K_ 任意 _0'）
- Signal（'15 分钟 _D1K_ 量柱 V221218_ 高量黄金柱 _9K_ 任意 _0'）
- Signal（'15 分钟 _D1K_ 量柱 V221218_ 高量黄金柱 _6K_ 任意 _0'）

Parameters

- **c** –CZSC 对象
- **kwargs** –di: 倒数第 di 根 K 线，加上这个参数就可以不用借助缓存就可以回溯

Returns

高量柱识别结果

vol_single_ma_V230214

czsc.signals.**vol_single_ma_V230214**(*c:* CZSC, ***kwargs*) → OrderedDict

均线辅助识别第三类买卖点，增加均线形态

参数模板："{freq}_D{di}VOL#{ma_type}#{timeperiod}_ 分类 V230214"

信号逻辑：

1. vol > ma，多头；反之，空头
2. ma[-1] > ma[-2]，向上；反之，向下

信号列表：

- Signal('15 分钟 _D1VOL#SMA#5_ 分类 V230214_ 空头 _ 向上 _ 任意 _0')
- Signal('15 分钟 _D1VOL#SMA#5_ 分类 V230214_ 空头 _ 向下 _ 任意 _0')
- Signal('15 分钟 _D1VOL#SMA#5_ 分类 V230214_ 多头 _ 向上 _ 任意 _0')
- Signal('15 分钟 _D1VOL#SMA#5_ 分类 V230214_ 多头 _ 向下 _ 任意 _0')

信号说明：

Parameters

- **c** –CZSC 对象
- **kwargs** –
  - ma_type: 均线类型，必须是 *ma_type_map* 中的 key
  - timeperiod: 均线计算周期
  - di: 信号计算截止倒数第 i 根 K 线

Returns

信号识别结果

vol_ti_suo_V221216

czsc.signals.**vol_ti_suo_V221216**(*c:* CZSC, ***kwargs*) → OrderedDict

梯量/缩量柱：顺势与逆势工具，贡献者：琅盎

参数模板："{freq}_D{di}K_ 量柱 V221216"

信号逻辑：

1. 只要比昨缩量的就能形成“缩量柱”，只要比昨日增高的就能形成“梯量柱”。严格地讲，连续三天缩量就是“缩量柱”，连续三天增量“阶梯状”就是梯量柱，它是当日量能连续同前二天的量相比。2. 缩量柱的形态是量柱明显走低，梯量柱的形态是成交量明显逐步走高，它们都有两种情况：量价同步和量价背离。3. “价升量缩”的“缩量柱”，体现了供不应求的局面，主力有主动买入的倾向；4. “量增价涨”的“梯量柱”，体现了努力上攻的态势，主力有被动买入的倾向。

信号列表：

- Signal('15 分钟 _D1K_ 量柱 V221216_ 梯量 _ 价平 _ 任意 _0')
- Signal('15 分钟 _D1K_ 量柱 V221216_ 梯量 _ 价跌 _ 任意 _0')

- Signal(‘15 分钟 _D1K_ 量柱 V221216_ 缩量 _ 价平 _ 任意 _0’)
- Signal(‘15 分钟 _D1K_ 量柱 V221216_ 缩量 _ 价跌 _ 任意 _0’)
- Signal(‘15 分钟 _D1K_ 量柱 V221216_ 梯量 _ 价升 _ 任意 _0’)
- Signal(‘15 分钟 _D1K_ 量柱 V221216_ 缩量 _ 价升 _ 任意 _0’)

信号说明：

Parameters

- **c** –CZSC 对象
- **kwargs** –di: 倒数第 di 根 K 线，加上这个参数就可以不用借助缓存就可以回溯

Returns

信号识别结果

vol_window_V230731

czsc.signals.**vol_window_V230731**(*c:* CZSC, ***kwargs*) → OrderedDict

指定窗口内成交量的特征

参数模板：”{freq}_D{di}W{w}M{m}N{n}_ 窗口能量 V230731”

信号逻辑：

取最近 m 根 K 线，计算每根 K 线的成交量，分成 n 层，最大值为 n，最小值为 1；最近 w 根 K 线的成交量分层最大值为 max_vol_layer，最小值为 min_vol_layer，以这两个值作为窗口内的成交量特征。

信号列表：

- Signal(‘60 分钟 _D2W5M100N10_ 窗口能量 V230731_ 高量 N9_ 低量 N4_ 任意 _0’)
- Signal(‘60 分钟 _D2W5M100N10_ 窗口能量 V230731_ 高量 N9_ 低量 N5_ 任意 _0’)
- Signal(‘60 分钟 _D2W5M100N10_ 窗口能量 V230731_ 高量 N9_ 低量 N2_ 任意 _0’)
- Signal(‘60 分钟 _D2W5M100N10_ 窗口能量 V230731_ 高量 N9_ 低量 N3_ 任意 _0’)
- Signal(‘60 分钟 _D2W5M100N10_ 窗口能量 V230731_ 高量 N10_ 低量 N4_ 任意 _0’)
- Signal(‘60 分钟 _D2W5M100N10_ 窗口能量 V230731_ 高量 N8_ 低量 N3_ 任意 _0’)
- Signal(‘60 分钟 _D2W5M100N10_ 窗口能量 V230731_ 高量 N10_ 低量 N3_ 任意 _0’)
- Signal(‘60 分钟 _D2W5M100N10_ 窗口能量 V230731_ 高量 N10_ 低量 N6_ 任意 _0’)
- Signal(‘60 分钟 _D2W5M100N10_ 窗口能量 V230731_ 高量 N10_ 低量 N7_ 任意 _0’)
- Signal(‘60 分钟 _D2W5M100N10_ 窗口能量 V230731_ 高量 N10_ 低量 N5_ 任意 _0’)
- Signal(‘60 分钟 _D2W5M100N10_ 窗口能量 V230731_ 高量 N9_ 低量 N6_ 任意 _0’)
- Signal(‘60 分钟 _D2W5M100N10_ 窗口能量 V230731_ 高量 N8_ 低量 N2_ 任意 _0’)

Parameters

- **c** –CZSC 对象
- **kwargs** –参数字典
  - **param di**
    信号计算截止倒数第 i 根 K 线

- **param w**
  观察的窗口大小。
- **param n**
  分层的数量。
- **param m**
  计算分位数所需取 K 线的数量。

Returns
信号识别结果

vol_window_V230801

czsc.signals.**vol_window_V230801**(*c:* CZSC, ***kwargs*) → OrderedDict

指定窗口内成交量的特征

参数模板："{freq}_D{di}W{w}_ 窗口能量 V230801"

信号逻辑：

观察一个固定窗口内的成交量特征，本信号以窗口内的最大成交量与最小成交量的先后顺序作为窗口成交量的特征。

信号列表：

- Signal('60 分钟 _D1W5_ 窗口能量 V230801_ 先缩后放 _ 任意 _ 任意 _0')
- Signal('60 分钟 _D1W5_ 窗口能量 V230801_ 先放后缩 _ 任意 _ 任意 _0')

Parameters

- **c** –CZSC 对象
- **kwargs** –参数字典
  - **param di**
    信号计算截止倒数第 i 根 K 线
  - **param w**
    观察的窗口大小。

Returns
信号识别结果

xl_bar_basis_V240411

czsc.signals.**xl_bar_basis_V240411**(*c:* CZSC, ***kwargs*) → OrderedDict

看涨吞没和看跌吞没形态

参数模板："{freq}_N{n}_ 形态 V240411"

信号逻辑：

1. 看涨吞没，第二根阳线实体覆盖第一根 K 线实体和上下影线
2. 看跌吞没，相反

信号列表：

- Signal('30 分钟 _N5_ 形态 V240411_ 看涨吞没 _ 任意 _ 任意 _0')
- Signal('30 分钟 _N5_ 形态 V240411_ 看跌吞没 _ 任意 _ 任意 _0')

Parameters

- **c** –CZSC 对象
- **kwargs** –

Returns

信号识别结果

xl_bar_basis_V240412

czsc.signals.**xl_bar_basis_V240412**(*c:* CZSC, ***kwargs*) → OrderedDict

长蜡烛形态

参数模板："{freq}_N{n}#TH{th}_ 形态 V240412"

信号逻辑：

1. 看涨长蜡烛形态，实体大于 (前 N 日 K 线实体长度之和) / N + 系数 * 标准差
2. 看跌长蜡烛形态

信号列表： - Signal('30 分钟 _N10#TH3_ 形态 V240412_ 看涨长蜡烛 _ 任意 _ 任意 _0') - Signal('30 分钟 _N10#TH3_ 形态 V240412_ 看跌长蜡烛 _ 任意 _ 任意 _0')

Parameters

- **c** –CZSC 对象
- **kwargs** –

Returns

信号识别结果

xl_bar_position_V240328

czsc.signals.**xl_bar_position_V240328**(*c:* CZSC, ***kwargs*) → OrderedDict

相对位置信号; 贡献者：谢磊

参数模板："{freq}_N{n}_BS 辅助 V240328"

信号逻辑：

1. 用当前价格与 EMA 的比值做一个偏离度指标
2. 当偏离度越高就越有可能是相对低点的位置

信号列表： - Signal('30 分钟 _N10_BS 辅助 V240328_ 相对低点 _ 任意 _ 任意 _0') - Signal('30 分钟 _N10_BS 辅助 V240328_ 相对高点 _ 任意 _ 任意 _0')

Parameters

- **c** –CZSC 对象
- **kwargs** –

Returns

信号识别结果

xl_bar_trend_V240329

czsc.signals.**xl_bar_trend_V240329**(*c:* CZSC, ***kwargs*) → OrderedDict

底部反转形态信号; 贡献者：谢磊

正向的十字孕线（Bullish Harami Cross）是一种看涨的蜡烛图形态，属于孕线形态的一种变体。这种形态出现在下跌趋势的末端，可能预示着趋势即将反转向上。正向的十字孕线由两根蜡烛图组成：第一根是一个长实体的阴线，显示了强劲的下跌趋势；第二根是一个十字线（或接近十字线的形态），其开盘价和收盘价都处于第一根阴线实体的中部以下，但实体部分较小，且颜色可以是阳线或阴线。

参数模板："{freq}_N{n}M{m}_ 十字线反转 V240329"

信号逻辑：

1，十字线定义，(h -l) / (c - o) 的绝对值大于 th，或 c == o

信号列表： - Signal('30 分钟 _N5M5_ 十字线反转 V240329_ 底部十字孕线 _ 任意 _ 任意 _0') - Signal('30 分钟 _N5M5_ 十字线反转 V240329_ 顶部十字孕线 _ 任意 _ 任意 _0')

Parameters

- **c** –CZSC 对象
- **kwargs** –

Returns

信号识别结果

xl_bar_trend_V240330

czsc.signals.**xl_bar_trend_V240330**(*c:* CZSC, ***kwargs*) → OrderedDict

完全分类，均线金叉过滤信号; 贡献者：谢磊

参数模板："{freq}_N{n}M{m}#{ma_type}_ 双均线过滤 V240330"

信号逻辑：

1，当 25 日均线大于 350 日均线，看多 2. 当 25 日均线小于 350 日均线，看空 3. 均线类型可选，平均线，EMA 线。

信号列表：

- Signal('15 分钟 _N5M21#SMA_ 双均线过滤 V240330_ 看空 _ 第 03 次 _ 任意 _0')
- Signal('15 分钟 _N5M21#SMA_ 双均线过滤 V240330_ 看空 _ 第 04 次 _ 任意 _0')
- Signal('15 分钟 _N5M21#SMA_ 双均线过滤 V240330_ 看空 _ 第 05 次 _ 任意 _0')
- Signal('15 分钟 _N5M21#SMA_ 双均线过滤 V240330_ 看空 _ 第 06 次 _ 任意 _0')
- Signal('15 分钟 _N5M21#SMA_ 双均线过滤 V240330_ 看空 _ 第 07 次 _ 任意 _0')
- Signal('15 分钟 _N5M21#SMA_ 双均线过滤 V240330_ 看空 _ 第 08 次 _ 任意 _0')
- Signal('15 分钟 _N5M21#SMA_ 双均线过滤 V240330_ 看空 _ 第 09 次 _ 任意 _0')
- Signal('15 分钟 _N5M21#SMA_ 双均线过滤 V240330_ 看空 _ 第 10 次 _ 任意 _0')
- Signal('15 分钟 _N5M21#SMA_ 双均线过滤 V240330_ 看多 _ 第 01 次 _ 任意 _0')
- Signal('15 分钟 _N5M21#SMA_ 双均线过滤 V240330_ 看多 _ 第 02 次 _ 任意 _0')
- Signal('15 分钟 _N5M21#SMA_ 双均线过滤 V240330_ 看多 _ 第 03 次 _ 任意 _0')
- Signal('15 分钟 _N5M21#SMA_ 双均线过滤 V240330_ 看多 _ 第 04 次 _ 任意 _0')
- Signal('15 分钟 _N5M21#SMA_ 双均线过滤 V240330_ 看空 _ 第 01 次 _ 任意 _0')
- Signal('15 分钟 _N5M21#SMA_ 双均线过滤 V240330_ 看空 _ 第 02 次 _ 任意 _0')
- Signal('15 分钟 _N5M21#SMA_ 双均线过滤 V240330_ 看多 _ 第 05 次 _ 任意 _0')
- Signal('15 分钟 _N5M21#SMA_ 双均线过滤 V240330_ 看多 _ 第 06 次 _ 任意 _0')
- Signal('15 分钟 _N5M21#SMA_ 双均线过滤 V240330_ 看多 _ 第 07 次 _ 任意 _0')
- Signal('15 分钟 _N5M21#SMA_ 双均线过滤 V240330_ 看多 _ 第 08 次 _ 任意 _0')
- Signal('15 分钟 _N5M21#SMA_ 双均线过滤 V240330_ 看多 _ 第 09 次 _ 任意 _0')
- Signal('15 分钟 _N5M21#SMA_ 双均线过滤 V240330_ 看多 _ 第 10 次 _ 任意 _0')

Parameters

- **c** –CZSC 对象
- **kwargs** –

Returns

信号识别结果

xl_bar_trend_V240331

czsc.signals.**xl_bar_trend_V240331**(*c:* CZSC, ***kwargs*) → OrderedDict

突破信号; 贡献者：谢磊

参数模板："{freq}_N{n}_ 突破信号 V240331"

信号逻辑：

1，突破前 N 日最高价，入场，做多 2. 跌破前 N 日最低价，入场，做空

信号列表：

- Signal('30 分钟 _N20_ 突破信号 V240331_ 做多 _ 任意 _ 任意 _0')
- Signal('30 分钟 _N20_ 突破信号 V240331_ 做空 _ 任意 _ 任意 _0')

Parameters

- **c** –CZSC 对象
- **kwargs** –

Returns

信号识别结果

zdy_bi_end_V230406

czsc.signals.**zdy_bi_end_V230406**(*c:* CZSC, ***kwargs*) → OrderedDict

分型停顿判断 K 线结束

参数模板："{freq}_D0 停顿分型 _BE 辅助 V230406"

信号逻辑：

1. 当分型形成后，等待后续出现某一根 K 线的收盘价站住第三根 K 线（此处的第三根 K 线是指形成分型的第三根 K 线）的极值，即可认为满足了停顿法的要求。2. 如果是两次停顿确认笔结束，要求内部至少还有一个相应的分型停顿。

信号列表：

- Signal('15 分钟 _D0 停顿分型 _BE 辅助 V230406_ 看空 _ 任意 _ 任意 _0')
- Signal('15 分钟 _D0 停顿分型 _BE 辅助 V230406_ 看多 _ 任意 _ 任意 _0')
- Signal('15 分钟 _D0 停顿分型 _BE 辅助 V230406_ 看多 _ 内部底停顿 _ 任意 _0')
- Signal('15 分钟 _D0 停顿分型 _BE 辅助 V230406_ 看空 _ 内部顶停顿 _ 任意 _0')
- Signal('15 分钟 _D0 停顿分型 _BE 辅助 V230406_ 看空 _ 任意 _ 顶分区间 _0')
- Signal('15 分钟 _D0 停顿分型 _BE 辅助 V230406_ 看多 _ 任意 _ 底分区间 _0')
- Signal('15 分钟 _D0 停顿分型 _BE 辅助 V230406_ 看空 _ 内部顶停顿 _ 顶分区间 _0')
- Signal('15 分钟 _D0 停顿分型 _BE 辅助 V230406_ 看多 _ 内部底停顿 _ 底分区间 _0')

相关信号：

- *czsc.signals.byi_bi_end_V230106()*

Parameters

c –CZSC 对象

Returns

zdy_bi_end_V230407

czsc.signals.**zdy_bi_end_V230407**(*c:* CZSC, ***kwargs*) → OrderedDict

分型停顿判断 K 线结束

参数模板："{freq}_D0 停顿分型 _BE 辅助 V230407"

信号逻辑：

1. 当分型形成后，等待后续出现某一根 K 线的收盘价站住第三根 K 线（此处的第三根 K 线是指形成分型的第三根 K 线）的极值，且后续的所有 K 线收盘价都满足，即可认为满足了停顿法的要求。
2. 如果是两次停顿确认笔结束，要求内部至少还有一个相应的分型停顿。

信号列表：

- Signal('15 分钟 _D0 停顿分型 _BE 辅助 V230407_ 看空 _ 任意 _ 任意 _0')
- Signal('15 分钟 _D0 停顿分型 _BE 辅助 V230407_ 看多 _ 任意 _ 任意 _0')
- Signal('15 分钟 _D0 停顿分型 _BE 辅助 V230407_ 看多 _ 内部底停顿 _ 任意 _0')
- Signal('15 分钟 _D0 停顿分型 _BE 辅助 V230407_ 看空 _ 内部顶停顿 _ 任意 _0')

Parameters

c –CZSC 对象

Returns

zdy_dif_V230527

czsc.signals.**zdy_dif_V230527**(*c:* CZSC, ***kwargs*) → OrderedDict

DIF 远离零轴辅助判断买卖点

参数模板："{freq}_N{n}T{t}_DIF 远离 V230527"

信号逻辑：

以多头远离为例：回溯 N 根 K 线，找到 DIF 最大值，然后绿柱回溯 10 跟找到 MACD 最大值，然后看最大值之间的距离判断原理。

信号列表：

- Signal('5 分钟 _N10T30_DIF 远离 V230527_ 空头远离 _ 任意 _ 任意 _0')
- Signal('5 分钟 _N10T30_DIF 远离 V230527_ 多头远离 _ 任意 _ 任意 _0')

Parameters

c –CZSC 对象

Returns

信号识别结果

zdy_dif_V230528

czsc.signals.**zdy_dif_V230528**(*c:* CZSC, ***kwargs*) → OrderedDict

DIF 远离零轴辅助判断买卖点

参数模板："{freq}_N{n}T{t}_DIF 远离 V230528"

信号逻辑：

以多头远离为例：回溯 1000 根 K 线，找到 DIF 的所有峰谷值，如最近一个是峰值，且大于所有峰值的 T% 分位数。

信号列表：

- Signal('5 分钟 _N20T70_DIF 远离 V230528_ 空头远离 _ 任意 _ 任意 _0')
- Signal('5 分钟 _N20T70_DIF 远离 V230528_ 多头远离 _ 任意 _ 任意 _0')

Parameters

c –CZSC 对象

Returns

信号识别结果

zdy_macd_V230518

czsc.signals.**zdy_macd_V230518**(*c:* CZSC, ***kwargs*) → OrderedDict

MACD 交叉次数

参数模板："{freq}_D{di}MACD 交叉 N{n}_BS 辅助 V230518"

信号逻辑：

1. MACD 大于 0，金叉；MACD 小于 0，死叉
2. 计算 MACD 连续大于 0 或者小于 0 的次数

信号列表：

- Signal('60 分钟 _D1MACD 交叉 N9_BS 辅助 V230518_ 死叉 _ 第 1 次 _ 任意 _0')
- Signal('60 分钟 _D1MACD 交叉 N9_BS 辅助 V230518_ 死叉 _ 第 2 次 _ 任意 _0')
- Signal('60 分钟 _D1MACD 交叉 N9_BS 辅助 V230518_ 死叉 _ 第 3 次 _ 任意 _0')
- Signal('60 分钟 _D1MACD 交叉 N9_BS 辅助 V230518_ 死叉 _ 第 4 次 _ 任意 _0')
- Signal('60 分钟 _D1MACD 交叉 N9_BS 辅助 V230518_ 死叉 _ 第 5 次 _ 任意 _0')
- Signal('60 分钟 _D1MACD 交叉 N9_BS 辅助 V230518_ 死叉 _ 第 6 次 _ 任意 _0')
- Signal('60 分钟 _D1MACD 交叉 N9_BS 辅助 V230518_ 死叉 _ 第 7 次 _ 任意 _0')
- Signal('60 分钟 _D1MACD 交叉 N9_BS 辅助 V230518_ 死叉 _ 第 8 次 _ 任意 _0')
- Signal('60 分钟 _D1MACD 交叉 N9_BS 辅助 V230518_ 死叉 _ 第 9 次 _ 任意 _0')

- Signal(‘60 分钟 _D1MACD 交叉 N9_BS 辅助 V230518_ 金叉 _ 第 1 次 _ 任意 _0’)
- Signal(‘60 分钟 _D1MACD 交叉 N9_BS 辅助 V230518_ 金叉 _ 第 2 次 _ 任意 _0’)
- Signal(‘60 分钟 _D1MACD 交叉 N9_BS 辅助 V230518_ 金叉 _ 第 3 次 _ 任意 _0’)
- Signal(‘60 分钟 _D1MACD 交叉 N9_BS 辅助 V230518_ 金叉 _ 第 4 次 _ 任意 _0’)
- Signal(‘60 分钟 _D1MACD 交叉 N9_BS 辅助 V230518_ 金叉 _ 第 5 次 _ 任意 _0’)
- Signal(‘60 分钟 _D1MACD 交叉 N9_BS 辅助 V230518_ 金叉 _ 第 6 次 _ 任意 _0’)
- Signal(‘60 分钟 _D1MACD 交叉 N9_BS 辅助 V230518_ 金叉 _ 第 7 次 _ 任意 _0’)
- Signal(‘60 分钟 _D1MACD 交叉 N9_BS 辅助 V230518_ 金叉 _ 第 8 次 _ 任意 _0’)
- Signal(‘60 分钟 _D1MACD 交叉 N9_BS 辅助 V230518_ 金叉 _ 第 9 次 _ 任意 _0’)

Parameters

- **c** –CZSC 对象
- **kwargs** –参数字典:return: 返回信号结果

zdy_macd_V230519

czsc.signals.**zdy_macd_V230519**(*c:* CZSC, ***kwargs*) → OrderedDict

MACD 连续缩柱

参数模板:”{freq}_D{di}N{n}MACD 缩柱 _BS 辅助 V230519”

信号逻辑:

1. 取 N 跟 K 线，如果 N 跟 K 线的 MACD 都在零轴上方，且 N 跟 K 线的 MACD 都比 N-1 跟 K-1 线的 MACD 小，则认为是多头连续缩柱
2. 反之为空头连续缩柱

信号列表:

- Signal(‘60 分钟 _D1N3MACD 缩柱 _BS 辅助 V230519_ 多头连续缩柱 _ 任意 _ 任意 _0’)
- Signal(‘60 分钟 _D1N3MACD 缩柱 _BS 辅助 V230519_ 空头连续缩柱 _ 任意 _ 任意 _0’)

Parameters

- **c** –CZSC 对象
- **kwargs** –参数字典

Returns

返回信号结果

zdy_macd_V230527

czsc.signals.**zdy_macd_V230527**(*c:* CZSC, ***kwargs*) → OrderedDict

DIF/DEA/MACD 远离零轴辅助判断买卖点

参数模板："{freq}_{key} 远离 W{w}N{n}T{t}_BS 辅助 V230527"

信号逻辑：

1. 获取最近 w 根 K 线，计算 DIF/DEA/MACD，计算绝对值的中位数和标准差
2. 如果最新的 n 个值的最大绝对值大于中位数 + t / 10 * 标准差，则认为远离零轴

信号列表：

- Signal('60 分钟 _DIF 远离 W100N10T10_BS 辅助 V230527_ 多头远离 _ 任意 _ 任意 _0')
- Signal('60 分钟 _DIF 远离 W100N10T10_BS 辅助 V230527_ 空头远离 _ 任意 _ 任意 _0')

Parameters

c –CZSC 对象

Returns

信号识别结果

zdy_macd_bc_V230422

czsc.signals.**zdy_macd_bc_V230422**(*c:* CZSC, ***kwargs*)

MACD 面积背驰

参数模板："{freq}_D{di}T{th}MACD 面积背驰 _BS 辅助 V230422"

信号逻辑：

以上涨背驰为例，反之为下跌背驰：

1. 背驰段的相应 macd 面积之和 <= 进入中枢段的相应面积之和 * th / 100
2. 中枢把黄白线拉到 0 轴附近，
3. 离开中枢的一笔，黄白线大于 0 且不新高

信号列表：

- Signal('15 分钟 _D1T50MACD 面积背驰 _BS 辅助 V230422_ 上涨 _9 笔 _ 任意 _0')
- Signal('15 分钟 _D1T50MACD 面积背驰 _BS 辅助 V230422_ 上涨 _7 笔 _ 任意 _0')
- Signal('15 分钟 _D1T50MACD 面积背驰 _BS 辅助 V230422_ 下跌 _5 笔 _ 任意 _0')
- Signal('15 分钟 _D1T50MACD 面积背驰 _BS 辅助 V230422_ 上涨 _5 笔 _ 任意 _0')
- Signal('15 分钟 _D1T50MACD 面积背驰 _BS 辅助 V230422_ 下跌 _7 笔 _ 任意 _0')
- Signal('15 分钟 _D1T50MACD 面积背驰 _BS 辅助 V230422_ 下跌 _9 笔 _ 任意 _0')

Parameters

- **c** –基础周期的 CZSC 对象
- **kwargs** –其他参数
  - di: 倒数第 di 根 K 线

- th: 背驰段的相应 macd 面积之和 <= 进入中枢段的相应面积之和 * th / 100

Returns

信号字典

zdy_macd_bs1_V230422

czsc.signals.**zdy_macd_bs1_V230422**(*c:* CZSC, ***kwargs*)

MACD 辅助判断第一类买卖点

参数模板："{freq}_D{di}T{th}MACD_BS1 辅助 V230422"

信号逻辑：

以上涨背驰为例，反之为下跌背驰：

1. 背驰段的相应 macd 面积之和 <= 进入中枢段的相应面积之和 * th / 100
2. 离开中枢的一笔，起点的黄白线在零轴附近，终点的黄白线在中枢黄白线的上方

信号列表：

- Signal('5 分钟 _D1T50MACD_BS1 辅助 V230422_ 看空 _ 上涨 5 笔 _ 任意 _0')
- Signal('5 分钟 _D1T50MACD_BS1 辅助 V230422_ 看多 _ 下跌 7 笔 _ 任意 _0')
- Signal('5 分钟 _D1T50MACD_BS1 辅助 V230422_ 看多 _ 下跌 5 笔 _ 任意 _0')
- Signal('5 分钟 _D1T50MACD_BS1 辅助 V230422_ 看空 _ 上涨 9 笔 _ 任意 _0')
- Signal('5 分钟 _D1T50MACD_BS1 辅助 V230422_ 看空 _ 上涨 7 笔 _ 任意 _0')
- Signal('5 分钟 _D1T50MACD_BS1 辅助 V230422_ 看多 _ 下跌 9 笔 _ 任意 _0')

Parameters

- **c** –基础周期的 CZSC 对象
- **kwargs** –其他参数
  - di: 倒数第 di 根 K 线
  - th: 背驰段的相应 macd 面积之和 <= 进入中枢段的相应面积之和 * th / 100

Returns

信号字典

zdy_macd_dif_V230516

czsc.signals.**zdy_macd_dif_V230516**(*c:* CZSC, ***kwargs*) → OrderedDict

MACD 柱子与 DIF 的关系

参数模板："{freq}_D{di}DIF 走平 _BS 辅助 V230516"

信号逻辑：

- DIF 走平看多：1）DIF 小于 MACD 柱子；2）DIF 相比前一周期下跌小于阈值
- DIF 走平看空：1）DIF 大于 MACD 柱子；2）DIF 相比前一周期上涨小于阈值

信号列表：

- Signal(‘60 分钟 _D1DIF 走平 _BS 辅助 V230516_ 看空 _ 红柱远离 _ 任意 _0’)
- Signal(‘60 分钟 _D1DIF 走平 _BS 辅助 V230516_ 看空 _ 柱子否定 _ 任意 _0’)
- Signal(‘60 分钟 _D1DIF 走平 _BS 辅助 V230516_ 看多 _ 绿柱远离 _ 任意 _0’)
- Signal(‘60 分钟 _D1DIF 走平 _BS 辅助 V230516_ 看多 _ 柱子否定 _ 任意 _0’)

Parameters

- **c** –CZSC 对象
- **kwargs** –参数字典

Returns

返回信号结果

zdy_macd_dif_V230517

czsc.signals.**zdy_macd_dif_V230517**(*c:* CZSC, ***kwargs*) → OrderedDict

MACD 三次开仓条件

参数模板：”{freq}_D{di}MACD 开仓 _BS 辅助 V230517”

信号逻辑：

以多头开仓为例：

1. DIF 在零轴下方运行很长时间后首次升破零轴
2. DIF 在零轴上方首次出现与 DEA 的飞吻
3. DIF 在零轴上方首次出现与 MACD 金叉

信号列表：

- Signal(‘60 分钟 _D1MACD 开仓 _BS 辅助 V230517_ 看空 _DIF 破零轴 _ 任意 _0’)
- Signal(‘60 分钟 _D1MACD 开仓 _BS 辅助 V230517_ 看空 _MACD 飞吻 _ 任意 _0’)
- Signal(‘60 分钟 _D1MACD 开仓 _BS 辅助 V230517_ 看空 _MACD 死叉 _ 任意 _0’)
- Signal(‘60 分钟 _D1MACD 开仓 _BS 辅助 V230517_ 看多 _DIF 破零轴 _ 任意 _0’)
- Signal(‘60 分钟 _D1MACD 开仓 _BS 辅助 V230517_ 看多 _MACD 飞吻 _ 任意 _0’)
- Signal(‘60 分钟 _D1MACD 开仓 _BS 辅助 V230517_ 看多 _MACD 金叉 _ 任意 _0’)

Parameters

- **c** –CZSC 对象
- **kwargs** –参数字典

Returns

返回信号结果

zdy_macd_dif_iqr_V230521

czsc.signals.**zdy_macd_dif_iqr_V230521**(*c:* CZSC, ***kwargs*) → OrderedDict

MACD 柱子与 DIF 的关系

参数模板："{freq}_D{di}DIF 走平 IQR_BS 辅助 V230521"

信号逻辑：

- DIF 走平看多：1）DIF 小于 MACD 柱子；2）最近 3 个周期 DIF 相比前一周期变化在四分位距内
- DIF 走平看空：1）DIF 大于 MACD 柱子；2）最近 3 个周期 DIF 相比前一周期变化在四分位距内

信号列表：

- Signal('60 分钟 _D1DIF 走平 IQR_BS 辅助 V230521_ 看空 _ 红柱远离 _ 任意 _0')
- Signal('60 分钟 _D1DIF 走平 IQR_BS 辅助 V230521_ 看空 _ 柱子否定 _ 任意 _0')
- Signal('60 分钟 _D1DIF 走平 IQR_BS 辅助 V230521_ 看多 _ 绿柱远离 _ 任意 _0')
- Signal('60 分钟 _D1DIF 走平 IQR_BS 辅助 V230521_ 看多 _ 柱子否定 _ 任意 _0')

Parameters

- **c** –CZSC 对象
- **kwargs** –参数字典

Returns

返回信号结果

zdy_stop_loss_V230406

czsc.signals.**zdy_stop_loss_V230406**(*cat:* CzscTrader, ***kwargs*) → OrderedDict

笔操作止损逻辑

参数模板："{freq1}_{pos_name}F{first_stop}_ 止损 V230406"

信号逻辑：

多头止损逻辑如下，反之为空头止损逻辑：

1. 进场后设定止损为固定 BP 止损。
2. 跌破开仓前最后一个分型的低点，止损。
3. 任何一笔在持仓状态下的上升笔结束后，若没有判断止盈走人，则将止损设定在该笔起始点的位置（或适当放低一些固定 BP）

信号列表：

- Signal('日线 _5 日线多头 F300_ 止损 V230406_ 多头止损 _ 任意 _ 任意 _0')
- Signal('日线 _5 日线多头 F300_ 止损 V230406_ 空头止损 _ 任意 _ 任意 _0')

Parameters

cat –CzscTrader 对象

Returns

zdy_take_profit_V230406

czsc.signals.**zdy_take_profit_V230406**(*cat:* CzscTrader, ***kwargs*) → OrderedDict

笔操作止盈逻辑

参数模板："{freq1}_{pos_name}_ 止盈 V230406"

信号逻辑：

多头止盈逻辑如下，反之为空头止盈逻辑：

1. 任何一笔在持仓状态下的上升笔结束，若升破前一下跌笔高点，继续持仓，如没有升破前一下跌笔高点，止盈走人。需要用两次停顿分型判断。

信号列表：

- Signal('日线 _5 日线多头 _ 止盈 V230406_ 多头止盈 _ 任意 _ 任意 _0')
- Signal('日线 _5 日线多头 _ 止盈 V230406_ 空头止盈 _ 任意 _ 任意 _0')

Parameters

cat – CzscTrader 对象

Returns

zdy_take_profit_V230407

czsc.signals.**zdy_take_profit_V230407**(*cat:* CzscTrader, ***kwargs*) → OrderedDict

笔操作止盈逻辑

参数模板："{freq1}_{pos_name}_ 止盈 V230407"

信号逻辑：

多头止盈逻辑如下，反之为空头止盈逻辑：1. 根据 K 线个数判断力度，提前止盈。(三买之后的上升笔中，如果笔数已经大于前一下跌笔的 K 线个数的 1.5 倍，并且还没有新高，直接止盈走人。

信号列表：

- Signal('日线 _5 日线多头 _ 止盈 V230407_ 多头止盈 _ 任意 _ 任意 _0')
- Signal('日线 _5 日线多头 _ 止盈 V230407_ 空头止盈 _ 任意 _ 任意 _0')

Parameters

cat – CzscTrader 对象

Returns

信号字典

zdy_vibrate_V230406

czsc.signals.**zdy_vibrate_V230406**(*cat:* CzscTrader, *freq1*, *freq2*, ***kwargs*) → OrderedDict

中枢震荡短差操作

参数模板："中枢震荡 _{freq1}#{freq2}_BS 辅助 V230406"

信号逻辑：

1、中枢笔数必须大于等于 3 笔。（排除掉盘背构成的中枢）2、中枢上沿做空，中枢下沿做多（不区分中枢方向）

以开空仓为例（中枢上沿），当向上一笔出现次级别分型停顿后，定义次级别收盘价为 P，次级别顶分型高点为 H，则需满足：

1. H>= 本级别中枢上沿价格 2.（H-本级别中枢上沿价格）< 本级别中枢高度 3.（H-P）*3 < 中枢高度 4. 本级别 MACD 黄白线死叉确立（已经走出并列向下走势）

稍微解释一下几个条件的内在逻辑：

1. H>= 本级别中枢上沿价格：确保是在中枢的上沿 2.（H-本级别中枢上沿价格）< 本级别中枢高度：确保中枢上沿不要太远的距离，太远了可能下跌就形成三买上去了。这种点不要。3.（H-P）*3 < 中枢高度：H-P，其实就是一个次级别分型停顿的高度了，这里就是为了排除掉奔走型中枢，区间特别小的中枢，没必要做。

信号列表：

- Signal（'中枢震荡 _5 分钟 #60 分钟 _BS 辅助 V230406_ 看多 _ 任意 _ 任意 _0'）
- Signal（'中枢震荡 _5 分钟 #60 分钟 _BS 辅助 V230406_ 看空 _ 任意 _ 任意 _0'）

Parameters

- **cat** – 交易员对象
- **freq1** – 次级别
- **freq2** – 本级别
- **kwargs** –

Returns

zdy_zs_V230423

czsc.signals.**zdy_zs_V230423**(*c:* CZSC, ***kwargs*)

约束中枢的形态和高度

参数模板："{freq}_D{di} 中枢形态 _BS 辅助 V230423"

信号逻辑：

以上涨中枢为例，反之为下跌中枢：

1. 进入笔和离开笔分别是高低点，笔的数量大于等于 5

2. 中枢高度 >= 进入中枢那一笔高度 / 3

信号列表：

- Signal(‘15 分钟 _D1 中枢形态 _BS 辅助 V230423_ 上涨 _5 笔 _ 任意 _0’)
- Signal(‘15 分钟 _D1 中枢形态 _BS 辅助 V230423_ 上涨 _7 笔 _ 任意 _0’)
- Signal(‘15 分钟 _D1 中枢形态 _BS 辅助 V230423_ 下跌 _5 笔 _ 任意 _0’)
- Signal(‘15 分钟 _D1 中枢形态 _BS 辅助 V230423_ 下跌 _7 笔 _ 任意 _0’)
- Signal(‘15 分钟 _D1 中枢形态 _BS 辅助 V230423_ 下跌 _9 笔 _ 任意 _0’)
- Signal(‘15 分钟 _D1 中枢形态 _BS 辅助 V230423_ 上涨 _9 笔 _ 任意 _0’)

Parameters

- **c** –基础周期的 CZSC 对象
- **kwargs** –其他参数
  - di: 倒数第 di 根 K 线

Returns

信号字典

zdy_zs_space_V230421

czsc.signals.**zdy_zs_space_V230421**(*c:* CZSC, ***kwargs*)

中枢空间形态约束

参数模板：”{freq}_D{di} 中枢空间 _BS 辅助 V230421”

信号逻辑：

以顶背驰为例：要求【离开笔】距离中枢上沿的高度，大于等于【进入笔】距离中枢下沿的高度

信号列表：

- Signal(‘15 分钟 _D1 中枢空间 _BS 辅助 V230421_ 上涨 _5 笔 _ 任意 _0’)
- Signal(‘15 分钟 _D1 中枢空间 _BS 辅助 V230421_ 下跌 _5 笔 _ 任意 _0’)
- Signal(‘15 分钟 _D1 中枢空间 _BS 辅助 V230421_ 上涨 _9 笔 _ 任意 _0’)
- Signal(‘15 分钟 _D1 中枢空间 _BS 辅助 V230421_ 上涨 _7 笔 _ 任意 _0’)
- Signal(‘15 分钟 _D1 中枢空间 _BS 辅助 V230421_ 下跌 _7 笔 _ 任意 _0’)
- Signal(‘15 分钟 _D1 中枢空间 _BS 辅助 V230421_ 下跌 _9 笔 _ 任意 _0’)

Parameters

- **c** –基础周期的 CZSC 对象
- **kwargs** –其他参数 - di: 倒数第 di 根 K 线

Returns

信号字典

2.5 czsc.sensors Package

2.5.1 Functions

discretizer(df, col[, n_bins, encode, strategy])	使用 KBinsDiscretizer 对连续变量在时间截面上进行离散化
get_index_beta(dc, sdt, edt[, freq, ...])	获取基准指数的 Beta
holds_concepts_effect(holds, concepts[, ...])	股票持仓列表的板块效应
turn_over_rate(df_holds)	计算持仓明细对应的组合换手率

discretizer

czsc.sensors.**discretizer**(*df: DataFrame*, *col: str*, *n_bins=20*, *encode='ordinal'*, *strategy='quantile'*)

使用 KBinsDiscretizer 对连续变量在时间截面上进行离散化

Parameters

- **df** –数据对象
- **col** –连续变量列名
- **n_bins** –参见 KBinsDiscretizer 文档 https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.KBinsDiscretizer.html
- **encode** –参见 KBinsDiscretizer 文档
- **strategy** –参见 KBinsDiscretizer 文档

Returns

get_index_beta

czsc.sensors.**get_index_beta**(*dc: TsDataCache*, *sdt: str*, *edt: str*, *freq='D'*, *file_xlsx=None*, *indices=None*)

获取基准指数的 Beta

Parameters

- **dc** –数据缓存对象
- **sdt** –开始日期
- **edt** –结束日期
- **freq** –K 线周期，D 日线，W 周线，M 月线
- **file_xlsx** –结果保存文件
- **indices** –定义指数列表

Returns

holds_concepts_effect

czsc.sensors.**holds_concepts_effect**(*holds: DataFrame*, *concepts: dict*, *top_n=20*, *min_n=3*, ***kwargs*)

股票持仓列表的板块效应

原理概述：在选股时，如果股票的概念板块与组合中的其他股票的概念板块有重合，那么这个股票的表现会更好。

函数计算逻辑:

1. 如果 kwargs 中存在’copy’键且对应值为 True，则将 holds 进行复制。
2. 为 holds 添加’概念板块’列，该列的值是 holds 中’symbol’列对应的股票的概念板块列表，如果没有对应的概念板块则填充为空。
3. 添加’概念数量’列，该列的值是每个股票的概念板块数量。
4. 从 holds 中筛选出概念数量大于 0 的行，赋值给 holds。
5. 创建空列表 new_holds 和空字典 dt_key_concepts。

6. 对 holds 按照’dt’进行分组，遍历每个分组，计算板块效应。a. 计算密集出现的概念，选取出现次数最多的前 top_n 个概念，赋值给 key_concepts 列表。b. 将日期 dt 和对应的 key_concepts 存入 dt_key_concepts 字典。c. 计算在密集概念中出现次数超过 min_n 的股票，将符合条件的股票添加到 new_holds 列表中。7. 使用 pd.concat 将 new_holds 中的 DataFrame 进行合并，忽略索引，赋值给 dfh。8. 创建 DataFrame dfk，其中包含日期 (dt) 和对应的强势概念 (key_concepts)。9. 返回 dfh 和 dfk。

Parameters

- **holds** –组合股票池数据，样例：

dt	symbol	weight
2023-05-09 00:00:00	601858.SH	0.00333333
2023-05-09 00:00:00	300502.SZ	0.00333333
2023-05-09 00:00:00	603258.SH	0.00333333
2023-05-09 00:00:00	300499.SZ	0.00333333
2023-05-09 00:00:00	300624.SZ	0.00333333

- **concepts** –股票的概念板块，样例：{

  ’002507.SZ’: [‘电子商务’，‘超级品牌’，‘国企改革’],‘002508.SZ’: [‘家用电器’，‘杭州亚运会’，‘恒大概念’]

  }

- **top_n** –选取前 n 个密集概念
- **min_n** –单股票至少要有 n 个概念在 top_n 中

Returns

过滤后的选股结果，每个时间点的 top_n 概念

turn_over_rate

czsc.sensors.**turn_over_rate**(*df_holds: DataFrame*) → [<class 'pandas.core.frame.DataFrame'>, <class 'float'>]

计算持仓明细对应的组合换手率

Parameters

df_holds –

每个交易日的持仓明细，数据样例如下

证券代码成分日期持仓权重

0 000576.SZ 2020-01-02 0.0099 1 000639.SZ 2020-01-02 0.0099 2 000803.SZ 2020-01-02 0.0099 3 000811.SZ 2020-01-02 0.0099 4 000829.SZ 2020-01-02 0.0099

Returns

组合换手率

2.5.2 Classes

CTAResearch(strategy, read_bars, ...)	
EventMatchSensor(events, symbols, read_bars, ...)	

CTAResearch

class czsc.sensors.**CTAResearch**(*strategy*, *read_bars*, *results_path*, ***kwargs*)

Bases: object

Methods Summary

backtest(symbols[, max_workers])	多进程执行 on bar 回测
check_signals(symbol[, sdt, edt])	在单个品种上检查信号
dummy(symbols[, sdt, edt, max_workers])	使用 DummyBacktest 进行 on sig 回测
replay(symbol[, sdt, edt, refresh])	单品种交易回放

Methods Documentation

backtest(*symbols*, *max_workers=3*, ***kwargs*)

多进程执行 on bar 回测

Parameters

- **symbols** –标的代码列表
- **max_workers** –最大进程数

Returns

None

check_signals(*symbol*, *sdt='20200101'*, *edt='20220101'*)

在单个品种上检查信号

Parameters

- **symbol** –标的代码
- **sdt** –开始时间
- **edt** –结束时间

Returns

None

dummy(*symbols*, *sdt='20200101'*, *edt='20220101'*, *max_workers=1*, ***kwargs*)

使用 DummyBacktest 进行 on sig 回测

Parameters

- **symbols** –品种列表
- **sdt** –回测开始时间
- **edt** –回测结束时间
- **max_workers** –最大进程数
- **kwargs** –

Returns

replay(*symbol*, *sdt='20200101'*, *edt='20220101'*, *refresh=True*)

单品种交易回放

Parameters

- **symbol** –标的代码
- **sdt** –开始时间
- **edt** –结束时间

- **refresh** –是否刷新

Returns

None

EventMatchSensor

class czsc.sensors.**EventMatchSensor**(*events: List[Dict[str, Any] | Event]*, *symbols: List[str]*, *read_bars: Callable*, ***kwargs*)

Bases: object

Methods Summary

get_event_csc(event_name)	获取事件的截面匹配次数

Methods Documentation

get_event_csc(*event_name: str*)

获取事件的截面匹配次数

csc = cross section count，表示截面匹配次数

函数执行逻辑：

1. 创建一个 self.data 的副本 df。
2. 在 df 中筛选出 event_name 列等于 1 的行。
3. **使用 groupby 方法按 symbol 和 dt 对筛选后的数据进行分组，并计算 event_name 列的总和。**
   结果将形成一个新的 DataFrame，其中索引为 (symbol, dt) 组合，只有一个列 event_name，表示每个组合的匹配次数。
4. **再次使用 groupby 方法按 dt 对上一步的结果进行分组，并计算 event_name 列的总和。这次得到的新 DataFrame**
   只有一个列 event_name，表示在每个时间点所有标的的事件匹配总数。

Parameters

event_name –事件名称

Returns

DataFrame

2.6 czsc.traders Package

2.6.1 Functions

check_signals_acc(bars, signals_config[, ...])	输入基础周期 K 线和想要验证的信号，输出信号识别结果的快照
clear_strategy(strategy_name[, redis_url, ...])	删除策略所有记录
combine_dates_and_pairs(dates, pairs, ...)	结合大盘日期择时和择时策略开平交易进行分析
combine_holds_and_pairs(holds, pairs, ...)	结合股票池和择时策略开平交易进行分析
generate_czsc_signals(bars, signals_config)	使用 CzscSignals 生成信号
get_ensemble_weight(trader[, method])	获取 CzscTrader 中所有 positions 按照 method 方法集成之后的权重
get_heartbeat_time([strategy_name, ...])	获取策略的最近一次心跳时间
get_signals_config(signals_seq[, signals_module])	获取信号列表对应的信号函数配置
get_signals_freqs(signals_seq)	获取信号列表对应的 K 线周期列表
get_strategy_mates([redis_url, ...])	获取 Redis 中的策略元数据
get_strategy_weights(strategy_name[, ...])	获取策略的持仓权重
get_unique_signals(bars, signals_config, ...)	获取信号函数中定义的所有信号列表
long_short_equity(factors, returns[, ...])	根据截面因子值与收益率，回测分析多空对冲组合的收益率
stock_holds_performance(dc, dfh, res_path)	计算 A 股日线持仓组合的表现
stoploss_by_direction(dfw[, stoploss])	按持仓方向进行止损

check_signals_acc

czsc.traders.**check_signals_acc**(*bars: List[RawBar]*, *signals_config: List[dict]*, *delta_days: int = 5*, ***kwargs*) → None

输入基础周期 K 线和想要验证的信号，输出信号识别结果的快照

函数执行逻辑：

1. 函数首先获取基础周期 K 线的 base_freq，并检查输入的 K 线数据 bars 是否按时间升序排列。如果 bars 的长度小于 600，函数直接返回。
2. 然后，函数调用 generate_czsc_signals 方法，生成 Czsc 信号，并将结果保存在 df 中。
3. 函数提取出 df 中所有的信号列 s_cols，并打印每一列的值的数量。然后，函数将所有的信号添加到 signals 列表中。
4. 函数将 bars 分为两部分，bars_left 和 bars_right，并获取信号配置 signals_config 中的所有 freqs。
5. 函数创建一个 BarGenerator 对象 bg，并使用 bars_left 中的 K 线数据来初始化它。
6. 函数创建一个 CzscSignals 对象 ct，并将 bg 和信号配置 signals_config 作为参数传入。
7. 函数创建一个字典 last_dt，用于存储每一个信号最后一次出现的时间。

8. 函数遍历 bars_right 中的每一根 K 线，对于每一根 K 线，函数调用 ct.update_signals(bar) 来更新信号。
9. 对于每一个信号，如果当前 K 线的时间与该信号最后一次出现的时间的差值大于 delta_days，并且该信号与当前的信号匹配，函数将创建一个 HTML 文件，保存信号识别结果的快照，并更新该信号最后一次出现的时间。

Parameters

- **bars** –原始 K 线
- **signals_config** –需要验证的信号列表
- **delta_days** –两次相同信号之间的间隔天数

Returns

None

clear_strategy

czsc.traders.**clear_strategy**(*strategy_name*, *redis_url=None*, *connection_pool=None*, *key_prefix='Weights'*, ***kwargs*)

删除策略所有记录

Parameters

- **strategy_name** –str, 策略名
- **redis_url** –str, redis 连接字符串，默认为 None，即从环境变量 RWC_REDIS_URL 中读取
- **connection_pool** –redis.ConnectionPool, redis 连接池
- **key_prefix** –str, redis 中 key 的前缀，默认为 Weights
- **kwargs** –dict, 其他参数

combine_dates_and_pairs

czsc.traders.**combine_dates_and_pairs**(*dates: list*, *pairs: DataFrame*, *results_path*)

结合大盘日期择时和择时策略开平交易进行分析

函数计算逻辑:

1. 将 dates 转换为日期类型，并赋值给变量 dates。
2. 将 pairs 复制到 dfp 变量。
3. 将 dfp 的’开仓时间’列转换为日期类型，并将日期部分提取出来赋值给’开仓日期’列。
4. 从 dfp 中选择开仓日期在 dates 中的数据，赋值给 df_pairs。
5. 从 dfp 中选择开仓时间在 df_pairs 的开仓时间范围内的数据，赋值给 dfp_sub。
6. 使用 dfp_sub 创建 PairsPerformance 对象 tp_old。
7. 使用 df_pairs 创建 PairsPerformance 对象 tp_new。

8. 打印原始交易的基本信息和平仓年度统计。
9. 打印组合过滤后的交易的基本信息和平仓年度统计。
10. 创建结果目录并保存评价结果和交易数据。
11. 返回 tp_old 和 tp_new 对象。

Parameters

- **dates** –大盘日期择时日期数据，数据样例 [‘2020-01-02’, …, ‘2022-01-06’]
- **pairs** –

  择时策略开平交易数据，数据格式如下

  标的代码交易方向最大仓位开仓时间累计开仓平仓时间 0 002698.SZ 多头 1 2015-01-12 13:30:00 24.02790 2015-01-13 09:45:00

  1 300031.SZ 多头 1 2015-01-12 10:30:00 53.87420 2015-01-13 09:45:00 2 300046.SZ 多头 1 2015-01-12 10:15:00 41.35824 2015-01-13 09:45:00 3 300076.SZ 多头 1 2015-01-12 10:30:00 57.84800 2015-01-13 09:45:00 4 300099.SZ 多头 1 2015-01-12 10:15:00 62.57308 2015-01-13 09:45:00

  累计平仓累计换手持仓 K 线数持仓天数盈亏金额交易盈亏盈亏比例

  0 23.38150 2 7 0.843750 -0.64640 -0.0269 -0.0269 1 52.71284 2 13 0.968750 -1.16136 -0.0215 -0.0215 2 40.72068 2 14 0.979167 -0.63756 -0.0154 -0.0154 3 55.45144 2 13 0.968750 -2.39656 -0.0414 -0.0414 4 61.50528 2 14 0.979167 -1.06780 -0.0170 -0.0170

- **results_path** –分析结果目录

Returns

combine_holds_and_pairs

czsc.traders.**combine_holds_and_pairs**(*holds*, *pairs*, *results_path*)

结合股票池和择时策略开平交易进行分析

函数计算逻辑:

1. **将 holds 和 pairs 数据进行处理和准备**。
   - 将 holds 复制到 dfh 变量。
   - 将 dfh 的’成分日期’列转换为日期类型。
   - 将 dfh 的’证券代码’列赋值给’标的代码’列。
   - 将 pairs 复制到 dfp 变量。
   - 将 dfp 的’开仓时间’列转换为日期类型，并将日期部分提取出来赋值给’开仓日期’列。
2. **合并数据并筛选交易对**。
   - 将 dfp 与 dfh 的 [[‘开仓日期’, ‘标的代码’, ‘持仓权重’]] 列进行左连接，**得到 dfp_**。

- 从 dfp_ 中选择持仓权重大于 0 的交易对，赋值给 df_pairs。
- 从 dfp 中选择开仓时间在 df_pairs 的开仓时间范围内的数据，赋值给 dfp_sub。

3. **进行评价和分析**。
   - 使用 dfp_sub 创建 PairsPerformance 对象 tp_old。
   - 使用 df_pairs 创建 PairsPerformance 对象 tp_new。
4. **创建结果目录并保存评价结果和交易数据**。
   - 使用 os.makedirs 创建结果目录。
   - 将 tp_old 的统计结果保存为 Excel 文件，文件名为”原始交易评价.xlsx”。
   - 将 tp_new 的统计结果保存为 Excel 文件，文件名为”组合过滤评价.xlsx”。
   - 将 df_pairs 的数据保存为 Feather 文件，文件名为”组合过滤交易.feather”。
5. 返回 tp_old 和 tp_new 对象。

Parameters

- **holds** –

  组合股票池数据，样例：
  成分日期证券代码 n1b 持仓权重

  0 2020-01-02 000001.SZ 183.758194 0.001232 1 2020-01-02 000002.SZ -156.633896 0.001232 2 2020-01-02 000063.SZ 310.296204 0.001232 3 2020-01-02 000066.SZ -131.824997 0.001232 4 2020-01-02 000069.SZ -38.561699 0.001232

- **pairs** –

  择时策略开平交易数据，数据格式如下
  标的代码交易方向最大仓位开仓时间累计开仓平仓时间 0 002698.SZ 多头 1 2015-01-12 13:30:00 24.02790 2015-01-13 09:45:00

  1 300031.SZ 多头 1 2015-01-12 10:30:00 53.87420 2015-01-13 09:45:00 2 300046.SZ 多头 1 2015-01-12 10:15:00 41.35824 2015-01-13 09:45:00 3 300076.SZ 多头 1 2015-01-12 10:30:00 57.84800 2015-01-13 09:45:00 4 300099.SZ 多头 1 2015-01-12 10:15:00 62.57308 2015-01-13 09:45:00

  累计平仓累计换手持仓 K 线数持仓天数盈亏金额交易盈亏盈亏比例

  0 23.38150 2 7 0.843750 -0.64640 -0.0269 -0.0269 1 52.71284 2 13 0.968750 -1.16136 -0.0215 -0.0215 2 40.72068 2 14 0.979167 -0.63756 -0.0154 -0.0154 3 55.45144 2 13 0.968750 -2.39656 -0.0414 -0.0414 4 61.50528 2 14 0.979167 -1.06780 -0.0170 -0.0170

- **results_path** –分析结果目录

Returns

generate_czsc_signals

czsc.traders.**generate_czsc_signals**(*bars: List[RawBar]*, *signals_config: List[dict]*, *sdt: AnyStr | datetime = '20170101'*, *init_n: int = 500*, *df=False*, ***kwargs*)

使用 CzscSignals 生成信号

函数执行逻辑：

1. 函数首先从信号配置 signals_config 中获取所有的 freqs。
2. 然后，函数将信号计算开始时间 sdt 转换为 datetime 类型，并将开始时间之前的 K 线数据分配给 bars_left，开始时间之后的 K 线数据分配给 bars_right。
3. 如果 bars_right 为空，即没有开始时间之后的 K 线数据，函数会发出一个警告，并返回一个空的 DataFrame 或空列表。
4. 函数创建一个 BarGenerator 对象 bg，并使用 bars_left 中的 K 线数据来初始化它。
5. 函数创建一个 CzscSignals 对象 cs，并将 bg 和信号配置 signals_config 作为参数传入。
6. 函数遍历 bars_right 中的每一根 K 线，对于每一根 K 线，函数调用 cs.update_signals(bar) 来更新信号，并将更新后的信号添加到 _sigs 列表中。
7. 最后，如果 df 参数为 True，函数将 _sigs 转换为 DataFrame 并返回；否则，直接返回 _sigs。

Parameters

- **bars** –基础周期 K 线序列
- **signals_config** –信号函数配置，格式如下：signals_config = [

  { ‘name’： ‘czsc.signals.tas_ma_base_V221101’, ‘freq’： ‘日线’, ‘di’： 1, ‘ma_type’： ‘SMA’, ‘timeperiod’： 5}, { ‘name’: ‘czsc.signals.tas_ma_base_V221101’, ‘freq’: ‘日线’, ‘di’: 5, ‘ma_type’: ‘SMA’, ‘timeperiod’: 5}, { ‘name’: ‘czsc.signals.tas_double_ma_V221203’, ‘freq’： ‘日线’, ‘di’: 1, ‘ma_seq’: (5, 20), ‘th’: 100}, { ‘name’： ‘czsc.signals.tas_double_ma_V221203’, ‘freq’： ‘日线’, ‘di’: 5, ‘ma_seq’: (5, 20), ‘th’: 100},

  ]
- **sdt** –信号计算开始时间
- **init_n** –用于 BarGenerator 初始化的基础周期 K 线数量
- **df** –是否返回 df 格式的信号计算结果，默认 False

Returns

信号计算结果

get_ensemble_weight

czsc.traders.**get_ensemble_weight**(*trader: CzscTrader*, *method: AnyStr | Callable = 'mean'*)

获取 CzscTrader 中所有 positions 按照 method 方法集成之后的权重

函数计算逻辑：

1. 获取 trader 持仓信息并转换为 DataFrame:
   - 遍历交易者的每个持仓位置。
   - 将每个位置的持仓信息转换为 DataFrame，并合并到一个整体的 DataFrame 中。
   - 将持仓列重命名为对应的位置名称。
2. 根据给定的方法计算权重:
   - 如果方法是可调用对象，将持仓信息转换为字典，并传递给该方法进行计算。
   - 如果方法是预定义字符串（”mean”、”max”、”min”、”vote”），根据相应的计算方式计算权重。
3. 返回包含日期、交易标的、权重和价格的 DataFrame:
   - 将计算得到的权重与其他相关列一起组成一个新的 DataFrame。
   - 将交易标的信息添加到新的 DataFrame 中。
   - 返回包含日期、交易标的、权重和价格的 DataFrame 副本。

Parameters

- **trader** –CzscTrader 缠论交易员对象
- **method** –str or callable

  集成方法，可选值包括：’mean’，‘max’，‘min’，‘vote’也可以传入自定义的函数，函数的输入为 dict，key 为 position.name，value 为 position.pos, 样例输入：

  {‘多头策略 A’：1，‘多头策略 B’：1，‘空头策略 A’：-1}
- **kwargs** –

Returns

pd.DataFrame columns = [‘dt’，‘symbol’，‘weight’，‘price’]

get_heartbeat_time

czsc.traders.**get_heartbeat_time**(*strategy_name=None*, *redis_url=None*, *connection_pool=None*, *key_prefix='Weights'*, ***kwargs*)

获取策略的最近一次心跳时间

Parameters

- **strategy_name** –str, 策略名，默认为 None, 即获取所有策略的心跳时间

- **redis_url** –str, redis 连接字符串，默认为 None，即从环境变量 RWC_REDIS_URL 中读取
- **connection_pool** –redis.ConnectionPool, redis 连接池
- **key_prefix** –str, redis 中 key 的前缀，默认为 Weights
- **kwargs** –dict, 其他参数
  - heartbeat_prefix: str, 心跳 key 的前缀，默认为 heartbeat

Returns

str, 最近一次心跳时间

get_signals_config

czsc.traders.**get_signals_config**(*signals_seq: List[str]*, *signals_module: str = 'czsc.signals'*) → List[Dict]

获取信号列表对应的信号函数配置

函数执行逻辑：

1. **首先创建了一个 SignalsParser 类的实例对象 sp，传入了参数 signals_module 进行初始化，**
   初始化工作主要是解析 signals_module 下的信号函数，生成了 sig_pats_map 信号参数模板字典和 sig_name_map 信号列表字典。
2. 然后使用 sp 实例调用 parse 方法，该方法解析 signals_seq 中的信号，并返回信号函数的配置信息。

Parameters

- **signals_seq** –信号列表
- **signals_module** –信号函数所在模块

Returns

信号函数配置

get_signals_freqs

czsc.traders.**get_signals_freqs**(*signals_seq: List*) → List[str]

获取信号列表对应的 K 线周期列表

函数执行逻辑：

1. 然后对于 signals_seq 中的每个信号进行以下操作：
   - 使用正则表达式从信号中提取信号周期，并将其存储在 _freqs 变量中。
   - 如果提取到了信号周期，则将其加入到 freqs 列表中。
2. 最后验证数据是否符合 sorted_freqs 列表规范，并且以 sorted_freqs 列表的排序进行返回。

Parameters

signals_seq –信号列表 / 信号函数配置列表

Returns

K 线周期列表

get_strategy_mates

czsc.traders.**get_strategy_mates**(*redis_url=None*, *connection_pool=None*, *key_pattern='Weights:META:*'*, ***kwargs*)

获取 Redis 中的策略元数据

Parameters

- **redis_url** –str, redis 连接字符串, 默认为 None, 即从环境变量 RWC_REDIS_URL 中读取
- **connection_pool** –redis.ConnectionPool, redis 连接池
- **key_pattern** –str, redis 中 key 的 pattern，默认为 Weights:META:*
- **kwargs** –dict, 其他参数

Returns

pd.DataFrame

get_strategy_weights

czsc.traders.**get_strategy_weights**(*strategy_name*, *redis_url=None*, *connection_pool=None*, *key_prefix='Weights'*, ***kwargs*)

获取策略的持仓权重

Parameters

- **strategy_name** –str, 策略名
- **redis_url** –str, redis 连接字符串, 默认为 None, 即从环境变量 RWC_REDIS_URL 中读取
- **connection_pool** –redis.ConnectionPool, redis 连接池
- **key_prefix** –str, redis 中 key 的前缀，默认为 Weights
- **kwargs** –dict, 其他参数
  - symbols : list, 品种列表, 默认为 None, 即获取所有品种的权重
  - sdt : str, 开始时间, eg: 20210924 10:19:00
  - edt : str, 结束时间, eg: 20220924 10:19:00
  - only_last : boolean, 是否只保留每个品种最近一次权重，默认为 False

Returns

pd.DataFrame

get_unique_signals

czsc.traders.**get_unique_signals**(*bars: List[RawBar]*, *signals_config: List[dict]*, ***kwargs*)

获取信号函数中定义的所有信号列表

函数执行逻辑：

1. 函数首先检查输入的 K 线数据 bars 是否按时间升序排列。如果 bars 的长度小于 600，函数直接返回一个空列表。
2. 然后，函数调用 generate_czsc_signals 方法，生成 CZSC 信号，并将结果保存在 df 中。
3. **函数遍历 df 中的所有列，对于每一列，如果列名包含三个部分，函数提取出该列中的所有唯一值，然后将列名和每一个唯一值组合成一个新的信号，**
   并添加到 _res 列表中。注意，如果唯一值中包含”其他”，则不会被添加到 _res 中。
4. 最后，函数返回 _res，其中包含了所有的唯一信号。

Parameters

- **bars** –基础 K 线数据
- **signals_config** –信号函数配置
- **kwargs** –传递给 generate_czsc_signals 方法的参数

Returns

信号列表

long_short_equity

czsc.traders.**long_short_equity**(*factors*, *returns*, *hold_period=2*, *rank=5*, ***kwargs*)

根据截面因子值与收益率，回测分析多空对冲组合的收益率

Parameters

- **factors** –

  截面因子，因子值越大，越偏向于做多，因子值越小，越偏向于做空；数据格式如下：
  SFIH9001 SFIF9001 SFIC9001

  dt 2022-08-31 1.403915 1.252826 0.968868 2022-09-01 1.376690 1.253377 0.972276 2022-09-02 1.380867 1.253929 0.974999 2022-09-05 1.370359 1.254482 0.977737 2022-09-06 0.685180 0.633634 0.493986

- **returns** –

  品种收益率矩阵，数据格式如下：
  SFIH9001 SFIF9001 SFIC9001

  dt 2021-01-04 0.007803 0.017228 0.004843 2021-01-05 0.014068 0.008300 0.000598 2021-01-06 0.024520 0.022766 0.004974 2021-01-07 -0.006193 -0.003698 0.005951 2021-01-08 -0.005651 -0.012263 -0.016441

- **hold_period** –持仓周期，dt 时刻的数量，如果是 2，则表示每两个交易时刻调仓一次
- **rank** –排序因子值前几名，或者排名因子值的前百分之几；如果是整数，则表示排名因子值前几名；如果是浮点数，则表示排名因子值的前百分之几。排名靠前，越偏向于做多；排名靠后，越偏向于做空。
- **kwargs** –

Returns

stock_holds_performance

czsc.traders.**stock_holds_performance**(*dc:* TsDataCache, *dfh*, *res_path*)

计算 A 股日线持仓组合的表现

Parameters

- **dc** –Tushare 数据缓存对象
- **dfh** –

  持仓组合，样例如下，其中【证券代码】要求是 tushare 的格式，成分日期当天状态为持仓
  成分日期证券代码持仓权重 n1b

  0 2020-01-03 300620.SZ 0.008403 141.861099 1 2020-01-03 300677.SZ 0.008403 767.124023 2 2020-01-03 300708.SZ 0.008403 93.029297 3 2020-01-03 002151.SZ 0.008403 -7.465500 4 2020-01-03 002156.SZ 0.008403 350.101715
- **res_path** –结果保存路径

Returns

stoploss_by_direction

czsc.traders.**stoploss_by_direction**(*dfw*, *stoploss=0.03*, ***kwargs*)

按持仓方向进行止损

Parameters

- **dfw** –pd.DataFrame, columns = [‘dt’, ‘symbol’, ‘weight’, ‘price’], 持仓权重数据，其中

  dt 为 K 线结束时间，必须是连续的交易时间序列，不允许有时间断层 symbol 为合约代码，weight 为 K 线结束时间对应的持仓权重，品种之间的权重是独立的，不会互相影响 price 为结束时间对应的交易价格，可以是当前 K 线的收盘价，或者下一根 K 线的开盘价，或者未来 N 根 K 线的 TWAP、VWAP 等

  数据样例如下：========== ======== ======== ======= dt symbol weight price ========== ======== ======== ======= 2019-01-02

09:01:00 DLi9001 0.5 961.695 2019-01-02 09:02:00 DLi9001 0.25 960.72 2019-01-02 09:03:00 DLi9001 0.25 962.669 2019-01-02 09:04:00 DLi9001 0.25 960.72 2019-01-02 09:05:00 DLi9001 0.25 961.695 ========== ======== ======== =======

- **stoploss** –止损比例
- **kwargs** –其他参数

Returns

pd.DataFrame，columns = [‘dt’，‘symbol’，‘weight’，‘raw_weight’，‘price’，‘returns’，

’ hold_returns’，‘min_hold_returns’，‘order_id’，‘is_stop’]

2.6.2 Classes

CzscSignals([bg])	缠中说禅技术分析理论之多级别信号计算
CzscTrader([bg, positions, ensemble_method])	缠中说禅技术分析理论之多级别联立交易决策类（支持多策略独立执行）
DummyBacktest(strategy, signals_path, ...)	
ExitsOptimize(read_bars, **kwargs)	基础策略出场优化流程
OpensOptimize(read_bars, **kwargs)	基础策略入场优化流程
PairsPerformance(df_pairs)	交易对效果评估
RedisWeightsClient(strategy_name[, ...])	策略持仓权重收发客户端
SignalsParser([signals_module])	解析一串信号，生成信号函数配置
WeightBacktest(dfw[, digits])	持仓权重回测

CzscSignals

class czsc.traders.**CzscSignals**(*bg:* BarGenerator | *None* = *None*, ***kwargs*)

Bases: object

缠中说禅技术分析理论之多级别信号计算

Methods Summary

get_signals_by_conf()	通过信号参数配置获取信号
open_in_browser([width, height])	直接在浏览器中打开分析结果
take_snapshot([file_html, width, height])	获取快照
update_signals(bar)	输入基础周期已完成 K 线，更新信号，更新仓位

Methods Documentation

get_signals_by_conf()

通过信号参数配置获取信号

函数执行逻辑：

1. 函数首先创建一个空的有序字典 s。
2. 如果 self.signals_config 不存在，函数直接返回空字典 s，否则，函数遍历其中的每一个配置。
3. 对于每一个参数，函数提取出信号名称和 freq，并根据这两个参数获取相应的信号，获取到的信号被添加到字典 s 中。
4. 函数最后返回字典 s，其中包含了所有获取到的信号。

信号参数配置，格式如下：

signals_config = [

{'name': 'czsc.signals.tas_ma_base_V221101', 'freq': '日线', 'di': 1, 'ma_type': 'SMA', 'timeperiod': 5}, {'name': 'czsc.signals.tas_ma_base_V221101', 'freq': '日线', 'di': 5, 'ma_type': 'SMA', 'timeperiod': 5}, {'name': 'czsc.signals.tas_double_ma_V221203', 'freq': '日线', 'di': 1, 'ma_seq': (5, 20), 'th': 100}, {'name': 'czsc.signals.tas_double_ma_V221203', 'freq': '日线', 'di': 5, 'ma_seq': (5, 20), 'th': 100},

]

Returns

信号字典

open_in_browser(*width='1400px'*, *height='580px'*)

直接在浏览器中打开分析结果

函数执行逻辑：

1. 首先创建一个 HTML 文件的路径 file_html，这个文件将被保存在用户的主目录下，文件名为”temp_czsc_advanced_trader.html”。
2. 然后，函数调用 self.take_snapshot 方法，将分析结果保存为一个 HTML 文件。

3. 最后，函数使用 webbrowser.open 方法打开这个 HTML 文件

take_snapshot(*file_html=None*, *width: str = '1400px'*, *height: str = '580px'*)

获取快照

函数执行逻辑：

1. 函数首先创建一个 Tab 对象，用于存储所有的图表和表格。
2. 函数遍历所有的 freq，对于每一个 freq，函数获取相应的 CZSC 对象，并将其转换为一个图表，然后添加到 Tab 对象中。
3. 函数提取出所有的信号，并按照 freq 分组。对于每一个 freq，函数创建一个表格，包含该 freq 下的所有信号，然后添加到 Tab 对象中。
4. 如果还有其他的信号，函数创建一个表格，包含所有的其他信号，然后添加到 Tab 对象中。
5. 最后，如果提供了 file_html 参数，函数将 Tab 对象渲染为一个 HTML 文件并保存；否则，函数返回 Tab 对象。

Parameters

- **file_html** –交易快照保存的 html 文件名
- **width** –图表宽度
- **height** –图表高度

Returns

update_signals(*bar:* RawBar)

输入基础周期已完成 K 线，更新信号，更新仓位

函数执行逻辑：

1. 函数首先调用 self.bg.update(bar)，输入一个已完成的基础周期 K 线 bar，更新各周期 K 线。
2. 然后，函数遍历所有的 K 线 freq 和对应的 K 线数据，对每一个 K 线数据，函数调用 self.kas[freq].update(b[-1])，更新对应的 CZSC 对象。
3. 函数提取出 K 线的标的代码 bar.symbol，并将其赋值给 self.symbol。
4. 函数提取出基础 freq 的最后一根 K 线 last_bar，并从中提取出结束时间 dt，K 线 IDid，以及收盘价 close，并将它们分别赋值给 self.end_dt，self.bid，和 self.latest_price。
5. 函数创建一个空的有序字典 s，并调用 self.get_signals_by_conf() 获取所有的信号，然后将这些信号更新到字典 s 中。
6. 最后，函数将 last_bar 的所有属性更新到字典 s 中。

Parameters

bar –基础周期已完成 K 线

Returns

None

CzscTrader

class czsc.traders.**CzscTrader**(*bg: BarGenerator | None = None*, *positions: List[Position] | None = None*, *ensemble_method: AnyStr | Callable = 'mean'*, ***kwargs*)

Bases: *CzscSignals*

缠中说禅技术分析理论之多级别联立交易决策类（支持多策略独立执行）

Attributes Summary

pos_changed	判断仓位是否发生变化

Methods Summary

get_ensemble_pos([method])	获取多个仓位的集成仓位
get_ensemble_weight([method])	获取 CzscTrader 中所有 positions 按照 method 方法集成之后的权重
get_position(name)	获取指定名称的仓位策略对象
on_bar(bar)	输入基础周期已完成 K 线，更新信号，更新仓位
on_sig(sig)	通过信号字典直接交易，用于快速回测场景
take_snapshot([file_html, width, height])	获取快照
update(bar)	输入基础周期已完成 K 线，更新信号，更新仓位
weight_backtest(**kwargs)	执行仓位集成权重的回测

Attributes Documentation

pos_changed

判断仓位是否发生变化

1. 函数首先检查 self.positions 是否为空。如果为空，即没有仓位，函数直接返回 False。
2. **如果 self.positions 不为空，函数遍历所有的仓位，对于每一个仓位，函数检查其 pos_changed 属性。**
   如果任何一个仓位的 pos_changed 属性为 True，即该仓位发生了变化，函数返回 True。

Returns

True/False

Methods Documentation

get_ensemble_pos(*method: AnyStr | Callable | None = None*) → float

获取多个仓位的集成仓位

函数执行逻辑：

1. 函数首先检查 self.positions 是否为空。如果为空，即没有仓位，函数直接返回 0。
2. 如果 self.positions 不为空，函数获取集成方法 method。如果没有传入 method 参数，函数使用 self.__ensemble_method 作为集成方法。
3. **如果 method 是一个字符串，函数将其转换为小写，然后获取所有仓位的仓位序列 pos_seq**。
    1. 如果 method 是” mean”，函数计算 pos_seq 的平均值作为集成仓位。
    2. 如果 method 是” vote”，函数计算 pos_seq 的和的符号作为集成仓位。
    3. 如果 method 是” max”，函数获取 pos_seq 的最大值作为集成仓位。
    4. 如果 method 不是以上任何一个值，函数抛出一个值错误。
4. 如果 method 不是一个字符串，即它是一个回调函数，函数将所有仓位的名称和仓位组成的字典作为参数传入 method，并将返回值作为集成仓位。

Parameters

method –多个仓位集成一个仓位的方法，可选值 mean, vote, max；也可以传入一个回调函数

假设有三个仓位对象，当前仓位分别是 1, 1, -1 mean - 平均仓位，pos = np.mean([1, 1, -1]) = 0.33 vote - 投票表决，pos = 1 max - 取最大，pos = 1

对于传入回调函数的情况，输入是 self.positions

Returns

pos, 集成仓位

get_ensemble_weight(*method: AnyStr | Callable | None = None*)

获取 CzscTrader 中所有 positions 按照 method 方法集成之后的权重

函数执行逻辑：

1. 函数首先接收一个参数 method，这是集成方法，可以是字符串或者一个回调函数。
2. 函数检查是否提供了 method 参数。如果没有提供，函数使用 self.__ensemble_method 作为集成方法；如果提供了，函数使用提供的 method 作为集成方法。
3. 函数调用 get_ensemble_weight 函数，输入 self 和 method，获取所有仓位按照指定方法集成之后的权重。

Parameters

- **method** –str or callable 集成方法，可选值包括：’mean’，‘max’，‘min’，‘vote’ 也可以传入自定义的函数，函数的输入为 dict，key 为 position.name，value 为 position.pos, 样例输入：

  {‘多头策略 A’: 1, ‘多头策略 B’: 1, ‘空头策略 A’: -1}

- **kwargs** –

Returns

pd.DataFrame columns = [‘dt’, ‘symbol’, ‘weight’, ‘price’]

get_position(*name: str*) → *Position* | None

获取指定名称的仓位策略对象

函数执行逻辑：

1. 函数首先接收一个参数 name，这是要查找的仓位名称。
2. 函数检查 self.positions 是否为空。如果为空，即没有仓位，函数直接返回 None。
3. 如果 self.positions 不为空，函数遍历所有的仓位，对于每一个仓位，函数检查其名称是否与输入的名称相同。如果相同，函数返回该仓位。
4. 如果遍历所有的仓位都没有找到与输入名称相同的仓位，函数返回 None。

Parameters

name –仓位名称

Returns

Position

on_bar(*bar:* RawBar) → None

输入基础周期已完成 K 线，更新信号，更新仓位

Parameters

bar –基础周期已完成 K 线

Returns

None

on_sig(*sig: dict*) → None

通过信号字典直接交易，用于快速回测场景

函数执行逻辑：

1. 函数首先接收一个参数 sig，这是一个信号字典，赋值给 self.s。
2. **函数从 sig 中提取出标的代码 symbol，结束时间 dt，K 线 ID id，以及收盘价 close，**并将它们分别赋值给 self.symbol，self.end_dt，self.bid，和 self.latest_price。

4. 如果 self.positions 不为空，即存在持仓策略，函数遍历所有 position，函数调用 position.update(self.s)，更新该仓位的状态

Parameters

sig –信号字典

Returns

None

take_snapshot(*file_html=None*, *width: str = '1400px'*, *height: str = '580px'*)

获取快照

Parameters

- **file_html** –交易快照保存的 html 文件名
- **width** –图表宽度
- **height** –图表高度

Returns

update(*bar:* RawBar) → None

输入基础周期已完成 K 线，更新信号，更新仓位

函数执行逻辑:

1. 函数首先接收一个参数 bar，这是一个已完成的基础周期 K 线。
2. 函数调用 self.update_signals(bar)，输入这个已完成的基础周期 K 线，更新信号。
3. 如果 self.positions 不为空，即存在仓位，函数遍历所有的仓位，对于每一个仓位，函数调用 position.update(self.s)，更新该仓位的状态。

Parameters

bar –基础周期已完成 K 线

Returns

None

weight_backtest(***kwargs*)

执行仓位集成权重的回测

Parameters

kwargs –

- method: str or callable，集成方法，参考 get_ensemble_weight 方法
- digits: int，权重小数点后保留的位数，例如 2 表示保留两位小数
- fee_rate: float，手续费率，例如 0.0002 表示万二

Returns

回测结果

DummyBacktest

class czsc.traders.**DummyBacktest**(*strategy*, *signals_path*, *results_path*, *read_bars*, ***kwargs*)

Bases: object

Methods Summary

execute(symbols[, n_jobs])	回测多个品种
one_pos_stats(pos_name)	分析单个持仓策略的表现
one_symbol_dummy(symbol)	回测单个品种
replay(symbol)	回放单个品种的交易

Methods Documentation

execute(*symbols*, *n_jobs=2*, ***kwargs*)

回测多个品种

Parameters

- **symbols** –品种列表
- **n_jobs** –进程数量，默认为 2 需要注意的是：1. 如果进程数过多，可能会导致内存不足 2. 多进程在 pycharm 的 ipython 中无法使用，需要在命令行中运行
- **kwargs** –

Returns

one_pos_stats(*pos_name*)

分析单个持仓策略的表现

one_symbol_dummy(*symbol*)

回测单个品种

replay(*symbol*)

回放单个品种的交易

ExitsOptimize

class czsc.traders.**ExitsOptimize**(*read_bars: Callable*, ***kwargs*)

Bases: object

基础策略出场优化流程

Methods Summary

execute([n_jobs])	批量优化策略

Methods Documentation

execute(*n_jobs=1*)

批量优化策略

Parameters

n_jobs –进程数量

Returns

OpensOptimize

class czsc.traders.**OpensOptimize**(*read_bars: Callable*, ***kwargs*)

Bases: object

基础策略入场优化流程

Methods Summary

execute([n_jobs])	批量优化策略

Methods Documentation

execute(*n_jobs=1*)

批量优化策略

Parameters

n_jobs –进程数量

Returns

PairsPerformance

class czsc.traders.**PairsPerformance**(*df_pairs: DataFrame*)

Bases: object

交易对效果评估

Attributes Summary

basic_info	写入基础信息

Methods Summary

agg_statistics(col)	按列聚合进行交易对评价
agg_to_excel(file_xlsx)	遍历聚合列，保存结果到 Excel 文件中
get_pairs_statistics(df_pairs)	统计一组交易的基本信息

Attributes Documentation

basic_info

写入基础信息

Methods Documentation

agg_statistics(*col: str*)

按列聚合进行交易对评价

agg_to_excel(*file_xlsx*)

遍历聚合列，保存结果到 Excel 文件中

static get_pairs_statistics(*df_pairs: DataFrame*)

统计一组交易的基本信息

Parameters

df_pairs –

Returns

RedisWeightsClient

class czsc.traders.**RedisWeightsClient**(*strategy_name*, *redis_url=None*, *connection_pool=None*, *send_heartbeat=True*, ***kwargs*)

Bases: object

策略持仓权重收发客户端

Attributes Summary

heartbeat_time	获取策略的最近一次心跳时间
metadata	获取策略元数据
version	

Methods Summary

clear_all([with_human])	删除该策略所有记录
get_all_weights([sdt, edt])	获取所有权重数据
get_hist_weights(symbol, sdt, edt)	获取单个品种的持仓权重历史数据
get_keys(pattern)	获取 redis 中指定 pattern 的 keys
get_last_times([symbols])	获取所有品种上策略最近一次发布信号的时间
get_last_weights([symbols, ignore_zero, lua])	获取最近的持仓权重
get_symbols()	获取策略交易的品种列表
publish(symbol, dt, weight[, price, ref, ...])	发布单个策略持仓权重
publish_dataframe(df[, overwrite, batch_size])	批量发布多个策略信号
register_lua_publish(client)	
set_metadata(base_freq, description, author, ...)	设置策略元数据
update_last(**kwargs)	设置策略最近一次更新时间，以及更新参数【可选】

Attributes Documentation

heartbeat_time

获取策略的最近一次心跳时间

metadata

获取策略元数据

version = 'V240303'

Methods Documentation

clear_all(*with_human=True*)

删除该策略所有记录

get_all_weights(*sdt=None*, *edt=None*, ***kwargs*) → DataFrame

获取所有权重数据

Parameters

- **sdt** –str, 开始时间, eg: 20210924 10:19:00
- **edt** –str, 结束时间, eg: 20220924 10:19:00

Returns

pd.DataFrame

get_hist_weights(*symbol*, *sdt*, *edt*) → DataFrame

获取单个品种的持仓权重历史数据

Parameters

- **symbol** –str, 品种代码
- **sdt** –str, 开始时间, eg: 20210924 10:19:00
- **edt** –str, 结束时间, eg: 20220924 10:19:00

Returns

pd.DataFrame

get_keys(*pattern*) → list

获取 redis 中指定 pattern 的 keys

get_last_times(*symbols=None*)

获取所有品种上策略最近一次发布信号的时间

Parameters

symbols –list, 品种列表, 默认为 None, 即获取所有品种

Returns

dict, {symbol: datetime}，如 {‘SFIF9001’: datetime(2021, 9, 24, 15, 19, 0)}

get_last_weights(*symbols=None*, *ignore_zero=True*, *lua=True*)

获取最近的持仓权重

Parameters

- **symbols** –list, 品种列表
- **ignore_zero** –boolean, 是否忽略权重为 0 的品种
- **lua** –boolean, 是否使用 lua 脚本获取，默认为 True 如果要全量获取，推荐使用 lua 脚本，速度更快；如果要获取指定 symbols，不推荐使用 lua 脚本。

Returns

pd.DataFrame

get_symbols()

获取策略交易的品种列表

publish(*symbol*, *dt*, *weight*, *price=0*, *ref=None*, *overwrite=False*)

发布单个策略持仓权重

Parameters

- **symbol** –str, eg; SFIF9001
- **dt** –py_datetime or pandas Timestamp
- **weight** –float, 信号值
- **price** –float, 产生信号时的价格
- **ref** –dict, 自定义数据
- **overwrite** –boolean, 是否覆盖已有记录

Returns

成功发布信号的条数

publish_dataframe(*df*, *overwrite=False*, *batch_size=10000*)

批量发布多个策略信号

Parameters

- **df** –pandas.DataFrame, 必需包含 [‘symbol’, ‘dt’, ‘weight’] 列, 可选 [‘price’, ‘ref’] 列, 如没有 price 则写 0, dtype 同 publish 方法
- **overwrite** –boolean, 是否覆盖已有记录

Returns

成功发布信号的条数

static register_lua_publish(*client*)

set_metadata(*base_freq*, *description*, *author*, *outsample_sdt*, ***kwargs*)

设置策略元数据

update_last(***kwargs*)

设置策略最近一次更新时间，以及更新参数【可选】

SignalsParser

class czsc.traders.**SignalsParser**(*signals_module: str = 'czsc.signals'*)

Bases: object

解析一串信号，生成信号函数配置

Methods Summary

config_to_keys(config)	将信号函数配置转换为信号 key 列表
get_function_name(signal)	获取信号对应的信号函数名称
parse(signal_seq)	解析信号序列
parse_params(name, signal)	获取信号函数参数

Methods Documentation

config_to_keys(*config: List[Dict]*)

将信号函数配置转换为信号 key 列表

函数执行逻辑：

1. 首先创建了一个空列表 keys 用于存储信号 key。
2. **对于传入的 config 列表中的每个配置字典 conf 进行以下操作：**
   - 获取信号函数的名称。
   - 如果该信号函数的名称在 self.sig_pats_map 中存在对应的模板，使用参数填充模板，并将结果添加到 keys 列表中。

Parameters

config–信号函数配置

config = [{‘freq’：‘日线’，‘max_overlap’：‘3’，‘name’：‘czsc.signals.cxt_bi_end_V230222’},
{‘freq1’：‘日线’，‘freq2’：‘60分钟’，‘name’：‘czsc.signals.cxt_zhong_shu_gong_zhen_V221221’}]

Returns

信号 key 列表

get_function_name(*signal: str*)

获取信号对应的信号函数名称

函数执行逻辑:

1. 创建一个 _signal 对象，通过传入的信号字符串进行初始化。
2. 通过遍历 sig_name_map 中的项目，找出那些与 _signal.k3 相匹配的键，并将它们存储在 _k3_match 列表中。
3. 如果只有一个匹配项，则返回该项；否则记录错误日志并返回 None。

Parameters

signal –信号，数据样例：15 分钟 _D1K_ 量柱 V221218_ 低量柱 _6K_ 任意 _0

Returns

信号函数名称

parse(*signal_seq: List[str]*)

解析信号序列

函数执行逻辑:

1. 接受一个 signal_seq 参数。
2. 定义一个空列表 res ，用于存储解析结果。
3. 遍历信号序列 signal_seq 中的每一个信号:
   - 调用 get_function_name 方法，以信号为参数，获取该信号对应的函数名。
   - 进行函数名存在性判断，name 在 sig_pats_map 中存在，调用 parse_params 方法，以函数名和信号为参数，解析参数并返回结果。

Parameters

signal_seq –信号序列, 样例：[‘15 分钟 _D1K_ 量柱 V221218_ 低量柱 _6K_ 任意 _0’，‘日线 _D1K_ 量柱 V221218_ 低量柱 _6K_ 任意 _0’]

Returns

信号函数配置

parse_params(*name*, *signal*)

获取信号函数参数

函数执行逻辑:

1. 首先根据传入的 name 和 signal 参数，通过 Signal(signal).key 获取一个键值。

2. 然后从实例变量 sig_pats_map 中获取与指定名称对应的参数模板，并将其存储在 pats 中。
3. 如果没有找到参数模板，则返回 None。
4. 最后将信号函数的完整名称存储在参数字典中，并返回参数字典。

Parameters

- **name** –信号函数名称, 如：cxt_bi_end_V230222
- **signal** –需要解析的信号, 如：15 分钟 _D1K_ 量柱 V221218_ 低量柱 _6K_ 任意 _0

Returns

WeightBacktest

class czsc.traders.**WeightBacktest**(*dfw*, *digits=2*, ***kwargs*)

Bases: object

持仓权重回测

飞书文档：https://s0cqcxuy3p.feishu.cn/wiki/Pf1fw1woQi4iJikbKJmcYToznxb

Attributes Summary

daily_return	品种等权费后日收益率
stats	回测绩效评价
version	

Methods Summary

backtest([n_jobs])	回测所有合约的收益率
get_symbol_daily(symbol)	获取某个合约的每日收益率
get_symbol_pairs(symbol)	获取某个合约的开平交易记录
process_symbol(symbol)	处理某个合约的回测数据
report(res_path)	回测报告

Attributes Documentation

daily_return

品种等权费后日收益率

stats

回测绩效评价

version = 'V231126'

Methods Documentation

backtest(*n_jobs=1*)

回测所有合约的收益率

函数计算逻辑：

1. 获取数据：遍历所有合约，调用 get_symbol_daily 方法获取每个合约的日收益，调用 get_symbol_pairs 方法获取每个合约的交易流水。
2. **数据处理：将每个合约的日收益合并为一个 DataFrame，使用 pd.pivot_table 方法将数据重塑为以日期为索引、合约为列、**
   收益率为值的表格，并将缺失值填充为 0。计算所有合约收益率的平均值，并将该列添加到 DataFrame 中。将结果存储在 res 字典中，键为合约名，值为包含日行情数据和交易对数据的字典。
3. **绩效评价：计算回测结果的开始日期和结束日期，调用 daily_performance 方法评估总收益率的绩效指标。将每个合约的交易对数据**
   合并为一个 DataFrame，调用 evaluate_pairs 方法评估交易对的绩效指标。将结果存储在 stats 字典中，并更新到绩效评价的字典中。
4. 返回结果：将合约的等权日收益数据和绩效评价结果存储在 res 字典中，并将该字典作为函数的返回结果。

get_symbol_daily(*symbol*)

获取某个合约的每日收益率

函数计算逻辑：

1. 从实例变量 self.dfw 中筛选出交易标的为 symbol 的数据，并复制到新的 DataFrame dfs。
2. 计算每条数据的收益（edge）：权重乘以下一条数据的价格除以当前价格减 1。
3. 计算每条数据的手续费（cost）：当前权重与前一条数据权重之差的绝对值乘以实例变量 self.fee_rate。
4. 计算每条数据扣除手续费后的收益（edge_post_fee）：收益减去手续费。
5. 根据日期进行分组，并对每组进行求和操作，得到每日的总收益、总扣除手续费后的收益和总手续费。

6. 重置索引，并将交易标的符号添加到 DataFrame 中。
7. 重命名列名，将’edge_post_fee’列改为’return’，将’dt’列改为’date’。
8. 选择需要的列，并返回包含日期、交易标的、收益、扣除手续费后的收益和手续费的 DataFrame。

Parameters

symbol –str，合约代码

Returns

pd.DataFrame，品种每日收益率，

columns = [‘date’，‘symbol’，‘edge’，‘return’，‘cost’] 其中

date 为交易日，symbol 为合约代码，edge 为每日收益率，return 为每日收益率减去交易成本后的真实收益，cost 为交易成本

数据样例如下：

date	symbol	edge	return	cost
2019-01-02	DLi9001	0.00230261	0.00195919	0.00085
2019-01-03	DLi9001	0.00425589	0.00310589	0.00115
2019-01-04	DLi9001	-0.0014209	-0.0024709	0.00105
2019-01-07	DLi9001	0.000988305	-0.000111695	0.0011
2019-01-08	DLi9001	-0.0004743	-0.0016243	0.00115

get_symbol_pairs(*symbol*)

获取某个合约的开平交易记录

函数计算逻辑：

1. 从实例变量 self.dfw 中筛选出交易标的为 symbol 的数据，并复制到新的 DataFrame dfs。
2. 将权重乘以 10 的 self.digits 次方，并转换为整数类型，作为 volume 列的值。
3. 生成 bar_id 列，从 1 开始递增，与行数对应。
4. 创建一个空列表 operates，用于存储开平仓交易记录。
5. 定义内部函数 __add_operate，用于向 operates 列表中添加开平仓交易记录。函数接受日期 dt、bar_id、交易量 volume、价格 price 和操作类型 operate 作为参数。函数根据交易量的绝对值循环添加交易记录到 operates 列表中。
6. 将 dfs 转换为字典列表 rows。
7. 处理第一个行记录。- 如果 volume 大于 0，则调用 __add_operate 函数添加”开多”操作的交易记录。- 如果 volume 小于 0，则调用 __add_operate 函数添加”开空”操作的交易记录。

8. 处理后续的行记录。- 使用 zip 函数遍历 rows[:-1] 和 rows[1:]，同时获取当前行 row1 和下一行 row2。- 根据 volume 的正负和变化情况，调用 __add_operate 函数添加对应的开平仓交易记录。
9. 创建空列表 pairs 和 opens，用于存储交易对和开仓记录。
10. 遍历 operates 列表中的交易记录。- 如果操作类型为”开多”或”开空”，将交易记录添加到 opens 列表中，并继续下一次循环。- 如果操作类型为”平多”或”平空”，将对应的开仓记录从 opens 列表中弹出。

    根据开仓和平仓的价格计算盈亏比例，并创建一个交易对字典，将其添加到 pairs 列表中。

11. 将 pairs 列表转换为 DataFrame，并返回包含交易标的的开平仓交易记录的 DataFrame。

process_symbol(*symbol*)

处理某个合约的回测数据

report(*res_path*)

回测报告

2.7 czsc.utils Package

2.7.1 Functions

cal_trade_price(bars[, decimals])	计算给定品种基础周期 K 线数据的交易价格
check_freq_and_market(time_seq[, freq])	检查时间序列是否为同一周期，是否为同一市场
check_gap_info(bars)	检查 bars 中的缺口信息
check_pressure_support(bars[, q_seq])	检查 bars 中的支撑、压力信息
clear_cache([path, subs, recreate])	清空缓存文件夹
count_last_same(seq)	统计与 seq 列表最后一个元素相似的连续元素数量
create_grid_params([prefix, multiply])	创建 grid search 参数组合
create_single_signal(**kwargs)	创建单个信号
cross_sectional_ic(df[, x_col, y_col, method])	分析 df 中 x_col 和 y_col 列的截面相关性（IC）
cross_sectional_ranker(df, x_cols, y_col, ...)	截面打分排序
daily_performance(daily_returns)	采用单利计算日收益数据的各项指标
dill_dump(data, file)	
dill_load(file)	
disk_cache([path, suffix, ttl])	缓存装饰器，支持多种数据格式

continues on next page

Table 4 – continued from previous page

empty_cache_path()	
fast_slow_cross(fast, slow)	计算 fast 和 slow 的交叉信息
format_standard_kline(df, freq)	格式化标准 K 线数据为 CZSC 标准数据结构 RawBar 列表
freq_end_time(dt, freq[, market])	A 股与期货市场精确的获取 dt 对应的 K 线周期结束时间
freqs_sorted(freqs)	K 线周期列表排序并去重，第一个元素是基础周期
get_dir_size(path)	获取目录大小，单位：Bytes
get_intraday_times([freq, market])	获取指定市场的交易时间段
get_py_namespace(file_py[, keys])	获取 python 脚本文件中的 namespace
get_sub_elements(elements[, di, n])	获取截止到倒数第 di 个元素的前 n 个元素
get_url_token(url)	获取指定 URL 数据接口的凭证码
heat_map(data[, x_label, y_label, title, ...])	绘制热力图
holds_performance(df, **kwargs)	组合持仓权重表现
import_by_name(name)	通过字符串导入模块、类、函数
index_composition(klines[, weights, base_point])	设置基点，按收益率加权合成指数
is_bis_down(bis)	判断 bis 中的连续笔是否是向下的
is_bis_up(bis)	判断 bis 中的连续笔是否是向上的
is_symmetry_zs(bis[, th])	对称中枢判断：中枢中所有笔的力度序列，标准差小于均值的一定比例
is_trading_time([dt, market])	判断指定时间是否是交易时间
kline_pro(kline[, fx, bi, xd, bs, title, ...])	绘制缠中说禅 K 线分析结果
net_value_stats(nv[, exclude_zero, sub_cost])	统计净值曲线的年化收益、夏普等
nmi_matrix(df[, heatmap])	计算高维标准化互信息并以矩阵形式输出
optuna_good_params(study[, keep])	获取 optuna 优化结果中的最优参数
optuna_study(objective[, direction, n_trials])	使用 optuna 进行参数优化
overlap(df, col, **kwargs)	给定 df 和 col，计算 col 中相同值的连续出现次数
print_df_sample(df[, n])	
psi(df, factor, segment, **kwargs)	PSI 群体稳定性指标，反映数据在不同分箱中的分布变化
read_json(file)	
resample_bars(df, target_freq[, raw_bars])	将给定的 K 线数据重新采样为目标周期的 K 线数据
resample_to_daily(df[, sdt, edt, ...])	将非日线数据转换为日线数据，以便进行日线级别的分析
risk_free_returns([start_date, end_date, ...])	创建无风险收益率序列
rolling_daily_performance(df, ret_col[, ...])	计算滚动日收益

continues on next page

Table 4 – continued from previous page

same_dir_counts(seq)	计算 seq 中与最后一个数字同向的数字数量
save_json(data, file)	
set_url_token(token, url)	设置指定 URL 数据接口的凭证码，通常一台机器只需要设置一次即可
single_linear(y[, x])	单变量线性拟合
subtract_fee(df[, fee])	依据单品种持仓信号扣除手续费
top_drawdowns(returns[, top])	分析最大回撤，返回最大回撤的波峰、波谷、恢复日期、回撤天数、恢复天数
update_bbars(da[, price_col, numbers])	在给定的 da 数据上计算并添加前面 n 根 bar 的累计收益列
update_nxb(df, **kwargs)	在给定的 df 上计算并添加后面 n 根 bar 的累计收益列
update_tbars(da, event_col)	计算带 Event 方向信息的未来收益
weekly_performance(weekly_returns)	采用单利计算周收益数据的各项指标
x_round(x[, digit])	用去尾法截断小数

cal_trade_price

czsc.utils.**cal_trade_price**(*bars: List[RawBar] | DataFrame*, *decimals=3*, ***kwargs*)

计算给定品种基础周期 K 线数据的交易价格

函数执行逻辑：

1. 首先，根据输入的 bars 参数类型（列表或 DataFrame），将其转换为 DataFrame 格式，并将其存储在变量 df 中。
2. 计算下一根 K 线的开盘价和收盘价，分别存储在新列 next_open 和 next_close 中。同时，将这两个新列名添加到 price_cols 列表中。
3. 计算 TWAP（时间加权平均价格）和 VWAP（成交量加权平均价格）。为此，函数使用了一个 for 循环，遍历 t_seq 参数（默认值为 (5, 10, 15, 20, 30, 60)）。在每次循环中：
   - 计算 TWAP：使用 rolling(t).mean().shift(-t) 方法计算时间窗口为 t 的滚动平均收盘价。
   - 计算 VWAP：首先计算滚动窗口内的成交量之和（sum_vol_t）和成交量乘以收盘价之和（sum_vcp_t），然后用后者除以前者，并向下移动 t 个单位。
   - 将 TWAP 和 VWAP 的列名添加到 price_cols 列表中。
4. 遍历 price_cols 列表中的每个列，将其中的 NaN 值替换为对应行的收盘价。
5. 从 DataFrame 中选择所需的列（包括基本的 K 线数据列和新计算的交易价格列），并使用 round(decimals) 方法保留指定的小数位数（默认为 3）。
6. 返回处理后的 DataFrame。

Parameters

- **bars** –基础周期 K 线数据，一般是 1 分钟周期的 K 线
- **decimals** –保留小数位数，默认值 3

Returns

交易价格表

check_freq_and_market

czsc.utils.**check_freq_and_market**(*time_seq: List*, *freq: AnyStr | None = None*)

检查时间序列是否为同一周期，是否为同一市场

函数计算逻辑：

1. 如果 `freq`在特定列表中，函数直接返回 `freq`和”默认”作为市场类型。
2. 如果 `freq`是’1 分钟’，函数会添加额外的时间点到 `time_seq`中。
3. 函数去除 `time_seq`中的重复时间点，并确保其长度至少为 2。
4. 函数遍历 `freq_market_times`字典，寻找与 `time_seq`匹配的项，并返回对应的 `freq_x`和 `market`。
5. 如果没有找到匹配的项，函数返回 None 和”默认”。

Parameters

- **time_seq** –时间序列，如 [‘11:00’，‘15:00’，‘23:00’，‘01:00’，‘02:30’]
- **freq** –时间序列对应的 K 线周期，可选参数，使用该参数可以加快检查速度。可选值：1 分钟、5 分钟、15 分钟、30 分钟、60 分钟、日线、周线、月线、季线、年线

Returns

- freq K 线周期
- market 交易市场

check_gap_info

czsc.utils.**check_gap_info**(*bars: List[*RawBar*]*)

检查 bars 中的缺口信息

Parameters

bars –K 线序列，按时间升序

Returns

check_pressure_support

czsc.utils.**check_pressure_support**(*bars: List[RawBar]*, *q_seq: List[float] | None = None*) → Dict

检查 bars 中的支撑、压力信息

1. 通过 round 函数对 K 线价格序列进行近似，统计价格出现次数，取出现次数超过 5 次的价位
2. 在出现次数最多的价格序列上计算分位数序列作为关键价格序列

Parameters

- **bars** –K 线序列，按时间升序
- **q_seq** –分位数序列

Returns

clear_cache

czsc.utils.**clear_cache**(*path: AnyStr | Path = PosixPath('/home/docs/.czsc')*, *subs=None*, *recreate=False*)

清空缓存文件夹

Parameters

- **path** –缓存文件夹路径
- **subs** –需要清空的子文件夹名称，如果为 None，则清空整个文件夹
- **recreate** –是否重新创建文件夹, True 时会重新创建文件夹, False 时不会重新创建文件夹

count_last_same

czsc.utils.**count_last_same**(*seq: List | array | Tuple*)

统计与 seq 列表最后一个元素相似的连续元素数量

Parameters

seq –数字序列

Returns

create_grid_params

czsc.utils.**create_grid_params**(*prefix: str = ''*, *multiply=3*, ***kwargs*) → dict

创建 grid search 参数组合

Parameters

- **prefix** –参数组前缀
- **multiply** –参数组合的位数，如果为 0，则使用 # 分隔参数

- **kwargs –任意参数的候选序列，参数值推荐使用 iterable

Returns

参数组合字典

Examples

>>>x = create_grid_params(“test”, x=(1, 2), y=(‘a’, ‘b’), detail=True) >>>print(x) Out[0]:

{‘test_x=1_y=a’: {‘x’: 1, ‘y’: ‘a’},

‘test_x=1_y=b’: {‘x’: 1, ‘y’: ‘b’}, ‘test_x=2_y=a’: {‘x’: 2, ‘y’: ‘a’},

‘test_x=2_y=b’: {‘x’: 2, ‘y’: ‘b’}}

单个参数传入单个值也是可以的，但类型必须是 int, float, str 中的任一 >>>x = create_grid_params(“test”, x=2, y=(‘a’, ‘b’), detail=False) >>>print(x) Out[1]:

{‘test001’: {‘x’: 2, ‘y’: ‘a’},

‘test002’: {‘x’: 2, ‘y’: ‘b’}}

create_single_signal

czsc.utils.**create_single_signal**(**kwargs) → OrderedDict

创建单个信号

cross_sectional_ic

czsc.utils.**cross_sectional_ic**(*df*, *x_col='open'*, *y_col='n1b'*, *method='spearman'*, ***kwargs*)

分析 df 中 x_col 和 y_col 列的截面相关性（IC）

:param df：数据，DateFrame 格式:param x_col：X 列:param y_col：Y 列，一般采用下期收益，也就是 n1b :param method：{‘pearson’, ‘kendall’, ‘spearman’} or callable

- pearson : standard correlation coefficient
- kendall : Kendall Tau correlation coefficient
- spearman : Spearman rank correlation
- callable: callable with input two 1d ndarrays and returning a float

Return：df，res

前者是每日相关系数结果，后者是每日相关系数的统计结果

cross_sectional_ranker

czsc.utils.**cross_sectional_ranker**(*df*, *x_cols*, *y_col*, ***kwargs*)

截面打分排序

Parameters

- **df** –因子数据，必须包含日期、品种、因子值、预测列，且按日期升序排列，样例数据如下：
- **x_cols** –因子列名
- **y_col** –预测列名
- **kwargs** –其他参数
  - model_params: dict, 模型参数，默认 {‘n_estimators’: 40, ‘learning_rate’: 0.01}，可调整，参考 lightgbm 文档
  - n_splits: int, 时间拆分次数，默认 5，即 5 段时间
  - rank_ascending: bool, 打分排序是否升序，默认 False-降序
  - copy: bool, 是否拷贝 df，True-拷贝，False-不拷贝

Returns

df, 包含预测分数和排序列

daily_performance

czsc.utils.**daily_performance**(*daily_returns*)

采用单利计算日收益数据的各项指标

函数计算逻辑：

1. 首先，将传入的日收益率数据转换为 NumPy 数组，并指定数据类型为 float64。
2. 然后，进行一系列判断：如果日收益率数据为空或标准差为零或全部为零，则返回一个字典，其中所有指标的值都为零。
3. 如果日收益率数据满足要求，则进行具体的指标计算：
   - 年化收益率 = 日收益率列表的和 / 日收益率列表的长度 * 252
   - 夏普比率 = 日收益率的均值 / 日收益率的标准差 * 标准差的根号 252
   - 最大回撤 = 累计日收益率的最高累积值 - 累计日收益率
   - 卡玛比率 = 年化收益率 / 最大回撤（如果最大回撤不为零，则除以最大回撤；否则为 10）
   - 日胜率 = 大于零的日收益率的个数 / 日收益率的总个数
   - 年化波动率 = 日收益率的标准差 * 标准差的根号 252
   - 非零覆盖 = 非零的日收益率个数 / 日收益率的总个数
4. 将所有指标的值存储在一个字典中，其中键为指标名称，值为相应的计算结果。

Parameters

daily_returns –日收益率数据，样例：[0.01, 0.02, -0.01, 0.03, 0.02, -0.02, 0.01, -0.01, 0.02, 0.01]

Returns

dict

dill_dump

czsc.utils.**dill_dump**(*data*, *file*)

dill_load

czsc.utils.**dill_load**(*file*)

disk_cache

czsc.utils.**disk_cache**(*path: AnyStr | Path = PosixPath('/home/docs/.czsc')*, *suffix: str = 'pkl'*, *ttl: int = -1*)

缓存装饰器，支持多种数据格式

Parameters

- **path** –缓存文件夹父路径，默认为 home_path，每个函数的缓存文件夹为 path/func_name
- **suffix** –缓存文件后缀，支持 pkl, json, txt, csv, xlsx, feather, parquet
- **ttl** –缓存文件有效期，单位：秒

empty_cache_path

czsc.utils.**empty_cache_path**()

fast_slow_cross

czsc.utils.**fast_slow_cross**(*fast*, *slow*)

计算 fast 和 slow 的交叉信息

Parameters

- **fast** –快线
- **slow** –慢线

Returns

format_standard_kline

czsc.utils.**format_standard_kline**(*df: DataFrame*, *freq: str*)

格式化标准 K 线数据为 CZSC 标准数据结构 RawBar 列表

Parameters

- **df** –标准 K 线数据，DataFrame 结构

dt	symbol	open	close	high	low	vol	amount
2023-11-17 00:00:00	689009.SI	33.52	33.41	33.69	33.38	1.97575e+0	6.61661e+07
2023-11-20 00:00:00	689009.SI	33.4	32.91	33.45	32.25	5.15016e+0	1.68867e+08

- **freq** –K 线级别

Returns

list of RawBar

freq_end_time

czsc.utils.**freq_end_time**(*dt: datetime*, *freq:* Freq | *AnyStr*, *market='A 股'*) → datetime

A 股与期货市场精确的获取 dt 对应的 K 线周期结束时间

Parameters

- **dt** –datetime
- **freq** –Freq

Returns

datetime

freqs_sorted

czsc.utils.**freqs_sorted**(*freqs*)

K 线周期列表排序并去重，第一个元素是基础周期

Parameters

freqs –K 线周期列表

Returns

K 线周期排序列表

get_dir_size

czsc.utils.**get_dir_size**(*path*)

获取目录大小，单位：Bytes

get_intraday_times

czsc.utils.**get_intraday_times**(*freq='1 分钟'*, *market='A 股'*)

获取指定市场的交易时间段

Parameters

market –市场名称，可选值：A 股、期货、默认

Returns

交易时间段列表

get_py_namespace

czsc.utils.**get_py_namespace**(*file_py: str*, *keys: list = []*) → dict

获取 python 脚本文件中的 namespace

Parameters

- **file_py** –python 脚本文件名
- **keys** –指定需要的对象名称

Returns

namespace

get_sub_elements

czsc.utils.**get_sub_elements**(*elements: List[Any]*, *di: int = 1*, *n: int = 10*) → List[Any]

获取截止到倒数第 di 个元素的前 n 个元素

Parameters

- **elements** –全部元素列表
- **di** –指定结束元素为倒数第 di 个
- **n** –指定需要的元素个数

Returns

部分元素列表

>>>x = [1, 2, 3, 4, 5, 6, 7, 8, 9] >>>y1 = get_sub_elements(x, di=1, n=3) >>>y2 = get_sub_elements(x, di=2, n=3)

get_url_token

czsc.utils.**get_url_token**(*url*)

获取指定 URL 数据接口的凭证码

heat_map

czsc.utils.**heat_map**(*data: List[dict]*, *x_label: List[str] | None = None*, *y_label: List[str] | None = None*, *title: str = '热力图'*, *width: str = '900px'*, *height: str = '680px'*) → HeatMap

绘制热力图

Parameters

- **data** –用于绘制热力图的数据，示例如下 [{‘x’：‘0hour’，‘y’：‘0day’，‘heat’：11},

  {‘x’：‘0hour’，‘y’：‘1day’，‘heat’：40}, {‘x’：‘0hour’，‘y’：‘2day’，‘heat’：38}, {‘x’：‘0hour’，‘y’：‘3day’，‘heat’：36}, {‘x’：‘0hour’，‘y’：‘4day’，‘heat’：11}]

- **x_label** –x 轴标签
- **y_label** –y 轴标签
- **title** –图表标题
- **width** –图表宽度
- **height** –图表高度

Returns

图表

holds_performance

czsc.utils.**holds_performance**(*df*, ***kwargs*)

组合持仓权重表现

Parameters

- **df** –pd.DataFrame, columns=[‘dt’，‘symbol’，‘weight’，‘n1b’] 数据说明，dt: 交易时间，symbol: 标的代码，weight: 权重，n1b: 名义收益率必须是每个时间点都有数据，如果某个时间点没有数据，可以增加一行数据，权重为 0
- **kwargs** –
  - fee: float, 单边费率，BP
  - digits: int, 保留小数位数

Returns

pd.DataFrame, columns=[‘date’, ‘change’, ‘edge_pre_fee’, ‘cost’, ‘edge_post_fee’]

import_by_name

czsc.utils.**import_by_name**(*name*)

通过字符串导入模块、类、函数

函数执行逻辑：

1. 检查 name 中是否包含点号（’.’）。如果没有，则直接使用内置的 import 函数来导入整个模块，并返回该模块对象。
2. **如果 name 包含点号，先处理一个相对路径。将 name 拆分为两部分：module_name 和 function_name**。
   使用 Python 内置的 rsplit 方法从右边开始分割，只取一次，这样可以确保我们将最后的一个点号前的部分作为 module_name，点号后面的部分作为 function_name。
3. **使用 import 函数导入指定的 module_name**。
   这里传入三个参数：globals() 和 locals() 分别代表当前全局和局部命名空间；[function_name] 是一个列表，用于指定要导入的子模块或属性名。这样做是为了避免一次性导入整个模块的所有内容，提高效率。
4. 使用 vars 函数获取模块的字典表示形式（即模块内所有的变量和函数），取出 function_name 对应的值，然后返回这个值。

Parameters

name –模块名，如：’czsc.objects.Factor’

Returns

模块对象

index_composition

czsc.utils.**index_composition**(*klines*, *weights=None*, *base_point=1000*, ***kwagrs*)

设置基点，按收益率加权合成指数

输入：

1. 成分标的 K 线行情
2. 每个时刻的成分权重
3. 基点

过程：

1. 计算成分标的每根 K 线的收益
2. 在每个时刻，按成分权重对标的收益加权计算，得到每个时刻的指数涨跌幅
3. 用基点和每个时刻的涨跌幅计算出每个时刻的指数价

输出：指数 K 线行情

Parameters

- **klines** –K 线行情，样例如下：

symbol	dt	open	close	high	low	vol	amount
000001.S	2021-01-04 09:31:00	3473.8	3469.3	3474.3	3468.8	10410142	13018854400
000001.S	2021-01-04 09:32:00	3470.1	3467.5	3471.9	3467.5	57036723	7721827328
000001.S	2021-01-04 09:33:00	3467.1	3466.9	3468.5	3466.9	66006048	8371049984
000001.S	2021-01-04 09:34:00	3466.8	3463.5	3466.8	3463.4	56393139	7435783168
000001.S	2021-01-04 09:35:00	3463.2	3460.3	3463.7	3460.3	50427190	6500695552

- **weights** –权重调整记录；索引为 dt，columns 为成分权重，每个时间截面的权重之和可以不为 1，样例如下：

dt	000001.SH	000016.SH	000300.SH	000905.SH
2021-01-04 09:31:00	0.244275	0.236822	0.271415	0.204674
2021-01-04 10:10:00	0.250273	0.140398	0.151955	0.294418
2021-01-04 10:54:00	0.127531	0.123941	0.199969	0.19682

注意：权重调整记录的时间截面必须是 K 线行情的时间截面的子集，且必须包含 K 线行情的第一根 K 线的时间截面

- **base_point** –基点，默认为 1000

Returns

指数 K 线行情，样例如下：

dt	returns	price	vol	amount
2021-01-04 09:31:00	0	1000	2195268112	31725149952
2021-01-04 09:32:00	-0.000230561	999.769	1187524832	18900989440
2021-01-04 09:33:00	-0.000165153	999.604	1330775568	19418287360
2021-01-04 09:34:00	-0.00103582	998.569	1133046300	17488768512
2021-01-04 09:35:00	-0.00067244	997.897	1025222108	15351070080

is_bis_down

czsc.utils.**is_bis_down**(*bis: List[BI]*)

判断 bis 中的连续笔是否是向下的

is_bis_up

czsc.utils.**is_bis_up**(*bis: List[BI]*)

判断 bis 中的连续笔是否是向上的

is_symmetry_zs

czsc.utils.**is_symmetry_zs**(*bis: List[BI]*, *th: float = 0.3*) → bool

对称中枢判断：中枢中所有笔的力度序列，标准差小于均值的一定比例

https://pic2.zhimg.com/80/v2-2f55ef49eda01972462531ebb6de4f19_1440w.jpg

Parameters

- **bis** –构成中枢的笔序列
- **th** –标准差小于均值的比例阈值

Returns

is_trading_time

czsc.utils.**is_trading_time**(*dt: datetime = datetime.datetime(2024, 5, 12, 11, 59, 22, 586058)*, *market='A 股'*)

判断指定时间是否是交易时间

kline_pro

czsc.utils.**kline_pro**(*kline: List[dict]*, *fx: List[dict] = []*, *bi: List[dict] = []*, *xd: List[dict] = []*, *bs: List[dict] = []*, *title: str = '缠中说禅 K 线分析'*, *t_seq: List[int] = []*, *width: str = '1400px'*, *height: str = '580px'*) → Grid

绘制缠中说禅 K 线分析结果

Parameters

- **kline** –K 线
- **fx** –分型识别结果
- **bi** –笔识别结果 {‘dt’: Timestamp(‘2020-11-26 00:00:00’),

'fx_mark': 'd', 'start_dt': Timestamp('2020-11-25 00:00:00'), 'end_dt': Timestamp('2020-11-27 00:00:00'), 'fx_high': 144.87, 'fx_low': 138.0, 'bi': 138.0}

- **xd** –线段识别结果
- **bs** –买卖点
- **title** –图表标题
- **t_seq** –均线系统
- **width** –图表宽度
- **height** –图表高度

Returns

用 Grid 组合好的图表

net_value_stats

czsc.utils.**net_value_stats**(*nv: DataFrame*, *exclude_zero: bool = False*, *sub_cost=True*) → dict

统计净值曲线的年化收益、夏普等

Parameters

- **nv** –净值数据，格式如下：

  dt edge cost

  0 2017-01-03 09:30:00 0.0 0.0 1 2017-01-03 10:00:00 0.0 0.0 2 2017-01-03 10:30:00 0.0 0.0 3 2017-01-03 11:00:00 0.0 0.0 4 2017-01-03 13:00:00 0.0 0.0

  列说明：dt: 交易时间 edge: 单利收益，单位：BP cost: 交易成本，单位：BP；可选列，如果没有成本列，则默认为 0

- **exclude_zero** –是否排除收益为 0 的情况，一般认为收益为 0 的情况是没有持仓的
- **sub_cost** –是否扣除成本

Returns

nmi_matrix

czsc.utils.**nmi_matrix**(*df: DataFrame*, *heatmap=False*) → DataFrame

计算高维标准化互信息并以矩阵形式输出

Parameters

- **df** –数据
- **heatmap** –是否绘制热力图

Returns

optuna_good_params

czsc.utils.**optuna_good_params**(*study: Study*, *keep=0.2*) → DataFrame

获取 optuna 优化结果中的最优参数

Parameters

- **study** –optuna.study.Study
- **keep** –float, 保留最优参数的比例, 默认 0.2 如果 keep 小于 0，则按比例保留；如果 keep 大于 0，则保留 keep 个参数组

Returns

pd.DataFrame, 最优参数组列表

optuna_study

czsc.utils.**optuna_study**(*objective*, *direction='maximize'*, *n_trials=100*, ***kwargs*)

使用 optuna 进行参数优化

overlap

czsc.utils.**overlap**(*df: DataFrame*, *col: str*, ***kwargs*)

给定 df 和 col，计算 col 中相同值的连续出现次数

Parameters

- **df** –pd.DataFrame, 至少包含 dt、symbol 和 col 列
- **col** –str，需要计算连续出现次数的列名
- **kwargs** –dict，其他参数
  - copy: bool, 是否复制 df，默认为 True
  - new_col: str, 计算结果的列名，默认为 f”{col}_overlap”
  - max_overlap: int, 最大允许连续出现次数，默认为 10

print_df_sample

czsc.utils.**print_df_sample**(*df*, *n=5*)

psi

czsc.utils.**psi**(*df: DataFrame*, *factor*, *segment*, ***kwargs*)

PSI 群体稳定性指标，反映数据在不同分箱中的分布变化

PSI = ∑(实际占比 - 基准占比) * ln(实际占比 / 基准占比)

参考：https://zhuanlan.zhihu.com/p/79682292 风控模型—群体稳定性指标 (PSI) 深入理解应用

Parameters

- **df** –数据, 必须包含 dt 和 col 列
- **factor** –分组因子
- **segment** –样本分组
- **kwargs** –

Returns

pd.DataFrame

read_json

czsc.utils.**read_json**(*file*)

resample_bars

czsc.utils.**resample_bars**(*df: DataFrame*, *target_freq: Freq | AnyStr*, *raw_bars=True*, ***kwargs*)

将给定的 K 线数据重新采样为目标周期的 K 线数据

函数计算逻辑：

1. 确定目标周期 `target_freq`的类型和市场类型。
2. 添加一个新列 `freq_edt`，表示每个数据点对应的目标周期的结束时间。
3. 根据 `freq_edt`对数据进行分组，并对每组数据进行聚合，得到目标周期的 K 线数据。
4. 重置索引，并选择需要的列。
5. 根据 `raw_bars`参数，决定返回的数据类型：如果为 True，转换为 `RawBar`对象；如果为 False，直接返回 DataFrame。
6. 如果 `drop_unfinished`参数为 True，删除最后一根未完成的 K 线。

Parameters

- **df** –

原始 K 线数据，必须包含以下列：symbol, dt, open, close, high, low, vol, amount。
样例如下：
symbol dt open close high low 0 000001.XSHG 2015-01-05 09:31:00 3258.63 3259.69 3262.85 3258.63

1 000001.XSHG 2015-01-05 09:32:00 3258.33 3256.19 3259.55 3256.19 2 000001.XSHG 2015-01-05 09:33:00 3256.10 3257.50 3258.42 3256.10 3 000001.XSHG 2015-01-05 09:34:00 3259.33 3261.76 3261.76 3257.98 4 000001.XSHG 2015-01-05 09:35:00 3261.71 3264.88 3265.48 3261.71

vol amount

0 1333523100 4.346872e+12 1 511386100 1.665170e+12 2 455375200 1.483385e+12 3 363393800 1.185303e+12 4 402854600 1.315272e+12

- **target_freq** –目标周期
- **raw_bars** –是否将转换后的 K 线序列转换为 RawBar 对象
- **kwargs** –
  - base_freq: 基础周期，如果不指定，则根据 df 中的 dt 列自动推断
  - drop_unfinished: 是否删除最后一根未完成的 K 线

Returns

转换后的 K 线序列

resample_to_daily

czsc.utils.**resample_to_daily**(*df: DataFrame*, *sdt=None*, *edt=None*, *only_trade_date=True*)

将非日线数据转换为日线数据，以便进行日线级别的分析

函数执行逻辑：

1. 首先，函数接收一个数据框 `df`，以及可选的开始日期 `sdt`，结束日期 `edt`，和一个布尔值 `only_trade_date`。
2. 函数将 `df`中的 `dt`列转换为日期时间格式。如果没有提供 `sdt`或 `edt`，则使用 `df`中的最小和最大日期作为开始和结束日期。
3. 创建一个日期序列。如果 `only_trade_date`为真，则只包含交易日期；否则，包含 `sdt`和 `edt`之间的所有日期。
4. 使用 `merge_asof`函数，找到每个日期在原始 `df`中对应的最近一个日期。
5. 创建一个映射，将每个日期映射到原始 `df`中的对应行。
6. 对于日期序列中的每个日期，复制映射中对应的数据行，并将日期设置为当前日期。
7. 最后，将所有复制的数据行合并成一个新的数据框，并返回。

Parameters

- **df** –日线以上周期的数据，必须包含 dt 列
- **sdt** –开始日期

- **edt** –结束日期
- **only_trade_date** –是否只保留交易日数据

Returns

pd.DataFrame

risk_free_returns

czsc.utils.**risk_free_returns**(*start_date='20180101'*, *end_date='20210101'*, *year_returns=0.03*)

创建无风险收益率序列

创建一个 Pandas DataFrame，包含两列：”date”和“returns”。”date”列包含从 trade_dates 获取的所有交易日期，“returns”列包含无风险收益率序列，计算方法是将年化收益率（year_returns）除以 252（一年的交易日数量，假设为每周 5 天）

Parameters

- **start_date** –起始日期
- **end_date** –截止日期
- **year_returns** –年化收益率

Returns

pd.DataFrame

rolling_daily_performance

czsc.utils.**rolling_daily_performance**(*df: DataFrame*, *ret_col*, *window=252*, *min_periods=100*, ***kwargs*)

计算滚动日收益

Parameters

- **df** –pd.DataFrame, 日收益数据，columns=[‘dt’, ret_col]
- **ret_col** –str, 收益列名
- **window** –int, 滚动窗口, 自然天数
- **min_periods** –int, 最小样本数
- **kwargs** –

same_dir_counts

czsc.utils.**same_dir_counts**(*seq: [typing.List, <built-in function array>]*)

计算 seq 中与最后一个数字同向的数字数量

Parameters

seq –数字序列

Returns

Example

>>>print(same_dir_counts([-1, -1, -2, -3, 0, 1, 2, 3, -1, -2, 1, 1, 2, 3])) >>>print(same_dir_counts([-1, -1, -2, -3, 0, 1, 2, 3]))

save_json

czsc.utils.**save_json**(*data*, *file*)

set_url_token

czsc.utils.**set_url_token**(*token*, *url*)

设置指定 URL 数据接口的凭证码，通常一台机器只需要设置一次即可

Parameters

- **token** –凭证码
- **url** –数据接口地址

single_linear

czsc.utils.**single_linear**(*y: array | list*, *x: array | list | None = None*) → dict

单变量线性拟合

Parameters

- **y** –目标序列
- **x** –单变量值

Return res

拟合结果，样例如下 {‘slope’：1.565，‘intercept’：67.9783，‘r2’：0.9967}

slope 标识斜率 intercept 截距 r2 拟合优度

subtract_fee

czsc.utils.**subtract_fee**(*df*, *fee=1*)

依据单品种持仓信号扣除手续费

函数执行逻辑：

1. 首先，函数对输入的 df 进行检查，确保其包含所需的列：’dt’（日期时间）和’pos’（持仓）。同时，检查’pos’列的值是否符合要求，即只能是 0、1 或-1。
2. 如果 df 中不包含’n1b’（名义收益率）列，函数会根据’price’列计算’n1b’列。
3. 然后，函数为输入的 DataFrame df 添加一个新列’date’，该列包含交易日期（从’dt’列中提取）。
4. 接下来，函数根据持仓（’pos’）和名义收益率（’n1b’）计算’edge_pre_fee’（手续费前收益）和’edge_post_fee’（手续费后收益）两列。
5. **函数根据持仓信号计算开仓和平仓的位置**。

   开仓位置（open_pos）是持仓信号发生变化的位置（即，当前持仓与前一个持仓不同），并且当前持仓不为 0。平仓位置（exit_pos）是持仓信号发生变化的位置（即，当前持仓与前一个持仓不同），并且前一个持仓不为 0。
6. 根据手续费规则，开仓时在第一个持仓 K 线上扣除手续费，平仓时在最后一个持仓 K 线上扣除手续费。函数通过将’edge_post_fee’列的值在开仓和平仓位置上分别减去手续费（fee）来实现这一逻辑。
7. 最后，函数返回修改后的 DataFrame df。

Parameters

- **df** –包含 dt、pos、price、n1b 列的 DataFrame
- **fee** –手续费，单位：BP

Returns

修改后的 DataFrame

top_drawdowns

czsc.utils.**top_drawdowns**(*returns: Series*, *top: int = 10*) → DataFrame

分析最大回撤，返回最大回撤的波峰、波谷、恢复日期、回撤天数、恢复天数

Parameters

- **returns** –pd.Series, 日收益率序列，index 为日期
- **top** –int, optional, 返回最大回撤的数量，默认 10

Returns

pd.DataFrame

update_bbars

czsc.utils.**update_bbars**(*da*, *price_col='close'*, *numbers=(1, 2, 5, 10, 20, 30)*) → None

在给定的 da 数据上计算并添加前面 n 根 bar 的累计收益列

函数的逻辑如下：

1. 首先，检查 price_col 是否在输入的 DataFrame（da）的列名中。如果不在，抛出 ValueError。
2. 使用 for 循环遍历 numbers 列表中的每个整数 n，对于每个整数 n，计算 n 根 bar 的累计收益。
3. 返回 None，表示这个函数会直接修改输入的 da，而不返回新的 DataFrame。

Parameters

- **da** –K 线数据，DataFrame 结构
- **price_col** –价格列
- **numbers** –考察的 bar 的数目的列表

Returns

bbars_cols: 后面 n 根 bar 的 bp 值列名

update_nxb

czsc.utils.**update_nxb**(*df: DataFrame*, ***kwargs*) → DataFrame

在给定的 df 上计算并添加后面 n 根 bar 的累计收益列

收益计量单位：BP；1 倍涨幅 = 10000BP

Parameters

- **df** –数据，DataFrame 结构，必须包含 dt, symbol, price 列
- **kwargs** –参数字典
  - nseq: 考察的 bar 的数目的列表，默认为 (1, 2, 3, 5, 8, 10, 13)
  - bp: 是否将收益转换为 BP，默认为 False

Returns

pd.DataFrame

update_tbars

czsc.utils.**update_tbars**(*da: DataFrame*, *event_col: str*) → None

计算带 Event 方向信息的未来收益

函数的逻辑如下：

1. 从输入的 da 的列名中提取所有以‘n’开头，以‘b’结尾的列名，这些列名表示未来 n 根 bar 的累计收益。将这些列名存储在 n_seq 列表中。
2. 使用 for 循环遍历 n_seq 列表中的每个整数 n。

3. **对于每个整数 n，计算带有 Event 方向信息的未来收益。**
   计算方法是：将前面 n 根 bar 的累计收益（列名 f’n{n}b’）与事件信号列（event_col）的值相乘。将计算结果存储在一个新的列中，列名为 f’t{n}b’。
4. 返回 None，表示这个函数会直接修改输入的 da，而不返回新的 DataFrame。

Parameters

- **da** –K 线数据，DataFrame 结构
- **event_col** –事件信号列名，含有 0, 1, -1 三种值，0 表示无事件，1 表示看多事件，-1 表示看空事件

Returns

None

weekly_performance

czsc.utils.**weekly_performance**(*weekly_returns*)

采用单利计算周收益数据的各项指标

Parameters

weekly_returns –周收益率数据, 样例: [0.01, 0.02, -0.01, 0.03, 0.02, -0.02, 0.01, -0.01, 0.02, 0.01]

Returns

dict

x_round

czsc.utils.**x_round**(*x: float | int*, *digit: int = 4*) → float | int

用去尾法截断小数

Parameters

- **x** –数字
- **digit** –保留小数位数

Returns

2.7.2 Classes

AliyunOSS(access_key_id, access_key_secret, ...)	
BarGenerator(base_freq, freqs[, max_count, ...])	
CrossSectionalPerformance(dfh, **kwargs)	根据截面持仓信息，计算截面绩效
DataClient([token, url, timeout])	
DiskCache([path])	
KlineChart([n_rows])	K 线绘图工具类
SignalAnalyzer(*args, **kwargs)	
SignalPerformance(*args, **kwargs)	信号表现分析
WordWriter([file_docx])	用 Word 文档记录信息

AliyunOSS

class czsc.utils.**AliyunOSS**(*access_key_id: str*, *access_key_secret: str*, *endpoint: str*, *bucket_name: str*)

Bases: object

Methods Summary

batch_download(oss_keys, local_paths[, ...])	批量从 OSS 下载文件。
batch_upload(filepaths, oss_keys[, replace, ...])	批量上传文件到 OSS。
create_folder(folder_path)	在 OSS 上创建文件夹。
delete_file(oss_key)	从 OSS 删除文件。
download(oss_key, filepath[, replace])	从 OSS 下载文件。
download_folder(oss_folder, local_folder[, ...])	从 OSS 下载指定文件夹。
file_exists(oss_key)	检查文件是否在 OSS 上存在。
get_file_stream(oss_key)	获取 OSS 上文件的数据流。
list_files([prefix, extensions])	列举 OSS 上的文件。
multipart_upload(filepath, oss_key)	分块上传大文件到 OSS。
upload(filepath, oss_key[, replace])	上传文件到 OSS。
upload_folder(local_folder, oss_folder[, ...])	上传本地文件夹到 OSS。

Methods Documentation

batch_download(*oss_keys: List[str]*, *local_paths: List[str]*, *replace: bool = False*, *threads: int = 5*)

批量从 OSS 下载文件。

Parameters

- **oss_keys** –list, 文件在 OSS 上的路径和名称列表。
- **local_paths** –list, 本地存储文件的路径列表。
- **threads** –int, 并行下载的线程数。默认为 5。

batch_upload(*filepaths: List[str]*, *oss_keys: List[str]*, *replace: bool = False*, *threads: int = 5*)

批量上传文件到 OSS。

Parameters

- **filepaths** –list, 本地文件的路径列表。
- **oss_keys** –list, 文件在 OSS 上的路径和名称列表。
- **replace** –boolean, 如果为 True，将覆盖 OSS 上的同名文件。默认为 False。
- **threads** –int, 并行上传的线程数。默认为 5。

create_folder(*folder_path: str*)

在 OSS 上创建文件夹。

Parameters

folder_path –string, 需要创建的文件夹的路径。

delete_file(*oss_key: str*) → bool

从 OSS 删除文件。

Parameters

oss_key –string, 文件在 OSS 上的路径和名称。

Returns

boolean, 如果删除成功，返回 True；否则，返回 False.

download(*oss_key: str*, *filepath: str*, *replace: bool = False*) → bool

从 OSS 下载文件。

Parameters

- **oss_key** –string, 文件在 OSS 上的路径和名称。
- **filepath** –string, 本地存储文件的路径。

Returns

boolean, 如果下载成功，返回 True；否则，返回 False。

download_folder(*oss_folder: str*, *local_folder: str*, *threads: int = 5*)

从 OSS 下载指定文件夹。

Parameters

- **oss_folder** –string, OSS 上的文件夹路径。
- **local_folder** –string, 本地存储文件夹的路径。
- **threads** –int, 并行下载的线程数。默认为 5。

file_exists(*oss_key: str*) → bool

检查文件是否在 OSS 上存在。

Parameters

oss_key –string, 文件在 OSS 上的路径和名称。

Returns

boolean, 如果文件存在，返回 True；否则，返回 False.

get_file_stream(*oss_key: str*) → BytesIO

获取 OSS 上文件的数据流。

Parameters

oss_key –string, 文件在 OSS 上的路径和名称。

Returns

BytesIO, 文件的数据流。

list_files(*prefix=''*, *extensions=None*)

列举 OSS 上的文件。

Parameters

- **prefix** –string, 列举的文件前缀，默认为空。
- **extensions** –list, 需要列举的文件的后缀名，默认为空，表示列举所有文件。

Returns

list, 列举的文件的名称列表。

multipart_upload(*filepath: str*, *oss_key: str*)

分块上传大文件到 OSS。

Parameters

- **filepath** –string, 本地文件的路径。
- **oss_key** –string, 文件在 OSS 上的路径和名称。

upload(*filepath: str*, *oss_key: str*, *replace: bool = False*) → bool

上传文件到 OSS。

Parameters

- **filepath** –string, 本地文件的路径。
- **oss_key** –string, 文件在 OSS 上的路径和名称。
- **replace** –boolean, 如果为 True，将覆盖 OSS 上的同名文件。默认为 False。

Returns

boolean, 如果上传成功，返回 True；否则，返回 False。

upload_folder(*local_folder: str*, *oss_folder: str*, *replace: bool = False*, *threads: int = 5*)

上传本地文件夹到 OSS。

Parameters

- **local_folder** –string, 本地文件夹的路径。
- **oss_folder** –string, OSS 上的目标文件夹路径。
- **replace** –boolean, 如果为 True，将覆盖 OSS 上的同名文件。默认为 False。
- **threads** –int, 并行上传的线程数。默认为 5。

BarGenerator

class czsc.utils.**BarGenerator**(*base_freq: str*, *freqs: List[str]*, *max_count: int = 5000*, *market='默认'*)

Bases: object

Attributes Summary

version

Methods Summary

init_freq_bars(freq, bars)	初始化某个周期的 K 线序列
update(bar)	更新各周期 K 线

Attributes Documentation

version = 'V231008'

Methods Documentation

init_freq_bars(*freq: str*, *bars: List[*RawBar*]*)

初始化某个周期的 K 线序列

函数计算逻辑：

1. 首先，它断言 `freq`必须是 `self.bars`的键之一。如果 `freq`不在 `self.bars`的键中，代码会抛出一个断言错误。
2. 然后，它断言 `self.bars[freq]`必须为空。如果 `self.bars[freq]`不为空，代码会抛出一个断言错误，并显示一条错误消息。
3. 如果以上两个断言都通过，它会将 `bars`赋值给 `self.bars[freq]`，从而初始化指定频率的 K 线序列。
4. 最后，它会将 `bars`列表中的最后一个 `RawBar`对象的 `symbol`属性赋值给 `self.symbol`。

Parameters

- **freq** –周期名称
- **bars** –K 线序列

update(*bar:* RawBar) → None

更新各周期 K 线

函数计算逻辑：

1. 首先，它获取基准频率 `base_freq`，并断言 `bar`的频率值等于 `base_freq`。
2. 然后，它将 `bar`的符号和日期时间设置为 `self.symbol`和 `self.end_dt`。
3. **接下来，它检查是否已经有一个与 `bar`日期时间相同的 K 线存在于 `self.bars[base_freq]`中**。
   如果存在，它会记录一个警告并返回，不进行任何更新。
4. 如果不存在重复的 K 线，它会遍历 `self.bars`的所有键（即所有的频率），并对每个频率调用 `self._update_freq`方法来更新该频率的 K 线。
5. 最后，它会限制在内存中的 K 线数量，确保每个频率的 K 线数量不超过 `self.max_count`。

Parameters

bar –必须是已经结束的 Bar

Returns

None

CrossSectionalPerformance

class czsc.utils.**CrossSectionalPerformance**(*dfh: DataFrame*, ***kwargs*)

Bases: object

根据截面持仓信息，计算截面绩效

Methods Summary

cal_turnover()	计算换手率
cross_net_value([by, values])	计算截面等权净值
report(file_docx)	

Methods Documentation

cal_turnover()

计算换手率

cross_net_value(*by='dt'*, *values='edge'*)

计算截面等权净值

Parameters

- **by** –按什么字段计算截面等权净值，默认按交易时间
- **values** –计算截面等权净值时，使用的字段，默认使用 edge，计算策略收益，可选值：edge, n1b 输入 edge，计算策略收益输入 n1b，计算基准收益

Returns

report(*file_docx*)

DataClient

class czsc.utils.**DataClient**(*token=None*, *url='http://api.tushare.pro'*, *timeout=300*, ***kwargs*)

Bases: object

Methods Summary

clear_cache()	清空缓存
post_request(api_name[, fields])	执行 API 数据查询

Methods Documentation

clear_cache()

清空缓存

post_request(*api_name*, *fields=''*, ***kwargs*)

执行 API 数据查询

Parameters

- **api_name** –str, 查询接口名称
- **fields** –str, 查询字段
- **kwargs** –dict, 查询参数
  - ttl: int, 缓存有效期，单位秒，-1 表示不过期

Returns

pd.DataFrame

DiskCache

class czsc.utils.**DiskCache**(*path=None*)

Bases: object

Methods Summary

get(k[, suffix])	读取缓存文件
is_found(k[, suffix, ttl])	判断缓存文件是否存在
remove(k[, suffix])	
set(k, v[, suffix])	写入缓存文件

Methods Documentation

get(*k: str*, *suffix: str = 'pkl'*) → Any

读取缓存文件

Parameters

- **k** –缓存文件名
- **suffix** –缓存文件后缀，支持 pkl, json, txt, csv, xlsx, feather, parquet

Returns

缓存文件内容

is_found(*k: str*, *suffix: str = 'pkl'*, *ttl=-1*) → bool

判断缓存文件是否存在

Parameters

- **k** –缓存文件名
- **suffix** –缓存文件后缀，支持 pkl, json, txt, csv, xlsx, feather, parquet
- **ttl** –缓存文件有效期，单位：秒，-1 表示永久有效

Returns

bool

remove(*k: str*, *suffix: str = 'pkl'*)

set(*k: str*, *v: Any*, *suffix: str = 'pkl'*)

写入缓存文件

Parameters

- **k** –缓存文件名
- **v** –缓存文件内容
- **suffix** –缓存文件后缀，支持 pkl, json, txt, csv, xlsx, feather, parquet

KlineChart

class czsc.utils.**KlineChart**(*n_rows=3*, ***kwargs*)

Bases: object

K 线绘图工具类

plotly 参数详解: https://www.jianshu.com/p/4f4daf47cc85

Methods Summary

add_bar_indicator(x, y, name, row[, color])	绘制条形图指标
add_indicator(dt[, scatters, scatter_names, ...])	绘制曲线叠加 bar 型指标
add_kline(kline[, name])	绘制 K 线
add_macd(kline[, row])	绘制 MACD 图
add_marker_indicator(x, y, name, row[, text])	绘制标记类指标
add_scatter_indicator(x, y, name, row[, text])	绘制线性/离散指标
add_sma(kline[, row, ma_seq, visible])	绘制均线图
add_vol(kline[, row])	绘制成交量图
open_in_browser([file_name])	在浏览器中打开

Methods Documentation

add_bar_indicator(*x*, *y*, *name: str*, *row: int*, *color=None*, ***kwargs*)

绘制条形图指标

绘图 API 文档：https://plotly.com/python-api-reference/generated/plotly.graph_objects.Bar.html

函数执行逻辑：

1. 获取自定义参数 hover_template、show_legend、visible 和 base。这些参数分别对应于鼠标悬停时显示的模板、是否显示图例、是否可见和基线（默认为 True）。
2. 如果 color 参数为空，则使用 self.color_red 作为颜色。
3. **使用给定的 x、y 数据创建一个 go.Bar 对象（条形图），并传入以下参数：**
    - x: 指标的 x 轴数据
    - y: 指标的 y 轴数据
    - marker_line_color: 条形边框的颜色
    - marker_color: 条形填充的颜色
    - name: 指标名称
    - showlegend: 是否显示图例
    - hovertemplate: 鼠标悬停时显示的模板
    - visible: 是否可见
    - base: 基线，默认为 True

4. 调用 self.fig.add_trace 方法将创建的 go.Bar 对象添加到指定子图中，并更新所有 traces 的 X 轴属性为“x1”。

Parameters

- **x** –指标的 x 轴
- **y** –指标的 y 轴
- **name** –指标名称
- **row** –放入第几个子图
- **color** –指标的颜色，可以是单个颜色，也可以是一个列表，列表长度和 y 的长度一致，指示每个 y 的颜色比如：color = ‘rgba(249,41,62,0.7)’或者 color = [‘rgba(249,41,62,0.7)’, ‘rgba(0,170,59,0.7)’]
- **kwargs** –

Returns

add_indicator(*dt*, *scatters: list | None = None*, *scatter_names: list | None = None*, *bar=None*, *bar_name=''*, *row=4*, ***kwargs*)

绘制曲线叠加 bar 型指标

1. 获取自定义参数 line_width，默认值为 0.6。
2. **如果 scatters（列表）不为空，则遍历 scatters 中的所有散点数据：**
   - **对于每个散点数据，调用 add_scatter_indicator 方法将其绘制为折线图。传递以下参数：**
     - x: 日期时间数据
     - y: 散点数据
     - name: 图例名称，来自 scatter_names 列表
     - row: 指定要添加指标的子图行数，默认值为 4
     - show_legend: 是否显示图例，默认值为 False
     - line_width: 线宽，默认值为 0.6
3. 如果 bar 不为空，则使用 np.where 函数根据 bar 值大于零的情况设置颜色：大于零使用红色（self.color_red），否则使用绿色（self.color_green）。
4. **调用 add_bar_indicator 方法将 bar 绘制为柱状图。传递以下参数：**
   - x: 日期时间数据
   - y: bar 数据
   - name: 图例名称，为传入的 bar_name 参数
   - row: 指定要添加指标的子图行数，默认值为 4

- color: 根据上一步计算的颜色设置
- show_legend: 是否显示图例，默认值为 False

add_kline(*kline: DataFrame*, *name: str = 'K 线'*, ***kwargs*)

绘制 K 线

函数执行逻辑：

1. 检查 kline 数据框是否包含‘text’列。如果没有，则添加一个空字符串列。
2. **使用 go.Candlestick 创建一个 K 线图，并传入以下参数：**
   - x: 日期时间数据
   - open, high, low, close: 开盘价、最高价、最低价和收盘价
   - text: 显示在每个 K 线上的文本标签
   - name: 图例名称
   - showlegend: 是否显示图例
   - increasing_line_color 和 decreasing_line_color: 上涨时的颜色和下跌时的颜色
   - increasing_fillcolor 和 decreasing_fillcolor: 上涨时填充颜色和下跌时填充颜色
   - **kwargs: 可以传递其他自定义参数给 Candlestick 函数。
3. 将创建的烛台图对象添加到 self.fig 中的第一个子图（row=1, col=1）。
4. 使用 fig.update_traces 更新所有 traces 的 xaxis 属性为“x1”。

add_macd(*kline: DataFrame*, *row=3*, ***kwargs*)

绘制 MACD 图

函数执行逻辑：

1. 首先，复制输入的 kline 数据框到 df。
2. 获取自定义参数 fastperiod、slowperiod 和 signalperiod。这些参数分别对应于计算 MACD 时使用的快周期、慢周期和信号周期，默认值分别为 12、26 和 9。
3. 使用 talib 库的 MACD 函数计算 MACD 值（diff, dea, macd）。
4. 创建一个名为 macd_colors 的 numpy 数组，根据 macd 值大于零的情况设置颜色：大于零使用红色（self.color_red），否则使用绿色（self.color_green）。
5. **调用 add_scatter_indicator 方法将 diff 和 dea 绘制为折线图。传递以下参数：**
   - x: 日期时间数据
   - y: diff 或 dea 数据
   - name: 图例名称，分别为“DIFF”和“DEA”
   - row: 指定要添加指标的子图行数，默认值为 3

- line_color: 线的颜色，分别为‘white’和‘yellow’
- show_legend: 是否显示图例，默认值为 False
- line_width: 线宽，默认值为 0.6

6. **调用 add_bar_indicator 方法将 macd 绘制为柱状图。传递以下参数：**
   - x: 日期时间数据
   - y: macd 数据
   - name: 图例名称，为“MACD”
   - row: 指定要添加指标的子图行数，默认值为 3
   - color: 根据 macd_colors 设置颜色
   - show_legend: 是否显示图例，默认值为 False

add_marker_indicator(*x*, *y*, *name: str*, *row: int*, *text=None*, ***kwargs*)

绘制标记类指标

函数执行逻辑：

1. **获取自定义参数 line_color、line_width、hover_template、show_legend 和 visible。**
   这些参数分别对应于折线颜色、宽度、鼠标悬停时显示的模板、是否显示图例和是否可见。
2. **使用给定的 x、y 数据创建一个 go.Scatter 对象（散点图），并传入以下参数：**
   - x: 指标的 x 轴数据
   - y: 指标的 y 轴数据
   - name: 指标名称
   - text: 文本说明
   - line_width: 线宽
   - line_color: 线颜色
   - hovertemplate: 鼠标悬停时显示的模板
   - showlegend: 是否显示图例
   - visible: 是否可见
   - opacity: 透明度
   - mode: 绘制模式，为‘markers’表示只绘制标记
   - marker: 标记的样式，包括大小、颜色和符号
3. 调用 self.fig.add_trace 方法将创建的 go.Scatter 对象添加到指定子图中，并更新所有 traces 的 X 轴属性为“x1”。

Parameters

- **x** –指标的 x 轴
- **y** –指标的 y 轴
- **name** –指标名称
- **row** –放入第几个子图
- **text** –文本说明
- **kwargs** –

Returns

add_scatter_indicator(*x*, *y*, *name: str*, *row: int*, *text=None*, ***kwargs*)

绘制线性/离散指标

绘图 API 文档：https://plotly.com/python-api-reference/generated/plotly.graph_objects.Scatter.html

函数执行逻辑：

1. 获取自定义参数 mode、hover_template、show_legend、opacity 和 visible。这些参数分别对应于绘图模式、鼠标悬停时显示的模板、是否显示图例、透明度和是否可见。
2. **使用给定的 x、y 数据创建一个 go.Scatter 对象（散点图），并传入以下参数：**
   - x: 指标的 x 轴数据
   - y: 指标的 y 轴数据
   - name: 指标名称
   - text: 文本说明
   - mode: 绘制模式，默认为‘text+lines’，表示同时绘制文本和线条
   - hovertemplate: 鼠标悬停时显示的模板
   - showlegend: 是否显示图例
   - visible: 是否可见
   - opacity: 透明度
3. 调用 self.fig.add_trace 方法将创建的 go.Scatter 对象添加到指定子图中，并更新所有 traces 的 X 轴属性为“x1”。

Parameters

- **x** –指标的 x 轴
- **y** –指标的 y 轴
- **name** –指标名称

- **row** –放入第几个子图
- **text** –文本说明
- **kwargs** –

Returns

add_sma(*kline: DataFrame*, *row=1*, *ma_seq=(5, 10, 20)*, *visible=False*, ***kwargs*)

绘制均线图

函数执行逻辑:

1. 复制输入的 kline 数据框到 df。
2. 获取自定义参数 line_width，默认值为 0.6。
3. **遍历 ma_seq 中的所有均线周期:**
   - 对每个周期使用 pandas rolling 方法计算收盘价的移动平均线。
   - **调用 add_scatter_indicator 方法将移动平均线数据绘制为折线图。传递以下参数:**
     - x: 日期时间数据
     - y: 移动平均线数据
     - name: 图例名称，格式为 “MA{ma}”，其中 {ma} 是当前的均线周期。
     - row: 指定要添加指标的子图行数，默认值为 1
     - line_width: 线宽，默认值为 0.6
     - visible: 是否可见，默认值为 False
     - show_legend: 是否显示图例，默认值为 True

add_vol(*kline: DataFrame*, *row=2*, ***kwargs*)

绘制成交量图

函数执行逻辑:

1. 首先，复制输入的 kline 数据框到 df。
2. 使用 np.where 函数根据收盘价（df[‘close’]）和开盘价（df[‘open’]）之间的关系为 df 创建一个新列‘vol_color’。如果收盘价大于开盘价，则使用红色（self.color_red），否则使用绿色（self.color_green）。
3. **调用 add_bar_indicator 方法绘制成交量图。传递以下参数:**
   - x: 日期时间数据
   - y: 成交量数据
   - color: 根据‘vol_color’列的颜色
   - name: 图例名称

- row: 指定要添加指标的子图行数，默认值为 2
- show_legend: 是否显示图例，默认值为 False

open_in_browser(*file_name: str | None = None*, ***kwargs*)

在浏览器中打开

SignalAnalyzer

class czsc.utils.**SignalAnalyzer**(**args*, ***kwargs*)

Bases: object

Methods Summary

execute([max_workers])	执行信号分析
find_valuable_signals(dfp)	根据信号表现，找出表现好的信号
generate_symbol_signals(symbol)	

Methods Documentation

execute(*max_workers=10*)

执行信号分析

static find_valuable_signals(*dfp*)

根据信号表现，找出表现好的信号

Parameters

dfp –信号表现分析结果

Returns

表现好的信号

generate_symbol_signals(*symbol*)

SignalPerformance

class czsc.utils.**SignalPerformance**(**args*, ***kwargs*)

Bases: object

信号表现分析

Methods Summary

analyze([mode])	分析信号出现前后的收益情况
report([file_xlsx])	

Methods Documentation

analyze(*mode='0b'*) → DataFrame

分析信号出现前后的收益情况

Parameters

mode –分析模式，0b 截面向前看 0n 截面向后看 1b 时序向前看 1n 时序向后看

Returns

report(*file_xlsx=None*)

WordWriter

class czsc.utils.**WordWriter**(*file_docx=None*)

Bases: object

用 Word 文档记录信息

Methods Summary

add_df_table(df[, style])	添加数据表
add_heading(text[, level])	
add_page_break()	添加分页符
add_paragraph(text[, style, bold, ...])	新增段落
add_picture(file[, width, height, alignment])	写入图片到文档中
add_title(text)	
save([file_docx])	保存结果到文件

Methods Documentation

add_df_table(*df: DataFrame*, *style='Table Grid'*, ***kwargs*)

添加数据表

https://www.jianshu.com/p/93e0df92cf16

Parameters

- **df** –数据表
- **style** –表格样式

Returns

add_heading(*text*, *level=1*)

add_page_break()

添加分页符

add_paragraph(*text*, *style=None*, *bold=False*, *first_line_indent=0.74*)

新增段落

Parameters

- **text** –文本
- **style** –段落样式
- **bold** –是否加粗
- **first_line_indent** –首行缩进，0.74 表示两个空格

Returns

add_picture(*file*, *width=None*, *height=None*, *alignment='center'*) → None

写入图片到文档中

Parameters

- **file** –图片文件路径
- **width** –图片宽度，默认单位 cm
- **height** –图片高度，默认单位 cm
- **alignment** –图片对齐，默认 center

Returns

add_title(*text*)

save(*file_docx=None*)

保存结果到文件

2.8 czsc.aphorism Module

2.8.1 Functions

print_one()	

print_one

czsc.aphorism.**print_one**()

2.9 czsc.enum Module

2.9.1 Classes

Direction(value)	An enumeration.
Enum(value)	Generic enumeration.
Freq(value)	An enumeration.
Mark(value)	An enumeration.
Operate(value)	An enumeration.

Direction

class czsc.enum.**Direction**(*value*)

Bases: Enum

An enumeration.

Attributes Summary

Down
Up

Attributes Documentation

Down = '向下'

Up = '向上'

Freq

class czsc.enum.**Freq**(*value*)

Bases: Enum

An enumeration.

Attributes Summary

D
F1
F10
F12
F120
F15
F2
F20
F3
F30
F4
F5
F6
F60
M
S
Tick
W
Y

Attributes Documentation

D = '日线'

F1 = '1 分钟'

F10 = '10 分钟'

F12 = '12 分钟'

F120 = '120 分钟'

F15 = '15 分钟'

F2 = '2 分钟'

F20 = '20 分钟'

F3 = '3 分钟'

F30 = '30 分钟'

F4 = '4 分钟'

F5 = '5 分钟'

F6 = '6 分钟'

F60 = '60 分钟'

M = '月线'

S = '季线'

Tick = 'Tick'

W = '周线'

Y = '年线'

Mark

class czsc.enum.**Mark**(*value*)

Bases: Enum

An enumeration.

Attributes Summary

D
G

Attributes Documentation

D = '底分型'

G = '顶分型'

Operate

class czsc.enum.**Operate**(*value*)

Bases: Enum

An enumeration.

Attributes Summary

HL
HO
HS
LE
LO
SE
SO

Attributes Documentation

HL = '持多'

HO = '持币'

HS = '持空'

LE = '平多'

LO = '开多'

SE = '平空'

SO = '开空'

2.10 czsc.envs Module

2.10.1 Functions

get_bi_change_th([v])	bi_change_th - 成笔需要超过 benchmark 的比例阈值
get_max_bi_num([v])	max_bi_num - 单个级别 K 线分析中，程序最大保存的笔数量
get_min_bi_len([v])	min_bi_len - 一笔的最小长度，也就是无包含 K 线的数量，7 是老笔的要求，6 是新笔的要求
get_verbose([verbose])	verbose - 是否输出执行过程的详细信息
get_welcome()	welcome - 是否输出版本标识和缠中说禅博客摘记

get_bi_change_th

czsc.envs.**get_bi_change_th**(*v: float | None = None*) → float

bi_change_th - 成笔需要超过 benchmark 的比例阈值

benchmark 是上一笔涨跌幅与最近五笔平均涨跌幅均值的最小值

设置成 -1，可以关闭根据当前笔涨跌幅达到 benchmark 的比例来确定笔

get_max_bi_num

czsc.envs.**get_max_bi_num**(*v: int | None = None*) → int

max_bi_num - 单个级别 K 线分析中，程序最大保存的笔数量

默认值为 50，仅使用内置的信号和因子，不需要调整这个参数。如果进行新的信号计算需要用到更多的笔，可以适当调大这个参数。

get_min_bi_len

czsc.envs.**get_min_bi_len**(*v: int | None = None*) → int

min_bi_len - 一笔的最小长度，也就是无包含 K 线的数量，7 是老笔的要求，6 是新笔的要求

get_verbose

czsc.envs.**get_verbose**(*verbose=None*)

verbose - 是否输出执行过程的详细信息

get_welcome

czsc.envs.**get_welcome**()

welcome - 是否输出版本标识和缠中说禅博客摘记

2.11 czsc.objects Module

2.11.1 Functions

cal_break_even_point(seq)	计算单笔收益序列的盈亏平衡点
create_fake_bis(fxs)	创建 fake_bis 列表
dataclass([cls, init, repr, eq, order, ...])	Returns the same class as was passed in, with dunder methods added based on the fields defined in the class.
deepcopy(x[, memo, _nil])	Deep copy operation on arbitrary Python objects.
deprecated(*args, **kwargs)	This is a decorator which can be used to mark functions as deprecated.
field(*[, default, default_factory, init, ...])	Return an object to identify dataclass fields.
single_linear(y[, x])	单变量线性拟合

cal_break_even_point

czsc.objects.**cal_break_even_point**(*seq: List[float]*) → float

计算单笔收益序列的盈亏平衡点

Parameters

seq –单笔收益序列

Returns

盈亏平衡点

create_fake_bis

czsc.objects.**create_fake_bis**(*fxs: List[FX]*) → List[*FakeBI*]

创建 fake_bis 列表

Parameters

fxs –分型序列，必须顶底分型交替

Returns

fake_bis

2.11.2 Classes

BI(symbol, fx_a, fx_b, fxs, direction, bars, ...)	
Direction(value)	An enumeration.
Event(operate, factors, signals_all, ...)	
FX(symbol, dt, mark, high, low, fx, ...)	
Factor(signals_all, signals_any, ...)	
FakeBI(symbol, sdt, edt, direction, high, ...)	虚拟笔：主要为笔的内部分析提供便利
Freq(value)	An enumeration.
Mark(value)	An enumeration.
NewBar(symbol, id, dt, freq, open, close, ...)	去除包含关系后的 K 线元素
Operate(value)	An enumeration.
Position(symbol, opens[, exits, interval, ...])	
RawBar(symbol, id, dt, freq, open, close, ...)	原始 K 线元素
Signal([signal, score, k1, k2, k3, v1, v2, v3])	
Tick(symbol[, name, price, vol])	
ZS(bis, cache)	中枢对象，主要用于辅助信号函数计算
datetime(year, month, day[, hour[, minute[, ...)	The year, month and day arguments are required.

BI

class czsc.objects.**BI**(*symbol: str*, *fx_a: czsc.objects.FX*, *fx_b: czsc.objects.FX*, *fxs: List*, *direction: czsc.enum.Direction*, *bars: List[czsc.objects.NewBar] = <factory>*, *cache: dict = <factory>*)

Bases: object

Attributes Summary

angle	笔的斜边与竖直方向的夹角，角度越大，力度越大
change	笔的涨跌幅
fake_bis	笔的内部分型连接得到近似次级别笔列表
high	
hypotenuse	笔的斜边长度
length	笔的无包含关系 K 线数量
low	
power	
power_price	价差力度
power_volume	成交量力度
raw_bars	构成笔的原始 K 线序列
rsq	笔的原始 K 线 close 单变量线性回归 r2

Methods Summary

get_cache_with_default(key, default)	带有默认值计算的缓存读取
get_price_linear([price_key])	计算 price 的单变量线性回归特征

Attributes Documentation

angle

笔的斜边与竖直方向的夹角，角度越大，力度越大

change

笔的涨跌幅

fake_bis

笔的内部分型连接得到近似次级别笔列表

high

hypotenuse

笔的斜边长度

length

笔的无包含关系 K 线数量

low

power

power_price

价差力度

power_volume

成交量力度

raw_bars

构成笔的原始 K 线序列

rsq

笔的原始 K 线 close 单变量线性回归 r2

Methods Documentation

get_cache_with_default(*key*, *default: Callable*)

带有默认值计算的缓存读取

Parameters

- **key** –缓存 key
- **default** –如果没有缓存数据，用来计算默认值并更新缓存的函数

Returns

get_price_linear(*price_key='close'*)

计算 price 的单变量线性回归特征

Parameters

price_key –指定价格类型，可选值 open close high low

Return value

单变量线性回归特征，样例如下 {‘slope’: 1.565, ‘intercept’: 67.9783, ‘r2’: 0.9967}

slope 标识斜率 intercept 截距 r2 拟合优度

Event

class czsc.objects.**Event**(*operate: czsc.enum.Operate, factors: List[czsc.objects.Factor], signals_all: List[czsc.objects.Signal] = <factory>, signals_any: List[czsc.objects.Signal] = <factory>, signals_not: List[czsc.objects.Signal] = <factory>, name: str = ''*)

Bases: object

Attributes Summary

name	
unique_signals	获取 Event 的唯一信号列表

Methods Summary

dump()	将 Event 对象转存为 dict
get_signals_config([signals_module])	获取事件的信号配置
is_match(s)	判断 event 是否满足
load(raw)	从 dict 中创建 Event

Attributes Documentation

name: str = ''

unique_signals

获取 Event 的唯一信号列表

Methods Documentation

dump() → dict

将 Event 对象转存为 dict

get_signals_config(*signals_module: str = 'czsc.signals'*) → List[Dict]

获取事件的信号配置

is_match(*s: dict*)

判断 event 是否满足

代码的执行逻辑如下：

1. 首先判断 signals_not 中的信号是否得到满足，如果满足任意一个信号，则直接返回 False，表示事件不满足。
2. 接着判断 signals_all 中的信号是否全部得到满足，如果有任意一个信号不满足，则直接返回 False，表示事件不满足。
3. 然后判断 signals_any 中的信号是否有一个得到满足，如果一个都不满足，则直接返回 False，表示事件不满足。
4. 最后判断因子是否满足，顺序遍历因子列表，找到第一个满足的因子就退出，并返回 True 和该因子的名称，表示事件满足。
5. 如果遍历完所有因子都没有找到满足的因子，则返回 False，表示事件不满足。

classmethod load(*raw: dict*)

从 dict 中创建 Event

Parameters

raw –样例如下 {‘name’：‘单测’，

’operate’：‘开多’，‘factors’：[{‘name’：‘测试’，

’signals_all’：[‘15 分钟 _ 倒 0 笔 _ 长度 _ 大于 5_ 其他 _ 其他 _0’], ‘signals_any’：[], ‘signals_not’：[]}],

’signals_all’：[‘15 分钟 _ 倒 0 笔 _ 方向 _ 向上 _ 其他 _ 其他 _0’], ‘signals_any’：[], ‘signals_not’：[]}

Returns

FX

class czsc.objects.**FX**(*symbol: str*, *dt: datetime.datetime*, *mark: czsc.enum.Mark*, *high: float*, *low: float*, *fx: float*, *elements: List = <factory>*, *cache: dict = <factory>*)

Bases: object

Attributes Summary

has_zs	构成分型的三根无包含 K 线是否有重叠中枢
new_bars	构成分型的无包含关系 K 线
power_str	
power_volume	成交量力度
raw_bars	构成分型的原始 K 线

Attributes Documentation

has_zs

构成分型的三根无包含 K 线是否有重叠中枢

new_bars

构成分型的无包含关系 K 线

power_str

power_volume

成交量力度

raw_bars

构成分型的原始 K 线

Factor

class czsc.objects.**Factor**(*signals_all: List[czsc.objects.Signal], signals_any: List[czsc.objects.Signal] = <factory>, signals_not: List[czsc.objects.Signal] = <factory>, name: str = ''*)

Bases: object

Attributes Summary

name	
unique_signals	获取 Factor 的唯一信号列表

Methods Summary

dump()	将 Factor 对象转存为 dict
is_match(s)	判断 factor 是否满足
load(raw)	从 dict 中创建 Factor

Attributes Documentation

name: str = ''

unique_signals

获取 Factor 的唯一信号列表

Methods Documentation

dump() → dict

将 Factor 对象转存为 dict

is_match(*s: dict*) → bool

判断 factor 是否满足

classmethod load(*raw: dict*)

从 dict 中创建 Factor

Parameters

raw –样例如下 {‘name’：‘单测’，‘signals_all’: [‘15 分钟 _ 倒 0 笔 _ 方向 _ 向上 _ 其他 _ 其他 _0’，‘15 分钟 _ 倒 0 笔 _ 长度 _ 大于 5_ 其他 _ 其他 _0’], ‘signals_any’: [], ‘signals_not’: []}

Returns

FakeBI

class czsc.objects.**FakeBI**(*symbol: str*, *sdt: ~datetime.datetime*, *edt: ~datetime.datetime*, *direction: ~czsc.enum.Direction*, *high: float*, *low: float*, *power: float*, *cache: dict = <factory>*)

Bases: object

虚拟笔：主要为笔的内部分析提供便利

NewBar

class czsc.objects.**NewBar**(*symbol: str*, *id: int*, *dt: ~datetime.datetime*, *freq: ~czsc.enum.Freq*, *open: float*, *close: float*, *high: float*, *low: float*, *vol: float*, *amount: float*, *elements: ~typing.List = <factory>*, *cache: dict = <factory>*)

Bases: object

去除包含关系后的 K 线元素

Attributes Summary

raw_bars	

Attributes Documentation

raw_bars

Position

class czsc.objects.**Position**(*symbol: str*, *opens: List[*Event*]*, *exits: List[*Event*] = []*, *interval: int = 0*, *timeout: int = 1000*, *stop_loss=1000*, *T0: bool = False*, *name=None*)

Bases: object

Attributes Summary

pairs	开平交易列表
unique_signals	获取所有事件的唯一信号列表

Methods Summary

dump([with_data])	将对象转换为 dict
evaluate([trade_dir])	评估交易表现
evaluate_holds([trade_dir])	按持仓信号评估交易表现
get_signals_config([signals_module])	获取事件的信号配置
load(raw)	从 dict 中创建 Position
update(s)	更新持仓状态

Attributes Documentation

pairs

开平交易列表

返回样例：

[{‘标的代码’：‘000001.SH’，

‘交易方向’：‘多头’，‘开仓时间’: Timestamp(‘2020-04-17 00:00:00’), ‘平仓时间’: Timestamp(‘2020-04-20 00:00:00’), ‘开仓价格’: 2838.49, ‘平仓价格’: 2852.55, ‘持仓 K 线数’: 1, ‘事件序列’：‘开多 @ 站上 SMA5 -> 开多 @ 站上 SMA5’，‘持仓天数’: 3.0, ‘盈亏比例’: 49.53},

{‘标的代码’：‘000001.SH’，
‘交易方向’：‘多头’，‘开仓时间’: Timestamp(‘2020-04-20 00:00:00’), ‘平仓时间’: Timestamp(‘2020-04-24 00:00:00’), ‘开仓价格’: 2852.55, ‘平仓价格’: 2808.53, ‘持仓 K 线数’: 4, ‘事件序列’：‘开多 @ 站上 SMA5 -> 平多 @100BP 止损’，‘持仓天数’: 4.0, ‘盈亏比例’: -154.32}]

数据说明：

1. 盈亏比例，单位是 BP
2. 持仓天数，单位是自然日
3. 持仓 K 线数，指基础周期 K 线数量

unique_signals

获取所有事件的唯一信号列表

Methods Documentation

dump(*with_data=False*)

将对象转换为 dict

evaluate(*trade_dir: str = '多空'*) → dict

评估交易表现

Parameters

trade_dir –交易方向，可选值 [‘多头’，‘空头’，‘多空’]

Returns

交易表现

evaluate_holds(*trade_dir: str = '多空'*) → dict

按持仓信号评估交易表现

Parameters

trade_dir –交易方向，可选值 [‘多头’，‘空头’，‘多空’]

Returns

交易表现

get_signals_config(*signals_module: str = 'czsc.signals'*) → List[Dict]

获取事件的信号配置

classmethod load(*raw: dict*)

从 dict 中创建 Position

Parameters

raw –样例如下

Returns

update(*s: dict*)

更新持仓状态

函数执行逻辑：

- 首先，检查最新信号的时间是否在上次信号之前，如果是则打印警告信息并返回。
- 初始化一些变量，包括操作类型（op）和操作描述（op_desc）。
- 遍历所有的事件，检查是否与最新信号匹配。如果匹配，则记录操作类型和操作描述，并跳出循环。
- 提取最新信号的相关信息，包括交易对符号、时间、价格和成交量。
- 更新持仓状态的结束时间为最新信号的时间。
- 如果操作类型是开仓（LO 或 SO），更新最后一个事件的信息。
- 定义一个内部函数 __create_operate，用于创建操作记录。
- 根据操作类型更新仓位和操作记录。
  - 如果操作类型是 LO（开多），检查是否满足开仓条件，如果满足则开多仓，否则只平空仓。
  - 如果操作类型是 SO（开空），检查是否满足开仓条件，如果满足则开空仓，否则只平多仓。
  - **如果当前持仓为多仓，进行多头出场的判断：**
    - 如果操作类型是 LE（平多），平多仓。
    - 如果当前价格相对于最后一个事件的价格的收益率小于止损阈值，平多仓。
    - 如果当前成交量相对于最后一个事件的成交量的增加量大于超时阈值，平多仓。
  - **如果当前持仓为空仓，进行空头出场的判断：**
    - 如果操作类型是 SE（平空），平空仓。
    - 如果当前价格相对于最后一个事件的价格的收益率小于止损阈值，平空仓。
    - 如果当前成交量相对于最后一个事件的成交量的增加量大于超时阈值，平空仓。
- 将当前持仓状态和价格记录到持仓列表中。

Parameters

s –最新信号字典

Returns

RawBar

class czsc.objects.**RawBar**(*symbol: str*, *id: int*, *dt: ~datetime.datetime*, *freq: ~czsc.enum.Freq*, *open: float*, *close: float*, *high: float*, *low: float*, *vol: float*, *amount: float*, *cache: dict = <factory>*)

Bases: object

原始 K 线元素

Attributes Summary

lower	下影
solid	实体
upper	上影

Attributes Documentation

lower

下影

solid

实体

upper

上影

Signal

class czsc.objects.**Signal**(*signal: str = ''*, *score: int = 0*, *k1: str = '任意'*, *k2: str = '任意'*, *k3: str = '任意'*, *v1: str = '任意'*, *v2: str = '任意'*, *v3: str = '任意'*)

Bases: object

Attributes Summary

k1	
k2	
k3	
key	获取信号名称
score	
signal	
v1	
v2	
v3	
value	获取信号值

Methods Summary

is_match(s)	判断信号是否与信号列表中的值匹配

Attributes Documentation

k1: str = '任意'

k2: str = '任意'

k3: str = '任意'

key

获取信号名称

score: int = 0

signal: str = ''

v1: str = '任意'

v2: str = '任意'

v3: str = '任意'

value

获取信号值

Methods Documentation

is_match(*s: dict*) → bool

判断信号是否与信号列表中的值匹配

代码的执行逻辑如下：

接收一个字典 s 作为参数，该字典包含了所有信号的信息。从字典 s 中获取名称为 key 的信号的值 v。如果 v 不存在，则抛出异常。从信号的值 v 中解析出 v1、v2、v3 和 score 四个变量。

如果当前信号的得分 score 大于等于目标信号的得分 self.score，则继续执行，否则返回 False。如果当前信号的第一个值 v1 等于目标信号的第一个值 self.v1 或者目标信号的第一个值为“任意”，则继续执行，否则返回 False。如果当前信号的第二个值 v2 等于目标信号的第二个值 self.v2 或者目标信号的第二个值为“任意”，则继续执行，否则返回 False。如果当前信号的第三个值 v3 等于目标信号的第三个值 self.v3 或者目标信号的第三个值为“任意”，则返回 True，否则返回 False。

Parameters

s –所有信号字典

Returns

bool

Tick

class czsc.objects.**Tick**(*symbol: str*, *name: str = ''*, *price: float = 0*, *vol: float = 0*)

Bases: object

Attributes Summary

name
price
vol

Attributes Documentation

name: str = ''

price: float = 0

vol: float = 0

ZS

class czsc.objects.**ZS**(*bis: ~typing.List[~czsc.objects.BI], cache: dict = <factory>*)

Bases: object

中枢对象，主要用于辅助信号函数计算

Attributes Summary

dd	中枢最低点
edir	中枢倒一笔方向，edir 是 end direction 的缩写
edt	中枢结束时间
gg	中枢最高点
is_valid	中枢是否有效
sdir	中枢第一笔方向，sdir 是 start direction 的缩写
sdt	中枢开始时间
zd	中枢下沿
zg	中枢上沿
zz	中枢中轴

Attributes Documentation

dd

中枢最低点

edir

中枢倒一笔方向，edir 是 end direction 的缩写

edt

中枢结束时间

gg

中枢最高点

is_valid

中枢是否有效

sdir

中枢第一笔方向，sdir 是 start direction 的缩写

sdt

中枢开始时间

zd

中枢下沿

zg

中枢上沿

zz

中枢中轴

2.12 czsc.strategies Module

2.12.1 Functions

abstractmethod(funcobj)	A decorator indicating abstract methods.
check_freq_and_market(time_seq[, freq])	检查时间序列是否为同一周期，是否为同一市场
create_cci_long(symbol[, is_stocks])	CCI 基础多头策略
create_cci_short(symbol[, is_stocks])	CCI 基础空头策略
create_emv_long(symbol[, is_stocks])	EMV 多头策略
create_emv_short(symbol[, is_stocks])	EMV 空头策略
create_macd_long(symbol[, is_stocks])	MACD 多头策略
create_macd_short(symbol[, is_stocks])	MACD 空头策略
create_single_ma_long(symbol, ma_name[, ...])	单均线多头策略
create_single_ma_short(symbol, ma_name[, ...])	单均线空头策略
create_third_buy_long(symbol[, is_stocks])	缠中说禅三买多头策略
create_third_sell_short(symbol[, is_stocks])	缠中说禅三卖空头策略
deepcopy(x[, memo, _nil])	Deep copy operation on arbitrary Python objects.
dill_dump(data, file)	
freqs_sorted(freqs)	K 线周期列表排序并去重，第一个元素是基础周期
get_signals_config(signals_seq[, signals_module])	获取信号列表对应的信号函数配置
get_signals_freqs(signals_seq)	获取信号列表对应的 K 线周期列表
read_json(file)	
save_json(data, file)	
x_round(x[, digit])	用去尾法截断小数

create_cci_long

czsc.strategies.**create_cci_long**(*symbol*, *is_stocks=False*, ***kwargs*) → *Position*

CCI 基础多头策略

用到的信号函数列表：

1. https://czsc.readthedocs.io/en/latest/api/czsc.signals.tas_cci_base_V230402.html
2. https://czsc.readthedocs.io/en/latest/api/czsc.signals.bar_zdt_V230331.html

Parameters

- **symbol** –标的代码
- **is_stocks** –是否是 A 股
- **kwargs** –其他参数 - base_freq: 基础级别 - freq: 信号级别 - cci_timeperiod: CCI 周期 - T0: 是否是 T0 策略 - interval: 同向开仓间隔时间 - timeout: 超时出场时间

Returns

create_cci_short

czsc.strategies.**create_cci_short**(*symbol*, *is_stocks=False*, ***kwargs*) → *Position*

CCI 基础空头策略

用到的信号函数列表：

1. https://czsc.readthedocs.io/en/latest/api/czsc.signals.tas_cci_base_V230402.html
2. https://czsc.readthedocs.io/en/latest/api/czsc.signals.bar_zdt_V230331.html

Parameters

- **symbol** –标的代码
- **is_stocks** –是否是 A 股
- **kwargs** –其他参数 - base_freq: 基础级别 - freq: 信号级别 - cci_timeperiod: CCI 周期 - T0: 是否是 T0 策略 - interval: 同向开仓间隔时间 - timeout: 超时出场时间

Returns

create_emv_long

czsc.strategies.**create_emv_long**(*symbol*, *is_stocks=False*, ***kwargs*) → *Position*

EMV 多头策略

Parameters

- **symbol** –标的代码
- **is_stocks** –是否是 A 股
- **kwargs** –其他参数
  - base_freq: 基础级别
  - freq: 信号级别
  - T0: 是否是 T0 策略

Returns

create_emv_short

czsc.strategies.**create_emv_short**(*symbol*, *is_stocks=False*, ***kwargs*) → *Position*

EMV 空头策略

Parameters

- **symbol** –标的代码
- **is_stocks** –是否是 A 股
- **kwargs** –其他参数
  - base_freq: 基础级别
  - freq: 信号级别
  - T0: 是否是 T0 策略

Returns

create_macd_long

czsc.strategies.**create_macd_long**(*symbol*, *is_stocks=False*, ***kwargs*) → *Position*

MACD 多头策略

https://czsc.readthedocs.io/en/latest/api/czsc.signals.tas_macd_base_V230320.html

Parameters

- **symbol** –标的代码
- **is_stocks** –是否是 A 股
- **kwargs** –其他参数 - base_freq: 基础级别 - freq: 信号级别 - max_overlap: 最大重叠数 - T0: 是否是 T0 策略

Returns

create_macd_short

czsc.strategies.**create_macd_short**(*symbol*, *is_stocks=False*, ***kwargs*) → *Position*

MACD 空头策略

https://czsc.readthedocs.io/en/latest/api/czsc.signals.tas_macd_base_V230320.html

Parameters

- **symbol** –标的代码
- **is_stocks** –是否是 A 股
- **kwargs** –其他参数

Returns

create_single_ma_long

czsc.strategies.**create_single_ma_long**(*symbol*, *ma_name*, *is_stocks=False*, ***kwargs*) → *Position*

单均线多头策略

https://czsc.readthedocs.io/en/latest/api/czsc.signals.tas_ma_base_V230313.html

Parameters

- **symbol** –
- **ma_name** –
- **is_stocks** –
- **kwargs** –

Returns

create_single_ma_short

czsc.strategies.**create_single_ma_short**(*symbol*, *ma_name*, *is_stocks=False*, ***kwargs*) → *Position*

单均线空头策略

https://czsc.readthedocs.io/en/latest/api/czsc.signals.tas_ma_base_V230313.html

Parameters

- **symbol** –标的代码
- **ma_name** –均线名称，如 SMA#5
- **is_stocks** –是否是 A 股
- **kwargs** –其他参数

Returns

create_third_buy_long

czsc.strategies.**create_third_buy_long**(*symbol*, *is_stocks=False*, ***kwargs*) → *Position*

缠中说禅三买多头策略

https://czsc.readthedocs.io/en/latest/api/czsc.signals.cxt_five_bi_V230619.html https://czsc.readthedocs.io/en/latest/api/czsc.signals.cxt_three_bi_V230618.html

Parameters

- **symbol** –标的代码
- **is_stocks** –是否是 A 股

- **kwargs** –其他参数
  - base_freq: 基础级别
  - freq: 信号级别
  - T0: 是否是 T0 策略

Returns

create_third_sell_short

czsc.strategies.**create_third_sell_short**(*symbol*, *is_stocks=False*, ***kwargs*) → *Position*

缠中说禅三卖空头策略

Parameters

- **symbol** –标的代码
- **is_stocks** –是否是 A 股
- **kwargs** –其他参数
  - base_freq: 基础级别
  - freq: 信号级别
  - T0: 是否是 T0 策略

Returns

2.12.2 Classes

ABC()	Helper class that provides a standard way to create an ABC using inheritance.
BarGenerator(base_freq, freqs[, max_count, ...])	
CzscJsonStrategy(**kwargs)	仅传入 Json 配置的 Positions 就完成策略创建
CzscStrategyBase(**kwargs)	择时交易策略的要素：
CzscStrategyExample2(**kwargs)	仅传入 Positions 就完成策略创建
CzscTrader([bg, positions, ensemble_method])	缠中说禅技术分析理论之多级别联立交易决策类（支持多策略独立执行）
Event(operate, factors, signals_all, ...)	
Factor(signals_all, signals_any, ...)	
Operate(value)	An enumeration.
Position(symbol, opens[, exits, interval, ...])	
RawBar(symbol, id, dt, freq, open, close, ...)	原始 K 线元素
Signal([signal, score, k1, k2, k3, v1, v2, v3])	
datetime(year, month, day[, hour[, minute[, ...)	The year, month and day arguments are required.
timedelta	Difference between two datetime values.
tqdm(*_, **__)	Decorate an iterable object, returning an iterator which acts exactly like the original iterable, but prints a dynamically updating progressbar every time a value is requested.

CzscJsonStrategy

class czsc.strategies.**CzscJsonStrategy**(***kwargs*)

Bases: *CzscStrategyBase*

仅传入 Json 配置的 Positions 就完成策略创建

执行逻辑:

1. 定义 CzscJsonStrategy 类，并继承自 CzscStrategyBase。这个类可以通过仅传入 Json 配置的 Positions 来完成策略创建。
2. 类中定义了一个名为 positions 的属性，使用 @property 装饰器将其标记为只读属性。
3. **在 positions 属性的 getter 方法中，执行以下操作：**

    - **从 self.kwargs 字典中获取键为”files_position”的值，并将其赋值给变量 files。**
      这里的 self.kwargs 可能是通过在实例化该类时传入的参数或其他方式设置的一个

字典，其中包含了策略配置文件的路径列表。

- 使用 self.kwargs.get 方法获取键为”check_position”的值，并设置默认值为 True，将其赋值给变量 check。这个值用于确定是否对 JSON 持仓策略进行 MD5 校验。
- **调用 self.load_positions(files, check) 方法，并返回其结果。这个方法可能是从父类 CzscStrategyBase 中继承的方法，**
  用于从配置文件中加载持仓策略。将文件列表和校验标志作为参数传递给该方法，并返回加载的持仓策略列表。

必须参数：

files_position: 以 json 文件配置的策略，每个 json 文件对应一个持仓策略配置 check_position: 是否对 json 持仓策略进行 MD5 校验，默认为 True

Attributes Summary

positions	持仓策略列表

Attributes Documentation

positions

CzscStrategyBase

class czsc.strategies.**CzscStrategyBase**(***kwargs*)

Bases: ABC

择时交易策略的要素：

1. 交易品种以及该品种对应的参数
2. K 线周期列表
3. 交易信号参数配置
4. 持仓策略列表

Attributes Summary

base_freq	基础 K 线周期
freqs	K 线周期列表
signals_config	交易信号参数配置
sorted_freqs	排好序的 K 线周期列表
symbol	交易标的
unique_signals	所有持仓策略中的交易信号列表

Methods Summary

backtest(bars, **kwargs)	
check(bars, res_path, **kwargs)	检查交易策略中的信号是否正确
dummy(sigs, **kwargs)	使用信号缓存进行策略回测
init_bar_generator(bars, **kwargs)	使用策略定义初始化一个 BarGenerator 对象
init_trader(bars, **kwargs)	使用策略定义初始化一个 CzscTrader 对象
load_positions(files[, check])	从配置文件中加载持仓策略
positions()	持仓策略列表
replay(bars, res_path, **kwargs)	交易策略交易过程回放
save_positions(path)	保存持仓策略配置

Attributes Documentation

base_freq

基础 K 线周期

freqs

K 线周期列表

signals_config

交易信号参数配置

sorted_freqs

排好序的 K 线周期列表

symbol

交易标的

unique_signals

所有持仓策略中的交易信号列表

Methods Documentation

backtest(*bars: List[*RawBar*]*, ***kwargs*) → *CzscTrader*

check(*bars: List[*RawBar*]*, *res_path*, ***kwargs*)

检查交易策略中的信号是否正确

Parameters

- **bars** –基础周期 K 线

- **res_path** –结果目录
- **kwargs** –bg 已经初始化好的 BarGenerator 对象，如果传入了 bg，则忽略 sdt 和 n 参数 sdt 初始化开始日期 n 初始化最小 K 线数量

Returns

dummy(*sigs: List[dict]*, ***kwargs*) → *CzscTrader*

使用信号缓存进行策略回测

Parameters

sigs –信号缓存，一般指 generate_czsc_signals 函数计算的结果缓存

Returns

完成策略回测后的 CzscTrader 对象

init_bar_generator(*bars: List[*RawBar*]*, ***kwargs*)

使用策略定义初始化一个 BarGenerator 对象

函数执行逻辑：

- 该方法的目的是使用策略定义初始化一个 BarGenerator 对象。BarGenerator 是用于生成 K 线数据的类。
- 参数 bars 表示基础周期的 K 线数据，**kwargs 用于接收额外的关键字参数。
- 首先，方法获取了基础 K 线的频率，并检查了是否已经有一个初始化好的 BarGenerator 对象传入。
- 然后，根据基础频率是否在排序后的频率列表中，确定要使用的频率列表。
- 如果没有传入 BarGenerator 对象，则根据传入的基础 K 线数据和其他参数创建一个新的 BarGenerator 对象，并使用部分 K 线数据初始化它。余下的 K 线数据将用于 trader 的初始化区间。
- 如果传入了 BarGenerator 对象，则会做一些断言检查，确保传入的基础 K 线数据与已有的 BarGenerator 对象的基础周期一致，并且 BarGenerator 的 end_dt 是 datetime 类型。然后，筛选出在 BarGenerator 的 end_dt 之后的 K 线数据。
- 最后，返回 BarGenerator 对象和余下的 K 线数据。

Parameters

- **bars** –基础周期 K 线
- **kwargs** –bg 已经初始化好的 BarGenerator 对象，如果传入了 bg，则忽略 sdt 和 n 参数 sdt 初始化开始日期 n 初始化最小 K 线数量

Returns

init_trader(*bars: List[*RawBar*]*, ***kwargs*) → *CzscTrader*

使用策略定义初始化一个 CzscTrader 对象

注意：这里会将所有持仓策略在 sdt 之后的交易信号计算出来并缓存在持仓策略实例内部，所以初始化的过程本身也是回测的过程。

函数执行逻辑：

- 首先，它通过调用 init_bar_generator 方法获取已经初始化好的 BarGenerator 对象和余下的 K 线数据。
- 然后，它创建一个 CzscTrader 对象，将 BarGenerator 对象、持仓策略的深拷贝、交易信号配置的深拷贝等参数传递给 CzscTrader 的构造函数。
- 接着，使用余下的 K 线数据对 CzscTrader 对象进行初始化，通过调用 trader.on_bar(bar) 方法处理每一根 K 线数据。
- 最后，返回初始化完成的 CzscTrader 对象。

Parameters

- **bars** –基础周期 K 线
- **kwargs** –bg 已经初始化好的 BarGenerator 对象，如果传入了 bg，则忽略 sdt 和 n 参数 sdt 初始化开始日期 n 初始化最小 K 线数量

Returns

完成策略初始化后的 CzscTrader 对象

load_positions(*files: List*, *check=True*) → List[*Position*]

从配置文件中加载持仓策略

Parameters

- **files** –以 json 格式保存的持仓策略文件列表
- **check** –是否校验 MD5 值，默认为 True

Returns

持仓策略列表

abstract positions() → List[*Position*]

持仓策略列表

replay(*bars: List[*RawBar*]*, *res_path*, ***kwargs*)

交易策略交易过程回放

函数执行逻辑：

- 该方法用于交易策略交易过程的回放。它接受基础周期的 K 线数据、结果目录以及额外的关键字参数作为输入。

- 首先，它检查 refresh 参数，如果为 True，则使用 shutil.rmtree 删除已存在的结果目录。
- 然后，它检查结果目录是否已存在，并且是否允许覆盖。如果目录已存在且不允许覆盖，则记录一条警告信息并返回。
- 通过调用 os.makedirs 创建结果目录，确保目录的存在。
- 接着，调用 init_bar_generator 方法初始化 BarGenerator 对象，并进行相关的初始化操作。
- 创建一个 CzscTrader 对象，并将初始化好的 BarGenerator 对象、持仓策略的深拷贝、交易信号配置的深拷贝等参数传递给 CzscTrader 的构造函数。
- 为每个持仓策略创建相应的目录。
- 遍历 K 线数据，调用 trader.on_bar(bar) 方法处理每一根 K 线数据。
- **在每根 K 线数据处理完成后，检查每个持仓策略是否有操作，并且操作的时间是否与当前 K 线的时间一致**。
  如果有操作，则生成相应的 HTML 文件名，并调用 trader.take_snapshot(file_html) 方法生成交易快照。
- 最后，遍历每个持仓策略，记录其评估信息，包括多空合并表现、多头表现、空头表现等。

Parameters

- **bars** –基础周期 K 线
- **res_path** –结果目录
- **kwargs** –bg 已经初始化好的 BarGenerator 对象，如果传入了 bg，则忽略 sdt 和 n 参数 sdt 初始化开始日期 n 初始化最小 K 线数量 refresh 是否刷新结果目录

Returns

save_positions(*path*)

保存持仓策略配置

Parameters

path –结果路径

Returns

None

CzscStrategyExample2

class czsc.strategies.**CzscStrategyExample2**(***kwargs*)

Bases: *CzscStrategyBase*

仅传入 Positions 就完成策略创建

Attributes Summary

positions	持仓策略列表

Methods Summary

create_pos_a()	
create_pos_b()	从 json 文件 / dict 中加载 Position

Attributes Documentation

positions

Methods Documentation

create_pos_a()

create_pos_b()

从 json 文件 / dict 中加载 Position

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search

INDEX

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W

X

Y

Z