一、案例：计算复权价

```
# -*- coding: utf-8 -*-
"""
author: xingbuxing
date: 2017年04月23日
功能：本程序主要通过综合运用pandas的基本操作，像大家展示如何计算复权价格
"""
import pandas as pd
pd.set_option('expand_frame_repr', False)  # 当列太多时不换行

# =====导入数据
# 因为数据是gbk编码，read_csv需要加上encoding='gbk'参数，不然会乱码
df = pd.read_csv('/Users/jxing/Desktop/201704课程/20170423_class4/data/sz300001.csv',
encoding='gbk')
# print df['股票代码']  # 发现有问题，报错
# print df[u'股票代码']  # 不报错
# print df.columns[0], type(df.columns[0])  # encoding之后所有的字符串都变成了unicode格式，包括列名称
# 将列名转变成string
df.columns = [i.encode('utf8') for i in df.columns]  # 每次打u太麻烦，将unicode再变成string

# 取我们需要的列，其他的列不要
df = df[['交易日期', '股票代码', '开盘价', '最高价', '最低价', '收盘价', '涨跌幅']]
# 将数据按照交易日期从小到大排序
df.sort_values(by=['交易日期'], inplace=True)


# =====计算复权价，最重要的就是涨跌幅要复权。下面考察我们的数据中的涨跌幅
# df['涨跌幅2'] = df['收盘价'].pct_change()  # 通过pct_change计算基于未复权收盘价的涨跌幅
# print df[abs(df['涨跌幅2'] - df['涨跌幅']) > 0.0001]  # 数据中的涨跌幅是复权后的涨跌幅
# del df['涨跌幅2']
# 有了复权涨跌幅，其他的复权价格都可以自己算。


# =====计算复权后的收盘价
# ===计算复权因子
df['复权因子'] = (df['涨跌幅'] + 1).cumprod()
# 计算出来的复权因子的意义：我在一开始投入1元买这个股票，最后的收益是多少，资金曲线是什么样子的。
# 如果要计算投入100万买入这个股票，资金曲线是什么样子的，df['复权因子'] * 100万即可

# ===后复权收盘价，等于用等于该股票上市价格的钱，买入该股票后的资金曲线。
initial_price = df.iloc[0]['收盘价'] / (1 + df.iloc[0]['涨跌幅'])  # 计算上市价格
df['收盘价_后复权'] = initial_price * df['复权因子']  # 相乘得到复权价


# ===如果计算复权的开盘价、最高价、最低价？
# 通过如下公式计算：'开盘价_复权' / '收盘价_复权' = '开盘价' / '收盘价'
df['开盘价_后复权'] = df['开盘价'] / df['收盘价'] * df['收盘价_后复权']
df['最高价_后复权'] = df['最高价'] / df['收盘价'] * df['收盘价_后复权']
df['最低价_后复权'] = df['最低价'] / df['收盘价'] * df['收盘价_后复权']
```

```
# ===计算前复权价格
# 作为本周作业


# =====导出数据
# df.to_csv('output_fuquan_sz300001.csv')
df.to_csv('output_fuquan_sz300001.csv', index=False,
          mode='w',  # mode='w'代表覆盖之前的文件，mode='a'，代表接在原来数据的末尾
          float_format='%.15f',  # 控制输出浮点数的精度
          header=None,  # 不输出表头
          encoding='gbk'
          )
```

二、择时策略框架

```
# -*- coding: utf-8 -*-
"""
@author: Xingbuxing
date: 2017年04月23日
初步演示择时策略框架
"""
import pandas as pd
pd.set_option('expand_frame_repr', False)  # 当列太多时不换行


# =====读入数据
df = pd.read_csv('/Users/jxing/Desktop/201704课程/20170423_class4/data/sz300001.csv',
encoding='gbk')
df.columns = [i.encode('utf8') for i in df.columns]
df = df[['交易日期', '股票代码', '开盘价', '最高价', '最低价', '收盘价', '涨跌幅']]
df.sort_values(by=['交易日期'], inplace=True)
df['交易日期'] = pd.to_datetime(df['交易日期'])
df.reset_index(inplace=True, drop=True)


# =====计算复权价
df['复权因子'] = (df['涨跌幅'] + 1).cumprod()
initial_price = df.iloc[0]['收盘价'] / (1 + df.iloc[0]['涨跌幅'])  # 计算上市价格
df['收盘价_后复权'] = initial_price * df['复权因子']  # 相乘得到复权价
df['开盘价_后复权'] = df['开盘价'] / df['收盘价'] * df['收盘价_后复权']
df['最高价_后复权'] = df['最高价'] / df['收盘价'] * df['收盘价_后复权']
df['最低价_后复权'] = df['最低价'] / df['收盘价'] * df['收盘价_后复权']
# df[['开盘价', '最高价', '最低价', '收盘价']] = df[['开盘价_后复权', '最高价_后复权', '最低价_后复
权', '收盘价_后复权']]
df = df[['交易日期', '股票代码', '开盘价', '最高价', '最低价', '收盘价', '涨跌幅', '开盘价_后复权',
'最高价_后复权', '最低价_后复权', '收盘价_后复权']]
```

```
# =====计算均线策略
# 均线策略：
# 当短期均线由下向上穿过长期均线的时候，第二天以开盘价全仓买入并在之后一直持有股票。
# 当短期均线由上向下穿过长期均线的时候，第二天以开盘价卖出全部股票并在之后一直空仓，直到下一次买入。

# ===计算均线
ma_short = 5  # 短期均线。ma代表：moving_average
ma_long = 50  # 长期均线
df['ma_short'] = df['收盘价_后复权'].rolling(ma_short, min_periods=1).mean()
df['ma_long'] = df['收盘价_后复权'].rolling(ma_long, min_periods=1).mean()
# 将缺失的均线数据补全
# df['ma_short'].fillna(value=df['收盘价'].expanding().mean(), inplace=True)
# df['ma_long'].fillna(value=df['收盘价'].expanding().mean(), inplace=True)
# print df
# exit()
# 补全数据的另外一种方式是使用rolling函数中的min_periods参数

# ===找出买入信号
# 当天的短期均线大于等于长期均线
condition1 = (df['ma_short'] >= df['ma_long'])
# 上个交易日的短期均线小于长期均线
condition2 = (df['ma_short'].shift(1) < df['ma_long'].shift(1))
# 将买入信号当天的signal设置为1
df.loc[condition1 & condition2, 'signal'] = 1


# ===找出卖出信号
# 当天的短期均线小于等于长期均线
condition1 = (df['ma_short'] <= df['ma_long'])
# 上个交易日的短期均线大于长期均线
condition2 = (df['ma_short'].shift(1) > df['ma_long'].shift(1))
# 将买入信号当天的signal设置为0
df.loc[condition1 & condition2, 'signal'] = 0
# 将无关的变量删除
df.drop(['ma_short', 'ma_long'], axis=1, inplace=True)


# =====由signal计算出实际的每天持有股票仓位
# ===计算仓位
# signal的计算运用了收盘价，是每天收盘之后产生的信号，到第二天的时候，仓位position才会改变。
# 例如2009-11-17产生买入信号，2009-11-18仓位才会编变成1。满仓用1表示，空仓用0表示
df['pos'] = df['signal'].shift()
df['pos'].fillna(method='ffill', inplace=True)
df['pos'].fillna(value=0, inplace=True)  # 将初始行数的position补全为0

# 这就是实际的仓位了吗？刚刚的计算逻辑是不是有没有什么问题？
# 查看20150501这几天的数据
# print df[df['交易日期'] > pd.to_datetime('20150501')][['交易日期', '开盘价_后复权', '收盘价_后复
权', '涨跌幅', 'signal', 'pos']]

# 涨跌停的时候是不得买卖股票的。很多人策略表现好，可能就是没有考虑这些限制。
# 这类策略和实际操作不吻合的问题，是经常犯的问题。

# 有的问题隐藏的很深，很多时候只有到了实盘交易的时候才会发现
```

```
# ===跌停时不得买卖股票考虑进来
# 找出开盘涨停的日期
# 今天的开盘价相对于昨天的收盘价上涨了9.7%。为什么用9.7%？不用10%
cond_cannot_buy = df['开盘价_后复权'] > df['收盘价_后复权'].shift(1) * 1.097
# 将开盘涨停日、并且当天position为1时的'pos'设置为空值
df.loc[cond_cannot_buy & (df['pos'] == 1), 'pos'] = None
# print df[df['交易日期'] > pd.to_datetime('20150501')][['交易日期', '开盘价_后复权', '收盘价_后复权', '涨跌幅', 'signal', 'pos']]

# 找出开盘跌停的日期
# 今天的开盘价相对于昨天的收盘价下跌了9.7%
cond_cannot_sell = df['开盘价_后复权'] < df['收盘价_后复权'].shift(1) * 0.903
# 将开盘跌停日、并且当天position为0时的'pos'设置为空值
df.loc[cond_cannot_sell & (df['pos'] == 0), 'pos'] = None

# position为空的日期，不能买卖。position只能和前一个交易日保持一致。
df['pos'].fillna(method='ffill', inplace=True)
# print df[df['交易日期'] > pd.to_datetime('20150501')][['交易日期', '开盘价_后复权', '收盘价_后复权', '涨跌幅', 'signal', 'pos']]

# ===截取上市一年之后的交易日
df = df.iloc[250-1:]
# 将第一天的仓位设置为0
df.iloc[0, -1] = 0
# print df[['交易日期', '开盘价_后复权', '收盘价_后复权', '涨跌幅', 'signal', 'pos']]
# exit()


# =====计算实际资金曲线(简单方法)
# 资金曲线是一个策略最终的结果。是评价一个策略最重要的标准。

# ==计算实际资金曲线
# 首先计算资金曲线每天的涨幅
# 当当天空仓时，pos为0，资产涨幅为0
# 当当天满仓时，pos为1，资产涨幅为股票本身的涨跌幅
df['equity_change'] = df['涨跌幅'] * df['pos']
# 根据每天的涨幅计算资金曲线
df['equity_curve'] = (df['equity_change'] + 1).cumprod()
# print df[['交易日期', '收盘价_后复权', 'pos', 'equity_change', 'equity_curve']]
# exit()

# 这样计算方式的缺点：
# 没有考虑交给券商的手续费,以及没有考虑交给国家的印花税
# 没有考虑交易的滑点
# 以及没有考虑...


# =====计算实际资金曲线(实际方法)
df = df[['交易日期', '股票代码', '开盘价', '最高价', '最低价', '收盘价', '涨跌幅', 'pos']]
df.reset_index(inplace=True, drop=True)


# ===设定参数
```

```
initial_money = 1000000  # 初始资金，默认为1000000元
slippage = 0.01  # 滑点，默认为0.01元
c_rate = 5.0 / 10000  # 手续费，commission fees，默认为万分之5
t_rate = 1.0 / 1000  # 印花税，tax，默认为千分之1

# ===第一天的情况
df.at[0, 'hold_num'] = 0  # 持有股票数量，此处也可用loc，但是定位单个元素at效率更高。
df.at[0, 'stock_value'] = 0  # 持仓股票市值
df.at[0, 'actual_pos'] = 0  # 每日的实际仓位
df.at[0, 'cash'] = initial_money  # 持有现金现金
df.at[0, 'equity'] = initial_money  # 总资产 = 持仓股票市值 + 现金
# print df[['交易日期', '开盘价', 'pos', 'hold_num', 'stock_value', 'actual_pos', 'cash',
'equity']]

# ===第一天之后每天的情况
# 从第二行开始，逐行遍历，逐行计算
for i in range(1, df.shape[0]):

    # 前一天持有的股票的数量
    hold_num = df.at[i - 1, 'hold_num']

    # 判断当天是否除权，若发生除权，需要调整hold_num
    # 若当天通过收盘价计算出的涨跌幅，和当天实际涨跌幅不同，说明当天发生了除权
    if abs((df.at[i, '收盘价'] / df.at[i-1, '收盘价'] - 1) - df.at[i, '涨跌幅']) > 0.001:
        stock_value = df.at[i - 1, 'stock_value']
        # 交易所会公布除权之后的价格
        last_price = df.at[i, '收盘价'] / (df.at[i, '涨跌幅'] + 1)
        hold_num = stock_value / last_price
        hold_num = int(hold_num)
        # if i > 1030:
        #     print stock_value, last_price, hold_num
        #     print df.iloc[1034:][['交易日期', '收盘价', '涨跌幅', 'pos', 'hold_num', 'cash',
'stock_value']]

    # 判断是否需要调整仓位：拿今天的仓位pos，和昨天的仓位pos进行比较，看是否相同
    # 需要调整仓位
    if df.at[i, 'pos'] != df.at[i - 1, 'pos']:

        # 对于需要调整到的仓位，需要买入多少股票
        # 昨天的总资产 * 今天的仓位 / 今天的收盘价，得到需要持有的股票数
        theory_num = df.at[i - 1, 'equity'] * df.at[i, 'pos'] / df.at[i, '开盘价']
        # 对需要持有的股票数取整
        theory_num = int(theory_num)  # 向下取整数，向上取整会出现钱不够的情况

        # 将theory_num和昨天持有股票相比较，判断加仓还是减仓
        # 加仓
        if theory_num >= hold_num:
            # 计算实际需要买入的股票数量
            buy_num = theory_num - hold_num
            # 买入股票只能整百，对buy_num进行向下取整百
            buy_num = int(buy_num / 100) * 100


            # 计算买入股票花去的现金
```

```
            buy_cash = buy_num * (df.at[i, '开盘价'] + slippage)
            # 计算买入股票花去的手续费，并保留2位小数
            commission = round(buy_cash * c_rate, 2)
            # 不足5元按5元收
            if commission < 5 and commission != 0:
                commission = 5
            df.at[i, '手续费'] = commission

            # 计算当天收盘时持有股票的数量和现金
            df.at[i, 'hold_num'] = hold_num + buy_num  # 持有股票，昨天持有的股票，加上今天买入的股
票
            df.at[i, 'cash'] = df.at[i - 1, 'cash'] - buy_cash - commission  # 剩余现金
            # print df[['交易日期', '开盘价', 'pos', 'hold_num', 'cash', '手续费']]
            # exit()

        # 减仓
        else:
            # 计算卖出股票数量，卖出股票可以不是整数，不需要取整百。
            sell_num = hold_num - theory_num

            # 计算卖出股票得到的现金
            sell_cash = sell_num * (df.at[i, '开盘价'] - slippage)
            # 计算手续费，不足5元按5元收，并保留2位小数
            commission = round(max(sell_cash * c_rate, 5), 2)
            df.at[i, '手续费'] = commission
            # 计算印花税，保留2位小数。历史上有段时间，买入也会收取印花税
            tax = round(sell_cash * t_rate, 2)
            df.at[i, '印花税'] = tax

            # 计算当天收盘时持有股票的数量和现金
            df.at[i, 'hold_num'] = hold_num - sell_num  # 持有股票
            df.at[i, 'cash'] = df.at[i - 1, 'cash'] + sell_cash - commission - tax  # 剩余现金
            # print df.iloc[50:100][['交易日期', '开盘价', 'pos', 'hold_num', 'cash', '手续费',
'印花税']]

    # 不需要调仓
    else:
        # 计算当天收盘时持有股票的数量和现金
        df.at[i, 'hold_num'] = hold_num  # 持有股票
        df.at[i, 'cash'] = df.at[i - 1, 'cash']  # 剩余现金。此处的cash可以乘以余额宝的收益率。
        # print df[['交易日期', 'pos', 'hold_num', 'cash']]

    # 以上的计算得到每天的hold_num和cash
    # 计算当天收盘时的各种资产数据
    df.at[i, 'stock_value'] = df.at[i, 'hold_num'] * df.at[i, '收盘价']  # 股票市值
    df.at[i, 'equity'] = df.at[i, 'cash'] + df.at[i, 'stock_value']  # 总资产
    df.at[i, 'actual_pos'] = df.at[i, 'stock_value'] / df.at[i, 'equity']  # 实际仓位

# print df[['交易日期', 'pos', 'cash', 'stock_value', 'equity', 'actual_pos']]




df = df[['交易日期', '收盘价', 'pos', 'hold_num', 'cash', 'stock_value', 'equity', 'actual_pos',
```

```
'手续费', '印花税']]
print df

# print df[['手续费', '印花税']].sum()
```