**Alpha\_ZD\_011**

class Alpha\_ZD\_011(Factor):

name = 'Alpha\_ZD\_011'

max\_window = 60

dependencies = ['sharpe\_ratio\_20','sharpe\_ratio\_60','sharpe\_ratio\_120','eps\_ttm']

def calc(self, data):

self.sharpe\_ratio\_20 = data['sharpe\_ratio\_20']

self.eps\_ttm = data['eps\_ttm'] # 每股收益TTM，过去12个月净利润TTM/总股本

# 过去一个月（20D）收益率 / 每股收益TTM

C = (self.sharpe\_ratio\_20 / self.eps\_ttm).rolling(20).mean()

return C.mean()

**Alpha\_ZD\_013**

class Alpha\_ZD\_013(Factor):

name = 'Alpha\_ZD\_013'

max\_window = 60

dependencies = ['sharpe\_ratio\_20','VOL20','eps\_ttm','VOL20','roa\_ttm']

def calc(self, data):

self.VOL20 = data['VOL20'] #20日平均换手率

self.roa\_ttm = data['roa\_ttm'] # 资产回报率TTM

self.sharpe\_ratio\_20 = data['sharpe\_ratio\_20']

self.eps\_ttm = data['eps\_ttm'] # 每股收益TTM，过去12个月净利润TTM/总股本

# 资产回报率TTM / VOL20

C = (self.roa\_ttm / self.VOL20).rolling(20).mean()

return C.mean()

**Alpha\_ZD\_014**

class Alpha\_ZD\_014(Factor):

name = 'Alpha\_ZD\_014'

max\_window = 60

dependencies = ['sharpe\_ratio\_20','circulating\_market\_cap','eps\_ttm','operating\_profit\_per\_share','current\_ratio']

def calc(self, data):

self.circulating\_market\_cap = data['circulating\_market\_cap'] # 流通市值

self.current\_ratio = data['current\_ratio'] # 流动比率（单季度）

# ln(circulating\_market\_cap) 流通市值对数

C = ((self.current\_ratio\*\*2)).rolling(10).mean()

return C.mean()

**Alpha\_ZD\_015**

class Alpha\_ZD\_015(Factor):

name = 'Alpha\_ZD\_015'

max\_window = 60

dependencies = ['operating\_revenue\_growth\_rate','total\_profit\_growth\_rate','ARBR']

def calc(self,data):

self.ARBR = data['ARBR']

self.operating\_revenue\_growth\_rate = data['operating\_revenue\_growth\_rate'] #营业收入增长率

self.total\_profit\_growth\_rate = data['total\_profit\_growth\_rate'] # 利润总额增长率

#

C = (self.ARBR.rank(axis=1) + self.total\_profit\_growth\_rate.rank(axis=1)).rolling(20).mean()

return C.mean()

**Alpha\_ZD\_021**

class Alpha\_ZD\_021(Factor):

name = 'Alpha\_ZD\_021'

max\_window = 60

dependencies = ['DEGM','net\_profit\_ratio','fixed\_asset\_ratio','cfo\_to\_ev']

def calc(self, data):

self.fixed\_asset\_ratio = data['fixed\_asset\_ratio'] # 固定资产比率

self.DEGM = data['DEGM'] # 毛利率增长

self.net\_profit\_ratio = data['net\_profit\_ratio'] # 销售净利率

self.cfo\_to\_ev = data['cfo\_to\_ev'] #经营活动产生的现金流量净额与企业价值之比TTM

C = (self.DEGM + self.cfo\_to\_ev ).rolling(30).mean()

return C.mean()

**Alpha\_ZD\_022**

class Alpha\_ZD\_022(Factor):

name = 'Alpha\_ZD\_022'

max\_window = 60

dependencies = ['equity\_turnover\_rate','net\_working\_capital']

def calc(self, data):

self.net\_working\_capital = data['net\_working\_capital']

self.equity\_turnover\_rate = data['equity\_turnover\_rate']

C = (self.net\_working\_capital\*\*2 / self.equity\_turnover\_rate ).rolling(30).mean()

return C.mean()

**Alpha\_ZD\_023**

class Alpha\_ZD\_023(Factor):

name = 'Alpha\_ZD\_023'

max\_window = 60

dependencies = ['long\_debt\_to\_asset\_ratio','operating\_profit\_ratio','roa\_ttm']

def calc(self, data):

self.roa\_ttm = data['roa\_ttm']

self.long\_debt\_to\_asset\_ratio = data['long\_debt\_to\_asset\_ratio']

self.operating\_profit\_ratio = data['operating\_profit\_ratio']

C = (self.roa\_ttm - self.operating\_profit\_ratio - self.long\_debt\_to\_asset\_ratio).rolling(30).mean()

return C.mean()

**Alpha\_ZD\_024**

class Alpha\_ZD\_024(Factor):

name = 'Alpha\_ZD\_024'

max\_window = 60

dependencies = ['total\_profit\_ttm','market\_cap']

def calc(self, data):

self.market\_cap = data['market\_cap'] # 市值

self.total\_profit\_ttm = data['total\_profit\_ttm'] #利润总额TTM

C = (self.market\_cap + self.total\_profit\_ttm ).rolling(30).mean()

return C.mean()

Alpha\_005

戈登模型：初始股息乘以不变增长股利贴现作为股票估价。贴现率用股票beta衡量，股利增长率用ROE\*留存比率（盈余公积金占未分配利润比例）假设初始股息相同，如果加入股息变量，效果应该更好。

公式：V=D0(1+g)/(y-g)=D1/(y-g)，其中的D0、D1分别是初期和第一期支付的[股息](https://baike.baidu.com/item/%E8%82%A1%E6%81%AF)，y是股东要求回报率，g是股利增长率

from jqfactor import analyze\_factor

from jqfactor import Factor

import pandas as pd

import numpy as np

class Liuqifan\_alpha\_005(Factor):

name='Gordon'

max\_window=1

dependencies=['beta','surplus\_reserve\_fund\_per\_share','retained\_profit\_per\_share'

,'close','roe\_ttm']

def calc(self,data):

beta=data['beta']

SRFPS=data['surplus\_reserve\_fund\_per\_share']

RPPS=data['retained\_profit\_per\_share']

close=data['close']

roe=data['roe\_ttm']

retention\_ratio=SRFPS/RPPS

growth=retention\_ratio\*roe

Gordon=beta/(beta-growth)

return Gordon.mean()

Alpha\_007

营运（operating）现金流增长率和利润增长率相乘，代表企业真实盈利能力。因为结合利润表和现金流量表可以看出公司是否财务有操纵情况。

from jqfactor import analyze\_factor

from jqfactor import Factor

import pandas as pd

import numpy as np

class Liuqifan\_alpha\_007(Factor):

name='CashflowAndEaringGrowth'

max\_window=1

dependencies=['net\_operate\_cashflow\_growth\_rate','np\_parent\_company\_owners\_growth\_rate']

def calc(self,data):

OCFGR=data['net\_operate\_cashflow\_growth\_rate']

NPGR=data['np\_parent\_company\_owners\_growth\_rate']

CashflowAndEaringGrowth=OCFGR\*NPGR

return CashflowAndEaringGrowth.mean()

Alpha\_008

戴维斯双击，低估+高增长。PB代表估值，净利润代表增长能力。

from jqfactor import analyze\_factor

from jqfactor import Factor

import pandas as pd

import numpy as np

class Liuqifan\_alpha\_008(Factor):

name='EPB'

max\_window=1

dependencies=['np\_parent\_company\_owners\_ttm','pb\_ratio']

def calc(self,data):

earing=data['np\_parent\_company\_owners\_ttm']

PB=data['pb\_ratio']

EPB=earing/PB

return EPB.mean()

Alpha\_011

ROIC，投入资本回报率，衡量公司投入的资本所产生的营业利润。投入资本（长期借款，股本，少数股东权益）越低，营业利润越高，因子越大，相应的收益越高，即高分位因子收益高。

from jqfactor import analyze\_factor

from jqfactor import Factor

import pandas as pd

import numpy as np

class Liuqifan\_alpha\_011(Factor):

name='ROIC'

max\_window=1

dependencies=['longterm\_loan','minority\_interests','capitalization',

'operating\_profit']

def calc(self,data):

longterm\_loan=data['longterm\_loan']

minority\_interests=data['minority\_interests']

capitalization=data['capitalization']

operating\_profit=data['operating\_profit']

ROIC=(operating\_profit)/(longterm\_loan+minority\_interests+capitalization)

return ROIC.mean()

Alpha\_015

可持续增长率/股价。可持续增长率越大，成长性越高，价格越低，估值越低，因此因子值越大，收益越高。

from jqfactor import analyze\_factor

from jqfactor import Factor

import pandas as pd

import numpy as np

class Liuqifan\_alpha\_015(Factor):

name='Growth/Price'

max\_window=1

dependencies=['surplus\_reserve\_fund\_per\_share','retained\_profit\_per\_share','close']

def calc(self,data):

close=data['close']

srfps=data['surplus\_reserve\_fund\_per\_share']

rpps=data['retained\_profit\_per\_share']

growthP=srfps/rpps/close

return growthP.mean()

far = analyze\_factor(factor=Liuqifan\_alpha\_015, start\_date='2013-05-01', end\_date='2019-06-01',

weight\_method='mktcap', universe='000905.XSHG', industry='jq\_11',

quantiles=5,periods=(10,20,30))

Alpha\_018

非主营业务占所有业务收入比重\*PB。非主营业务占比越高，估值越高，代表公司可能处于转型，因此后续收益大。相比于alpha\_017(只考虑非主营业务占比），这里的因子构造解释了企业转型的因素。回顾017策略，非主营占比越高，收益反而越低，即公司越不务正业，表现越差。

from jqfactor import analyze\_factor

from jqfactor import Factor

import pandas as pd

import numpy as np

class Liuqifan\_alpha\_018(Factor):

name='NOR/TOR'

max\_window=1

dependencies=['non\_operating\_revenue','total\_operating\_revenue'

,'book\_to\_price\_ratio']

def calc(self,data):

nor=data['non\_operating\_revenue']

tor=data['total\_operating\_revenue']

bp=data['book\_to\_price\_ratio']

return (nor/tor/bp).mean()

Alpha\_021

主力资金占成交额比/市值\*股价偏离60日最高价程度。来源于股价操纵，市值越小，主力资金占比越大的股票越容易操纵，同是如果价格越低，未来上涨的幅度和概率应该越大。

from jqfactor import analyze\_factor

from jqfactor import Factor

import pandas as pd

import numpy as np

class Liuqifan\_alpha\_021(Factor):

name='ZL'

max\_window=60

dependencies=['net\_pct\_main','close','size']

def calc(self,data):

npm=data['net\_pct\_main']

close=data['close']

size=data['size']

ZL=npm[-1:]/(size[-1:])\*(np.max(close)-close[-1:])/close[-1:]

return ZL.mean()

Alpha\_023

RRV的PB分解，将市值分解为各个财务指标并计算价格和价值偏离程度,参考RRV的PB分解，这里进一步简化，不考虑行业因素。

from jqfactor import analyze\_factor

from jqfactor import Factor

import pandas as pd

import numpy as np

class Liuqifan\_alpha\_023(Factor):

name='bm'

max\_window=1

dependencies=['debt\_to\_equity\_ratio','net\_profit\_ttm','size','total\_owner\_equities']

def calc(self,data):

de=data['debt\_to\_equity\_ratio']

netprofit=data['net\_profit\_ttm']

m=data['size']

e=data['total\_owner\_equities']

lev=1+de

b=np.log(e)

ni=np.log(abs(netprofit))

Ini=(netprofit<0)

bm=m-(7.428071+0.529429\*b+0.198214\*ni-0.03514\*Ini+0.041929\*lev)

return bm.mean()

**Alpha\_020:**

class Clvnshi\_Alpha\_Custom\_020(Factor):

name = 'clvnshi\_alpha\_custom\_020'

max\_window = 20

dependencies = ['financial\_assets','financial\_liability']

def calc(self, data):

financial\_assets = data['financial\_assets']

financial\_liability = data['financial\_liability']

df = (financial\_assets - financial\_liability)/ financial\_liability

df = df.rolling(20).mean()

return df.replace([-np.inf, np.inf], 0).fillna(value=0).mean()

含义：金融资产/金融负债 – 1； 若负债高则应股票表现较差；

**Alpha\_020:**

class Clvnshi\_Alpha\_Custom\_020(Factor):

name = 'clvnshi\_alpha\_custom\_020'

max\_window = 20

dependencies = ['financial\_assets','financial\_liability']

def calc(self, data):

financial\_assets = data['financial\_assets']

financial\_liability = data['financial\_liability']

df = (financial\_assets - financial\_liability)/ financial\_liability

df = df.rolling(20).mean()

return df.replace([-np.inf, np.inf], 0).fillna(value=0).mean()

含义：金融资产/金融负债 – 1； 若负债高则应股票表现较差；

**Alpha\_044:**

class Clvnshi\_Alpha\_Custom\_044(Factor):

name = 'clvnshi\_alpha\_custom\_044'

max\_window = 20

dependencies = ['retained\_earnings']

def calc(self, data):

earning = data['retained\_earnings']

df = earning.rolling(6).mean() \* volume.apply(log)

return df.replace([-np.inf, np.inf], 0).fillna(value=0).mean()

含义：留存收益移动平均的对数；留存收益可以提现盈利能力

**Alpha\_046:**

class Clvnshi\_Alpha\_Custom\_046(Factor):

name = 'clvnshi\_alpha\_custom\_046'

max\_window = 5

dependencies = ['eps\_ttm','market\_cap']

def calc(self, data):

market\_cap = data['market\_cap']

eps\_ttm = data['eps\_ttm']

ratio = eps\_ttm / market\_cap

df = ratio

return df.replace([-np.inf, np.inf], 0).fillna(value=0).mean()

含义：每股收益/市值；体现盈利能力

**Alpha\_047:**

class Clvnshi\_Alpha\_Custom\_047(Factor):

name = 'clvnshi\_alpha\_custom\_047'

max\_window = 20

dependencies = ['administration\_expense\_ttm','market\_cap']

def calc(self, data):

administration\_expense\_ttm = data['administration\_expense\_ttm']

market\_cap = data['market\_cap']

df = administration\_expense\_ttm / market\_cap

return df.replace([-np.inf, np.inf], 0).fillna(value=0).mean()

含义：管理费用/市值 适度管理费用有助企业增长；

**Alpha\_049:**

class Clvnshi\_Alpha\_Custom\_049(Factor):

name = 'clvnshi\_alpha\_custom\_049'

max\_window = 5

dependencies = ['non\_recurring\_gain\_loss','net\_working\_capital']

def calc(self, data):

net\_working\_capital = data['net\_working\_capital']

non\_recurring\_gain\_loss = data['non\_recurring\_gain\_loss']

ratio = non\_recurring\_gain\_loss / net\_working\_capital

df = ratio

return df.replace([-np.inf, np.inf], 0).fillna(value=0).mean()

含义：非经常性损益 / 经运营资本； 非经常性损益越大，可能会对公司经营造成影响

Alpha\_2: asset turnover ratio

class asset\_turnover\_ratio(Factor):

name = 'asset\_turnover\_ratio'

max\_window = 1

dependencies = ['operating\_revenue''total\_assets']

def calc(self data):

f = data['operating\_revenue']/data['total\_assets']

return f.mean()

Alpha\_4: current ratio

class current\_ratio(Factor):

name = 'current\_ratio'

max\_window = 1

dependencies = ['total\_current\_assets''total\_current\_liability'] def calc(self data):

f = data['total\_current\_assets']/data['total\_current\_liability']

return f.mean()

Alpha\_5: dividend rate

class dividend\_rate(Factor):

name = 'dividend\_rate'

max\_window = 1

dependencies = ['dividend\_payable''close']

def calc(self data):

f = data['dividend\_payable']/data['close']

return f.mean()

Alpha\_6: high low open close

class COHL\_ratio(Factor):

name = 'COHL\_ratio'

max\_window = 1

dependencies = ['close''open''high''low']

def calc(self data):

f=(data['close']-data['open'])/(data['high']-data['low']+0.001)

return -f.mean()

Alpha\_19: YOY profit

class PE\_ratio(Factor):

name = 'PE\_ratio'

max\_window = 1

dependencies = ['inc\_net\_profit\_year\_on\_year']

def calc(self data):

f = data['inc\_net\_profit\_year\_on\_year']

return f.mean()

1. 是否低估（pb）+质量因子（ROA）

class Alpha\_new\_002(Factor):

name = 'alpha\_new\_002'

max\_window = 1

dependencies = ['pb\_ratio', 'roa']

def calc(self, data):

pb = data['pb\_ratio']

roa = data['roa']

alpha = (500 - pb.rank(axis = 1)) + roa.rank(axis = 1)

return alpha.mean()

1. net profit/cap

class Alpha\_new\_003(Factor):

name = 'alpha\_new\_003'

max\_window = 1

dependencies = ['net\_profit', 'net\_profit\_1', 'net\_profit\_2', 'net\_profit\_3',

'market\_cap']

def calc(self, data):

net\_profit\_ttm = data['net\_profit'] + data['net\_profit\_1'] + data['net\_profit\_2'] + data['net\_profit\_3']

alpha = net\_profit\_ttm / data['market\_cap']

return alpha.mean()

1. 质量因子（ROA）+资本结构（cash flow/liability）

class Alpha\_new\_004(Factor):

name = 'alpha\_new\_004'

max\_window = 1

dependencies = ['net\_operate\_cash\_flow', 'total\_liability', 'roa']

def calc(self, data):

roa = data['roa']

structure = data['net\_operate\_cash\_flow'] / data['total\_liability']

alpha = roa.rank(axis=1) + structure.rank(axis=1)

return alpha.mean()

1. 开发支出/总资产

class Alpha\_new\_005(Factor):

name = 'alpha\_new\_005'

max\_window = 1

dependencies = ['development\_expenditure', 'total\_assets']

def calc(self, data):

development = data['development\_expenditure']

asset = data['total\_assets']

alpha = development / asset

return alpha.mean()

1. 成长因子的综合指标（投资回报率roe、业绩增长profit、效率提升turn、边际投资回报率invest的排名之和）

class Alpha\_new\_012(Factor):

name = 'alpha\_new\_012'

max\_window = 1

dependencies = ['net\_profit\_y', 'net\_profit\_y1',

'income\_tax\_expense\_y', 'income\_tax\_expense\_y1',

'financial\_expense\_y', 'financial\_expense\_y1',

'net\_invest\_cash\_flow\_y',

'net\_invest\_cash\_flow\_y1',

'operating\_profit\_growth\_rate', 'OperatingCycle', 'roe']

def calc(self, data):

ebit = data['net\_profit\_y'] + data['income\_tax\_expense\_y'] + data['financial\_expense\_y']

ebit\_1 = data['net\_profit\_y1'] + data['income\_tax\_expense\_y1'] + data['financial\_expense\_y1']

profit = data['operating\_profit\_growth\_rate']

turn = data['OperatingCycle']

roe = data['roe']

invest = (ebit - ebit\_1) / (data['net\_invest\_cash\_flow\_y']-data['net\_invest\_cash\_flow\_y1'])

alpha = roe.rank(axis=1) + profit.rank(axis=1) + turn.rank(axis=1) + invest.rank(axis=1)

return alpha.mean()

1. 盈利能力（ROA）+增长（ROA-GARP：净利润同比变化 / 一年平均总资产 / pb ）

class Alpha\_new\_005(Factor):

name = 'alpha\_new\_005'

max\_window = 1

dependencies = ['inc\_net\_profit\_year\_on\_year',

'total\_assets', 'total\_assets\_1', 'total\_assets\_2',

'total\_assets\_3', 'pb\_ratio', 'roa']

def calc(self, data):

roa = data['roa']

assets = (data['total\_assets']+data['total\_assets\_1']+data['total\_assets\_2']+data['total\_assets\_3'])/4

growth = data['inc\_net\_profit\_year\_on\_year'] / assets / data['pb\_ratio']

alpha = roa.rank(axis=1) + growth.rank(axis=1)

return alpha.mean()

1. 低波动率，尤其是残差波动率，能够带来较大的alpha

class Alpha\_new\_012(Factor):

name = 'alpha\_new\_012'

max\_window = 1

dependencies = ['residual\_volatility']

def calc(self, data):

vol = data['residual\_volatility']

alpha = -vol

return alpha.mean()

1. Dividend yield和return正相关

class Alpha\_new\_012(Factor):

name = 'alpha\_new\_012'

max\_window = 1

dependencies = ['dividend\_interest\_payment', 'market\_cap']

def calc(self, data):

alpha = data['dividend\_interest\_payment'] / data['market\_cap']

return alpha.mean()

1. 估值 + 总资产增长率

class Alpha\_new\_012(Factor):

name = 'alpha\_new\_012'

max\_window = 1

dependencies = ['total\_assets', 'total\_assets\_1','pb\_ratio']

def calc(self, data):

pb = data['pb\_ratio']

change = data['total\_assets'] / data['total\_assets\_1']

alpha = 500 - pb.rank(axis=1) + change.rank(axis = 1)

return alpha.mean()

1. leverage（debt-equity ratio）+ pb

class Alpha\_new\_012(Factor):

name = 'alpha\_new\_012'

max\_window = 1

dependencies = ['debt\_to\_equity\_ratio', 'pb\_ratio']

def calc(self, data):

risk = data['debt\_to\_equity\_ratio']

pb = data['pb\_ratio']

alpha = 500 - risk.rank(axis=1) + 500 - pb.rank(axis=1)

return alpha.mean()

1. 估值因子（pb）+成长因子（ROA、cash flow ROA、5年ROA标准差、5年sales标准差、开发支出比例）

class Alpha\_new\_012(Factor):

name = 'alpha\_new\_012'

max\_window = 1

dependencies = ['roa\_y', 'roa\_y1', 'roa\_y2', 'roa\_y3', 'roa\_y4',

'net\_operate\_cash\_flow\_y', 'total\_assets\_y1','pb\_ratio',

'net\_profit\_y', 'development\_expenditure\_y',

'net\_profit\_y1', 'net\_profit\_y2', 'net\_profit\_y3', 'net\_profit\_y4'

]

def calc(self, data):

pb = data['pb\_ratio']

roa = data['roa\_y']

cf = data['net\_operate\_cash\_flow\_y']

asset = data['total\_assets\_y1']

ni = data['net\_profit\_y']

cf\_roa = cf / asset

accrual = cf - ni

rd = data['development\_expenditure\_y'] / asset

roa\_table = pd.concat([data['roa\_y'], data['roa\_y1'], data['roa\_y2'],

data['roa\_y3'], data['roa\_y4']],axis=0,ignore\_index=False)

roa\_v = roa\_table.std()

ni\_table = pd.concat([ni, data['net\_profit\_y1'], data['net\_profit\_y2'],

data['net\_profit\_y3'], data['net\_profit\_y4']],axis=0,ignore\_index=False)

ni\_v = ni\_table.std()

alpha = 500 - pb.rank(axis=1) + roa.rank(axis=1) + cf\_roa.rank(axis=1) + rd.rank(axis=1) - roa\_v.rank() - ni\_v.rank()

return alpha.mean()

1. 五因子（Earnings yield、growth、residual volatility、momentum、liquidity）

class Alpha\_new\_012(Factor):

name = 'alpha\_new\_012'

max\_window = 1

dependencies = ['earnings\_yield','growth',

'residual\_volatility', 'momentum', 'liquidity']

def calc(self, data):

value = data['earnings\_yield']

growth = data['growth']

vol = data['residual\_volatility']

mom = data['momentum']

liquidity = data['liquidity']

alpha = value.rank(axis=1)+growth.rank(axis=1)+500-vol.rank(axis=1)+500 - liquidity.rank(axis=1) + mom.rank(axis=1)

return alpha.mean()

1. 盈利能力eps

class Alpha\_new\_012(Factor):

name = 'alpha\_new\_012'

max\_window = 1

dependencies = ['eps']

def calc(self, data):

eps = data['eps']

alpha = eps

return alpha.mean()

1. 0.3×质量因子（ROA）+0.5×估值因子（PB）+0.2×低波动因子（收益率标准差）

class Alpha\_new\_012(Factor):

name = 'alpha\_new\_012'

max\_window = 60

dependencies = ['close', 'roa', 'pb\_ratio']

def calc(self, data):

close = data['close']

vol = close.pct\_change().std()

roa = data['roa'].mean()

pb = data['pb\_ratio'].mean()

alpha = 0.3 \* roa.rank() + 0.5 \* (500 - pb.rank()) + 0.2 \* (500 - vol.rank())

return alpha

1. accruals、size、pb、momentum

class Alpha\_new\_012(Factor):

name = 'alpha\_new\_012'

max\_window = 1

dependencies = ['total\_current\_assets', 'total\_current\_assets\_1',

'cash\_equivalents', 'cash\_equivalents\_1',

'total\_current\_liability', 'total\_current\_liability\_1','size','pb\_ratio','momentum']

def calc(self, data):

accruals = data['total\_current\_assets'] - data['total\_current\_assets\_1'] - data['cash\_equivalents'] + data['cash\_equivalents\_1'] - data['total\_current\_liability'] + data['total\_current\_liability\_1']

size = data['size']

pb = data['pb\_ratio']

mom = data['momentum']

alpha = accruals.rank(axis=1)+500-pb.rank(axis=1)+mom.rank(axis=1)

return alpha.mean()

1. 盈利增长能力（净利润同比+环比）

class Alpha\_new\_012(Factor):

name = 'alpha\_new\_012'

max\_window = 1

dependencies = ['inc\_net\_profit\_annual','inc\_net\_profit\_year\_on\_year']

def calc(self, data):

value = data['inc\_net\_profit\_annual']

growth = data['inc\_net\_profit\_year\_on\_year']

alpha = value.rank(axis=1)+growth.rank(axis=1)

return alpha.mean()

1. 成长性（开发支出增长与净利润增长之比）

class Alpha\_new\_012(Factor):

name = 'alpha\_new\_012'

max\_window = 1

dependencies = ['development\_expenditure\_y', 'development\_expenditure\_y1',

'net\_profit\_y', 'net\_profit\_y1']

def calc(self, data):

rd = data['development\_expenditure\_y']/data['development\_expenditure\_y1']

pro = data['net\_profit\_y']/data['net\_profit\_y1']

alpha = rd/pro

return alpha.mean()

1. 质量因子+估值因子（营业收入增长率+净利润增长率+PB）

class Alpha\_new\_012(Factor):

name = 'alpha\_new\_012'

max\_window = 1

dependencies = ['pb\_ratio', 'operating\_revenue\_growth\_rate',

'net\_profit\_growth\_rate']

def calc(self, data):

gor = data['operating\_revenue\_growth\_rate']

gnp = data['net\_profit\_growth\_rate']

pb = data['pb\_ratio']

alpha = 500-pb.rank(axis=1)+gor.rank(axis=1)+gnp.rank(axis=1)

return alpha.mean()

1. beta+size+pb+leverage+eps

class Alpha\_new\_012(Factor):

name = 'alpha\_new\_012'

max\_window = 1

dependencies = ['beta', 'size', 'pb\_ratio', 'leverage', 'eps']

def calc(self, data):

beta = data['beta']

size = data['size']

pb = data['pb\_ratio']

lev = data['leverage']

eps = data['eps']

alpha = 500-beta.rank(axis=1)+500-size.rank(axis=1)+500-pb.rank(axis=1)+500-lev.rank(axis=1)+eps.rank(axis=1)

return alpha.mean()

1. DuPont分析

class Alpha\_new\_012(Factor):

name = 'alpha\_new\_012'

max\_window = 1

dependencies = ['total\_asset\_turnover\_rate', 'profit\_margin\_ttm']

def calc(self, data):

alpha = data['total\_asset\_turnover\_rate'].rank(axis=1) + data['profit\_margin\_ttm'].rank(axis=1)

return alpha.mean()

1. 总资产变化率

class Alpha\_new\_012(Factor):

name = 'alpha\_new\_012'

max\_window = 1

dependencies = ['total\_assets', 'total\_assets\_4']

def calc(self, data):

alpha = data['total\_assets\_4']/(data['total\_assets']-1

return alpha.mean()

1. quality+value+low risk+payout

class Alpha\_new\_012(Factor):

name = 'alpha\_new\_012'

max\_window = 1

dependencies = ['gross\_profit\_margin', 'roe', 'roa',

'net\_invest\_cash\_flow', 'net\_operate\_cash\_flow',

'net\_finance\_cash\_flow', 'total\_assets', 'pb\_ratio',

'debt\_to\_equity\_ratio']

def calc(self, data):

roe = data['roe']

roa = data['roa']

cf = data['net\_invest\_cash\_flow'] + data['net\_operate\_cash\_flow'] + data['net\_finance\_cash\_flow']

cfoa = cf/data['total\_assets']

gmar = data['gross\_profit\_margin']

pb = data['pb\_ratio']

risk = data['debt\_to\_equity\_ratio']

profit = zscore(roe.rank(axis=1).mean())+zscore(roa.rank(axis=1).mean())+zscore(cfoa.rank(axis=1).mean())+zscore(gmar.rank(axis=1).mean())

alpha = profit / 4 + zscore((500 - pb.rank(axis=1)).mean()) + zscore((500 - risk.rank(axis=1)).mean())

return alpha

1. turnover+pb+beta+size

class Alpha\_new\_012(Factor):

name = 'alpha\_new\_012'

max\_window = 1

dependencies = ['liquidity', 'pb\_ratio', 'beta', 'size']

def calc(self, data):

liquidity = data['liquidity']

pb = data['pb\_ratio']

beta = data['beta']

size = data['size']

alpha = 500 - liquidity.rank(axis=1) + 500 - pb.rank(axis=1) + 500 - (beta\*\*2).rank(axis=1) + 500 - size.rank(axis=1)

return alpha.mean()

1. 价值因子+成长因子

class Alpha\_new\_012(Factor):

name = 'alpha\_new\_012'

max\_window = 1

dependencies = ['pe\_ratio', 'pb\_ratio', 'ps\_ratio',

'pcf\_ratio', 'dividend\_interest\_payment',

'operating\_revenue\_growth\_rate',

'net\_profit\_growth\_rate']

def calc(self, data):

pe = data['pe\_ratio']

pb = data['pb\_ratio']

ps = data['ps\_ratio']

pcf = data['pcf\_ratio']

gor = data['operating\_revenue\_growth\_rate']

gnp = data['net\_profit\_growth\_rate']

value = (1/pe+1/pb+1/ps+1/pcf)/4

growth = (gor + gnp)/2

alpha = value + growth

return alpha.mean()

1. systematic+idiosyncratic volatility

class Alpha\_new\_012(Factor):

name = 'alpha\_new\_012'

max\_window = 1

dependencies = ['beta','residual\_volatility']

def calc(self, data):

beta = data['beta']

vol = data['residual\_volatility']

alpha = -beta \* beta - vol \* vol

return alpha.mean()

1. firm productivity(return on invested capital)

class Alpha\_new\_012(Factor):

name = 'alpha\_new\_012'

max\_window = 1

dependencies = ['total\_assets', 'total\_liability', 'net\_profit']

def calc(self, data):

alpha = data['net\_profit']/(data['total\_assets']-data['total\_liability'])

return alpha.mean()

### 因子11

因子11表示为盈利能力的5日最大值与最小值差。因子11的alpha值和beta值较小，平均收益显单调递增性，多空组合非线性。

代码：

class Alpha\_zeting\_109(Factor):

name = 'alpha\_zeting\_109'

max\_window = 5

dependencies = ['earnings\_yield']

def calc(self,data):

'''盈利能力的5日最大值与最小值差'''

self.earnings\_yield = data['earnings\_yield']

alpha\_zeting\_109 = self.earnings\_yield.max()-self.earnings\_yield.min()

return alpha\_zeting\_109

### 因子13

因子13表示为60日无形资产与固定资产的比值。因子13的alpha值较高，beta值较小，平均收益显凸性，多空组合显单调递增性。

代码：

class Alpha\_zeting\_112(Factor):

name = 'alpha\_zeting\_112'

max\_window = 60

dependencies = ['intangible\_asset\_ratio','fixed\_asset\_ratio']

def calc(self,data):

'''无形资产/固定资产'''

self.intangible\_asset = data['intangible\_asset\_ratio']

self.fixed\_asset = data['fixed\_asset\_ratio']

alpha\_zeting\_112 = self.intangible\_asset/self.fixed\_asset

return -alpha\_zeting\_112.mean()

Alpha\_9:商誉与总资产的比值的20天平均

Alpha\_16\_yy= good\_will/total\_assets. rolling(window=20).mean()

构造因子的逻辑：商誉的价值会对股价有影响

class Alpha\_16\_yy(Factor):

name = 'Alpha\_16\_yy'

max\_window =20

dependencies = ['good\_will','total\_assets']

def calc(self, data):

good\_will = data['good\_will']

total\_assets = data['total\_assets']

Alpha\_16\_yy= good\_will/total\_assets

return -Alpha\_16\_yy.mean()

Alpha\_10：利润总额与总资产的比值的20天平均

Alpha\_17\_yy= (total\_profit/total\_assets).rolling(window=20).mean()

构造因子的逻辑：利润总额与总资产的比值与ROA不同，但同样也能反映出该公司的价值

class Alpha\_17\_yy(Factor):

name = 'Alpha\_17\_yy'

max\_window =20

dependencies = ['total\_profit','total\_assets']

def calc(self, data):

total\_profit = data['total\_profit']

total\_assets = data['total\_assets']

Alpha\_17\_yy= total\_profit/total\_assets

return Alpha\_17\_yy.mean()

Alpha\_11：中单净额的20天平均

Alpha\_19\_yy= net\_amount\_m.rolling(window=20).mean()

构造因子的逻辑：中单净额大小能反映出一般投资者的喜好

class Alpha\_19\_yy(Factor):

name = 'Alpha\_19\_yy'

max\_window =20

dependencies = ['net\_amount\_m']

def calc(self, data):

net\_amount\_m = data['net\_amount\_m']

Alpha\_19\_yy= net\_amount\_m

return -Alpha\_19\_yy.mean()

Alpha\_15: 小单净占比

Alpha\_22\_yy= -net\_pct\_s.rolling(window=1).mean()

构造因子的逻辑： 小单净占比能够反映出专业投资机构对该股票的评估

class Alpha\_22\_yy(Factor):

name = 'Alpha\_22\_yy'

max\_window =1

dependencies = ['net\_pct\_s']

def calc(self, data):

net\_pct\_s = data['net\_pct\_s']

Alpha\_22\_yy= net\_pct\_s

return -Alpha\_22\_yy.mean()