环境：

Window 7

Anaconda

<https://www.continuum.io/downloads>

一、 数据准备：

**1）市场数据：**

Benchmark、treasuries、交易日信息

market\_config\_location：市场数据文件夹路径（直到最后一层文件夹名）

如：E:\Anaconda\Lib\site-packages\zipline\_china\zipline\cache\MarketConfig

按规则准备好benchmark\_daily.csv或benchmark\_minute1.csv，和treasuries.csv，使用tools文件夹里的convert\_benchmark.py工具处理数据（得到 pkl文件）

tradingdates.csv：按规则准备好数据

treasuries.csv: 从Wind更新trasuries.csv

**2）K线数据：**

*Daily数据和自定义数据：*

market\_data\_location: 数据文件夹路径（直到最后一层文件夹名）

如：E:\Anaconda\Lib\site-packages\zipline\_china\zipline\cache\DailyData

用tools文件夹里的convert\_dailydata.py工具处理k线数据（从Wind下载的原始数据），得到相应的pkl文件（分为volume, close, high, low, open）

*自定义数据：*

同样的方法

注意：输出的pkl文件名称必须是volume, close, high, low, open（不带.pkl），对自定义数据名称自己取，但之后要在handle\_data（history）里访问时必须引用和文件名相同的变量名，如”cap”

*分钟数据：*

使用convert\_minutedata.py，需要给出原高频数据文件夹和输出文件夹（可以是同一个，覆盖原数据），以及最后的文件（最好以.pkl结尾）

**3）Fundamental数据：**

下载的fundamental数据表都放在同一个folder内，

每一张表是一种data，表名和data名相同，例如”roe.csv”

在这个文件夹内需要放额外的一个csv文件”announcement\_date.csv”，可从Wind下载

每一张表（包括announcement\_date）都必须有相同的维度

短期可以改进的地方：

写一些自动连接Wind数据库的api，不用手动处理data

远期：

建一个数据库方便管理

二、回测：

举例：

实例1.

import sys

sys.path.insert(0,"E:\Anaconda\Lib\site-packages\zipline\_china")

import pandas as pd

import datetime

import pickle

import pytz

from zipline.DailyData import dataGen

import time

source = dataGen.dailyData()

#data and config

market\_config\_location = r'D:\Data\MarketConfig'

market\_data\_location = r'D:\Data\DailyData'

RegularSourceType = 'pickle'

CustomFileList = [r'D:\Data\DailyData\cap']

CustomSourceType = 'pickle'

datasource = source(CustomSourceType = CustomSourceType, RegularSourceType = RegularSourceType, RegularSourceDirectory = market\_data\_location,\

CustomFileList = CustomFileList)

#backtest setting"

data\_frequency = 'daily'

emission\_rate = 'daily'

period\_start = pytz.utc.localize(datetime.datetime.strptime("2005/01/01",'%Y/%m/%d'))

period\_end = pytz.utc.localize(datetime.datetime.strptime("2015/01/01",'%Y/%m/%d'))

from zipline.api import (

add\_history,

add\_fundamental,

history,

get\_fundamentals,

order\_target\_percent,

cancel\_order,

record,

symbol,

set\_slippage,

set\_commission

)

import zipline as zp

import math

from zipline.finance.slippage import SlippageModel, create\_transaction

from zipline.finance.commission import PerDollar

def initialize(context):

# Register 2 histories that track daily prices,

# one with a 100 window and one with a 300 day window

add\_history(101, '1d', 'price')

add\_history(1,'1d','cap')

context.old\_status = None

context.timespent1 = []

context.timespent2 = []

context.touchlimit = 0

set\_slippage(MySlippage(0.1, 0.0))

set\_commission(PerDollar())

class MySlippage(SlippageModel):

def \_\_init\_\_(self, volume\_limit=0.1, price\_impact=0.0):

self.volume\_limit = volume\_limit

self.price\_impact = price\_impact

def \_\_repr\_\_(self):

return """

{class\_name}(

volume\_limit={volume\_limit},

price\_impact={price\_impact})

""".strip().format(class\_name=self.\_\_class\_\_.\_\_name\_\_,

volume\_limit=self.volume\_limit,

price\_impact=self.price\_impact)

def process\_order(self, event, order):

max\_volume = self.volume\_limit \* event.volume

# price impact accounts for the total volume of transactions

# created against the current minute bar

remaining\_volume = max\_volume - self.volume\_for\_bar

if remaining\_volume <= 100:

# we can't fill any more transactions

return

# the current order amount will be the min of the

# volume available in the bar or the open amount.

# volume needs to be the multiple of 100

cur\_volume = int(min(math.floor(remaining\_volume/100)\*100, abs(order.open\_amount)))

if cur\_volume < 100:

return

# tally the current amount into our total amount ordered.

# total amount will be used to calculate price impact

total\_volume = self.volume\_for\_bar + cur\_volume

# volume\_share is the percentage of our volume of trade in the total volume for that security

volume\_share = min(total\_volume / event.volume,

self.volume\_limit)

simulated\_impact = volume\_share \*\* 2 \

\* math.copysign(self.price\_impact, order.direction) \

\* event.price

return create\_transaction(

event,

order,

# In the future, we may want to change the next line

# for limit pricing

event.price + simulated\_impact,

math.copysign(cur\_volume, order.direction)

)

def handle\_data(context, data):

# Skip first 300 days to get full windows

# Compute averages

# history() has to be called with the same params

# from above and returns a pandas dataframe.

print "time: %s"%context.datetime

prices = history(101, '1d', 'price')

index = symbol('000001.SH')

index\_return = (prices[index].iloc[100] - prices[index].iloc[0])/prices[index].iloc[0]

if index\_return > -0.03:

context.selected\_sids = None

cap = history(1, '1d', 'cap')

cap = cap.iloc[0]

cap.sort()

small\_caps = cap.iloc[:50]

context.selected\_sids = list(small\_caps.index)

small\_caps\_prices = prices[context.selected\_sids]

returns = (small\_caps\_prices.iloc[100] - small\_caps\_prices.iloc[0])/small\_caps\_prices.iloc[0]

with\_low\_returns = list(returns.index[returns < index\_return - 0.3])

context.selected = with\_low\_returns

context.N = len(context.selected)

for sym in context.selected:

order\_target\_percent(sym, 1.0/context.N)

record(cap = cap.ix[sym], returns = returns.ix[sym])

existing\_orders = context.get\_open\_orders()

#cancel open orders of those not selected

for sym, orders\_sym in existing\_orders.iteritems():

if sym not in context.selected:

for order in orders\_sym:

cancel\_order(order)

#clear positions of those not selected

current\_positions = context.portfolio.positions

for sym in current\_positions:

if sym not in context.selected:

order\_target\_percent(sym, 0.0)

else:

existing\_orders = context.get\_open\_orders()

for sym, orders\_sym in existing\_orders.iteritems():

for order in orders\_sym:

cancel\_order(order)

order\_target\_percent(sym, 0.0)

# Save values for later inspection

TradingDictionary = {'initialize' : initialize,

'handle\_data' : handle\_data,

'market\_config\_location': market\_config\_location,

'data\_frequency': data\_frequency,

'emission\_rate': emission\_rate,

'period\_start': period\_start,

'period\_end': period\_end,

'warming\_period':10,

}

algo = zp.TradingAlgorithm(\*\*TradingDictionary)

#do not use source period start and period end

results = algo.run(datasource, overwrite\_sim\_params = False)

with open(r'D:\Data\RunResults\SmallCap030', 'w') as f:

pickle.dump([algo.sim\_params, algo.blotter.orders, algo.perf\_tracker.perf\_periods, algo.perf\_tracker.cumulative\_risk\_metrics], f)

三、分析回测结果

algo.sim\_params #回测参数

algo.perf\_tracker.returns #每日收益率

algo.perf\_tracker.cumulative\_risk\_metrics.algorithm\_returns #每日收益率

algo.perf\_tracker.cumulative\_risk\_metrics.benchmark\_returns #每日基准收益率

algo.perf\_tracker.cumulative\_performance.returns #总收益率

algo.perf\_tracker.todays\_performance.to\_dict() #看orders

algo.perf\_tracker.cumulative\_risk\_metrics #看整个时间段的risk报告

algo.perf\_tracker.intraday\_risk\_metrics #每日的risk报告

results #每日资产收益报告

algo.blotter.orders #所有order

algo.blotter.open\_orders #目前的pending orders