鼎松科技-量化研究总结表

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 策略名称 | “东方红4号”基金收益率的分解 | | | | | |
| 作者 | 王奚为 | | | | | |
| 所属分类 | 1.多因子 | 2.量化择时 | 3.行业配置 | 4.收益增强 | 5.衍生品对冲 | 6.其它() |
| a.基本面分析 | b.技术分析 | c.机器学习 | d.其它() | | |
| 思路来源 | 东方红4号基金的平均年复合回报接近30%。同伯克希尔哈萨维的收益一样，东方红4号基金的收益可以被分解为多个因子的共同作用之和。这意味着如果可以通过回归的方法找出因子对收益的贡献度，并检验他们的显著性，如果因子是显著的意味着因子对投资组合的收益率的确存在贡献度。同时，在回归过程中应该检验的显著性，并尝试将的显著性降为最低，这意味着投资组合中不存在不可解释的因素，即不会出现基金经理获取内幕消息的情况。 | | | | | |
| 参考资料(名称/网址/作者) | BUFFETT’S ALPHA(Andrea Frazzini, 2013) | | | | | |
| 研究构思 | 1. 因子确定   首先分析基金经理陈光明在公开媒体上所表示的选股策略，例如他坚持较低的换手率，坚持买高质量，便宜的股票，且曾经公开表示更信任价值投资的复合效益。结合论文在分解伯克希尔哈萨维的收益率的过程中所使用的因子，我在模型构建中优先选择了这六个因子进行回归。   1. 因子构建   从Wind数据库中获取东方红基金从2009年4月13日到2016年12月31日的每日净值数据，通过处理转换为月度数据。然后从聚宽数据库中获取上述所选的5个因子的数据并进行分层回测，使用评分最高的一层股票减去评分最低的一层股票，构建多空组合，获得全部因子的数据，选用沪深300指数数据作为MKT因子。以年收益率4%作为无风险利率，计算月度收益率。  3. 回归方程构建  （1）通过回归确定各个因子的显著性，剔除一些几乎完全不显著的因子。并多次尝试加入一些显著性较强的因子进行回归，将的显著性降到最低。  （2）然后将东方红4号的超额收益率对MKT因子做回归，代表了市场因子此时可以在多大程度上解释基金的超额收益率。此时MKT因子前对应的系数即为  （3）用东方红4号的超额收益率减去陈光明的投资组合可以被市场因子解释的那一部分，剩下的部分代表了由于陈光明独特的选股策略所产生的超额收益部分。因此再次对六因子进行回归。即  （4）  由于之前的回归中已经将变得不显著了，因此上述等式右侧因子的构建部分就代表了陈光明积极地选股策略。  （5）最后将陈光明的积极的投资策略加上投资组合的市场敞口所对应的回报率和无风险利率，可以得到东方红4号基金特有的收益率。 | | | | | |
| 实验流程(包括重要假设、细节、修改、改进等) | 1. 时间序列回归  在使用时间序列数据做回归之前，必须先进行序列相关性检验。对所有因子以及基金收益率的使用Adfuller检验，都可以得到相当小的p值，因此序列是平稳的。  2. 基准确定  在选用基准的过程中，分别将超额收益率和沪深300、中证500、中证700、上证50做回归。发现唯有沪深300的相关性最高，因此在分层回测过程中选用的基准为沪深300  3. 因子的选择  在实际的回归当中，发现动量因子(UMD)和价值因子(HML)并不显著，因此去除了动量因子。HML因子的不显著和我们通常设想的不符，鉴于价值因子在常规的多因子模型中的有效性，因此重新使用聚宽数据库调用了与价值因子相关的其他因子，选用了与之相近的三个因子roa, roic和roe，经过多次测试发现。roa表现稍好于roe，roic的显著性最差，因此保留了价值因子。  很多之前认为会相当显著的因子在实际检验中都不显著，如size, book\_to\_price\_ratio等因子。事实上，EP因子要比BP因子更合适。 | | | | | |
| 结果展示 | **Add liquidity only**    **Add Variance 60 only**    **Add leverage only:**    **Leverage+Variance60**    **Leverage + liquidity**    **Liquidity + Variance60**    **Liquidity + Variance60 + leverage** | | | | | |
| 研究结论 | 由回归结果可以得到，始终显著的是因子是市场因子，相较于BP因子，EP因子更加显著。更适合用来回归。  相对来讲，liquidity 和leverage 的组合能够使的显著性降到最低  显著性不够的原因有可能  （1）所选的股票池太小，使用中证800代替也许会更好  （2）陈光明可以获得内幕信息 | | | | | |
| 评价反思 | 1. 始终有着较高的显著性，仍然不能很好的消除不能解释的超额收益。   2. BAB因子的构建可以进一步改进，目前已经有更好的构建方法。  3. 扩大股票池，使用中证800的股票池 | | | | | |
| 代码(附注释) | #stratified\_backtest.ipynb  #1 先导入所需要的程序包  import datetime  import numpy as np  import pandas as pd  import time  from jqdata import \*  from pandas import Series, DataFrame  import matplotlib.pyplot as plt  import seaborn as sns  import itertools  import copy  import pickle  import os  import csv  '''  注意：  size选用排序为False  book\_to\_price\_ratio排序为False  roe\_ttm选用排序为True  beta选用True  momentum选用True  money\_flow\_20 选用False  liquidity选用False  Variance60选用False  BIAS60选用False  earnings\_yield选用False  financial\_expense\_rate选用False  turnover\_volatility选用False  residual\_volatility选用False  leverage选用False  '''  f='leverage'  if (os.path.exists('C:/Users/dsquant06/Desktop/test/'+f)==False):  os.mkdir('C:/Users/dsquant06/Desktop/test/'+f)  path='C:/Users/dsquant06/Desktop/test/test.csv'  # 定义类'参数分析'  class parameter\_analysis(object):    # 定义函数中不同的变量  def \_\_init\_\_(self, algorithm\_id=None):  self.algorithm\_id = algorithm\_id # 回测id    self.params\_df = pd.DataFrame() # 回测中所有调参备选值的内容，列名字为对应修改面两名称，对应回测中的 g.XXXX  self.results = {} # 回测结果的回报率，key 为 params\_df 的行序号，value 为  self.evaluations = {} # 回测结果的各项指标，key 为 params\_df 的行序号，value 为一个 dataframe  self.backtest\_ids = {} # 回测结果的 id    # 新加入的基准的回测结果 id，可以默认为空 ''，则使用回测中设定的基准  self.benchmark\_id = '000300.XSHG'    self.benchmark\_returns = [] # 新加入的基准的回测回报率  self.returns = {} # 记录所有回报率  self.excess\_returns = {} # 记录超额收益率  self.log\_returns = {} # 记录收益率的 log 值  self.log\_excess\_returns = {} # 记录超额收益的 log 值  self.dates = [] # 回测对应的所有日期  self.excess\_max\_drawdown = {} # 计算超额收益的最大回撤  self.excess\_annual\_return = {} # 计算超额收益率的年化指标  self.evaluations\_df = pd.DataFrame() # 记录各项回测指标，除日回报率外    # 定义排队运行多参数回测函数  def run\_backtest(self, #  algorithm\_id=None, # 回测策略id  running\_max=10, # 回测中同时巡行最大回测数量  start\_date='2009-04-12', # 回测开始时间  end\_date='2017-01-01', # 回测结束日期  frequency='day', # 回测的运行频率  initial\_cash='100000000', # 回测的初始持仓金额  param\_names=[], # 回测中调整参数涉及的变量  param\_values=[] # 回测中每个变量的备选参数值  ):  # 当此处回测策略的 id 没有给出时，调用类输入的策略 id  if algorithm\_id == None: algorithm\_id=self.algorithm\_id    # 生成所有参数组合并加载到 df 中  # 包含了不同参数具体备选值的排列组合中一组参数的 tuple 的 list  param\_combinations = list(itertools.product(\*param\_values))  #print(param\_combinations)  # 生成一个 dataframe， 对应的列为每个调参的变量，每个值为调参对应的备选值  to\_run\_df = pd.DataFrame(param\_combinations)  # 修改列名称为调参变量的名字  to\_run\_df.columns = param\_names    # 设定运行起始时间和保存格式  start = time.time()  # 记录结束的运行回测  finished\_backtests = {}  # 记录运行中的回测  running\_backtests = {}  # 计数器  pointer = 0  # 总运行回测数目，等于排列组合中的元素个数  total\_backtest\_num = len(param\_combinations)  # 记录回测结果的回报率  all\_results = {}  # 记录回测结果的各项指标  all\_evaluations = {}    # 在运行开始时显示  print('【已完成|运行中|待运行】:')  # 当运行回测开始后，如果没有全部运行完全的话：  while len(finished\_backtests)<total\_backtest\_num:  # 显示运行、完成和待运行的回测个数  print('[%s|%s|%s].' % (len(finished\_backtests),  len(running\_backtests),  (total\_backtest\_num-len(finished\_backtests)-len(running\_backtests)) )),  # 记录当前运行中的空位数量  to\_run = min(running\_max-len(running\_backtests), total\_backtest\_num-len(running\_backtests)-len(finished\_backtests))  # 把可用的空位进行跑回测  for i in range(pointer, pointer+to\_run):  # 备选的参数排列组合的 df 中第 i 行变成 dict，每个 key 为列名字，value 为 df 中对应的值  params = to\_run\_df.iloc[i].to\_dict()  # 记录策略回测结果的 id，调整参数 extras 使用 params 的内容  backtest = create\_backtest(algorithm\_id = algorithm\_id,  start\_date = start\_date,  end\_date = end\_date,  frequency = frequency,  initial\_cash = initial\_cash,  extras = params,  # 再回测中把改参数的结果起一个名字，包含了所有涉及的变量参数值  name = str(params)  )  # 记录运行中 i 回测的回测 id  running\_backtests[i] = backtest  # 计数器计数运行完的数量  pointer = pointer+to\_run    # 获取回测结果  failed = []  finished = []  # 对于运行中的回测，key 为 to\_run\_df 中所有排列组合中的序数  for key in running\_backtests.keys():  # 研究调用回测的结果，running\_backtests[key] 为运行中保存的结果 id  bt = get\_backtest(running\_backtests[key])  #print(bt.backtest\_id)  # 获得运行回测结果的状态，成功和失败都需要运行结束后返回，如果没有返回则运行没有结束  status = bt.get\_status()  # 当运行回测失败  if status == 'failed':  # 失败 list 中记录对应的回测结果 id  failed.append(key)  # 当运行回测成功时  elif status == 'done':  # 成功 list 记录对应的回测结果 id，finish 仅记录运行成功的  finished.append(key)  # 回测回报率记录对应回测的回报率 dict， key to\_run\_df 中所有排列组合中的序数， value 为回报率的 dict  # 每个 value 一个 list 每个对象为一个包含时间、日回报率和基准回报率的 dict  all\_results[key] = bt.get\_results()  # 回测回报率记录对应回测结果指标 dict， key to\_run\_df 中所有排列组合中的序数， value 为回测结果指标的 dataframe  all\_evaluations[key] = bt.get\_risk()  # 记录运行中回测结果 id 的 list 中删除失败的运行  for key in failed:  print('第'+str(key)+'组回测失败！')  finished\_backtests[key] = running\_backtests.pop(key)  # 在结束回测结果 dict 中记录运行成功的回测结果 id，同时在运行中的记录中删除该回测  for key in finished:  finished\_backtests[key] = running\_backtests.pop(key)  # 当一组同时运行的回测结束时报告时间  if len(finished\_backtests) != 0 and len(finished\_backtests) % running\_max == 0 and to\_run !=0:  # 记录当时时间  middle = time.time()  # 计算剩余时间，假设没工作量时间相等的话  remain\_time = (middle - start) \* (total\_backtest\_num - len(finished\_backtests)) / len(finished\_backtests)  # print 当前运行时间  print('[已用%s时,尚余%s时,请不要关闭浏览器].' % (str(round((middle - start) / 60.0 / 60.0,3)),  str(round(remain\_time / 60.0 / 60.0,3)))),  # 5秒钟后再跑一下  time.sleep(5)  # 记录结束时间  end = time.time()  print('')  print('【回测完成】总用时：%s秒(即%s小时)。' % (str(int(end-start)),  str(round((end-start)/60.0/60.0,2)))),  # 对应修改类内部对应  self.params\_df = to\_run\_df  self.results = all\_results  self.evaluations = all\_evaluations  self.backtest\_ids = finished\_backtests    #7 最大回撤计算方法  def find\_max\_drawdown(self, returns):  # 定义最大回撤的变量  result = 0  # 记录最高的回报率点  historical\_return = 0  # 遍历所有日期  for i in range(len(returns)):  # 最高回报率记录  historical\_return = max(historical\_return, returns[i])  # 最大回撤记录  drawdown = 1-(returns[i] + 1) / (historical\_return + 1)  # 记录最大回撤  result = max(drawdown, result)  # 返回最大回撤值  return result  # log 收益、新基准下超额收益和相对与新基准的最大回撤  def organize\_backtest\_results(self, benchmark\_id=None):  # 若新基准的回测结果 id 没给出  if benchmark\_id==None:  # 使用默认的基准回报率，默认的基准在回测策略中设定  #print(self.results)  self.benchmark\_returns = [x['benchmark\_returns'] for x in self.results[0]]  # 当新基准指标给出后  else:  # 基准使用新加入的基准回测结果  self.benchmark\_returns = [x['returns'] for x in get\_backtest(benchmark\_id).get\_results()]  # 回测日期为结果中记录的第一项对应的日期  self.dates = [x['time'] for x in self.results[0]]    # 对应每个回测在所有备选回测中的顺序 （key），生成新数据  # 由 {key：{u'benchmark\_returns': 0.022480100091729405,  # u'returns': 0.03184566700000002,  # u'time': u'2006-02-14'}} 格式转化为：  # {key: []} 格式，其中 list 为对应 date 的一个回报率 list  for key in self.results.keys():  self.returns[key] = [x['returns'] for x in self.results[key]]  # 生成对于基准（或新基准）的超额收益率  for key in self.results.keys():  self.excess\_returns[key] = [(x+1)/(y+1)-1 for (x,y) in zip(self.returns[key], self.benchmark\_returns)]  # 生成 log 形式的收益率  for key in self.results.keys():  self.log\_returns[key] = [log(x+1) for x in self.returns[key]]  # 生成超额收益率的 log 形式  for key in self.results.keys():  self.log\_excess\_returns[key] = [log(x+1) for x in self.excess\_returns[key]]  # 生成超额收益率的最大回撤  for key in self.results.keys():  self.excess\_max\_drawdown[key] = self.find\_max\_drawdown(self.excess\_returns[key])  # 生成年化超额收益率  for key in self.results.keys():  self.excess\_annual\_return[key] = (self.excess\_returns[key][-1]+1)\*\*(252./float(len(self.dates)))-1  # 把调参数据中的参数组合 df 与对应结果的 df 进行合并  self.evaluations\_df = pd.concat([self.params\_df, pd.DataFrame(self.evaluations).T], axis=1)  # self.evaluations\_df =  # 获取最总分析数据，调用排队回测函数和数据整理的函数  def get\_backtest\_data(self,  algorithm\_id=None, # 回测策略id  benchmark\_id=None, # 新基准回测结果id  file\_name='results.pkl', # 保存结果的 pickle 文件名字  running\_max=10, # 最大同时运行回测数量  start\_date='2009-04-12', # 回测开始时间  end\_date='2017-01-01', # 回测结束日期  frequency='day', # 回测的运行频率  initial\_cash='100000000', # 回测初始持仓资金  param\_names=[], # 回测需要测试的变量  param\_values=[] # 对应每个变量的备选参数  ):  # 调运排队回测函数，传递对应参数  self.run\_backtest(algorithm\_id=algorithm\_id,  running\_max=running\_max,  start\_date=start\_date,  end\_date=end\_date,  frequency=frequency,  initial\_cash=initial\_cash,  param\_names=param\_names,  param\_values=param\_values  )  # 回测结果指标中加入 log 收益率和超额收益率等指标  self.organize\_backtest\_results(benchmark\_id)  # 生成 dict 保存所有结果。  results = {'returns':self.returns,  'excess\_returns':self.excess\_returns,  'log\_returns':self.log\_returns,  'log\_excess\_returns':self.log\_excess\_returns,  'dates':self.dates,  'benchmark\_returns':self.benchmark\_returns,  'evaluations':self.evaluations,  'params\_df':self.params\_df,  'backtest\_ids':self.backtest\_ids,  'excess\_max\_drawdown':self.excess\_max\_drawdown,  'excess\_annual\_return':self.excess\_annual\_return,  'evaluations\_df':self.evaluations\_df}  # 保存 pickle 文件  pickle\_file = open(file\_name, 'wb')  pickle.dump(results, pickle\_file)  pickle\_file.close()  # 读取保存的 pickle 文件，赋予类中的对象名对应的保存内容  def read\_backtest\_data(self, file\_name='results.pkl'):  pickle\_file = open(file\_name, 'rb')  results = pickle.load(pickle\_file)  self.returns = results['returns']  self.excess\_returns = results['excess\_returns']  self.log\_returns = results['log\_returns']  self.log\_excess\_returns = results['log\_excess\_returns']  self.dates = results['dates']  self.benchmark\_returns = results['benchmark\_returns']  self.evaluations = results['evaluations']  self.params\_df = results['params\_df']  self.backtest\_ids = results['backtest\_ids']  self.excess\_max\_drawdown = results['excess\_max\_drawdown']  self.excess\_annual\_return = results['excess\_annual\_return']  self.evaluations\_df = results['evaluations\_df']    # 回报率折线图  def plot\_returns(self):  # 通过figsize参数可以指定绘图对象的宽度和高度，单位为英寸；  fig = plt.figure(figsize=(20,8))  ax = fig.add\_subplot(111)  # 作图  k = sorted(self.returns.keys())  for key in k:  ax.plot(range(len(self.returns[key])), self.returns[key], label=key)  # 设定benchmark曲线并标记  ax.plot(range(len(self.benchmark\_returns)), self.benchmark\_returns, label='benchmark', c='k', linestyle='--')  ticks = [int(x) for x in np.linspace(0, len(self.dates)-1, 11)]  plt.xticks(ticks, [self.dates[i] for i in ticks])  # 设置图例样式  ax.legend(loc = 2, fontsize = 10)  # 设置y标签样式  ax.set\_ylabel('returns',fontsize=20)  # 设置x标签样式  ax.set\_yticklabels([str(round(x\*100,0))+'% 'for x in ax.get\_yticks()])  # 设置图片标题样式  ax.set\_title("Strategy's performances with different parameters", fontsize=21)  plt.xlim(0, len(self.returns[0]))  plt.savefig("C:/Users/dsquant06/Desktop/test/"+f+"/回报率折现图.pdf")  # 超额收益率图  def plot\_excess\_returns(self):  # 通过figsize参数可以指定绘图对象的宽度和高度，单位为英寸；  fig = plt.figure(figsize=(20,8))  ax = fig.add\_subplot(111)  # 作图  k = sorted(self.returns.keys())  for key in k:  ax.plot(range(len(self.excess\_returns[key])), self.excess\_returns[key], label=key)  # 设定benchmark曲线并标记  ax.plot(range(len(self.benchmark\_returns)), [0]\*len(self.benchmark\_returns), label='benchmark', c='k', linestyle='--')  ticks = [int(x) for x in np.linspace(0, len(self.dates)-1, 11)]  plt.xticks(ticks, [self.dates[i] for i in ticks])  # 设置图例样式  ax.legend(loc = 2, fontsize = 10)  # 设置y标签样式  ax.set\_ylabel('excess returns',fontsize=20)  # 设置x标签样式  ax.set\_yticklabels([str(round(x\*100,0))+'% 'for x in ax.get\_yticks()])  # 设置图片标题样式  ax.set\_title("Strategy's performances with different parameters", fontsize=21)  plt.xlim(0, len(self.excess\_returns[0]))  plt.savefig("C:/Users/dsquant06/Desktop/test/"+f+"/超额收益率图.pdf")    # 多空组合折线图（long第一组，short最后一组）  # 指标计算方法见 https://www.joinquant.com/help/api/help?name=api#%E9%A3%8E%E9%99%A9%E6%8C%87%E6%A0%87  def plot\_long\_short\_portfolio\_returns(self):  #显示所有列  pd.set\_option('display.max\_columns', None)  #显示所有行  pd.set\_option('display.max\_rows', None)  # 通过figsize参数可以指定绘图对象的宽度和高度，单位为英寸；  fig = plt.figure(figsize=(20,8))  ax = fig.add\_subplot(111)  trading\_days = self.evaluations\_df['trading\_days'][0]    # 作图  k = [0,len(self.returns.keys())-1]  for key in k:  ax.plot(range(len(self.returns[key])), self.returns[key], label=key)  first\_group\_cumulative\_return = self.returns[0].copy()  last\_group\_cumulative\_return = self.returns[len(self.returns.keys())-1].copy()  first\_group\_net\_value = [x+1 for x in first\_group\_cumulative\_return]  last\_group\_net\_value = [x+1 for x in last\_group\_cumulative\_return]    #计算每日涨跌幅  first\_group\_daily\_return = [(first\_group\_net\_value[i+1]-first\_group\_net\_value[i])/first\_group\_net\_value[i] for i in range(len(first\_group\_net\_value)-1)]  last\_group\_daily\_return = [(last\_group\_net\_value[i+1]-last\_group\_net\_value[i])/last\_group\_net\_value[i] for i in range(len(last\_group\_net\_value)-1)]  ####################################################################################  portfolio\_daily\_return = [first\_group\_daily\_return[i]-last\_group\_daily\_return[i] for i in range(len(first\_group\_daily\_return))]    #################################################################################  #portfolio\_daily\_return为输出的结果，得到的为每日的回报率，计算方法为每日的（return+1）相乘最后减一  #计算累计涨跌幅  portfolio\_cumulative\_return = [0]  pf\_net\_value = [1]  net\_value = 1  for i in range(len(first\_group\_net\_value)-1):  net\_value = net\_value \* (1+first\_group\_daily\_return[i]-last\_group\_daily\_return[i])  pf\_net\_value.append(net\_value)  portfolio\_cumulative\_return.append(net\_value-1)  ax.plot(range(len(self.returns[k[0]])), portfolio\_cumulative\_return, label='portfolio')    # 计算benchmark的每日收益  bm\_cumulative\_return = self.benchmark\_returns.copy()  bm\_net\_value = [x+1 for x in bm\_cumulative\_return]  bm\_daily\_return = [(bm\_net\_value[i+1]-bm\_net\_value[i])/bm\_net\_value[i] for i in range(len(bm\_cumulative\_return)-1)]  pf = pd.DataFrame(portfolio\_daily\_return,columns=['pf'])  bm = pd.DataFrame(bm\_daily\_return,columns=['bm'])  cov\_pf\_bm = pf['pf'].cov(bm['bm'])  bm\_var = bm['bm'].var()  pf\_var = pf['pf'].var()    #计算各个指标  attributes = []  columns = ['total\_returns','total\_annualized\_returns', 'benchmark\_returns', 'alpha','beta','sharpe\_ratio','volatility','max\_drawdown']  Rf = 0.04 #无风险收益率    total\_returns = portfolio\_cumulative\_return[-1]  attributes.append(str(round(total\_returns \* 100, 2))+'%')    bm\_total\_returns = bm\_cumulative\_return[-1]  total\_annualized\_returns = (1+total\_returns)\*\*(250\*1.0/trading\_days) - 1  attributes.append(str(round(total\_annualized\_returns \* 100, 2))+'%')  attributes.append(str(round(bm\_total\_returns \* 100, 2))+'%')    bm\_total\_annualized\_returns = (1+bm\_total\_returns)\*\*(250\*1.0/trading\_days) - 1  beta = cov\_pf\_bm/bm\_var  alpha = total\_annualized\_returns - (Rf + beta \* (bm\_total\_annualized\_returns - Rf))  attributes.append(round(alpha,2))  attributes.append(round(beta,2))    volatility = np.sqrt(250 \* pf\_var)  sharpe\_ratio = (total\_annualized\_returns - Rf) / volatility  attributes.append(round(sharpe\_ratio,2))  attributes.append(str(round(volatility \* 100, 2))+'%')    drawdown = {}  for i in range(len(pf\_net\_value)):  for j in range(i+1,len(pf\_net\_value)):  drawdown[(i,j)] = (pf\_net\_value[i]-pf\_net\_value[j])/pf\_net\_value[i]  drawdown = sorted(drawdown.items(), key=lambda drawdown:drawdown[1],reverse = True)  (start, end) = drawdown[0][0]  max\_drawdown = drawdown[0][1]    ax.scatter(start, portfolio\_cumulative\_return[start],color='r',s=20)  ax.scatter(end, portfolio\_cumulative\_return[end],color='r',s=20)  attributes.append(str(round(max\_drawdown \* 100, 2))+'%')    df = pd.DataFrame(attributes,index=columns,columns=[''])  df = df.T  print(df)      # 设定benchmark曲线并标记  ax.plot(range(len(self.benchmark\_returns)), self.benchmark\_returns, label='benchmark', c='k', linestyle='--')  ticks = [int(x) for x in np.linspace(0, len(self.dates)-1, 11)]  plt.xticks(ticks, [self.dates[i] for i in ticks])  # 设置图例样式  ax.legend(loc = 2, fontsize = 10)  # 设置y标签样式  ax.set\_ylabel('portfolio returns',fontsize=20)  # 设置x标签样式  ax.set\_yticklabels([str(round(x\*100,0))+'% 'for x in ax.get\_yticks()])  # 设置图片标题样式  ax.set\_title("Strategy's performances with different parameters", fontsize=21)  plt.xlim(0, len(self.returns[0]))  plt.savefig("C:/Users/dsquant06/Desktop/test/"+f+"/多空组合折线图.pdf")    # log回报率图  def plot\_log\_returns(self):  # 通过figsize参数可以指定绘图对象的宽度和高度，单位为英寸；  fig = plt.figure(figsize=(20,8))  ax = fig.add\_subplot(111)  # 作图  k = sorted(self.returns.keys())  for key in k:  ax.plot(range(len(self.log\_returns[key])), self.log\_returns[key], label=key)  # 设定benchmark曲线并标记  ax.plot(range(len(self.benchmark\_returns)), [log(x+1) for x in self.benchmark\_returns], label='benchmark', c='k', linestyle='--')  ticks = [int(x) for x in np.linspace(0, len(self.dates)-1, 11)]  plt.xticks(ticks, [self.dates[i] for i in ticks])  # 设置图例样式  ax.legend(loc = 2, fontsize = 10)  # 设置y标签样式  ax.set\_ylabel('log returns',fontsize=20)  # 设置图片标题样式  ax.set\_title("Strategy's performances with different parameters", fontsize=21)  plt.xlim(0, len(self.log\_returns[0]))  plt.savefig("C:/Users/dsquant06/Desktop/test/"+f+"/log回报率图.pdf")    # 超额收益率的 log 图  def plot\_log\_excess\_returns(self):  # 通过figsize参数可以指定绘图对象的宽度和高度，单位为英寸；  fig = plt.figure(figsize=(20,8))  ax = fig.add\_subplot(111)  # 作图  k = sorted(self.returns.keys())  for key in k:  ax.plot(range(len(self.log\_excess\_returns[key])), self.log\_excess\_returns[key], label=key+1)  # 设定benchmark曲线并标记  ax.plot(range(len(self.benchmark\_returns)), [0]\*len(self.benchmark\_returns), label='benchmark', c='k', linestyle='--')  ticks = [int(x) for x in np.linspace(0, len(self.dates)-1, 11)]  plt.xticks(ticks, [self.dates[i] for i in ticks])  # 设置图例样式  ax.legend(loc = 2, fontsize = 10)  # 设置y标签样式  ax.set\_ylabel('log excess returns',fontsize=20)  # 设置图片标题样式  ax.set\_title("Strategy's performances with different parameters", fontsize=21)  plt.xlim(0, len(self.log\_excess\_returns[0]))  plt.savefig("C:/Users/dsquant06/Desktop/test/"+f+"/超额收益率log图.pdf")    # 回测的4个主要指标，包括总回报率、最大回撤夏普率和波动  def get\_eval4\_bar(self, sort\_by=[]):    sorted\_params = self.params\_df  for by in sort\_by:  sorted\_params = sorted\_params.sort(by)  indices = sorted\_params.index    fig = plt.figure(figsize=(20,7))  # 定义位置  ax1 = fig.add\_subplot(221)  # 设定横轴为对应分位，纵轴为对应指标  ax1.bar(range(len(indices)),  [self.evaluations[x]['algorithm\_return'] for x in indices], 0.6, label = 'Algorithm\_return')  plt.xticks([x+0.3 for x in range(len(indices))], indices)  # 设置图例样式  ax1.legend(loc='best',fontsize=15)  # 设置y标签样式  ax1.set\_ylabel('Algorithm\_return', fontsize=15)  # 设置y标签样式  ax1.set\_yticklabels([str(round(x\*100,0))+'% 'for x in ax1.get\_yticks()])  # 设置图片标题样式  ax1.set\_title("Strategy's of Algorithm\_return performances of different quantile", fontsize=15)  # x轴范围  plt.xlim(0, len(indices))  # 定义位置  ax2 = fig.add\_subplot(224)  # 设定横轴为对应分位，纵轴为对应指标  ax2.bar(range(len(indices)),  [self.evaluations[x]['max\_drawdown'] for x in indices], 0.6, label = 'Max\_drawdown')  plt.xticks([x+0.3 for x in range(len(indices))], indices)  # 设置图例样式  ax2.legend(loc='best',fontsize=15)  # 设置y标签样式  ax2.set\_ylabel('Max\_drawdown', fontsize=15)  # 设置x标签样式  ax2.set\_yticklabels([str(round(x\*100,0))+'% 'for x in ax2.get\_yticks()])  # 设置图片标题样式  ax2.set\_title("Strategy's of Max\_drawdown performances of different quantile", fontsize=15)  # x轴范围  plt.xlim(0, len(indices))  # 定义位置  ax3 = fig.add\_subplot(223)  # 设定横轴为对应分位，纵轴为对应指标  ax3.bar(range(len(indices)),  [self.evaluations[x]['sharpe'] for x in indices], 0.6, label = 'Sharpe')  # print(indices)  # print(self.evaluations[indices[0]])  plt.xticks([x+0.3 for x in range(len(indices))], indices)  # 设置图例样式  ax3.legend(loc='best',fontsize=15)  # 设置y标签样式  ax3.set\_ylabel('Sharpe', fontsize=15)  # 设置x标签样式  ax3.set\_yticklabels([str(round(x\*100,0))+'% 'for x in ax3.get\_yticks()])  # 设置图片标题样式  ax3.set\_title("Strategy's of Sharpe performances of different quantile", fontsize=15)  # x轴范围  plt.xlim(0, len(indices))  # 定义位置  ax4 = fig.add\_subplot(222)  # 设定横轴为对应分位，纵轴为对应指标  ax4.bar(range(len(indices)),  [self.evaluations[x]['algorithm\_volatility'] for x in indices], 0.6, label = 'Algorithm\_volatility')  plt.xticks([x+0.3 for x in range(len(indices))], indices)  # 设置图例样式  ax4.legend(loc='best',fontsize=15)  # 设置y标签样式  ax4.set\_ylabel('Algorithm\_volatility', fontsize=15)  # 设置x标签样式  ax4.set\_yticklabels([str(round(x\*100,0))+'% 'for x in ax4.get\_yticks()])  # 设置图片标题样式  ax4.set\_title("Strategy's of Algorithm\_volatility performances of different quantile", fontsize=15)  # x轴范围  plt.xlim(0, len(indices))    #13 年化回报和最大回撤，正负双色表示  def get\_eval(self, sort\_by=[]):  sorted\_params = self.params\_df  for by in sort\_by:  sorted\_params = sorted\_params.sort(by)  indices = sorted\_params.index    # 大小  fig = plt.figure(figsize = (20, 8))  # 图1位置  ax = fig.add\_subplot(111)  # 生成图超额收益率的最大回撤  ax.bar([x+0.3 for x in range(len(indices))],\  [-self.evaluations[x]['max\_drawdown'] for x in indices], color = '#32CD32',  width = 0.6, label = 'Max\_drawdown', zorder=10)  # 图年化超额收益  ax.bar([x for x in range(len(indices))],\  [self.evaluations[x]['annual\_algo\_return'] for x in indices], color = 'r',  width = 0.6, label = 'Annual\_return')  plt.xticks([x+0.3 for x in range(len(indices))], indices)  # 设置图例样式  ax.legend(loc='best',fontsize=15)  # 基准线  plt.plot([0, len(indices)], [0, 0], c='k',\  linestyle='--', label='zero')  # 设置图例样式  ax.legend(loc='best',fontsize=15)  # 设置y标签样式  ax.set\_ylabel('Max\_drawdown', fontsize=15)  # 设置x标签样式  ax.set\_yticklabels([str(round(x\*100,0))+'% 'for x in ax.get\_yticks()])  # 设置图片标题样式  ax.set\_title("Strategy's performances of different quantile", fontsize=15)  # 设定x轴长度  plt.xlim(0, len(indices))  #14 超额收益的年化回报和最大回撤  # 加入新的benchmark后超额收益和  def get\_excess\_eval(self, sort\_by=[]):  sorted\_params = self.params\_df  for by in sort\_by:  sorted\_params = sorted\_params.sort(by)  indices = sorted\_params.index    # 大小  fig = plt.figure(figsize = (20, 8))  # 图1位置  ax = fig.add\_subplot(111)  # 生成图超额收益率的最大回撤  ax.bar([x+0.3 for x in range(len(indices))],\  [-self.excess\_max\_drawdown[x] for x in indices], color = '#32CD32',  width = 0.6, label = 'Excess\_max\_drawdown')  # 图年化超额收益  ax.bar([x for x in range(len(indices))],\  [self.excess\_annual\_return[x] for x in indices], color = 'r',  width = 0.6, label = 'Excess\_annual\_return')  plt.xticks([x+0.3 for x in range(len(indices))], indices)  # 设置图例样式  ax.legend(loc='best',fontsize=15)  # 基准线  plt.plot([0, len(indices)], [0, 0], c='k', \  linestyle='--', label='zero')  # 设置图例样式  ax.legend(loc='best',fontsize=15)  # 设置y标签样式  ax.set\_ylabel('Max\_drawdown', fontsize=15)  # 设置x标签样式  ax.set\_yticklabels([str(round(x\*100,0))+'% 'for x in ax.get\_yticks()])  # 设置图片标题样式  ax.set\_title("Strategy's performances of different quantile", fontsize=15)  # 设定x轴长度  plt.xlim(0, len(indices))  pa = parameter\_analysis('70')  #3 运行回测  pa.get\_backtest\_data(file\_name = 'results.pkl',  running\_max = 10,  benchmark\_id = None,  start\_date='2009-04-12',  end\_date='2017-01-01',  frequency = 'day',  initial\_cash = '100000000',  param\_names = ['factor', 'quantile'],  param\_values = [[f], tuple(zip(range(0,100,10), range(10,101,10)))]  )  #4 数据读取  pa.read\_backtest\_data('results.pkl')  #5 回测参数的 Dataframe  pa.params\_df  #6 查看回测结果指标  pa.evaluations\_df  #7 回报率折线图  pa.plot\_returns()  #8 超额收益率图  pa.plot\_excess\_returns()  #9 多空组合折线图  pa.plot\_long\_short\_portfolio\_returns()  #10 log回报率图  pa.plot\_log\_returns()  #11 超额收益率的 log 图  pa.plot\_log\_excess\_returns()  #12 回测的4个主要指标，包括总回报率、最大回撤夏普率和波动  # get\_eval4\_bar(self, sort\_by=[])  pa.get\_eval4\_bar()  #13 年化回报和最大回撤，正负双色显示  # get\_eval(self, sort\_by=[])  pa.get\_eval()  #14 超额收益的年化回报和最大回撤  # 加入新的benchmark后超额收益和  # get\_excess\_eval(self, sort\_by=[])  pa.get\_excess\_eval()  # test 测试最后bar图中的sort\_by对应内容  param\_names=['abc','x\_y\_z']  param\_values=[['a','b','c'], [1,2]]  param\_combinations = list(itertools.product(\*param\_values))  to\_run\_df = pd.DataFrame(param\_combinations)  to\_run\_df.columns = param\_names  # to\_run\_df.ix[1].to\_dict()  to\_run\_df  # sort\_by = ['abc']  sort\_by = ['abc', 'x\_y\_z']  # sort\_by = ['x\_y\_z']  # sort\_by = ['x\_y\_z','abc']  sorted\_params = to\_run\_df  for by in sort\_by:  sorted\_params = sorted\_params.sort\_values(by)  indices = sorted\_params.index  sorted\_params  def date\_output(path\_return):  with open(path\_return,'w') as f:  f.write('{0}\n'.format('time'))  for j in range(len(pa.results[1])):  f.write('{0}\n'.format(pa.results[1][j]['time']))    def concat\_csv(path\_return,factor,portfolio):  if (os.path.exists(path\_return)==False):  date\_output(path\_return)  cache=pd.DataFrame(portfolio,columns=[factor])  cache=cache+1  temp=pd.read\_csv(path\_return)  if f in temp.columns.values:  temp.drop(columns=f,axis=1,inplace=True)  print("已删除"+f)  temp=pd.concat([temp,cache],axis=1)  if (os.path.exists(path\_return)):  os.remove(path\_return)  temp.to\_csv(path\_return,index=False,header=True)  first\_group\_cumulative\_return = pa.returns[0].copy()  last\_group\_cumulative\_return = pa.returns[len(pa.returns.keys())-1].copy()  first\_group\_net\_value = [x+1 for x in first\_group\_cumulative\_return]  last\_group\_net\_value = [x+1 for x in last\_group\_cumulative\_return]  #计算每日涨跌幅  first\_group\_daily\_return = [(first\_group\_net\_value[i+1]-first\_group\_net\_value[i])/first\_group\_net\_value[i] for i in range(len(first\_group\_net\_value)-1)]  last\_group\_daily\_return = [(last\_group\_net\_value[i+1]-last\_group\_net\_value[i])/last\_group\_net\_value[i] for i in range(len(last\_group\_net\_value)-1)]  ####################################################################################  portfolio\_daily\_return = [first\_group\_daily\_return[i]-last\_group\_daily\_return[i] for i in range(len(first\_group\_daily\_return))]  #################################################################################  #portfolio\_daily\_return为输出的结果，得到的为每日的回报率，计算方法为每日的（return+1）相乘最后减一  #计算累计涨跌幅  portfolio\_cumulative\_return = [0]  pf\_net\_value = [1]  net\_value = 1  for i in range(len(first\_group\_net\_value)-1):  net\_value = net\_value \* (1+first\_group\_daily\_return[i]-last\_group\_daily\_return[i])  pf\_net\_value.append(net\_value)  portfolio\_cumulative\_return.append(net\_value-1)  concat\_csv(path,f,portfolio\_cumulative\_return)  #因子、基金、股指月度收益率数据以及基金和股指的回归.ipynb  import pandas as pd  import csv  import datetime  import numpy as np  from statsmodels.api import OLS  import matplotlib.pyplot as plt  import os  import re  import statsmodels.api as sm  from scipy import stats  #处理每个因子的月度收益率  #index='beta'#处理的数据字段  time='time'#对应的时间字段  path2='F:/test/month.csv'  path\_return='F:/test/test.csv'  def process\_csv(path2,docket,index):  if (os.path.exists(path2)==False):  with open(path2,'w') as f:  f.write('monthly rate input\n')  #path\_return为净值的csv  cache=pd.DataFrame(docket,columns=[index])  temp=pd.read\_csv(path2)  temp=pd.concat([temp,cache],axis=1)  messup(temp,path2)  temp.to\_csv(path2,index=False,header=True)  print('数据计算完成')  def process\_month\_return(index,time,path\_return,path2):  #path路径为读取保存每日净值文件的csv  df=pd.read\_csv(path\_return,encoding="gbk")  date=len(df.loc[:,[time]])-1  #处理股指的月收益率  docket=[]  value=df.loc[[date],[index]].values[0][0]  temp=(datetime.datetime.strptime(df.loc[[0],[time]].values[0][0],'%Y/%m/%d %H:%M')).month  for i in range(0,date,1):  if (temp!=(datetime.datetime.strptime(df.loc[[i],[time]].values[0][0],'%Y/%m/%d %H:%M')).month):  docket.append((df.loc[[i],[index]].values[0][0]-value)/value)  value=df.loc[[i],[index]].values[0][0]  temp=(datetime.datetime.strptime(df.loc[[i],[time]].values[0][0],'%Y/%m/%d %H:%M')).month  docket.append((df.loc[[0],[index]].values[0][0]-value)/value) #加上最后一天的月收益率    process\_csv(path2,docket,index)  #表格规整  def messup(df,path):  for i in range(10):  if 'Unnamed: 0' in df.columns.values:  df.drop(columns='Unnamed: 0',axis=1,inplace=True)  print('已删除多余的Unnamed: 0列')  elif 'Unnamed: 0.1' in df.columns.values:  df.drop(columns='Unnamed: 0.1',axis=1,inplace=True)  print('已删除多余的Unnamed: 0.1列')  elif 'Unnamed: 0.1.1' in df.columns.values:  df.drop(columns='Unnamed: 0.1.1',axis=1,inplace=True)  print('已删除多余的Unnamed: 0.1.1列')  elif 'Unnamed: 0.1.1.1' in df.columns.values:  df.drop(columns='Unnamed: 0.1.1.1',axis=1,inplace=True)  print('已删除多余的Unnamed: 0.1.1.1列')  elif 'Unnamed: 0.1.1.1.1' in df.columns.values:  df.drop(columns='Unnamed: 0.1.1.1.1',axis=1,inplace=True)  print('已删除多余的Unnamed: 0.1.1.1.1列')  if (os.path.exists(path)):  os.remove(path)  if (os.path.exists(path2)):  os.remove(path2)  print("原文件删除成功")  else:  print("原文件不存在，创建新的原文件")  #有时时间格式会发生报错，要么换成%Y-%m-%d %H:%M:%S要么换成%Y/%m/%d %H:%M:%S要么根据报错提示换成别的形式  p=pd.read\_csv(path\_return)  for x in p.columns[1:]:  process\_month\_return(x,time,path\_return,path2)  print("计算完成")  #股指数据的选择并保存在本地 股指收益率文件index.csv  #股指选择沪深300或者中证指数 基金收益率文件fund.csv  #沪深300 000300.XSHG 中证700 000907.XSHG  # https://www.joinquant.com/data/dict/indexData  stock='000300.XSHG'  info=get\_price(stock, start\_date='2009-4-13', end\_date='2016-12-31', frequency='daily', fields='open', skip\_paused=True, fq='pre')  info.to\_csv('F:/test/index.csv')  #东方红基金月收益率计算  #股指月收益率计算  # -\*- coding: utf-8 -\*-  #df=pd.read\_csv(path,encoding="unicode\_escape")  df\_index=pd.read\_csv("F:/test/index.csv",encoding="gbk")  df\_fund=pd.read\_csv("F:/test/fund.csv",encoding="gbk")  index='fuquan'#处理的数据字段  time='riqi'#对应的时间字段  date=len(df\_fund.loc[:,[time]])-1  #处理股指的月收益率  docketfund=[]  value=df\_fund.loc[[date],[index]].values[0][0]  temp=(datetime.datetime.strptime(df\_fund.loc[[date],[time]].values[0][0],'%Y/%m/%d')).month  for i in range(date,0,-1):    if (temp!=(datetime.datetime.strptime(df\_fund.loc[[i],[time]].values[0][0],'%Y/%m/%d')).month):  docketfund.append((df\_fund.loc[[i],[index]].values[0][0]-value)/value)  value=df\_fund.loc[[i],[index]].values[0][0]    temp=(datetime.datetime.strptime(df\_fund.loc[[i],[time]].values[0][0],'%Y/%m/%d')).month  docketfund.append((df\_fund.loc[[0],[index]].values[0][0]-value)/value) #加上最后一天的月收益率  process\_csv(path2,docketfund,index)  index='MKT'#处理的数据字段  time='time'#对应的时间字段  #从平台上直接读取的数据有问题，需要索引规整  df\_index=df\_index.rename(columns={'Unnamed: 0':'time','open':'MKT'})  date=len(df\_index.loc[:,[time]])-1  #处理股指的月收益率  docketindex=[]  value=df\_index.loc[[date],[index]].values[0][0]  temp=(datetime.datetime.strptime(df\_index.loc[[0],[time]].values[0][0],'%Y-%m-%d')).month  for i in range(0,date,1):    if (temp!=(datetime.datetime.strptime(df\_index.loc[[i],[time]].values[0][0],'%Y-%m-%d')).month):  docketindex.append((df\_index.loc[[i],[index]].values[0][0]-value)/value)  value=df\_index.loc[[i],[index]].values[0][0]    temp=(datetime.datetime.strptime(df\_index.loc[[i],[time]].values[0][0],'%Y-%m-%d')).month  docketindex.append((df\_index.loc[[0],[index]].values[0][0]-value)/value) #加上最后一天的月收益率  process\_csv(path2,docketindex,index)  #设定年化率4%，在此情况下选用月收益率作为无风险利率  r=pow(1.04,12)-1  #生成premium和时间索引两列  df=pd.read\_csv(path2)  df['t']=list(range(1,len(df)+1))#生成时间  df['premium']=df.apply(lambda x: x['fuquan'] - 0.01\*r, axis=1)#生成超额收益  messup(df,path2)  df.to\_csv(path2,index=False,header=True)#重新存取到csv文件  #一元线性回归部分，确定指数  #综合比较沪深300、中证500、中证800、上证50，显然沪深300的相关性最好  time='time'#对应的时间字段  path\_month='F:/test/month.csv'  x=pd.read\_csv(path\_month)  model=sm.OLS(list(x.fuquan.values),list(x.MKT.values))  reg=model.fit()  print(reg.summary())  #多元线性回归.ipynb  import pandas as pd  import csv  import datetime  import numpy as np  from statsmodels.api import OLS  import matplotlib.pyplot as plt  import os  import re  import statsmodels.api as sm  from scipy import stats  #此回归是不带巴菲特特有beta的回归，运行此回归的目的是将alpha变得不显著  path='F:/test/month.csv'  path\_store="F:/test/multi-regression"  path\_regression='F:/test/reg.csv'  df=pd.read\_csv(path)  df.columns.values  #independent=['size', 'book\_to\_price\_ratio', 'beta','MKT','money\_flow\_20','liquidity','residual\_volatility','BIAS60','earnings\_yield','financial\_expense\_rate','roa\_ttm']#将要进行回归的自变量加入到列表中去  independent=['size','beta','roa\_ttm','MKT','EP','momentum']  dependent=['premium']  x=df[independent]  y=df[dependent]  #model=sm.OLS(pd.DataFrame(df.loc[:,['open']].values.T).values[0],pd.DataFrame(df.loc[:,['t']].values.T).values[0])  x=sm.add\_constant(x)  reg=sm.OLS(y,x).fit()  print(reg.summary())  #时间序列平稳性检验.ipynb  # -\*- coding:utf-8 -\*-  from statsmodels.tsa.stattools import adfuller  import statsmodels.tsa.stattools as ta  import statsmodels.api as sm  import pandas as pd  import matplotlib.pyplot as plt  import numpy as np  from statsmodels.graphics.tsaplots import plot\_acf, plot\_pacf  import os  path='C:/Users/dsquant06/Desktop/test/month.csv'  # 移动平均图  def draw\_trend(timeSeries, size):  f = plt.figure(facecolor='white')  # 对size个数据进行移动平均  rol\_mean = timeSeries.rolling(window=size).mean()  # 对size个数据进行加权移动平均  #rol\_weighted\_mean = pd.ewma(timeSeries, span=size)  rol\_weighted\_mean = timeSeries.ewm(size)  timeSeries.plot(color='blue', label='Original')  rolmean.plot(color='red', label='Rolling Mean')  rol\_weighted\_mean.plot(color='black', label='Weighted Rolling Mean')  plt.legend(loc='best')  plt.title('Rolling Mean')  plt.show()  def draw\_ts(timeSeries):  f = plt.figure(facecolor='white')  timeSeries.plot(color='blue')  plt.xlabel('samples')  plt.ylabel('return')  plt.savefig("C:/Users/dsquant06/Desktop/test/股指回归/趋势图.pdf")  plt.show()  '''  　　Unit Root Test  The null hypothesis of the Augmented Dickey-Fuller is that there is a unit  root, with the alternative that there is no unit root. That is to say the  bigger the p-value the more reason we assert that there is a unit root  '''  def testStationarity(ts):  dftest = adfuller(ts)  # 对上述函数求得的值进行语义描述  dfoutput = pd.Series(dftest[0:4], index=['Test Statistic','p-value','#Lags Used','Number of Observations Used'])  for key,value in dftest[4].items():  dfoutput['Critical Value (%s)'%key] = value  return dfoutput  # 自相关和偏相关图，默认阶数为31阶  def draw\_acf(ts, lags=92):  f = plt.figure(facecolor='white')  #ax1 = f.add\_subplot(211)  plot\_acf(ts, lags=92)  plt.xlabel('lags')  plt.ylabel('correlation coefficient')  plt.subplots\_adjust(left=None, bottom=None, right=None, top=None,wspace=0.1, hspace=1)  plt.savefig("C:/Users/dsquant06/Desktop/test/股指回归/自相关图.pdf")  plt.show()      def draw\_pacf(ts,lags=46):  f = plt.figure(facecolor='white')  #ax2 = f.add\_subplot(212)  plot\_pacf(ts, lags=46)  plt.xlabel('lags')  plt.ylabel('correlation coefficient')  plt.subplots\_adjust(left=None, bottom=None, right=None, top=None,wspace=0.1, hspace=1)  plt.savefig("C:/Users/dsquant06/Desktop/test/股指回归/偏相关图.pdf")  plt.show()    df=pd.read\_csv(path)  df.columns.values[1:]  index='jingzhi'  #将dataframe的数据类型转换为array的行向量  df=df.loc[:,[index]]  ts=np.array(pd.DataFrame(df.values.T))[0]  draw\_ts(df)  draw\_acf(ts,92)  draw\_pacf(ts,46)  #beta系数回归.ipynb  # -\*- coding:utf-8 -\*-  from statsmodels.tsa.stattools import adfuller  import statsmodels.tsa.stattools as ta  import statsmodels.api as sm  import pandas as pd  import matplotlib.pyplot as plt  import numpy as np  from statsmodels.graphics.tsaplots import plot\_acf, plot\_pacf  import os  path='C:/Users/dsquant06/Desktop/test/month.csv'  dependent=['premium']  independent=['MKT']  #第二步需要用r-rf-beta\*MKT求得回归方程的左侧  independent0=['roic\_ttm', 'size', 'book\_to\_price\_ratio', 'beta','MKT']#这一项是接下来回归的右侧的几个因子  #表格规整  def messup(df,path):  for i in range(10):  if 'Unnamed: 0' in df.columns.values:  df.drop(columns='Unnamed: 0',axis=1,inplace=True)  print('已删除多余的Unnamed: 0列')  elif 'Unnamed: 0.1' in df.columns.values:  df.drop(columns='Unnamed: 0.1',axis=1,inplace=True)  print('已删除多余的Unnamed: 0.1列')  elif 'Unnamed: 0.1.1' in df.columns.values:  df.drop(columns='Unnamed: 0.1.1',axis=1,inplace=True)  print('已删除多余的Unnamed: 0.1.1列')  elif 'Unnamed: 0.1.1.1' in df.columns.values:  df.drop(columns='Unnamed: 0.1.1.1',axis=1,inplace=True)  print('已删除多余的Unnamed: 0.1.1.1列')  elif 'Unnamed: 0.1.1.1.1' in df.columns.values:  df.drop(columns='Unnamed: 0.1.1.1.1',axis=1,inplace=True)  print('已删除多余的Unnamed: 0.1.1.1.1列')  if (os.path.exists(path)):  os.remove(path)  df=pd.read\_csv(path)  x=df[independent]  y=df[dependent]  #model=sm.OLS(pd.DataFrame(df.loc[:,['open']].values.T).values[0],pd.DataFrame(df.loc[:,['t']].values.T).values[0])  #首先将premium作为因变量，将MKT作为自变量，做回归所得到自变量前的系数为所求的beta\_buffet  x=sm.add\_constant(x)  reg=sm.OLS(y,x).fit()  print(reg.summary())  x1=df[independent0]  y1=df.apply(lambda x: x['premium']-(reg.params[independent])\*(x['MKT']), axis=1)#这一项是r-rf-beta\*MKT左侧的值  x1=sm.add\_constant(x1)  reg1=sm.OLS(y1,x1).fit()  print(reg1.summary())  #接下来需要求得rA  #如果原来的表中有以前所得到的回归，那么删除这一列  if 'ra' in df.columns.values:  print('已删除原有的rA值')  df.drop(columns='ra',axis=1,inplace=True)  #根据各个因子的系数计算rA  for i in independent0:  df['ra']=df.apply(lambda x: (reg1.params[i])\*(x[i]), axis=1)#生成ractive  #规整表格，输出rA的值  messup(df,path)  df.to\_csv(path)  #设定年化率4%，在此情况下选用月收益率作为无风险利率  r=pow(1.04,12)-1  #接下来需要求得rA  #如果原来的表中有以前所得到的回归，那么删除这一列  if 'style' in df.columns.values:  print('已删除原有的style值')  df.drop(columns='style',axis=1,inplace=True)  #最后计算r\_Buffetstyle  df['style']=df.apply(lambda x: r + (x['MKT'])\*(reg.params[independent])+x['ra'], axis=1)#生成ractive  messup(df,path)  df.to\_csv(path) | | | | | |