```python
#!/usr/bin/env python
# -*- coding: utf-8 -*-
# Author: 李珂宇
import pandas as pd
import numpy as np
import sys
import time
import os
import psutil
import matplotlib.pyplot as plt
import seaborn as sns


def memory_info():
    # 模获得当前进程的pid
    pid = os.getpid()
    # 根据pid找到进程，进而找到占用的内存值
    p = psutil.Process(pid)
    info = p.memory_full_info()
    memory = info.uss / 1024
    return memory


def read_clean_db(db_name):
    path_root = r"./Database/"
    time_0 = time.time()
    # 似乎np.nan和'Nan'效果一样 skiprows是神器不解释
    db = pd.read_csv(path_root + db_name, encoding="gbk", na_values=np.nan, skiprows=[1, 2])
    # 注意 由于导入的数据中混杂中英文导致类型混淆,对后文计算有影响
    # 解决办法是读取后删除注释行并修改index,并用low_memory=True和.astype(float)
    # self.database.drop(index=[0, 1], inplace=True)
    # self.database.reset_index(drop=True, inplace=True)
    # 更好的办法是在读取函数中跳过那两行skiprows
    list_1 = []
    # 股票代码填充到六位
    for i in range(db['Stkcd'].size):
        i_str = str(db.loc[i, 'Stkcd'])
        length = len(i_str)
        if length != 6:
            i_new = '0' * (6 - length) + i_str
            list_1.append(i_new)
            # 直接单个赋值效率极低，放弃，尝试整列修改效率极大提升
            # dirty_db.loc[i,'Stkcd'] = i_new
        else:
            list_1.append(i_str)
    db['Stkcd'] = list_1
    # 日期格式化 如果有的话
    # TODO:一个更通用化的逻辑
    if db_name == 'TRD_Mnth.CSV':
        db['Trdmnt'] = pd.to_datetime(db['Trdmnt'])
    else:
        db['Accper'] = pd.to_datetime(db['Accper'])
    time_1 = time.time()
    print('读取并清洗' + db_name + '用时' + str(round(time_1 - time_0, 4)) + 's')
    # print(db)
    return db


# 0 清理数据
# TODO:以后用装饰器来修改运行时间和占用内存部分
# TODO:清理数据应当放到外面 不然每次实例都要重新清理
memory_0 = memory_info()
db_pm = read_clean_db('PriceMultiples.CSV')
db_bs = read_clean_db('FS_Combas.CSV')
db_is = read_clean_db('FS_Comins.CSV')
db_cfd = read_clean_db('FS_Comscfd.CSV')
db_cfi = read_clean_db('FS_Comscfi.CSV')
db_p_month = read_clean_db('TRD_Mnth.CSV')
memory_1 = memory_info()
print('全部读取并清洗完成，占用内存' + str(memory_1 - memory_0) + ' KB')

```

```python

class RelativeV:
    def __init__(self, code_id, db_pm=db_pm, db_bs=db_bs, db_is=db_is, db_cfd=db_cfd,
     db_cfi=db_cfi,db_p_month=db_p_month):
        # 相对价值指标数据库self.database
        self.database = db_pm
        # 资产负债表数据库
        self.db_bs = db_bs
        # 利润表数据库
        self.db_is = db_is
        # 现金流量表数据库(直接法)
        self.db_cfd = db_cfd
        # 现金流量表数据库(间接法)
        self.db_cfi = db_cfi
        # 月都股票数据
        self.db_p_month = db_p_month
        self.code_id = code_id
        self.industry_id = ''
        self.target_df = pd.DataFrame
        self.code_list = []
        self.calender = []
        self.df_industry = pd.DataFrame
        self.df_average = pd.DataFrame
        self.df_fs_result = pd.DataFrame
        self.df_relative_result = pd.DataFrame
        self.df_fs_result_price = pd.DataFrame
        self.df_relative_result_price = pd.DataFrame
        self.price_cache = []
        self.df_summary = pd.DataFrame

    # 1 提取目标公司信息
    def basic_db(self):
        df = self.database.loc[self.database['Stkcd'] == self.code_id]
        df = df.sort_values(by='Accper', ascending=False)
        df.reset_index(drop=True, inplace=True)
        self.calender = df['Accper'].tolist()
        if len(self.calender) == 0:
            print('未查询到相关数据')
            sys.exit(1)
        df = df.set_index(['Accper'])
        industry_id = set(df['Indcd'])
        if len(industry_id) == 1:
            self.industry_id = list(industry_id)[0]
        else:
            print('公司行业发生变化，需要检查数据库是否出错')
            sys.exit(1)
        self.target_df = df
        pd.DataFrame.to_csv(df, "./Result/target.csv", encoding='gbk')

    # 2 选出所有同行业公司
    def industry_match(self):
        # 根据行业指标查询所有同行业公司 df_0
        df_0 = self.database.loc[self.database['Indcd'] == self.industry_id]
        # 剔除目标公司 df_1
        df_1 = df_0.loc[self.database['Stkcd'] != self.code_id]
        industry_id = set(df_1['Indcd'])
        if len(industry_id) != 1:
            print('行业数据不唯一，需要检查数据库是否出错')
            sys.exit(1)
        self.code_list = list(set(df_1['Stkcd']))
        self.df_industry = df_1
        pd.DataFrame.to_csv(df_1, "./Result/industry_exc_target.csv", encoding='gbk')

    # 3 计算行业平均价格倍数
    def average_db(self):
        # 以日期和指标为两坐标轴
        df_list_date = [self.df_industry.loc[self.df_industry['Accper'] == i] for i
        in self.calender]
        df_index = self.calender
        df_column = self.target_df.columns.values.tolist()
        df_column = df_column[df_column.index('F100101B'):]
        # 计算平均值时剔除NaN np里用nanmean(),df里不知道,也没有自动屏蔽nan的mean
```

```
142             # TODO:没有剔除0
143             df_list_average = [[np.nanmean(i[j].to_numpy()) for j in df_column] for i in
                df_list_date]
144             # type(i[j].values) = <class 'numpy.ndarray'>, to_numpy()是更先进的方法
145             df_average = pd.DataFrame(df_list_average, index=df_index, columns=df_column)
146             self.df_average = df_average
147             pd.DataFrame.to_csv(df_average, "./Result/industry_average.csv", encoding=
                'gbk')
148
149         # 4 提取相应财务数据(数据不全,做不了完整版)
150         def fs_inquiry(self):
151             # 选出目标公司
152             df_cache_bs = self.db_bs.loc[self.db_bs['Stkcd'] == self.code_id].loc[self.
                db_bs['Typrep'] == 'A']
153             df_cache_is = self.db_is.loc[self.db_is['Stkcd'] == self.code_id].loc[self.
                db_is['Typrep'] == 'A']
154             df_cache_cfd = self.db_cfd.loc[self.db_cfd['Stkcd'] == self.code_id].loc[self
                .db_cfd['Typrep'] == 'A']
155             df_cache_cfi = self.db_cfi.loc[self.db_cfi['Stkcd'] == self.code_id].loc[self
                .db_cfi['Typrep'] == 'A']
156
157             # 具体数据
158             # 实收资本A003101000
159             paid_in_capital_cache = [df_cache_bs.loc[df_cache_bs['Accper'] == i][
                'A003101000'].to_numpy() for i in self.calender]
160             paid_in_capital = [i[0] if len(i) != 0 else np.nan for i in
                paid_in_capital_cache]
161             # B002000000 [净利润]
162             ni_cache = [df_cache_is.loc[df_cache_is['Accper'] == i]['B002000000'].
                to_numpy() for i in self.calender]
163             ni = [i[0] if len(i) != 0 else np.nan for i in ni_cache]
164             # 调整因子 12/(间隔月份)(1 1.33 2 4)
165             adj = [12.0 / i.month for i in self.calender]
166             # B001100000 [营业总收入]
167             revenue_cache = [df_cache_is.loc[df_cache_is['Accper'] == i]['B001100000'].
                to_numpy() for i in self.calender]
168             revenue = [i[0] if len(i) != 0 else np.nan for i in revenue_cache]
169             # A003000000 [所有者权益合计]
170             equity_cache = [df_cache_bs.loc[df_cache_bs['Accper'] == i]['A003000000'].
                to_numpy() for i in self.calender]
171             equity = [i[0] if len(i) != 0 else np.nan for i in equity_cache]
172             # B002000101 [归属于母公司所有者的净利润]
173             ni_parent_cache = [df_cache_is.loc[df_cache_is['Accper'] == i]['B002000101'].
                to_numpy() for i in self.calender]
174             ni_parent = [i[0] if len(i) != 0 else np.nan for i in ni_parent_cache]
175             # A003100000 [归属于母公司所有者权益合计]
176             equity_parent_cache = [df_cache_bs.loc[df_cache_bs['Accper'] == i][
                'A003100000'].to_numpy() for i in self.calender]
177             equity_parent = [i[0] if len(i) != 0 else np.nan for i in equity_parent_cache]
178             # A001000000 [资产总计]
179             asset_cache = [df_cache_bs.loc[df_cache_bs['Accper'] == i]['A001000000'].
                to_numpy() for i in self.calender]
180             asset = [i[0] if len(i) != 0 else np.nan for i in asset_cache]
181             # A001218000 [无形资产净额]
182             intangible_cache = [df_cache_bs.loc[df_cache_bs['Accper'] == i]['A001218000'
                ].to_numpy() for i in self.calender]
183             intangible = [i[0] if len(i) != 0 else np.nan for i in intangible_cache]
184             # A001220000 [商誉净额]
185             goodwill_cache = [df_cache_bs.loc[df_cache_bs['Accper'] == i]['A001220000'].
                to_numpy() for i in self.calender]
186             goodwill = [i[0] if len(i) != 0 else np.nan for i in goodwill_cache]
187             # 有形资产
188             tangible = np.array(asset) - np.array(intangible) - np.array(goodwill)
189             # C001000000 [经营活动产生的现金流量净额] 直接法
190             cfo_d_cache = [df_cache_cfd.loc[df_cache_cfd['Accper'] == i]['C001000000'].
                to_numpy() for i in self.calender]
191             cfo_d = [i[0] if len(i) != 0 else np.nan for i in cfo_d_cache]
192             # D000100000 [经营活动产生的现金流量净额] 间接法
193             cfo_i_cache = [df_cache_cfi.loc[df_cache_cfi['Accper'] == i]['D000100000'].
                to_numpy() for i in self.calender]
194             cfo_i = [i[0] if len(i) != 0 else np.nan for i in cfo_i_cache]
195
```

```
196             # 分析结果
197             # 上一年用 date.replace(year=date.year-1)
198             # F100101B [市盈率1] - 今收盘价当期值/（净利润上年年报值/实收资本本期期末值）
199             eps_1 = [ni[self.calender.index(date.replace(year=date.year-1))]/
                paid_in_capital[self.calender.index(date)]
200                      if date.replace(year=date.year-1) in self.calender else np.nan
                         for date in self.calender]
201             # F100102B [市盈率2] -
                今收盘价当期值/（调整因子*净利润当期值/实收资本本期期末值）
202             eps_2 = [adj[self.calender.index(date)]*ni[self.calender.index(date)]/
                paid_in_capital[self.calender.index(date)]
203                      for date in self.calender]
204             # F100103C [市盈率TTM] - 今收盘价当期值/（净利润TTM/实收资本本期期末值）
205             # F100201B [市销率1] -
                今收盘价当期值/（营业总收入上年年报值/实收资本本期期末值）
206             sps_1 = [revenue[self.calender.index(date.replace(year=date.year-1))]/
                paid_in_capital[self.calender.index(date)]
207                      if date.replace(year=date.year-1) in self.calender else np.nan
                         for date in self.calender]
208             # F100202B [市销率2] -
                今收盘价当期值/（调整因子*营业总收入当期值/实收资本本期期末值）
209             sps_2 = [adj[self.calender.index(date)]*revenue[self.calender.index(date)]/
                paid_in_capital[self.calender.index(date)]
210                      for date in self.calender]
211             # F100203C [市销率TTM] - 今收盘价当期值/（营业总收入TTM/实收资本本期期末值）
212             # F100301B [市现率1] -
                今收盘价当期值/（经营活动产生的现金流量净额上年年报值/实收资本本期期末值）
213             cfops_d_1 = [cfo_d[self.calender.index(date.replace(year=date.year-1))]/
                paid_in_capital[self.calender.index(date)]
214                      if date.replace(year=date.year-1) in self.calender else np.nan
                         for date in self.calender]
215             cfops_i_1 = [cfo_i[self.calender.index(date.replace(year=date.year-1))]/
                paid_in_capital[self.calender.index(date)]
216                      if date.replace(year=date.year-1) in self.calender else np.nan
                         for date in self.calender]
217             # F100302B [市现率2] -
                今收盘价当期值/（调整因子*经营活动产生的现金流量净额当期值/实收资本本期期末值
                ）
218             cfops_d_2 = [adj[self.calender.index(date)]*cfo_d[self.calender.index(date)]/
                paid_in_capital[self.calender.index(date)]
219                      for date in self.calender]
220             cfops_i_2 = [adj[self.calender.index(date)]*cfo_i[self.calender.index(date)]/
                paid_in_capital[self.calender.index(date)]
221                      for date in self.calender]
222             # F100303C [市现率TTM] -
                今收盘价当期值/（经营活动产生的现金流量净额TTM/实收资本本期期末值）
223             # F100401A [市净率] -
                今收盘价当期值/（所有者权益合计期末值/实收资本本期期末值）
224             bps = [equity[self.calender.index(date)]/paid_in_capital[self.calender.index(
                date)] for date in self.calender]
225             # F100501A [市值有形资产比] -
                今收盘价当期值/[（资产总计-无形资产净额-商誉净额）期末值/实收资本本期期末值]
226             tps = [tangible[self.calender.index(date)]/paid_in_capital[self.calender.
                index(date)] for date in self.calender]
227             # F100601B [市盈率母公司1] -
                今收盘价当期值/（归属于母公司所有者的净利润上年年报值/实收资本本期期末值）
228             eps_parent_1 = [ni_parent[self.calender.index(date.replace(year=date.year-1
                ))]/paid_in_capital[self.calender.index(date)]
229                      if date.replace(year=date.year-1) in self.calender else np.nan
                         for date in self.calender]
230             # F100602B [市盈率母公司2] -
                今收盘价当期值/（调整因子*归属于母公司所有者的净利润当期值/实收资本本期期末值
                ）
231             eps_parent_2 = [adj[self.calender.index(date)]*ni_parent[self.calender.index(
                date)]/paid_in_capital[self.calender.index(date)]
232                      for date in self.calender]
233             # F100603C [市盈率母公司TTM] -
                今收盘价当期值/[（归属于母公司所有者的净利润）TTM/实收资本本期期末值]
234             # F100701A [市净率母公司] -
                今收盘价当期值/（归属于母公司所有者权益合计期末值/实收资本本期期末值）
235             bps_parent = [equity_parent[self.calender.index(date)]/paid_in_capital[self.
                calender.index(date)] for date in self.calender]
```

```
236
237             # 财务数据提取结果表
238             dic_fs_result = {
239                 'eps_1': eps_1,
240                 'eps_2': eps_2,
241                 'sps_1': sps_1,
242                 'sps_2': sps_2,
243                 'cfops_d_1': cfops_d_1,
244                 'cfops_i_1': cfops_i_1,
245                 'cfops_d_2': cfops_d_2,
246                 'cfops_i_2': cfops_i_2,
247                 'bps': bps,
248                 'tps': tps,
249                 'eps_parent_1': eps_parent_1,
250                 'eps_parent_2': eps_parent_2,
251                 'bps_parent': bps_parent
252             }
253             df_fs_result = pd.DataFrame(dic_fs_result, index=self.calender)
254             self.df_fs_result = df_fs_result
255             pd.DataFrame.to_csv(df_fs_result, "./Result/fs_result.csv", encoding='gbk')
256
257         # 4_a 直接用当时股价和财务指标来反求财务数据,再用这个数据求价格倍数平均估计值
258         def fs_result_price(self):
259             # 选出目标公司
260             df_cache_p_month = self.db_p_month.loc[self.db_p_month['Stkcd'] == self.
                code_id]
261             # 选出对应月份(TODO:暂定只有季度月,以后改为用self.calender中出现的所有月)
262             df_p_month = df_cache_p_month.loc[[True if date.month in [3, 6, 9, 12] else
                False for date in df_cache_p_month['Trdmnt']]]\
263                 .sort_values(by='Trdmnt', ascending=False)
264             df_p_month.reset_index(drop=True, inplace=True)
265             # 修改月底以匹配数据
266             date_clean = [date.replace(day=31) if date.month in [1, 3, 5, 7, 8, 10, 12]
267                           else date.replace(day=30) for date in df_p_month['Trdmnt']]
268             df_p_month['Trdmnt'] = pd.Series(date_clean)
269             df_p_month = df_p_month.set_index(['Trdmnt'])
270             # 到这里就清洗完成了
271             # Mclsprc [月收盘价] 匹配数据 若价格数据不足则用nan填充
272             price_cache_0 = pd.DataFrame(df_p_month['Mclsprc'])
273             # concat变得不好用了 改用join 处理index更方便
274             # df_target_price = pd.concat([self.target_df, price_cache_0],
                axis=1).sort_index(ascending=False)
275             df_target_price = self.target_df.join(other=price_cache_0,how='left').
                sort_index(ascending=False)
276             price_cache = df_target_price['Mclsprc'].to_numpy()
277             self.price_cache = price_cache
278             # print(df_target_price)
279
280             # 反推结果
281             eps_1 = [price_cache[self.calender.index(date)]/self.target_df.loc[date,
                'F100101B'] for date in self.calender]
282             eps_2 = [price_cache[self.calender.index(date)]/self.target_df.loc[date,
                'F100102B'] for date in self.calender]
283             sps_1 = [price_cache[self.calender.index(date)]/self.target_df.loc[date,
                'F100201B'] for date in self.calender]
284             sps_2 = [price_cache[self.calender.index(date)]/self.target_df.loc[date,
                'F100202B'] for date in self.calender]
285             cfops_d_1 = [price_cache[self.calender.index(date)]/self.target_df.loc[date,
                'F100301B'] for date in self.calender]
286             cfops_i_1 = [price_cache[self.calender.index(date)]/self.target_df.loc[date,
                'F100301B'] for date in self.calender]
287             cfops_d_2 = [price_cache[self.calender.index(date)]/self.target_df.loc[date,
                'F100302B'] for date in self.calender]
288             cfops_i_2 = [price_cache[self.calender.index(date)]/self.target_df.loc[date,
                'F100302B'] for date in self.calender]
289             bps = [price_cache[self.calender.index(date)]/self.target_df.loc[date,
                'F100401A'] for date in self.calender]
290             tps = [price_cache[self.calender.index(date)]/self.target_df.loc[date,
                'F100501A'] for date in self.calender]
291             eps_parent_1 = [price_cache[self.calender.index(date)]/self.target_df.loc[
                date, 'F100601B'] for date in self.calender]
292             eps_parent_2 = [price_cache[self.calender.index(date)]/self.target_df.loc[
```

```
            date, 'F100602B'] for date in self.calender]
293         bps_parent = [price_cache[self.calender.index(date)]/self.target_df.loc[date,
             'F100701A'] for date in self.calender]
294
295         # 财务数据提取结果表
296         dic_fs_result_price = {
297             'eps_1': eps_1,
298             'eps_2': eps_2,
299             'sps_1': sps_1,
300             'sps_2': sps_2,
301             'cfops_d_1': cfops_d_1,
302             'cfops_i_1': cfops_i_1,
303             'cfops_d_2': cfops_d_2,
304             'cfops_i_2': cfops_i_2,
305             'bps': bps,
306             'tps': tps,
307             'eps_parent_1': eps_parent_1,
308             'eps_parent_2': eps_parent_2,
309             'bps_parent': bps_parent
310         }
311         df_fs_result_price = pd.DataFrame(dic_fs_result_price, index=self.calender)
312         self.df_fs_result_price = df_fs_result_price
313         pd.DataFrame.to_csv(df_fs_result_price, "./Result/fs_result_price.csv",
            encoding='gbk')
314
315     # 5 根据目标公司具体财务数据计算相对估值结果
316     def relative_result(self):
317         p_pe_1 = [self.df_average.loc[date, 'F100101B']*self.df_fs_result.loc[date,
            'eps_1'] for date in self.calender]
318         p_pe_2 = [self.df_average.loc[date, 'F100102B']*self.df_fs_result.loc[date,
            'eps_2'] for date in self.calender]
319         p_ps_1 = [self.df_average.loc[date, 'F100201B']*self.df_fs_result.loc[date,
            'sps_1'] for date in self.calender]
320         p_ps_2 = [self.df_average.loc[date, 'F100202B']*self.df_fs_result.loc[date,
            'sps_2'] for date in self.calender]
321         p_pcf_d_1 = [self.df_average.loc[date, 'F100301B']*self.df_fs_result.loc[date
            , 'cfops_d_1'] for date in self.calender]
322         p_pcf_i_1 = [self.df_average.loc[date, 'F100301B']*self.df_fs_result.loc[date
            , 'cfops_i_1'] for date in self.calender]
323         p_pcf_d_2 = [self.df_average.loc[date, 'F100302B']*self.df_fs_result.loc[date
            , 'cfops_d_2'] for date in self.calender]
324         p_pcf_i_2 = [self.df_average.loc[date, 'F100302B']*self.df_fs_result.loc[date
            , 'cfops_i_2'] for date in self.calender]
325         p_pb = [self.df_average.loc[date, 'F100401A']*self.df_fs_result.loc[date,
            'bps'] for date in self.calender]
326         p_evt = [self.df_average.loc[date, 'F100501A']*self.df_fs_result.loc[date,
            'tps'] for date in self.calender]
327         p_pe_parent_1 = [self.df_average.loc[date, 'F100601B']*self.df_fs_result.loc[
            date, 'eps_parent_1'] for date in self.calender]
328         p_pe_parent_2 = [self.df_average.loc[date, 'F100602B']*self.df_fs_result.loc[
            date, 'eps_parent_2'] for date in self.calender]
329         p_pb_parent = [self.df_average.loc[date, 'F100701A']*self.df_fs_result.loc[
            date, 'bps_parent'] for date in self.calender]
330         dic_relative_result = {
331             'p_pe_1': p_pe_1,
332             'p_pe_2': p_pe_2,
333             'p_ps_1': p_ps_1,
334             'p_ps_2': p_ps_2,
335             'p_pcf_d_1': p_pcf_d_1,
336             'p_pcf_i_1': p_pcf_i_1,
337             'p_pcf_d_2': p_pcf_d_2,
338             'p_pcf_i_2': p_pcf_i_2,
339             'p_pb': p_pb,
340             'p_evt': p_evt,
341             'p_pe_parent_1': p_pe_parent_1,
342             'p_pe_parent_2': p_pe_parent_2,
343             'p_pb_parent': p_pb_parent
344         }
345         df_relative_result = pd.DataFrame(dic_relative_result, index=self.calender)
346         self.df_relative_result = df_relative_result
347         pd.DataFrame.to_csv(self.df_relative_result, "./Result/relative_result.csv",
            encoding='gbk')
```

```
348
349         # 5_a 根据目标公司具体财务数据计算相对估值结果
350         def relative_result_price(self):
351             p_pe_1 = [self.df_average.loc[date, 'F100101B']*self.df_fs_result_price.loc[
                date, 'eps_1'] for date in self.calender]
352             p_pe_2 = [self.df_average.loc[date, 'F100102B']*self.df_fs_result_price.loc[
                date, 'eps_2'] for date in self.calender]
353             p_ps_1 = [self.df_average.loc[date, 'F100201B']*self.df_fs_result_price.loc[
                date, 'sps_1'] for date in self.calender]
354             p_ps_2 = [self.df_average.loc[date, 'F100202B']*self.df_fs_result_price.loc[
                date, 'sps_2'] for date in self.calender]
355             p_pcf_d_1 = [self.df_average.loc[date, 'F100301B']*self.df_fs_result_price.
                loc[date, 'cfops_d_1'] for date in self.calender]
356             p_pcf_i_1 = [self.df_average.loc[date, 'F100301B']*self.df_fs_result_price.
                loc[date, 'cfops_i_1'] for date in self.calender]
357             p_pcf_d_2 = [self.df_average.loc[date, 'F100302B']*self.df_fs_result_price.
                loc[date, 'cfops_d_2'] for date in self.calender]
358             p_pcf_i_2 = [self.df_average.loc[date, 'F100302B']*self.df_fs_result_price.
                loc[date, 'cfops_i_2'] for date in self.calender]
359             p_pb = [self.df_average.loc[date, 'F100401A']*self.df_fs_result_price.loc[
                date, 'bps'] for date in self.calender]
360             p_evt = [self.df_average.loc[date, 'F100501A']*self.df_fs_result_price.loc[
                date, 'tps'] for date in self.calender]
361             p_pe_parent_1 = [self.df_average.loc[date, 'F100601B']*self.
                df_fs_result_price.loc[date, 'eps_parent_1'] for date in self.calender]
362             p_pe_parent_2 = [self.df_average.loc[date, 'F100602B']*self.
                df_fs_result_price.loc[date, 'eps_parent_2'] for date in self.calender]
363             p_pb_parent = [self.df_average.loc[date, 'F100701A']*self.df_fs_result_price.
                loc[date, 'bps_parent'] for date in self.calender]
364             dic_relative_result_price = {
365                 'p_pe_1': p_pe_1,
366                 'p_pe_2': p_pe_2,
367                 'p_ps_1': p_ps_1,
368                 'p_ps_2': p_ps_2,
369                 'p_pcf_d_1': p_pcf_d_1,
370                 'p_pcf_i_1': p_pcf_i_1,
371                 'p_pcf_d_2': p_pcf_d_2,
372                 'p_pcf_i_2': p_pcf_i_2,
373                 'p_pb': p_pb,
374                 'p_evt': p_evt,
375                 'p_pe_parent_1': p_pe_parent_1,
376                 'p_pe_parent_2': p_pe_parent_2,
377                 'p_pb_parent': p_pb_parent
378             }
379             df_relative_result_price = pd.DataFrame(dic_relative_result_price, index=self
                .calender)
380             self.df_relative_result_price = df_relative_result_price
381             pd.DataFrame.to_csv(self.df_relative_result_price,
                "./Result/relative_result_price.csv", encoding='gbk')
382
383         # 6 总结
384         def summary(self):
385             p_pe = np.nanmean(np.array([self.df_relative_result['p_pe_1'].tolist(),
386                                         self.df_relative_result['p_pe_2'].tolist()]),
                                        axis=0)
387             p_ps = np.nanmean(np.array([self.df_relative_result['p_ps_1'].tolist(),
388                                         self.df_relative_result['p_ps_2'].tolist()]),
                                        axis=0)
389             p_pcf = np.nanmean(np.array([self.df_relative_result['p_pcf_d_1'].tolist(),
390                                          self.df_relative_result['p_pcf_i_1'].tolist(),
391                                          self.df_relative_result['p_pcf_d_2'].tolist(),
392                                          self.df_relative_result['p_pcf_i_2'].tolist()]),
                                           axis=0)
393             p_pb = np.nanmean(np.array([self.df_relative_result['p_pb'].tolist()]), axis=
                0)
394             p_evt = np.nanmean(np.array([self.df_relative_result['p_evt'].tolist()]),
                axis=0)
395             p = np.nanmean(np.array([p_pe, p_ps, p_pcf, p_pb, p_evt]), axis=0)
396
397             p_pe_price = np.nanmean(np.array([self.df_relative_result_price['p_pe_1'].
                tolist(),
398                                               self.df_relative_result_price['p_pe_2'].
```

```
                                            tolist()]), axis=0)
            p_ps_price = np.nanmean(np.array([self.df_relative_result_price['p_ps_1'].
            tolist(),
                                              self.df_relative_result_price['p_ps_2'].
                                              tolist()]), axis=0)
            p_pcf_price = np.nanmean(np.array([self.df_relative_result_price['p_pcf_d_1'
            ].tolist(),
                                               self.df_relative_result_price['p_pcf_i_1'
                                               ].tolist(),
                                               self.df_relative_result_price['p_pcf_d_2'
                                               ].tolist(),
                                               self.df_relative_result_price['p_pcf_i_2'
                                               ].tolist()]), axis=0)
            p_pb_price = np.nanmean(np.array([self.df_relative_result_price['p_pb'].
            tolist()]), axis=0)
            p_evt_price = np.nanmean(np.array([self.df_relative_result_price['p_evt'].
            tolist()]), axis=0)
            p_price = np.nanmean(np.array([p_pe_price, p_ps_price, p_pcf_price,
            p_pb_price, p_evt_price]), axis=0)
            dic_summery = {
                'relative_P_F': p,
                'relative_P_P': p_price,
                'real_P': self.price_cache
            }
            df_summary = pd.DataFrame(dic_summery, index=self.calender)
            self.df_summary = df_summary
            pd.DataFrame.to_csv(df_summary, "./Result/summary.csv", encoding='gbk')
            plt.figure(figsize=(32, 18))
            sns.lineplot(data=df_summary)
            plt.savefig('./Result/summary.png')

        # 测试用函数
        def test(self):
            time_x = time.time()
            self.basic_db()
            self.industry_match()
            self.average_db()
            self.fs_inquiry()
            self.relative_result()
            self.fs_result_price()
            self.relative_result_price()
            self.summary()
            time_y = time.time()
            print('计算完成, 用时' + str(round(time_y - time_x, 4)) + 's,
            结果见Result文件夹')

        # 测试用函数
        def test2(self):
            self.basic_db()
            self.fs_result_price()


    if __name__ == "__main__":
        sid = input('请输入目标公司的六位证券代码')
        while len(sid) != 6:
            print('输入无效,请重新输入')
            sid = input('请输入目标公司的六位证券代码')
        test = RelativeV(sid)
        test.test()
        # test.test2()
```