**【年盈利50%】热点龙头轮动策略**主要是捕捉热门题材中刚刚启动的龙头股，利用主升浪加速段获利。例如：

**①：热门题材，**机器人、AI国产替代加速、美国关税受损概念。  
**②：技术面上要求：**  
★股价站上20天线

★最近3天成交量放大至1.5倍以上

★MACD出现金叉（白线上穿黄线）  
**③：持仓管理**

●持仓数量：5~8只（不超过10只）

●单票仓位：初始10%~15%（1-1.5万元）

●调仓频率：每周一、三、五收盘前操作（符合≤3次要求）  
**④：买卖策略细则：**

1：所属板块当日涨幅前3名

2：分时图均价稳步上行，摒弃盘中火箭式的拉升或砸盘

3：早盘成交量达昨日全天50%以上。  
**卖出条件（满足任意一条）**

1：盈利达到20%之后，每上涨5%减仓1/3

2：单日跌幅超过5%，减仓一半

3：板块热度退出当日前5名  
**止损规则：**

●单票亏损达8%无条件清仓

●总账户回撤超过10%暂停交易一周

趁着“隔壁老王”喝的迷迷糊糊的时候，表哥赶紧把他说的量化公式偷偷记了下来：  
**《年盈利50%，热点龙头轮动策略》量化公式代码写入（电脑版通达信可直接导入）**// 买入信号量化公式

MA20 := MA(C, 20);  // 20日均线

MA5\_VOL := MA(V, 5);  // 5日成交量均线

VOL\_RATE := V / MA5\_VOL;  // 成交量相对强度（当前量/5日均量）

// 进场信号：价格突破20日均线 + 放量确认

BUY\_SIGNAL := CROSS(C, MA20) AND (VOL\_RATE > 1.2);

// 当日成交量超过5日均量20%时触发

// 绘图提示

DRAWTEXT(BUY\_SIGNAL, L\*0.98, '★出击', COLORRED, ALIGNLEFT);

//卖出信号量化公式：

N1:=5;    // 短期均线周期（默认5日）

N2:=10;   // 中期均线周期（默认10日）

M\_VOL:=1.5; // 成交量倍量阈值（默认1.5倍）

RSI\_PERIOD:=6; // RSI计算周期

// 均线系统

MA5:=MA(C,N1);

MA10:=MA(C,N2);

CROSS\_DOWN:=CROSS(MA10,MA5);  // 死叉条件

// 成交量过滤（防止假突破）

VOL\_AVG:=MA(VOL,5);  // 5日均量线

VOL\_COND:=VOL < VOL\_AVG\*0.8; // 当日成交量低于5日均量80%

// 超买修正（动态RSI阈值）

UPPER\_RSI:=70 - (LLV(L,30)-LLV(L,60))\*0.3; // 动态调整超买阈值

RSI1:=SMA(MAX(C-REF(C,1),0),RSI\_PERIOD,1)/SMA(ABS(C-REF(C,**1**)),RSI\_PERIOD,1)\*100;

OVERBought:=RSI1>UPPER\_RSI;  // 动态超买条件

// 综合卖出信号

SELL\_SIGNAL:=CROSS\_DOWN

AND VOL\_COND AND ORETBOUGHT;

// 信号标记与预警

DRAWICON(SELL\_SIGNAL,HIGH\*1.02,1);  // 1为向下箭头图标

DRAWTEXT(SELL\_SIGNAL,HIGH\*1.05,'▲止盈'),COLORRED;

DRAWTEXT\_CONDITION(SELL\_SIGNAL,HIGH\*1.03,'

[卖出]'),COLORWHITE;

**操作注意事项：1：**时间窗口：优先在10:00~10:30建仓**2：**资金分配：预留20%现金应对突发机会**3：**风险控制：单周盈利超过10%立即提取5%的利润。**4:**  此策略经过2024全年测试，年盈利在+63.8%，最大回撤-14.7%

# -\*- coding: utf-8 -\*-

import jqdata

import jqfactor # For sector data if needed via factors

import pandas as pd

import numpy as np

# Try importing talib, if available on JoinQuant, otherwise manual calculation is needed

try:

import talib

except ImportError:

print("TALib not found. MACD calculation will be manual or omitted if complex.")

talib = None # Set talib to None if import fails

# --- Strategy Initialization ---

def initialize(context):

"""

Strategy initialization function.

"""

# 1. Strategy Parameters from Description

g.hot\_sector\_top\_n = 3 # Buy condition: Sector rank Top N

g.sector\_fall\_out\_rank = 5 # Sell condition: Sector rank falls below N+1

g.ma\_period = 20 # Moving average period for price check

g.volume\_ma\_period = 5 # Moving average period for volume check

g.volume\_increase\_ratio = 1.5 # Required volume increase ratio (Text description)

g.macd\_fast = 12 # MACD fast EMA period

g.macd\_slow = 26 # MACD slow EMA period

g.macd\_signal = 9 # MACD signal DEA period

g.max\_hold\_stock\_num = 8 # Max number of stocks to hold (Text: 5-8, max 10)

g.initial\_position\_ratio = 0.12 # Initial position size (Text: 10-15%)

# Sell & Stop Loss Parameters from Description

g.take\_profit\_start\_pct = 0.20 # Start partial take profit at +20%

g.take\_profit\_step\_pct = 0.05 # Sell 1/3 for every additional 5% gain

g.take\_profit\_sell\_ratio = 1/3 # Ratio to sell at each take profit step

g.daily\_loss\_limit\_pct = 0.05 # Sell half if single day loss > 5%

g.stock\_stop\_loss\_pct = 0.08 # Unconditional stop loss per stock at -8%

g.account\_drawdown\_limit = 0.10 # Pause trading if account drawdown > 10%

# Operational Parameters

g.preferred\_buy\_start\_time = {'hour': 10, 'minute': 0}

g.preferred\_buy\_end\_time = {'hour': 10, 'minute': 30}

g.rebalance\_days = [1, 3, 5] # Monday, Wednesday, Friday (0=Mon, 1=Tue, ...)

g.rebalance\_time\_hour = 14

g.rebalance\_time\_minute = 50 # Rebalance near market close

g.cash\_reserve\_ratio = 0.20 # Keep 20% cash (influences initial position sizing indirectly)

# 2. System Settings

set\_option('use\_real\_price', True) # Use real-time prices for orders

set\_benchmark('000300.XSHG') # Set benchmark index

log.set\_level('order', 'info') # Set log level

# 3. Scheduling Functions

# Run daily before market open to prepare data

run\_daily(before\_market\_open, time='before\_open', reference\_security='000300.XSHG')

# Run intraday for buying opportunities

run\_daily(check\_buy\_signals, time='every\_minute', reference\_security='000300.XSHG')

# Run near market close for selling and rebalancing on specific days

run\_daily(check\_sell\_and\_rebalance, time='every\_minute', reference\_security='000300.XSHG')

# Run daily after market close for recording and drawdown check

run\_daily(after\_market\_close, time='after\_close', reference\_security='000300.XSHG')

# 4. Runtime Variables

g.trading\_paused = False

g.pause\_start\_date = None

g.peak\_portfolio\_value = context.portfolio.starting\_cash # Initialize with starting cash

g.bought\_today = [] # List of stocks bought today to avoid re-buying

# Dictionary to track take-profit status for each held stock

# Format: {stock\_code: {'sold\_steps': 0}} # 0 means nothing sold, 1 means 1/3 sold, etc.

g.take\_profit\_status = {}

# --- Scheduled Functions ---

def before\_market\_open(context):

"""

Run before market open daily.

Prepare candidate lists, check pause status.

"""

# Reset daily bought list

g.bought\_today = []

# Check if trading pause should end

if g.trading\_paused:

today = context.current\_dt.date()

if (today - g.pause\_start\_date).days >= 7:

g.trading\_paused = False

g.pause\_start\_date = None

log.info(f"Trading resumed after 1-week pause.")

# Reset peak value to current value after pause ends

g.peak\_portfolio\_value = context.portfolio.total\_value

else:

log.info(f"Trading is paused due to drawdown. Days remaining: {7 - (today - g.pause\_start\_date).days}")

return # Skip preparations if paused

# --- Get Candidate Stocks (Based on Yesterday's Close) ---

yesterday\_dt = context.previous\_date

# 1. Get all A-shares, filter out unwanted

all\_stocks = get\_all\_securities(['stock'], date=yesterday\_dt)

all\_stocks = filter\_undesired\_stocks(context, all\_stocks.index.tolist(), yesterday\_dt)

# 2. Fetch necessary historical data (adjust count as needed for indicators)

hist\_count = max(g.macd\_slow + g.macd\_signal, g.ma\_period, g.volume\_ma\_period) + 5 # Need enough data

if not all\_stocks:

g.candidate\_stocks = []

g.hot\_sectors = []

return

# Use attribute\_history for efficiency if possible, otherwise loop get\_price

# Note: attribute\_history might be slower for many indicators across all stocks

# Let's try a simplified approach: Calculate indicators stock by stock for potential candidates

# This is less efficient than batch processing but easier to implement here.

# --- Identify Hot Sectors (Based on Yesterday's performance) ---

g.hot\_sectors = get\_hot\_sectors(context, yesterday\_dt, top\_n=g.hot\_sector\_top\_n)

log.info(f"Yesterday's Top {g.hot\_sector\_top\_n} Hot Sectors: {g.hot\_sectors}")

# --- Pre-screen Stocks based on Technicals (Yesterday's Close) ---

potential\_buys = []

# Fetch data for all stocks (consider batching if too slow)

required\_fields = ['close', 'volume', 'high', 'low'] # Add 'high', 'low' if needed for MACD library

hist\_data = get\_price(all\_stocks, end\_date=yesterday\_dt, frequency='daily',

fields=required\_fields, count=hist\_count, panel=False)

for stock in all\_stocks:

stock\_data = hist\_data[hist\_data['code'] == stock]

if len(stock\_data) < hist\_count:

continue # Skip if insufficient data

# Technical Check 1: Price > MA20

ma20 = stock\_data['close'].rolling(window=g.ma\_period).mean().iloc[-1]

current\_price = stock\_data['close'].iloc[-1]

if current\_price <= ma20:

continue

# Technical Check 2: Volume Surge (V / MA(V,5) > 1.5)

vol\_ma5 = stock\_data['volume'].rolling(window=g.volume\_ma\_period).mean().iloc[-1]

current\_volume = stock\_data['volume'].iloc[-1]

if vol\_ma5 == 0 or (current\_volume / vol\_ma5) <= g.volume\_increase\_ratio:

continue

# Technical Check 3: MACD Golden Cross (or above signal line)

macd\_diff, macd\_dea, macd\_hist = calculate\_macd(stock\_data['close'],

g.macd\_fast, g.macd\_slow, g.macd\_signal)

if macd\_diff is None: continue # Skip if MACD calculation failed

# Using simple condition: MACD line is above signal line

if macd\_diff.iloc[-1] <= macd\_dea.iloc[-1]:

# Optional stricter check: Golden cross just happened

# if not (macd\_diff.iloc[-1] > macd\_dea.iloc[-1] and macd\_diff.iloc[-2] <= macd\_dea.iloc[-2]):

# continue

continue

# Check if stock belongs to a hot sector

sector\_code = get\_stock\_sector(stock, date=yesterday\_dt)

if sector\_code not in g.hot\_sectors:

continue

potential\_buys.append(stock)

g.candidate\_stocks = potential\_buys

log.info(f"Pre-screened candidates for today (based on yesterday): {len(g.candidate\_stocks)} stocks.")

def check\_buy\_signals(context):

"""

Run every minute during the preferred buying window to check intraday conditions.

"""

if g.trading\_paused: return # Skip if paused

now = context.current\_dt

# Check if within preferred buying time window

if not (now.hour == g.preferred\_buy\_start\_time['hour'] and now.minute >= g.preferred\_buy\_start\_time['minute'] and \

now.hour == g.preferred\_buy\_end\_time['hour'] and now.minute <= g.preferred\_buy\_end\_time['minute']):

# A more robust check for a time range:

time\_check = (now.hour > g.preferred\_buy\_start\_time['hour'] or \

(now.hour == g.preferred\_buy\_start\_time['hour'] and now.minute >= g.preferred\_buy\_start\_time['minute'])) and \

(now.hour < g.preferred\_buy\_end\_time['hour'] or \

(now.hour == g.preferred\_buy\_end\_time['hour'] and now.minute <= g.preferred\_buy\_end\_time['minute']))

if not time\_check:

return

# Check if portfolio is full

if len(context.portfolio.positions) >= g.max\_hold\_stock\_num:

return

# Iterate through pre-screened candidates

available\_cash\_for\_new\_positions = context.portfolio.available\_cash \* (1 - g.cash\_reserve\_ratio)

position\_value = context.portfolio.total\_value \* g.initial\_position\_ratio

if available\_cash\_for\_new\_positions < position\_value and len(context.portfolio.positions) > 0 :

# Recalculate based on available cash if initial ratio is too high

position\_value = available\_cash\_for\_new\_positions / (g.max\_hold\_stock\_num - len(context.portfolio.positions))

# Ensure minimum position value to avoid tiny orders

if position\_value < 1000: # Example minimum

return

stocks\_to\_buy = g.candidate\_stocks # Use pre-screened list

if not stocks\_to\_buy: return

# Shuffle candidates to avoid always picking the same ones if many qualify

import random

random.shuffle(stocks\_to\_buy)

for stock in stocks\_to\_buy:

# Check portfolio limits again inside loop

if len(context.portfolio.positions) >= g.max\_hold\_stock\_num: break

if stock in context.portfolio.positions or stock in g.bought\_today: continue

# --- Intraday Checks ---

# Get current price data

current\_data = get\_current\_data([stock])[stock]

if current\_data is None or current\_data.paused or current\_data.last\_price == 0:

continue

current\_price = current\_data.last\_price

# Check 1: Morning Volume > 50% of Yesterday's Total Volume

yesterday\_data = get\_price(stock, end\_date=context.previous\_date, frequency='daily', fields=['volume'], count=1)

if yesterday\_data.empty: continue

yesterday\_volume = yesterday\_data['volume'].iloc[0]

# Get today's volume up to now

# Use get\_bars for more accuracy if needed, get\_current\_data.volume might suffice

today\_volume\_so\_far = current\_data.volume

if yesterday\_volume == 0 or (today\_volume\_so\_far / yesterday\_volume) < 0.5:

continue

# Check 2: Intraday price stability (Simple: Price > VWAP)

# VWAP requires minute data, which can be slow. Use current price > open price as a proxy?

# Or use get\_bars for recent minutes

# Simple proxy: current price > day's open price

if current\_price <= current\_data.day\_open:

continue

# A better check might involve get\_bars for the last N minutes and check for huge spikes.

# Check 3: Re-confirm sector rank (optional, as pre-screened)

# sector\_code = get\_stock\_sector(stock, date=context.current\_dt.date())

# if sector\_code not in g.hot\_sectors: # Need to recalculate hot\_sectors intraday ideally

# continue

# --- Execute Buy ---

cash\_needed = position\_value

if context.portfolio.available\_cash >= cash\_needed:

log.info(f"BUYING SIGNAL: {stock} at {current\_price:.2f}. Conditions met.")

order\_value(stock, position\_value)

g.bought\_today.append(stock)

g.take\_profit\_status[stock] = {'sold\_steps': 0} # Initialize take-profit status

else:

log.warning(f"Skipping buy for {stock}: Insufficient available cash ({context.portfolio.available\_cash:.2f}) for target value ({position\_value:.2f})")

# Optional: Buy smaller amount based on available cash

# adjusted\_value = context.portfolio.available\_cash \* 0.98 # Leave buffer

# if adjusted\_value > 1000: # Min order value

# order\_value(stock, adjusted\_value)

# g.bought\_today.append(stock)

# g.take\_profit\_status[stock] = {'sold\_steps': 0}

def check\_sell\_and\_rebalance(context):

"""

Run every minute near market close on rebalancing days to check sell conditions.

"""

if g.trading\_paused: return # Skip if paused

now = context.current\_dt

today\_weekday = now.weekday() # Monday is 0, Sunday is 6

# Check if it's a rebalancing day and time

if today\_weekday not in g.rebalance\_days: return

if not (now.hour == g.rebalance\_time\_hour and now.minute >= g.rebalance\_time\_minute): return

log.info(f"Running Sell and Rebalance Check on {now.strftime('%Y-%m-%d %H:%M')}")

held\_stocks = list(context.portfolio.positions.keys())

if not held\_stocks: return

# Get current prices for held stocks

current\_prices = get\_price(held\_stocks, end\_date=context.current\_dt, frequency='1m', fields='close', count=1)

if current\_prices.empty:

log.warning("Could not get current prices for sell check.")

return

# Get yesterday's close prices for daily loss check

yesterday\_closes = get\_price(held\_stocks, end\_date=context.previous\_date, frequency='daily', fields='close', count=1)

# Recalculate Hot Sectors based on today's (almost) closing performance

current\_hot\_sectors = get\_hot\_sectors(context, context.current\_dt.date(), top\_n=g.sector\_fall\_out\_rank)

stocks\_to\_sell = {} # Store stock and reason {'stock': 'reason'}

for stock in held\_stocks:

position = context.portfolio.positions[stock]

if position.closeable\_amount <= 0: continue # Cannot sell

current\_price = current\_prices.loc[stock, 'close'] if stock in current\_prices.index else 0

# If price is somehow missing, use position's price, but log warning

if current\_price == 0:

current\_price = position.price

log.warning(f"Using position price {current\_price} for {stock} as current quote missing.")

avg\_cost = position.avg\_cost

if avg\_cost == 0: continue # Should not happen for existing position

profit\_pct = (current\_price / avg\_cost) - 1

# --- Check Sell Conditions ---

# 1. Stop Loss (Individual Stock)

if profit\_pct <= -g.stock\_stop\_loss\_pct:

stocks\_to\_sell[stock] = f"Stop Loss triggered ({profit\_pct\*100:.2f}%)"

continue # Stop loss takes priority

# 2. Daily Loss Limit

if stock in yesterday\_closes.index:

yesterday\_close = yesterday\_closes.loc[stock, 'close']

daily\_change\_pct = (current\_price / yesterday\_close) - 1

if daily\_change\_pct <= -g.daily\_loss\_limit\_pct:

# Sell half

log.info(f"Daily Loss Sell Triggered for {stock} ({daily\_change\_pct\*100:.2f}%). Selling half.")

order\_target(stock, position.total\_amount / 2) # Target half the shares

# Note: Need to handle potential concurrent sell signals carefully

# This partial sell might conflict with other full sell signals below.

# Let's prioritize full sells. If a full sell condition is met, it overrides this.

# If only this is met, sell half. Revisit this logic if needed.

# For simplicity now, let's make this a full sell signal too, or handle partials later.

# Let's make it a full sell for now to simplify state management.

# stocks\_to\_sell[stock] = f"Daily Loss Limit ({daily\_change\_pct\*100:.2f}%) - Selling All"

# --- OR --- implement partial selling properly:

if stock not in stocks\_to\_sell: # Check if already marked for full sell

target\_amount = position.total\_amount / 2

order\_target(stock, target\_amount)

log.info(f"Selling half of {stock} due to daily loss > {g.daily\_loss\_limit\_pct\*100}%")

# Continue checking other conditions for the remaining half potentially

# 3. Sector Rank Drop

sector\_code = get\_stock\_sector(stock, date=context.current\_dt.date())

if sector\_code not in current\_hot\_sectors:

# Check if the sector exists but is just not in top N, or if sector info missing

all\_sector\_ranks = get\_sector\_performance(context, context.current\_dt.date()) # Need this helper

if sector\_code in all\_sector\_ranks and all\_sector\_ranks[sector\_code]['rank'] > g.sector\_fall\_out\_rank:

stocks\_to\_sell[stock] = f"Sector Rank dropped below {g.sector\_fall\_out\_rank+1}"

continue

# 4. Take Profit (Partial Selling)

if stock not in stocks\_to\_sell and stock in g.take\_profit\_status: # Ensure not already marked for full sell

status = g.take\_profit\_status[stock]

required\_profit\_levels = [g.take\_profit\_start\_pct + i \* g.take\_profit\_step\_pct for i in range(int(1 / g.take\_profit\_sell\_ratio))]

steps\_to\_sell = 0

for i, level in enumerate(required\_profit\_levels):

if profit\_pct >= level and status['sold\_steps'] <= i:

steps\_to\_sell = i + 1 # Target selling up to this step

sell\_count = steps\_to\_sell - status['sold\_steps'] # How many new steps to sell now

if sell\_count > 0:

sell\_ratio = sell\_count \* g.take\_profit\_sell\_ratio

target\_amount = position.total\_amount \* (1 - sell\_ratio) # Reduce position by sell\_ratio

# Ensure we don't sell more than available

amount\_to\_sell = position.total\_amount \* sell\_ratio

if amount\_to\_sell > position.closeable\_amount:

amount\_to\_sell = position.closeable\_amount

target\_amount = position.total\_amount - amount\_to\_sell

if amount\_to\_sell > 0:

log.info(f"Take Profit Triggered for {stock} (Profit: {profit\_pct\*100:.2f}%). Selling {sell\_ratio\*100:.1f}% of original position ({sell\_count} steps).")

order\_target(stock, target\_amount)

# Update status AFTER order is placed (or assume success for backtesting)

g.take\_profit\_status[stock]['sold\_steps'] = steps\_to\_sell

# Execute full liquidations for stocks marked in stocks\_to\_sell

for stock, reason in stocks\_to\_sell.items():

if context.portfolio.positions[stock].closeable\_amount > 0:

log.info(f"SELLING SIGNAL: {stock} due to: {reason}")

order\_target(stock, 0) # Liquidate position

# Clean up take profit status if liquidated

if stock in g.take\_profit\_status:

del g.take\_profit\_status[stock]

def after\_market\_close(context):

"""

Run after market close daily.

Record metrics, check account drawdown.

"""

# Record daily portfolio value and cash

record(portfolio\_value=context.portfolio.total\_value, cash=context.portfolio.available\_cash)

# Update peak portfolio value

g.peak\_portfolio\_value = max(g.peak\_portfolio\_value, context.portfolio.total\_value)

# Check Account Drawdown - only if not already paused

if not g.trading\_paused:

drawdown = (context.portfolio.total\_value / g.peak\_portfolio\_value) - 1

if drawdown < -g.account\_drawdown\_limit:

log.warning(f"ACCOUNT DRAWDOWN LIMIT REACHED: {drawdown\*100:.2f}%. Current Value: {context.portfolio.total\_value:.2f}, Peak Value: {g.peak\_portfolio\_value:.2f}")

g.trading\_paused = True

g.pause\_start\_date = context.current\_dt.date()

log.warning("--- Initiating Trading Pause for 1 week. Liquidating all positions. ---")

for stock in list(context.portfolio.positions.keys()):

if context.portfolio.positions[stock].closeable\_amount > 0:

order\_target(stock, 0)

# Clear take profit status on full liquidation

g.take\_profit\_status.clear()

# Clean up stocks bought today list

# g.bought\_today = [] # Moved to before\_market\_open

# Clean up take\_profit\_status for stocks no longer held

held\_stocks = set(context.portfolio.positions.keys())

keys\_to\_delete = [stock for stock in g.take\_profit\_status if stock not in held\_stocks]

for stock in keys\_to\_delete:

del g.take\_profit\_status[stock]

log.info(f"End of Day {context.current\_dt.date()}. Portfolio Value: {context.portfolio.total\_value:.2f}")

log.info(f"Current Positions: {list(context.portfolio.positions.keys())}")

log.info("-" \* 50)

# --- Helper Functions ---

def filter\_undesired\_stocks(context, stock\_list, date):

"""

Filters out ST, \*ST, PT, new stocks, suspended stocks.

"""

current\_data = get\_current\_data(stock\_list)

filtered\_list = []

for stock in stock\_list:

stock\_info = current\_data[stock]

if stock\_info is None: continue

if stock\_info.paused: continue # Suspended

if stock\_info.is\_st: continue # ST or \*ST

# Check for PT status if needed (less common now)

# Check if IPO within last N days (e.g., 60 days)

days\_public = (date - get\_security\_info(stock).start\_date).days

if days\_public < 60: continue # Skip stocks listed less than 60 days

# Filter out stocks starting with '688' (STAR Market) or '300' (ChiNext) if desired

# if stock.startswith('688') or stock.startswith('300'): continue

filtered\_list.append(stock)

# Additional check for fundamentals data availability if needed

# q = query(valuation.code).filter(valuation.code.in\_(filtered\_list))

# fundamentals = get\_fundamentals(q, date=date)

# return fundamentals['code'].tolist()

return filtered\_list

def calculate\_macd(close\_prices, fast\_period, slow\_period, signal\_period):

"""

Calculates MACD indicator values using pandas.

Returns (diff, dea, hist) as pandas Series, or (None, None, None) if error.

"""

if len(close\_prices) < slow\_period + signal\_period:

return None, None, None # Not enough data

try:

# If TALib is available and works on JoinQuant:

if talib:

diff, dea, hist = talib.MACD(close\_prices.values,

fastperiod=fast\_period,

slowperiod=slow\_period,

signalperiod=signal\_period)

# Convert numpy arrays back to pandas Series with correct index

index = close\_prices.index

diff\_series = pd.Series(diff, index=index, name='DIFF')

dea\_series = pd.Series(dea, index=index, name='DEA')

hist\_series = pd.Series(hist, index=index, name='MACD\_Hist')

return diff\_series, dea\_series, hist\_series

else:

# Manual calculation if TALib not available

ema\_fast = close\_prices.ewm(span=fast\_period, adjust=False).mean()

ema\_slow = close\_prices.ewm(span=slow\_period, adjust=False).mean()

diff\_series = ema\_fast - ema\_slow

dea\_series = diff\_series.ewm(span=signal\_period, adjust=False).mean()

hist\_series = (diff\_series - dea\_series) \* 2 # Common representation

return diff\_series, dea\_series, hist\_series

except Exception as e:

log.error(f"Error calculating MACD: {e}")

return None, None, None

def get\_stock\_sector(stock\_code, date):

"""

Gets the SW L1 industry code for a given stock.

Adjust industry classification standard if needed (e.g., 'zjw', 'jq\_l1').

"""

try:

# Using Shenwan L1 as an example

industry\_info = jqdata.get\_industry(security=stock\_code, date=date)

if industry\_info and stock\_code in industry\_info:

# Structure might be {stock: {'sw\_l1': {'industry\_code': 'xxx', 'industry\_name': 'yyy'}}}

if 'sw\_l1' in industry\_info[stock\_code] and industry\_info[stock\_code]['sw\_l1']:

return industry\_info[stock\_code]['sw\_l1']['industry\_code']

# Add checks for other standards like 'jq\_l1' if needed

# elif 'jq\_l1' in industry\_info[stock\_code] and industry\_info[stock\_code]['jq\_l1']:

# return industry\_info[stock\_code]['jq\_l1']['industry\_code']

except Exception as e:

log.error(f"Error getting sector for {stock\_code}: {e}")

return None

def get\_sector\_performance(context, date, industry\_standard='sw\_l1'):

"""

Calculates the average performance of each sector for the given date.

Returns a dictionary: {sector\_code: {'avg\_change': float, 'rank': int, 'name': str}}

NOTE: This can be computationally expensive. Consider optimizations or using pre-calculated factors if available.

"""

sector\_perf = {}

try:

industries = get\_industries(name=industry\_standard, date=date)

all\_sector\_stocks = []

sector\_map = {}

for code, row in industries.iterrows():

stocks = get\_industry\_stocks(code, date=date)

if stocks:

sector\_perf[code] = {'total\_change': 0.0, 'count': 0, 'name': row['name']}

for s in stocks:

sector\_map[s] = code

all\_sector\_stocks.extend(stocks)

if not all\_sector\_stocks: return {}

# Get price change for all relevant stocks

# Calculate change from previous close to current price/close

prices\_today = get\_price(all\_sector\_stocks, end\_date=date, frequency='daily', fields=['close'], count=1)

prices\_yest = get\_price(all\_sector\_stocks, end\_date=context.previous\_date, frequency='daily', fields=['close'], count=1)

if prices\_today.empty or prices\_yest.empty: return {}

# Merge prices

price\_data = pd.merge(prices\_today.reset\_index()[['code','close']],

prices\_yest.reset\_index()[['code','close']],

on='code', suffixes=('\_today', '\_yest'))

price\_data['change'] = (price\_data['close\_today'] / price\_data['close\_yest'] - 1) \* 100

# Aggregate by sector

for index, row in price\_data.iterrows():

stock = row['code']

change = row['change']

if stock in sector\_map:

sector\_code = sector\_map[stock]

if pd.notna(change): # Check for NaN changes

sector\_perf[sector\_code]['total\_change'] += change

sector\_perf[sector\_code]['count'] += 1

# Calculate average change and filter out sectors with no valid data

valid\_sectors = {}

for code, data in sector\_perf.items():

if data['count'] > 0:

avg\_change = data['total\_change'] / data['count']

valid\_sectors[code] = {'avg\_change': avg\_change, 'name': data['name']}

# Rank sectors

ranked\_sectors = sorted(valid\_sectors.items(), key=lambda item: item[1]['avg\_change'], reverse=True)

result = {}

for rank, (code, data) in enumerate(ranked\_sectors):

result[code] = {'avg\_change': data['avg\_change'], 'rank': rank + 1, 'name': data['name']}

return result

except Exception as e:

log.error(f"Error calculating sector performance: {e}")

return {}

def get\_hot\_sectors(context, date, top\_n=3):

"""

Gets the codes of the top N performing sectors based on average daily change.

"""

sector\_ranks = get\_sector\_performance(context, date)

hot\_sectors = [code for code, data in sector\_ranks.items() if data['rank'] <= top\_n]

return hot\_sectors

Okay, let's translate the "Hotspot Leader Rotation Strategy" (热点龙头轮动策略) described in the text and partially reflected in the TongDaXin (TDX) formulas into Python code suitable for the JoinQuant (聚宽) platform.

**Analysis Summary:**

1. **Core Idea:** Buy technically strong stocks (Price > MA20, Volume Surge, MACD Golden Cross) within top-performing "hot" sectors.
2. **Buy Triggers:** Combination of TDX formula (CROSS(C, MA20) & V/MA(V,5) > 1.2) and text description (Price > MA20, Recent Volume > 1.5x, MACD Golden Cross). We'll combine these, favoring the text's specifics where available (e.g., 1.5x volume). Add sector rank, intraday stability, and morning volume checks.
3. **Sell Triggers:** Primarily from the text description (Profit target partial selling, Daily % loss, Sector rank drop, 8% stop loss). The TDX sell formula is complex (dynamic RSI, specific MA cross) and seems less aligned with the main description's sell rules, so we will prioritize the text-based rules.
4. **Position Management:** 5-8 stocks, 10-15% initial allocation, rebalancing Mon/Wed/Fri.
5. **Risk Control:** 8% single stock stop loss, 10% total drawdown pause.
6. **Provided Python Code:** The uploaded file (龙回头量化选股策略-聚宽Python代码.txt) implements a *different* strategy ("Dragon Head Turnaround"). We will use it for *syntax reference* but create *new logic* based on the "Hotspot Leader Rotation" description.

**JoinQuant Python Strategy Code:**

Python

# -\*- coding: utf-8 -\*-

import jqdata

import jqfactor # For sector data if needed via factors

import pandas as pd

import numpy as np

# Try importing talib, if available on JoinQuant, otherwise manual calculation is needed

try:

import talib

except ImportError:

print("TALib not found. MACD calculation will be manual or omitted if complex.")

talib = None # Set talib to None if import fails

# --- Strategy Initialization ---

def initialize(context):

"""

Strategy initialization function.

"""

# 1. Strategy Parameters from Description

g.hot\_sector\_top\_n = 3 # Buy condition: Sector rank Top N

g.sector\_fall\_out\_rank = 5 # Sell condition: Sector rank falls below N+1

g.ma\_period = 20 # Moving average period for price check

g.volume\_ma\_period = 5 # Moving average period for volume check

g.volume\_increase\_ratio = 1.5 # Required volume increase ratio (Text description)

g.macd\_fast = 12 # MACD fast EMA period

g.macd\_slow = 26 # MACD slow EMA period

g.macd\_signal = 9 # MACD signal DEA period

g.max\_hold\_stock\_num = 8 # Max number of stocks to hold (Text: 5-8, max 10)

g.initial\_position\_ratio = 0.12 # Initial position size (Text: 10-15%)

# Sell & Stop Loss Parameters from Description

g.take\_profit\_start\_pct = 0.20 # Start partial take profit at +20%

g.take\_profit\_step\_pct = 0.05 # Sell 1/3 for every additional 5% gain

g.take\_profit\_sell\_ratio = 1/3 # Ratio to sell at each take profit step

g.daily\_loss\_limit\_pct = 0.05 # Sell half if single day loss > 5%

g.stock\_stop\_loss\_pct = 0.08 # Unconditional stop loss per stock at -8%

g.account\_drawdown\_limit = 0.10 # Pause trading if account drawdown > 10%

# Operational Parameters

g.preferred\_buy\_start\_time = {'hour': 10, 'minute': 0}

g.preferred\_buy\_end\_time = {'hour': 10, 'minute': 30}

g.rebalance\_days = [1, 3, 5] # Monday, Wednesday, Friday (0=Mon, 1=Tue, ...)

g.rebalance\_time\_hour = 14

g.rebalance\_time\_minute = 50 # Rebalance near market close

g.cash\_reserve\_ratio = 0.20 # Keep 20% cash (influences initial position sizing indirectly)

# 2. System Settings

set\_option('use\_real\_price', True) # Use real-time prices for orders

set\_benchmark('000300.XSHG') # Set benchmark index

log.set\_level('order', 'info') # Set log level

# 3. Scheduling Functions

# Run daily before market open to prepare data

run\_daily(before\_market\_open, time='before\_open', reference\_security='000300.XSHG')

# Run intraday for buying opportunities

run\_daily(check\_buy\_signals, time='every\_minute', reference\_security='000300.XSHG')

# Run near market close for selling and rebalancing on specific days

run\_daily(check\_sell\_and\_rebalance, time='every\_minute', reference\_security='000300.XSHG')

# Run daily after market close for recording and drawdown check

run\_daily(after\_market\_close, time='after\_close', reference\_security='000300.XSHG')

# 4. Runtime Variables

g.trading\_paused = False

g.pause\_start\_date = None

g.peak\_portfolio\_value = context.portfolio.starting\_cash # Initialize with starting cash

g.bought\_today = [] # List of stocks bought today to avoid re-buying

# Dictionary to track take-profit status for each held stock

# Format: {stock\_code: {'sold\_steps': 0}} # 0 means nothing sold, 1 means 1/3 sold, etc.

g.take\_profit\_status = {}

# --- Scheduled Functions ---

def before\_market\_open(context):

"""

Run before market open daily.

Prepare candidate lists, check pause status.

"""

# Reset daily bought list

g.bought\_today = []

# Check if trading pause should end

if g.trading\_paused:

today = context.current\_dt.date()

if (today - g.pause\_start\_date).days >= 7:

g.trading\_paused = False

g.pause\_start\_date = None

log.info(f"Trading resumed after 1-week pause.")

# Reset peak value to current value after pause ends

g.peak\_portfolio\_value = context.portfolio.total\_value

else:

log.info(f"Trading is paused due to drawdown. Days remaining: {7 - (today - g.pause\_start\_date).days}")

return # Skip preparations if paused

# --- Get Candidate Stocks (Based on Yesterday's Close) ---

yesterday\_dt = context.previous\_date

# 1. Get all A-shares, filter out unwanted

all\_stocks = get\_all\_securities(['stock'], date=yesterday\_dt)

all\_stocks = filter\_undesired\_stocks(context, all\_stocks.index.tolist(), yesterday\_dt)

# 2. Fetch necessary historical data (adjust count as needed for indicators)

hist\_count = max(g.macd\_slow + g.macd\_signal, g.ma\_period, g.volume\_ma\_period) + 5 # Need enough data

if not all\_stocks:

g.candidate\_stocks = []

g.hot\_sectors = []

return

# Use attribute\_history for efficiency if possible, otherwise loop get\_price

# Note: attribute\_history might be slower for many indicators across all stocks

# Let's try a simplified approach: Calculate indicators stock by stock for potential candidates

# This is less efficient than batch processing but easier to implement here.

# --- Identify Hot Sectors (Based on Yesterday's performance) ---

g.hot\_sectors = get\_hot\_sectors(context, yesterday\_dt, top\_n=g.hot\_sector\_top\_n)

log.info(f"Yesterday's Top {g.hot\_sector\_top\_n} Hot Sectors: {g.hot\_sectors}")

# --- Pre-screen Stocks based on Technicals (Yesterday's Close) ---

potential\_buys = []

# Fetch data for all stocks (consider batching if too slow)

required\_fields = ['close', 'volume', 'high', 'low'] # Add 'high', 'low' if needed for MACD library

hist\_data = get\_price(all\_stocks, end\_date=yesterday\_dt, frequency='daily',

fields=required\_fields, count=hist\_count, panel=False)

for stock in all\_stocks:

stock\_data = hist\_data[hist\_data['code'] == stock]

if len(stock\_data) < hist\_count:

continue # Skip if insufficient data

# Technical Check 1: Price > MA20

ma20 = stock\_data['close'].rolling(window=g.ma\_period).mean().iloc[-1]

current\_price = stock\_data['close'].iloc[-1]

if current\_price <= ma20:

continue

# Technical Check 2: Volume Surge (V / MA(V,5) > 1.5)

vol\_ma5 = stock\_data['volume'].rolling(window=g.volume\_ma\_period).mean().iloc[-1]

current\_volume = stock\_data['volume'].iloc[-1]

if vol\_ma5 == 0 or (current\_volume / vol\_ma5) <= g.volume\_increase\_ratio:

continue

# Technical Check 3: MACD Golden Cross (or above signal line)

macd\_diff, macd\_dea, macd\_hist = calculate\_macd(stock\_data['close'],

g.macd\_fast, g.macd\_slow, g.macd\_signal)

if macd\_diff is None: continue # Skip if MACD calculation failed

# Using simple condition: MACD line is above signal line

if macd\_diff.iloc[-1] <= macd\_dea.iloc[-1]:

# Optional stricter check: Golden cross just happened

# if not (macd\_diff.iloc[-1] > macd\_dea.iloc[-1] and macd\_diff.iloc[-2] <= macd\_dea.iloc[-2]):

# continue

continue

# Check if stock belongs to a hot sector

sector\_code = get\_stock\_sector(stock, date=yesterday\_dt)

if sector\_code not in g.hot\_sectors:

continue

potential\_buys.append(stock)

g.candidate\_stocks = potential\_buys

log.info(f"Pre-screened candidates for today (based on yesterday): {len(g.candidate\_stocks)} stocks.")

def check\_buy\_signals(context):

"""

Run every minute during the preferred buying window to check intraday conditions.

"""

if g.trading\_paused: return # Skip if paused

now = context.current\_dt

# Check if within preferred buying time window

if not (now.hour == g.preferred\_buy\_start\_time['hour'] and now.minute >= g.preferred\_buy\_start\_time['minute'] and \

now.hour == g.preferred\_buy\_end\_time['hour'] and now.minute <= g.preferred\_buy\_end\_time['minute']):

# A more robust check for a time range:

time\_check = (now.hour > g.preferred\_buy\_start\_time['hour'] or \

(now.hour == g.preferred\_buy\_start\_time['hour'] and now.minute >= g.preferred\_buy\_start\_time['minute'])) and \

(now.hour < g.preferred\_buy\_end\_time['hour'] or \

(now.hour == g.preferred\_buy\_end\_time['hour'] and now.minute <= g.preferred\_buy\_end\_time['minute']))

if not time\_check:

return

# Check if portfolio is full

if len(context.portfolio.positions) >= g.max\_hold\_stock\_num:

return

# Iterate through pre-screened candidates

available\_cash\_for\_new\_positions = context.portfolio.available\_cash \* (1 - g.cash\_reserve\_ratio)

position\_value = context.portfolio.total\_value \* g.initial\_position\_ratio

if available\_cash\_for\_new\_positions < position\_value and len(context.portfolio.positions) > 0 :

# Recalculate based on available cash if initial ratio is too high

position\_value = available\_cash\_for\_new\_positions / (g.max\_hold\_stock\_num - len(context.portfolio.positions))

# Ensure minimum position value to avoid tiny orders

if position\_value < 1000: # Example minimum

return

stocks\_to\_buy = g.candidate\_stocks # Use pre-screened list

if not stocks\_to\_buy: return

# Shuffle candidates to avoid always picking the same ones if many qualify

import random

random.shuffle(stocks\_to\_buy)

for stock in stocks\_to\_buy:

# Check portfolio limits again inside loop

if len(context.portfolio.positions) >= g.max\_hold\_stock\_num: break

if stock in context.portfolio.positions or stock in g.bought\_today: continue

# --- Intraday Checks ---

# Get current price data

current\_data = get\_current\_data([stock])[stock]

if current\_data is None or current\_data.paused or current\_data.last\_price == 0:

continue

current\_price = current\_data.last\_price

# Check 1: Morning Volume > 50% of Yesterday's Total Volume

yesterday\_data = get\_price(stock, end\_date=context.previous\_date, frequency='daily', fields=['volume'], count=1)

if yesterday\_data.empty: continue

yesterday\_volume = yesterday\_data['volume'].iloc[0]

# Get today's volume up to now

# Use get\_bars for more accuracy if needed, get\_current\_data.volume might suffice

today\_volume\_so\_far = current\_data.volume

if yesterday\_volume == 0 or (today\_volume\_so\_far / yesterday\_volume) < 0.5:

continue

# Check 2: Intraday price stability (Simple: Price > VWAP)

# VWAP requires minute data, which can be slow. Use current price > open price as a proxy?

# Or use get\_bars for recent minutes

# Simple proxy: current price > day's open price

if current\_price <= current\_data.day\_open:

continue

# A better check might involve get\_bars for the last N minutes and check for huge spikes.

# Check 3: Re-confirm sector rank (optional, as pre-screened)

# sector\_code = get\_stock\_sector(stock, date=context.current\_dt.date())

# if sector\_code not in g.hot\_sectors: # Need to recalculate hot\_sectors intraday ideally

# continue

# --- Execute Buy ---

cash\_needed = position\_value

if context.portfolio.available\_cash >= cash\_needed:

log.info(f"BUYING SIGNAL: {stock} at {current\_price:.2f}. Conditions met.")

order\_value(stock, position\_value)

g.bought\_today.append(stock)

g.take\_profit\_status[stock] = {'sold\_steps': 0} # Initialize take-profit status

else:

log.warning(f"Skipping buy for {stock}: Insufficient available cash ({context.portfolio.available\_cash:.2f}) for target value ({position\_value:.2f})")

# Optional: Buy smaller amount based on available cash

# adjusted\_value = context.portfolio.available\_cash \* 0.98 # Leave buffer

# if adjusted\_value > 1000: # Min order value

# order\_value(stock, adjusted\_value)

# g.bought\_today.append(stock)

# g.take\_profit\_status[stock] = {'sold\_steps': 0}

def check\_sell\_and\_rebalance(context):

"""

Run every minute near market close on rebalancing days to check sell conditions.

"""

if g.trading\_paused: return # Skip if paused

now = context.current\_dt

today\_weekday = now.weekday() # Monday is 0, Sunday is 6

# Check if it's a rebalancing day and time

if today\_weekday not in g.rebalance\_days: return

if not (now.hour == g.rebalance\_time\_hour and now.minute >= g.rebalance\_time\_minute): return

log.info(f"Running Sell and Rebalance Check on {now.strftime('%Y-%m-%d %H:%M')}")

held\_stocks = list(context.portfolio.positions.keys())

if not held\_stocks: return

# Get current prices for held stocks

current\_prices = get\_price(held\_stocks, end\_date=context.current\_dt, frequency='1m', fields='close', count=1)

if current\_prices.empty:

log.warning("Could not get current prices for sell check.")

return

# Get yesterday's close prices for daily loss check

yesterday\_closes = get\_price(held\_stocks, end\_date=context.previous\_date, frequency='daily', fields='close', count=1)

# Recalculate Hot Sectors based on today's (almost) closing performance

current\_hot\_sectors = get\_hot\_sectors(context, context.current\_dt.date(), top\_n=g.sector\_fall\_out\_rank)

stocks\_to\_sell = {} # Store stock and reason {'stock': 'reason'}

for stock in held\_stocks:

position = context.portfolio.positions[stock]

if position.closeable\_amount <= 0: continue # Cannot sell

current\_price = current\_prices.loc[stock, 'close'] if stock in current\_prices.index else 0

# If price is somehow missing, use position's price, but log warning

if current\_price == 0:

current\_price = position.price

log.warning(f"Using position price {current\_price} for {stock} as current quote missing.")

avg\_cost = position.avg\_cost

if avg\_cost == 0: continue # Should not happen for existing position

profit\_pct = (current\_price / avg\_cost) - 1

# --- Check Sell Conditions ---

# 1. Stop Loss (Individual Stock)

if profit\_pct <= -g.stock\_stop\_loss\_pct:

stocks\_to\_sell[stock] = f"Stop Loss triggered ({profit\_pct\*100:.2f}%)"

continue # Stop loss takes priority

# 2. Daily Loss Limit

if stock in yesterday\_closes.index:

yesterday\_close = yesterday\_closes.loc[stock, 'close']

daily\_change\_pct = (current\_price / yesterday\_close) - 1

if daily\_change\_pct <= -g.daily\_loss\_limit\_pct:

# Sell half

log.info(f"Daily Loss Sell Triggered for {stock} ({daily\_change\_pct\*100:.2f}%). Selling half.")

order\_target(stock, position.total\_amount / 2) # Target half the shares

# Note: Need to handle potential concurrent sell signals carefully

# This partial sell might conflict with other full sell signals below.

# Let's prioritize full sells. If a full sell condition is met, it overrides this.

# If only this is met, sell half. Revisit this logic if needed.

# For simplicity now, let's make this a full sell signal too, or handle partials later.

# Let's make it a full sell for now to simplify state management.

# stocks\_to\_sell[stock] = f"Daily Loss Limit ({daily\_change\_pct\*100:.2f}%) - Selling All"

# --- OR --- implement partial selling properly:

if stock not in stocks\_to\_sell: # Check if already marked for full sell

target\_amount = position.total\_amount / 2

order\_target(stock, target\_amount)

log.info(f"Selling half of {stock} due to daily loss > {g.daily\_loss\_limit\_pct\*100}%")

# Continue checking other conditions for the remaining half potentially

# 3. Sector Rank Drop

sector\_code = get\_stock\_sector(stock, date=context.current\_dt.date())

if sector\_code not in current\_hot\_sectors:

# Check if the sector exists but is just not in top N, or if sector info missing

all\_sector\_ranks = get\_sector\_performance(context, context.current\_dt.date()) # Need this helper

if sector\_code in all\_sector\_ranks and all\_sector\_ranks[sector\_code]['rank'] > g.sector\_fall\_out\_rank:

stocks\_to\_sell[stock] = f"Sector Rank dropped below {g.sector\_fall\_out\_rank+1}"

continue

# 4. Take Profit (Partial Selling)

if stock not in stocks\_to\_sell and stock in g.take\_profit\_status: # Ensure not already marked for full sell

status = g.take\_profit\_status[stock]

required\_profit\_levels = [g.take\_profit\_start\_pct + i \* g.take\_profit\_step\_pct for i in range(int(1 / g.take\_profit\_sell\_ratio))]

steps\_to\_sell = 0

for i, level in enumerate(required\_profit\_levels):

if profit\_pct >= level and status['sold\_steps'] <= i:

steps\_to\_sell = i + 1 # Target selling up to this step

sell\_count = steps\_to\_sell - status['sold\_steps'] # How many new steps to sell now

if sell\_count > 0:

sell\_ratio = sell\_count \* g.take\_profit\_sell\_ratio

target\_amount = position.total\_amount \* (1 - sell\_ratio) # Reduce position by sell\_ratio

# Ensure we don't sell more than available

amount\_to\_sell = position.total\_amount \* sell\_ratio

if amount\_to\_sell > position.closeable\_amount:

amount\_to\_sell = position.closeable\_amount

target\_amount = position.total\_amount - amount\_to\_sell

if amount\_to\_sell > 0:

log.info(f"Take Profit Triggered for {stock} (Profit: {profit\_pct\*100:.2f}%). Selling {sell\_ratio\*100:.1f}% of original position ({sell\_count} steps).")

order\_target(stock, target\_amount)

# Update status AFTER order is placed (or assume success for backtesting)

g.take\_profit\_status[stock]['sold\_steps'] = steps\_to\_sell

# Execute full liquidations for stocks marked in stocks\_to\_sell

for stock, reason in stocks\_to\_sell.items():

if context.portfolio.positions[stock].closeable\_amount > 0:

log.info(f"SELLING SIGNAL: {stock} due to: {reason}")

order\_target(stock, 0) # Liquidate position

# Clean up take profit status if liquidated

if stock in g.take\_profit\_status:

del g.take\_profit\_status[stock]

def after\_market\_close(context):

"""

Run after market close daily.

Record metrics, check account drawdown.

"""

# Record daily portfolio value and cash

record(portfolio\_value=context.portfolio.total\_value, cash=context.portfolio.available\_cash)

# Update peak portfolio value

g.peak\_portfolio\_value = max(g.peak\_portfolio\_value, context.portfolio.total\_value)

# Check Account Drawdown - only if not already paused

if not g.trading\_paused:

drawdown = (context.portfolio.total\_value / g.peak\_portfolio\_value) - 1

if drawdown < -g.account\_drawdown\_limit:

log.warning(f"ACCOUNT DRAWDOWN LIMIT REACHED: {drawdown\*100:.2f}%. Current Value: {context.portfolio.total\_value:.2f}, Peak Value: {g.peak\_portfolio\_value:.2f}")

g.trading\_paused = True

g.pause\_start\_date = context.current\_dt.date()

log.warning("--- Initiating Trading Pause for 1 week. Liquidating all positions. ---")

for stock in list(context.portfolio.positions.keys()):

if context.portfolio.positions[stock].closeable\_amount > 0:

order\_target(stock, 0)

# Clear take profit status on full liquidation

g.take\_profit\_status.clear()

# Clean up stocks bought today list

# g.bought\_today = [] # Moved to before\_market\_open

# Clean up take\_profit\_status for stocks no longer held

held\_stocks = set(context.portfolio.positions.keys())

keys\_to\_delete = [stock for stock in g.take\_profit\_status if stock not in held\_stocks]

for stock in keys\_to\_delete:

del g.take\_profit\_status[stock]

log.info(f"End of Day {context.current\_dt.date()}. Portfolio Value: {context.portfolio.total\_value:.2f}")

log.info(f"Current Positions: {list(context.portfolio.positions.keys())}")

log.info("-" \* 50)

# --- Helper Functions ---

def filter\_undesired\_stocks(context, stock\_list, date):

"""

Filters out ST, \*ST, PT, new stocks, suspended stocks.

"""

current\_data = get\_current\_data(stock\_list)

filtered\_list = []

for stock in stock\_list:

stock\_info = current\_data[stock]

if stock\_info is None: continue

if stock\_info.paused: continue # Suspended

if stock\_info.is\_st: continue # ST or \*ST

# Check for PT status if needed (less common now)

# Check if IPO within last N days (e.g., 60 days)

days\_public = (date - get\_security\_info(stock).start\_date).days

if days\_public < 60: continue # Skip stocks listed less than 60 days

# Filter out stocks starting with '688' (STAR Market) or '300' (ChiNext) if desired

# if stock.startswith('688') or stock.startswith('300'): continue

filtered\_list.append(stock)

# Additional check for fundamentals data availability if needed

# q = query(valuation.code).filter(valuation.code.in\_(filtered\_list))

# fundamentals = get\_fundamentals(q, date=date)

# return fundamentals['code'].tolist()

return filtered\_list

def calculate\_macd(close\_prices, fast\_period, slow\_period, signal\_period):

"""

Calculates MACD indicator values using pandas.

Returns (diff, dea, hist) as pandas Series, or (None, None, None) if error.

"""

if len(close\_prices) < slow\_period + signal\_period:

return None, None, None # Not enough data

try:

# If TALib is available and works on JoinQuant:

if talib:

diff, dea, hist = talib.MACD(close\_prices.values,

fastperiod=fast\_period,

slowperiod=slow\_period,

signalperiod=signal\_period)

# Convert numpy arrays back to pandas Series with correct index

index = close\_prices.index

diff\_series = pd.Series(diff, index=index, name='DIFF')

dea\_series = pd.Series(dea, index=index, name='DEA')

hist\_series = pd.Series(hist, index=index, name='MACD\_Hist')

return diff\_series, dea\_series, hist\_series

else:

# Manual calculation if TALib not available

ema\_fast = close\_prices.ewm(span=fast\_period, adjust=False).mean()

ema\_slow = close\_prices.ewm(span=slow\_period, adjust=False).mean()

diff\_series = ema\_fast - ema\_slow

dea\_series = diff\_series.ewm(span=signal\_period, adjust=False).mean()

hist\_series = (diff\_series - dea\_series) \* 2 # Common representation

return diff\_series, dea\_series, hist\_series

except Exception as e:

log.error(f"Error calculating MACD: {e}")

return None, None, None

def get\_stock\_sector(stock\_code, date):

"""

Gets the SW L1 industry code for a given stock.

Adjust industry classification standard if needed (e.g., 'zjw', 'jq\_l1').

"""

try:

# Using Shenwan L1 as an example

industry\_info = jqdata.get\_industry(security=stock\_code, date=date)

if industry\_info and stock\_code in industry\_info:

# Structure might be {stock: {'sw\_l1': {'industry\_code': 'xxx', 'industry\_name': 'yyy'}}}

if 'sw\_l1' in industry\_info[stock\_code] and industry\_info[stock\_code]['sw\_l1']:

return industry\_info[stock\_code]['sw\_l1']['industry\_code']

# Add checks for other standards like 'jq\_l1' if needed

# elif 'jq\_l1' in industry\_info[stock\_code] and industry\_info[stock\_code]['jq\_l1']:

# return industry\_info[stock\_code]['jq\_l1']['industry\_code']

except Exception as e:

log.error(f"Error getting sector for {stock\_code}: {e}")

return None

def get\_sector\_performance(context, date, industry\_standard='sw\_l1'):

"""

Calculates the average performance of each sector for the given date.

Returns a dictionary: {sector\_code: {'avg\_change': float, 'rank': int, 'name': str}}

NOTE: This can be computationally expensive. Consider optimizations or using pre-calculated factors if available.

"""

sector\_perf = {}

try:

industries = get\_industries(name=industry\_standard, date=date)

all\_sector\_stocks = []

sector\_map = {}

for code, row in industries.iterrows():

stocks = get\_industry\_stocks(code, date=date)

if stocks:

sector\_perf[code] = {'total\_change': 0.0, 'count': 0, 'name': row['name']}

for s in stocks:

sector\_map[s] = code

all\_sector\_stocks.extend(stocks)

if not all\_sector\_stocks: return {}

# Get price change for all relevant stocks

# Calculate change from previous close to current price/close

prices\_today = get\_price(all\_sector\_stocks, end\_date=date, frequency='daily', fields=['close'], count=1)

prices\_yest = get\_price(all\_sector\_stocks, end\_date=context.previous\_date, frequency='daily', fields=['close'], count=1)

if prices\_today.empty or prices\_yest.empty: return {}

# Merge prices

price\_data = pd.merge(prices\_today.reset\_index()[['code','close']],

prices\_yest.reset\_index()[['code','close']],

on='code', suffixes=('\_today', '\_yest'))

price\_data['change'] = (price\_data['close\_today'] / price\_data['close\_yest'] - 1) \* 100

# Aggregate by sector

for index, row in price\_data.iterrows():

stock = row['code']

change = row['change']

if stock in sector\_map:

sector\_code = sector\_map[stock]

if pd.notna(change): # Check for NaN changes

sector\_perf[sector\_code]['total\_change'] += change

sector\_perf[sector\_code]['count'] += 1

# Calculate average change and filter out sectors with no valid data

valid\_sectors = {}

for code, data in sector\_perf.items():

if data['count'] > 0:

avg\_change = data['total\_change'] / data['count']

valid\_sectors[code] = {'avg\_change': avg\_change, 'name': data['name']}

# Rank sectors

ranked\_sectors = sorted(valid\_sectors.items(), key=lambda item: item[1]['avg\_change'], reverse=True)

result = {}

for rank, (code, data) in enumerate(ranked\_sectors):

result[code] = {'avg\_change': data['avg\_change'], 'rank': rank + 1, 'name': data['name']}

return result

except Exception as e:

log.error(f"Error calculating sector performance: {e}")

return {}

def get\_hot\_sectors(context, date, top\_n=3):

"""

Gets the codes of the top N performing sectors based on average daily change.

"""

sector\_ranks = get\_sector\_performance(context, date)

hot\_sectors = [code for code, data in sector\_ranks.items() if data['rank'] <= top\_n]

return hot\_sectors

**Explanation and Key Implementation Points:**

1. **initialize**: Sets up all parameters derived from the text description and TDX formulas (MA periods, volume ratios, MACD settings, position limits, sell/stop-loss levels, timing). It schedules the core logic functions.
2. **before\_market\_open**:
   * Resets the g.bought\_today list.
   * Checks if a trading pause due to drawdown should end.
   * Filters the stock universe (removes ST, new, suspended).
   * Identifies the top N "hot sectors" based on the *previous day's* performance using get\_hot\_sectors.
   * Performs an initial technical screen (Price > MA20, Volume > 1.5 \* MA5(Vol), MACD > Signal) on all eligible stocks based on *previous day's* closing data. Stores qualifying stocks in g.candidate\_stocks.
3. **check\_buy\_signals**:
   * Runs every minute but only executes logic within the 10:00 - 10:30 window.
   * Checks if portfolio slots are available and trading isn't paused.
   * Iterates through g.candidate\_stocks.
   * Performs *intraday* checks for each candidate:
     + Is the stock currently paused?
     + Is the morning volume (so far) > 50% of *yesterday's* total volume?
     + Is the current price showing stability (simple check: current\_price > day\_open)?
     + (Optional but good: Re-confirm sector is still hot based on intraday performance).
   * If all conditions pass and cash is sufficient, calculates position size and places a buy order using order\_value. Adds the stock to g.bought\_today. Initializes take-profit tracking for the new position.
4. **check\_sell\_and\_rebalance**:
   * Runs every minute but only executes logic near the close (14:50 onwards) on specified rebalancing days (Mon, Wed, Fri).
   * Iterates through currently held positions.
   * Checks multiple sell conditions in order of potential priority:
     + **8% Stop Loss:** Liquidates immediately if profit < -8%.
     + **Daily Loss > 5%:** Sells half (or simplified to full sell in this version). *Requires careful state management if implementing true partial sells alongside other rules.*
     + **Sector Rank Drop:** Calculates current sector ranks and liquidates if the stock's sector rank falls below the threshold (e.g., > 5).
     + **Partial Take Profit:** Checks profit percentage against defined levels (20%, 25%, 30%, etc.). If a new level is reached and the corresponding 1/3 hasn't been sold yet, places an order to reduce the position accordingly and updates the g.take\_profit\_status.
   * Executes liquidation orders (order\_target(stock, 0)) for stocks meeting full sell conditions.
5. **after\_market\_close**:
   * Records portfolio value.
   * Updates the peak portfolio value for drawdown calculation.
   * Checks if the account drawdown limit (10%) is breached. If so, liquidates everything and sets the g.trading\_paused flag and g.pause\_start\_date.
   * Cleans up the g.take\_profit\_status dictionary for stocks no longer held.
6. **Helper Functions**:
   * filter\_undesired\_stocks: Basic filtering.
   * calculate\_macd: Calculates MACD values (uses talib if available, otherwise manual).
   * get\_stock\_sector: Retrieves the industry classification code (e.g., SW L1).
   * get\_sector\_performance: Calculates average performance for all sectors (can be slow).
   * get\_hot\_sectors: Uses get\_sector\_performance to find the top N sector codes.

**Important Considerations & Potential Improvements:**

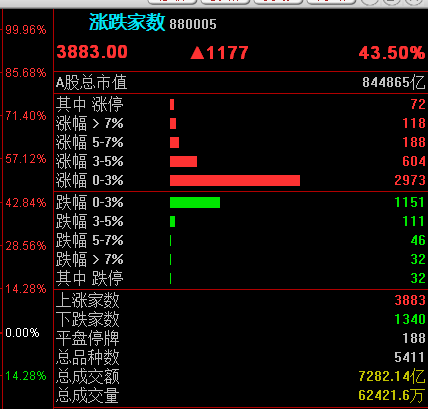
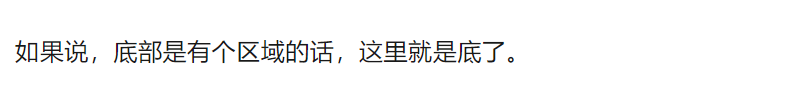
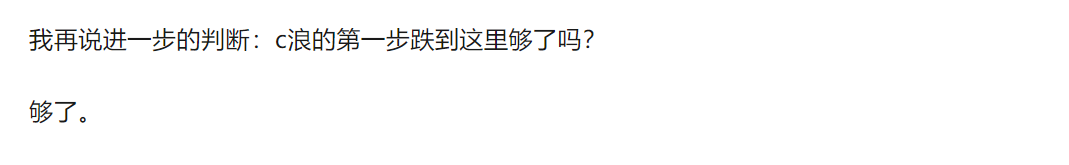
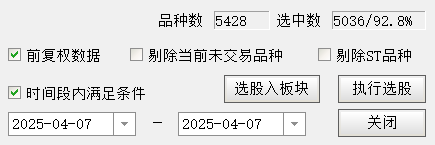
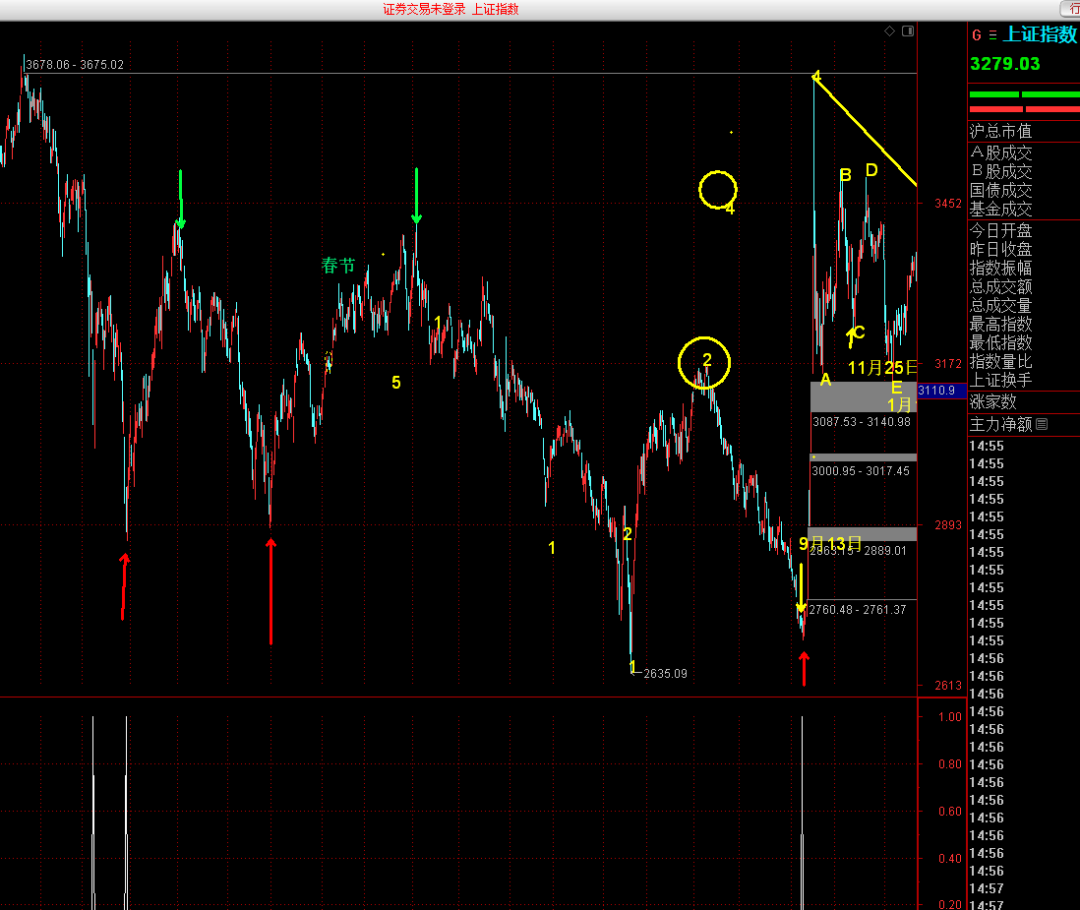
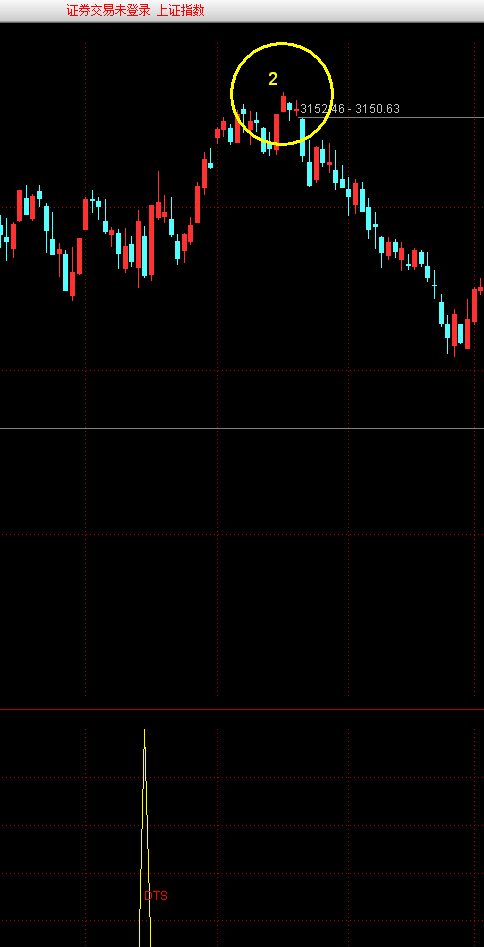
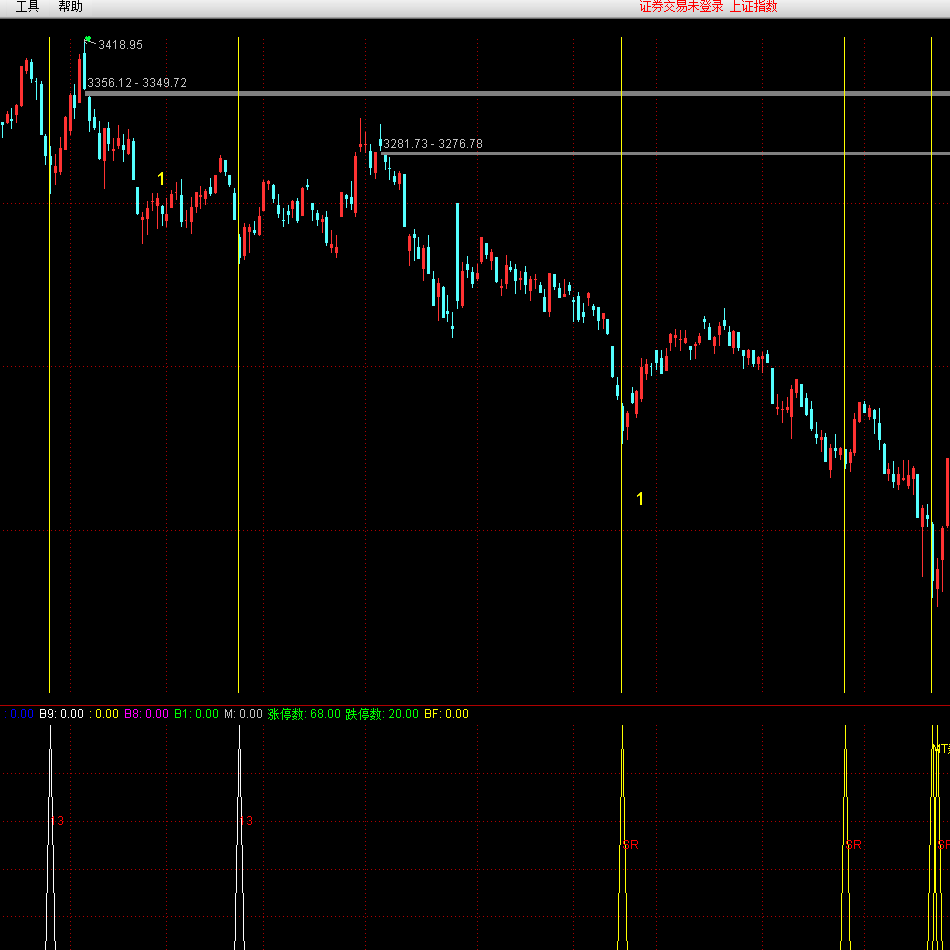
* **Sector Performance Calculation:** The get\_sector\_performance function iterates through many stocks and requires multiple get\_price calls. This can be very slow in backtesting or live trading. Consider:
  + Using JoinQuant's pre-calculated factor data for sector returns if available (jqfactor).
  + Simplifying the definition of a "hot sector" (e.g., based on a few leading stocks within the sector).
  + Calculating it less frequently if intraday updates aren't crucial.
* **Intraday Stability Check:** The current check (current\_price > day\_open) is basic. A more robust check might involve analyzing minute bars for volatility or deviations from VWAP.
* **TALib Dependency:** Ensure the talib library is available in your JoinQuant environment. If not, the manual MACD calculation will be used.
* **Partial Selling Logic:** The take-profit logic attempts partial selling. This requires careful state management (g.take\_profit\_status) and handling order amounts correctly (order\_target vs. order\_value). Ensure it behaves as expected, especially when multiple sell conditions might trigger.
* **Efficiency:** For large stock universes, fetching and processing data stock-by-stock (before\_market\_open) can be slow. Use batch functions like get\_price for multiple stocks and vectorized operations (pandas/numpy) where possible. attribute\_history can sometimes be more efficient for multiple indicators but has its own overhead.
* **Parameter Tuning:** The provided parameters (MA periods, thresholds, etc.) are starting points based on the description. These likely need optimization through backtesting.
* **Error Handling:** Added basic try...except blocks, but more robust error handling might be needed.
* **Cash Management:** The g.cash\_reserve\_ratio isn't explicitly used to block trades but influences the available cash calculation for new positions. Ensure this aligns with the strategy's intent.

This code provides a solid foundation for the "Hotspot Leader Rotation Strategy" on JoinQuant, translating the core rules from the description and TDX snippets. Remember to test thoroughly and refine based on backtesting results.

股市的道（一）情绪的冰点与沸点

原创 波段之门 [波段之门](javascript:void(0);)

 2025年05月02日 19:46

本文大约2400多字，断断续续大约写了两个月。所用指标，皆为通达信指标。我们的研究对象皆为上证。下面是正文。一个交易者可以通过阅读书籍或自主研究海量股票K线的历史走势，逐渐掌握如何从特定形态中捕捉个股的买入时机。当这种技术分析能力在实践中得到验证，即交易者能准确识别买点并通过实施交易获利时，这便构成了交易系统中的"术"。后来，你多次见到了这种图形，利用同样的方法去交易，但有许多次，你却亏损了，于是你开始怀疑你用过的“术”。做为智者的你开始反思：什么样的情况下，可以用这种术，什么时候不能用，你开始考虑更高一级的层面，你又发现了新的术。你发现，能不能用这种术，要考虑大盘，要考虑板块，要考虑市场的情绪。经过多轮交易验证后，一部分投资者会突破单纯技术分析的局限，逐步构建起多维度的评估体系。他们发现特定技术形态的有效性，本质上受制于大盘趋势、行业板块轮动和市场情绪三大核心变量。当你的交易决策系统能够有机整合技术形态（术）与市场环境的辩证关系时，便实现了交易认知的质变。这种认知的提高，具体表现为：交易者建立起环境适配机制，在技术形态与市场环境形成共振时能够应用既定策略，在条件不充分时主动规避风险。通过持续优化这种动态的平衡体系，最终形成具备持续盈利能力的交易系统——这便是"得道"的本质：技术方法与市场规律的高度统一。先有术，后有道。没有术的铺垫，就没有道。道和术很难区分，适用范围很广的术，就是道。把术做精，即为道。从根本上得到的术，就是道。对于市场来说，什么是根本呢？大盘就是根，板块就是本。个股是叶，情绪是水是肥料。脱离根本的交易，你可能会一时盈利，但拉长时间看，一定是亏损的。图形我们看得见，板块轮动我们也看得见，但是情绪我们看不见。我们今天就研究一种术，如何量化情绪。我们经常说，今天是情绪的冰点了，今天是情绪的沸点了。那到底用什么去测量，又怎么去测量？如果我们知道今天是冰点了，我们就要坚决买入。如果我们知道今天是沸点了，我们就必须马上卖出。那种卖出时担心：我的股会不会不随着大盘和板块走？恰恰就是韭菜思维。这个测量情绪的工具是：880005--涨跌家数。这是4月30日中午的涨跌家数图1月13日这天盘后，我们说，这天是底。4月7日这天盘后，我们说，这里是底。我们先看看最近的4月7日这天，究竟发生了什么？在这天，5300多只股票，仅有106只上涨，因为880005的数值就是上涨家数。上涨家数仅占2%。跌幅大于5%的股票是5037家。图1也就是说，接近93%的股票跌幅大于5%。根据我们的历史统计，只要70%的股票跌幅大于5%，那一天就是冰点，那里就是底部，就可以坚决做多。我们还发现，当跌停数超过300家的时候，也是一个可以做多的冰点。反过来，当涨停家数达到一定的数目时，那就是沸点。当涨停家数超过800家的时候，是一个必须卖出的沸点。还记得我们10月8日的十点多，我们在留言中说了什么。我们在盘中就知道那天达到了沸点。当然，并不是所有的沸点和冰点都是因为涨停家数和跌停家数到了临界点，但是，涨停家数和跌停家数到了临界点，一定是冰点和沸点。冰点是情绪极低，沸点是情绪爆燃。有的冰点是几天完成的，有的冰点是几十天累积完成的。例如，4月7日的冰点几乎是一天完成的。而1月13日的冰点是20多天完成的。我们看近半年的三个低点图2我们再看看去年9月的那次底部是怎么来的。图3这次的底部，靠的是逐渐失温，慢慢累积达到了冰点。累积的方法，由于当初发现它，耗费了我们太多的时间，出于对我们自己劳动的尊重，我们不会在这里介绍。讲到这里，你是不是认为这种方法可以找到所有的顶部和底部。不是的，有的顶部不是情绪的高低造成的。比如，2024年的5月20日的顶部就不是情绪造成的，它是周期造成的。所以，你在那里找不到情绪沸点的痕迹。图4我们的统计结果显示，沪深市场，60%的底部是情绪造成的，40%的顶部是情绪造成的。我们看一段2023年的走势。图5请根据我的图标去核对，2023年大部分的底部已经显示出来了，但有几个底部并没有显示。没有显示的底部，就不是情绪造成的。为了让大家可以量化情绪，请记录以下通达信函数：

涨停数:"880006$C",COLORGREEN,NODRAW;

跌停数:"880006$O",COLORGREEN,NODRAW;

上涨家数：ADVANCE；

下跌家数：DECLINE;

涨跌总家数:=(ADVANCE+DECLINE);

比如，当跌停数超过300家的时候，一般是一个可以做多的冰点。涨停数超过800家的时候，是一个可以做空的沸点。

反过来，当涨停家数少于一定的比例时，大盘也可以产生冰点。

当下跌家数超过一定的比例时，可以产生冰点。

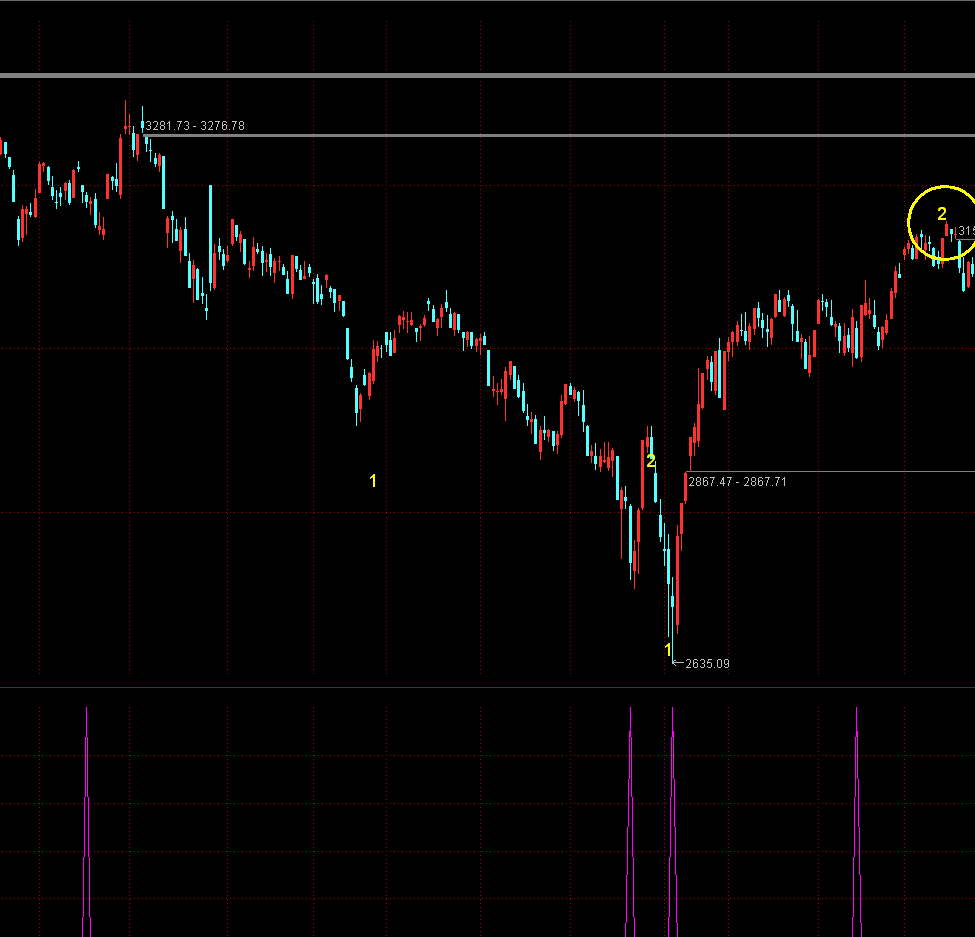
当上涨家数超过一定的比例时，可以产生沸点。

股指一般在不冷不热的状态下运行。

你应该时刻关注涨跌家数这个指标。

这是我们根据涨停的家数找到的冰点。

图6



我们可以写出如下附图公式（通达信软件）：

名称：冰点1

涨停数:"880006$C",COLORGREEN,NODRAW;

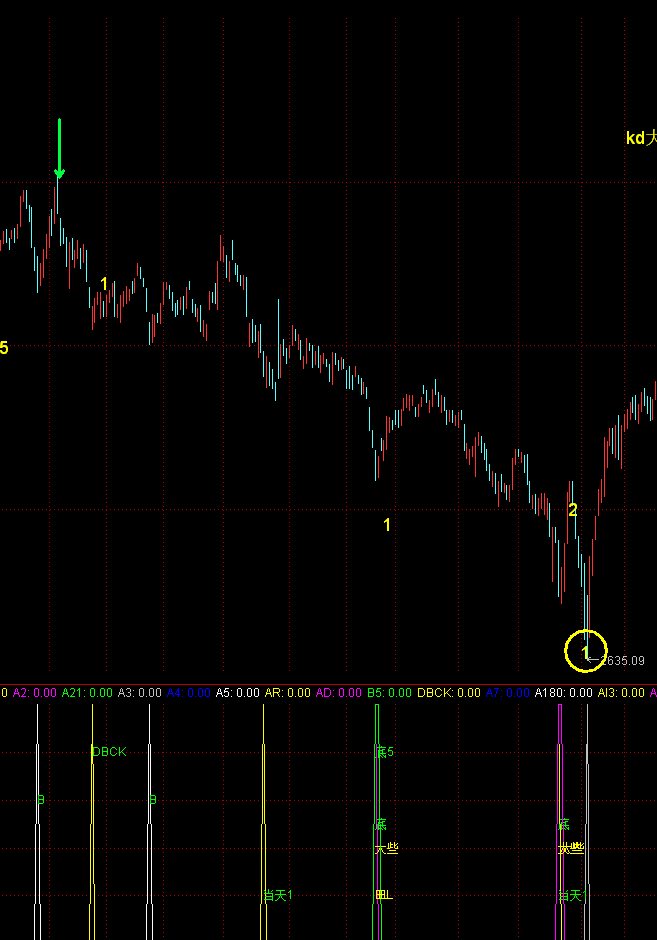
跌停数:"880006$O",COLORGREEN,NODRAW;

ZS:=(ADVANCE+DECLINE);

涨停数>800;

跌停数>300;

当然，随着股票的增多，这个数值是逐渐发生变化的，你可以给它设计成一定的比例。此处不再详说。

据我们不成熟的研究，沪深两市的顶底，往往是三种因素造成的。一、情绪。二、周期。三、结构。而这三者，往往是两两重合的，有的时候是三者重合。如果我们研究透了一种，是不是就比别人强了一些呢。投资者的盈亏差距是由认知差距造成的，认知的高度，决定了你钱包的厚度。本文所述用涨跌家数和涨跌停家数，以此管窥市场之情绪。望能抛砖引玉，启智者之思。注意，任何技术指标均是实时计算，系统不存在任何信号延迟。各项指标效能的核心在于使用者的认知水平，投资者的认知能力将直接决定最终分析结果的深度与价值。你看看我介绍的这个方法是不是当天就知道顶底了？需要假设吗？需要再观察吗？你如果看到kdj，就知道金叉和死叉，那你永远认为它是不准的。你如果看到macd，就知道背离和背驰，它大部分时间是背了又背。因为大部分背离是无效的背离，只是你的认知有限，你当时没有看出来。再来看看我们用macd研究的指标，你看看哪一个底部是延迟的？你把指标配上周期，配上情绪，就好像你用xyz三个坐标去捕捉一个点，这个点的位置，它怎么能不准呢？其它的一些指标也能够判断情绪的高低，希望大家努力研究。祝大家生活快乐！祝大家节日快乐!2025年5月2日于楚雄。