文章思路：

1. 简单介绍算法交易和历史VWAP模型，并说明本文动机：由于目前网上缺少VWAP相关研究，本文根据广发研报结论选取适当参数对VWAP结果做简单测试，判断其绩效，并简单对不同参数情况进行比较讨论。
2. 选取上证50综指以及具有行业代表性、流通性较好的蓝筹股：招商银行、阳泉煤业、南方航空、紫金矿业、万科A，采用研报中适当参数对四只标的进行TWAP与VWAP实证研究，对订单执行情况和成本节约情况进行绩效评估，关注四个指标（两个量两个价）：成交量分布预测准确度（vdfa，Volume distribution forecast accuracy）、订单执行率（oer，Order execution rate）、与市场均价误差比（maper，Market average price error ratio）、与前日收盘价误差比（pdcper，Previous day closing price error ratio），分析其均值、方差，画图分析。
3. 简要总结与展望下一步——VWAP适用条件、参数寻优和动态VWAP策略实施。

作者介绍：

在下某华东985金工小硕一枚，信仰价值投资，坚信以基本面为基础的量化投资是中国投资界的未来，潜心钻研并常以此为乐！一年5倍者如过江之卿，五年1倍者却廖若繁星！与君更勉，欢迎添加微信（13912998609, yangye-123456）多多交流！

前言：

半个月前关注“量化投资与机器学习”公众号，得知其与万矿有个合作福利，即半个月内在社区发布一策略研究贴，通过审核后便可初步获得数据高级权限。在下不才，于实习中老板要求研究算法交易，并发现现阶段网上尤其是万矿平台上缺少相关研究，在此抛砖引玉，对算法交易进行简要介绍与实证测试，并在接下来进行应用条件、参数寻优和动态VWAP研究。（由于时间紧急，本文没做公式编辑）

一、算法交易概述

算法交易（Algorithmic Trading），指事先设计好交易策略，编制成计算机程序，并通过程序计算出的结果进行自动下单交易。算法交易专注择时和下单策略，依赖复杂的数学公式与计算机程序。具体来说，交易者在金融市场中进行较大规模的交易时，规模较大的单笔交易对于流动性相对较差的市场具有强烈的冲击，从而会造成市场瞬间的剧烈波动。为了减少市场波动对交易所造成的不利影响，交易者通常会将需要进行交易的订单拆细，即将大规模交易拆分为若干小规模交易，并在合适的时机分别对其进行分散交易，从而降低相关交易成本——特别是冲击成本，使得整个交易过程中价格能够达到最优水平。因此，算法交易的目标可总结为：通过拆单隐藏交易动机，减少冲击成本，在合理的价格水平下完成大额交易，获取Alpha收益。

算法交易的完整流程包括四步，分别是算法模型（策略）研究、交易系统的设计与开发、交易的执行与交易后分析。

二、发展历程

回顾历史，算法交易的发展主要可以分为几个阶段：第一阶段，算法较为简单，注重交易的执行效率，以时间和交易量为纲，以TWAP与Berkowitz于1988年提出的VWAP策略为代表，VWAP策略发展至今已经较为成为，迄今为止依然是市场内最为流行、用途最广的算法交易策略；第二阶段，以Almgren和Chriss的执行差额IS策略为代表，更加注重交易质量的提升；第三阶段，主要意图在于隐藏流动性，但受交易规则桎梏，该阶段内产生的大部分策略仅适用于海外市场，如夜鹰（Nighthawk）、狙击兵（Sniper）等；第四阶段，该阶段算法交易策略更趋于智能交易，如复杂事件处理（Complex Event，CEP）、新闻交易（News Trading）等。

三、TWAP、VWAP策略原理简介

本文是算法交易系列文章的第一篇，因此主要关注的是最基础的TWAP、VWAP模型及其应用效果与不足。

1.TWAP策略原理

TWAP（Time Weighted Average Price），时间加权平均价格算法，是最为简单的一种传统算法交易策略。该模型将交易时间进行均匀分割，并在每个分割节点上将均匀拆分的订单进行提交。例如，A 股市场一个交易日的交易时间为4小时，即240分钟。首先将这240分钟均匀分为N份（或将240分钟中的某一部分均匀分割），如240份。 TWAP 策略会将该交易日需要执行的订单均匀分配在这 240 个节点上去执行，从而使得交易均价跟踪TWAP为该交易日1分钟收盘价的算术平均值。

TWAP存在的主要问题是在订单规模很大的情况下，均匀分配到每个节点上的下单量仍然较为可观，仍有可能对市场造成一定的冲击，可能会出现无法完全交易的情况。另一方面，真实市场的成交量是在波动变化的，将所有的订单均匀分配到每个节点上显然是不够合理的。因此，人们很快建立了基于成交量变动预测的 VWAP 模型。不过，由于 TWAP 操作和理解起来非常简单，因此其对于流动性较好的市场和订单规模较小的交易仍然较为适用。

2.VWAP策略原理

VWAP（Volume Weighted Average Price），成交量加权平均价格算法，是目前市场上最为流行的算法交易策略之一，也是很多其它算法交易模型的原型。首先定义VWAP，它是一段时间内证券价格按成交量加权的平均值。VWAP算法交易策略的目的就是尽可能地使订单拆分所成交的VWAP盯住市场的VWAP，那么在拆分订单时需要按照市场真实的成交量分时按比例进行提交，这就需要对市场分时成交量进行预测。传统历史VWAP策略利用股票的历史成交数据来估算日内成交量的分布，即T日第i时段的下单量比例为T日前ndays日该i时段交易量的算术平均值所占T日前ndays日交易量算数平均值的比例。最后，VWAP策略的目标是最小化实际加权成交价与市场加权均价的差异。

四、TWAP、VWAP测试

本文参数主要参考广发证券算法交易系列研报，其中模拟每日交易量选取前一日交易量的20%，VWAP交易量分布估计时间窗口选取60天。

In [1]:

```
import numpy as np
import pandas as pd
import datetime
from WindCharts import *
from scipy import stats
import matplotlib.pyplot as plt
```

1. 首先是TWAP函数。

In [7]:

```
def twap_vol_trade(data,ratio):
    # 根据历史交易量计算下一日总订单量
    data['tradeday'] = pd.to_datetime(data['tradeday'],format='%Y/%m/%d')
    data['date_time'] = pd.to_datetime(data['date_time'],format='%Y/%m/%d %H:%M:%S')
    data.drop_duplicates(subset = ['sec_code','date_time'], keep = 'first', inplace = True)
    data.sort_values(by = ['sec_code', 'date_time'], inplace = True)
    # 删除停牌交易日观测
    data = data[data.groupby(['sec_code','tradeday'])['volume'].apply(lambda x:pd.Series(np.full(len(x),x.sum()>0))).values]
    ## 计算日交易量，此处假定是市场前日交易量的ratio比例
    output = pd.DataFrame()
    output[['sec_code','tradeday']] = data[['sec_code','tradeday']].drop_duplicates()
    output['volume_trade'] = (data.groupby(['sec_code','tradeday'])['volume'].sum()*ratio).values
    output['tradeday'] = output.groupby('sec_code')['tradeday'].shift(-1)
    data1 = pd.merge(data, output, on = ['sec_code','tradeday'], how = 'left')
    return data1

def twap_vol_dis(data1):
    ## 根据TWAP策略进行拆单交易
    data2 = data1.copy()
    data2['weight_forcast'] = 1
    data2['weight_forcast'] = data2.groupby(['sec_code','tradeday'])['weight_forcast'].apply(lambda x: x/x.sum())
    data2['volume_trade_i'] = data2['volume_trade'] * data2['weight_forcast']
    data2['weight_market'] = data2.groupby(['sec_code','tradeday'])['volume'].apply(lambda x:x/x.sum())
    data2['volume_real_i'] = data2.apply(lambda x: min(x['volume'], x['volume_trade_i']), axis = 1)
    data2['weight_real'] = data2.groupby(['sec_code','tradeday'])['volume_real_i'].apply(lambda x:x/x.sum())
    return data2


def twap_result(data2):
    # 计算绩效评价指标，判断策略执行效果(多变量分组计算后赋值的一般方法是？)
    output = pd.DataFrame()
    output[['sec_code','tradeday']] = data2[['sec_code','tradeday']].drop_duplicates()
    output['vdfa'] = data2.groupby(['sec_code','tradeday']).apply(lambda x: pow((x['weight_forcast'] - x['weight_market']),2).sum()).values
    output['oer'] = data2.groupby(['sec_code','tradeday']).apply(lambda x:((x['volume_real_i'].sum())/x['volume_trade']).mean()).values
    output['vwap_market'] = data2.groupby(['sec_code','tradeday']).apply(lambda x: (x['p_close'] * x['weight_market']).sum()).values
    output['twap_real'] = data2.groupby(['sec_code','tradeday']).apply(lambda x: (x['p_close'] * x['weight_real']).sum()).values
    output['maper'] = (output['twap_real'] - output['vwap_market'])/output['vwap_market']
    output['p_close_pre'] = data2[data2['date_time'].apply(lambda x:x.strftime('%H:%M:%S') == '15:00:00')]['p_close'].shift(1).values
    output['pdcper'] = (output['twap_real'] - output['p_close_pre'])/output['p_close_pre']
    return output

def twap(data,ratio):
    '''@author: langzi_ye
    利用算法交易拆单进行交易，减小市场冲击，并进行模拟交易，回测其四大指标vdfa，oer，maper，pdcper
    删除停牌的观测
    输入值
    ratio Float 阈值，默认值为0.2
    data DataFrame 数据框，包括各标的的分钟收盘价、交易量等数据
    输出值
    data2 DataFrame 数据框，在data基础上添加模拟交易数据字段
    output DataFrame 数据框，回测结果，包括vwap和四大指标数据
    '''
    data1 = twap_vol_trade(data,ratio)
    data2 = twap_vol_dis(data1)
    output = twap_result(data2)
    return data2,output
```

1. 然后是VWAP函数。

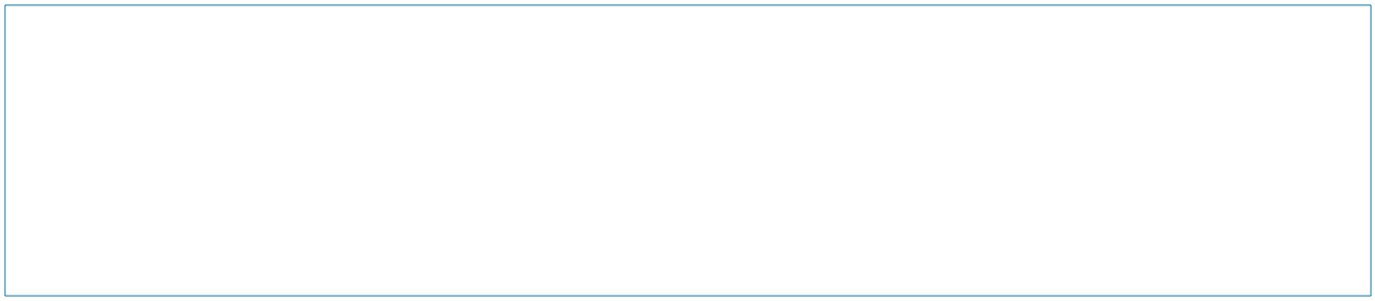
In [8]:

```
def vwap_vol_trade(data,ratio):
    # 根据历史交易量计算下一日总订单量
    data['tradeday'] = pd.to_datetime(data['tradeday'],format='%Y/%m/%d')
    data['date_time'] = pd.to_datetime(data['date_time'],format='%Y/%m/%d %H:%M:%S')
    data.drop_duplicates(subset = ['sec_code','date_time'], keep = 'first', inplace = True)
    data.sort_values(by = ['sec_code', 'date_time'], inplace = True)
    # 删除停牌交易日观测
    data = data[data.groupby(['sec_code','tradeday'])['volume'].apply(lambda x:pd.Series(np.full(len(x),x.sum()>0))).values]
    ## 计算日交易量，此处假定是市场前日交易量的ratio比例
    output = pd.DataFrame()
    output[['sec_code','tradeday']] = data[['sec_code','tradeday']].drop_duplicates()
    output['volume_trade'] = (data.groupby(['sec_code','tradeday'])['volume'].sum()*ratio).values
    output['tradeday'] = output.groupby('sec_code')['tradeday'].shift(-1)
    data1 = pd.merge(data, output, on = ['sec_code','tradeday'], how = 'left')
    return data1

def vwap_vol_dis(data1,ndays):
    ## 根据历史交易数据估计t日交易量分布，并进行拆单模拟交易
    time = data1['date_time'].apply(lambda x:x.strftime('%H:%M:%S')).drop_duplicates()
    data11 = pd.DataFrame()
    for i in range(len(time)):
        data1_i = data1[data1['date_time'].apply(lambda x:x.strftime('%H:%M:%S')) == time[i]]
        data1_i['volume_forcast'] = data1_i.groupby('sec_code')['volume'].apply(pd.rolling_mean, ndays)
        data1_i['date_time'] = data1_i.groupby('sec_code')['date_time'].shift(-1)
        data1_i = data1_i[['sec_code','date_time','volume_forcast']]
        data11 = pd.concat([data11,data1_i],axis = 0)

    data11.sort_values(by = ['sec_code','date_time'], inplace = True)
    data2 = pd.merge(data1, data11, on = ['sec_code','date_time'], how = 'left')
    data2['weight_forcast'] = data2.groupby(['sec_code','tradeday'])['volume_forcast'].apply(lambda x:x/x.sum())
    data2['volume_trade_i'] = data2['volume_trade'] * data2['weight_forcast']
    data2['weight_market'] = data2.groupby(['sec_code','tradeday'])['volume'].apply(lambda x:x/x.sum())
    data2['volume_real_i'] = data2.apply(lambda x: min(x['volume'], x['volume_trade_i']), axis = 1)
    data2['weight_real'] = data2.groupby(['sec_code','tradeday'])['volume_real_i'].apply(lambda x:x/x.sum())
    return data2

def vwap_result(data2):
    # 计算绩效评价指标，判断策略执行效果(多变量分组计算后赋值的一般方法是？)
    output = pd.DataFrame()
    output[['sec_code','tradeday']] = data2[['sec_code','tradeday']].drop_duplicates()
    output['vdfa'] = data2.groupby(['sec_code','tradeday']).apply(lambda x: pow((x['weight_forcast'] - x['weight_market']),2).sum()).values
    output['oer'] = data2.groupby(['sec_code','tradeday']).apply(lambda x:((x['volume_real_i'].sum())/x['volume_trade']).mean()).values
    output['vwap_market'] = data2.groupby(['sec_code','tradeday']).apply(lambda x: (x['p_close'] * x['weight_market']).sum()).values
    output['vwap_real'] = data2.groupby(['sec_code','tradeday']).apply(lambda x: (x['p_close'] * x['weight_real']).sum()).values
    output['maper'] = (output['vwap_real'] - output['vwap_market'])/output['vwap_market']
    output['p_close_pre'] = data2[data2['date_time'].apply(lambda x:x.strftime('%H:%M:%S') == '15:00:00')]['p_close'].shift(1).values
    output['pdcper'] = (output['vwap_real'] - output['p_close_pre'])/output['p_close_pre']
    return output

def vwap(data,ndays,ratio):
    '''@author: 87374
    利用算法交易拆单进行交易，减小市场冲击，并进行模拟交易，回测其四大指标vdfa，oer，maper，pdcper
    删除停牌的观测
    输入值
    ndays float 时间窗口长度，默认值为5
    ratio Float 阈值，默认值为0.2
    data DataFrame 数据框，包括各标的的分钟收盘价、交易量等数据
    输出值
    data2 DataFrame 数据框，在data基础上添加模拟交易数据字段
    output DataFrame 数据框，回测结果，包括vwap和四大指标数据
    '''
    data1 = vwap_vol_trade(data,ratio)
    data2 = vwap_vol_dis(data1,ndays)
    output = vwap_result(data2)
    return data2,output
```

结果如下：

In [9]:

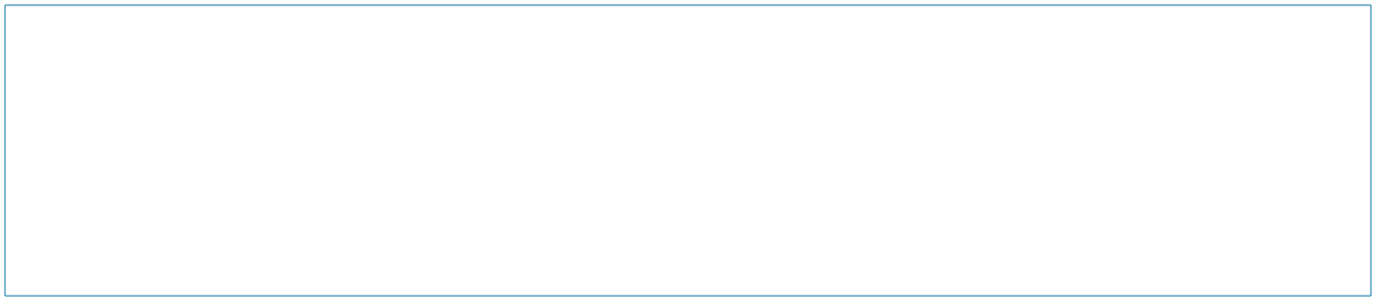
```
input_data = pd.read_csv('data/data.csv', encoding='gbk')
data_twap,output_twap = twap(input_data, 0.2)
data_vwap,output_vwap = vwap(input_data, 60, 0.2)
```



```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
<pbl> in <module>


----> 1 input_data = pd.read_csv('data/data.csv', encoding='gbk')
 2 data_twap,output_twap = twap(input_data, 0.2)
 3 data_vwap,output_vwap = vwap(input_data, 60, 0.2)

/opt/conda/lib/python3.6/site-packages/pandas/io/parsers.py in parser_f(filepath_or_buffer, sep, delimiter, header, names, index_col, usecols, squeeze, prefix, mangle_dupe_cols, d
type, engine, converters, true_values, false_values, skipinitialspace, skiprows, skipfooter, nrows, na_values, keep_default_na, na_filter, verbose, skip_blank_lines, parse_dates,
infer_datetime_format, keep_date_col, date_parser, dayfirst, iterator, chunksize, compression, thousands, decimal, lineterminator, quotechar, quoting, doublequote, escapechar, com
ment, encoding, dialect, tupleize_cols, error_bad_lines, warn_bad_lines, delim_whitespace, low_memory, memory_map, float_precision)
    700                     skip_blank_lines=skip_blank_lines)
    701
--> 702         return _read(filepath_or_buffer, kwds)
    703
    704     parser_f.__name__ = name

/opt/conda/lib/python3.6/site-packages/pandas/io/parsers.py in _read(filepath_or_buffer, kwds)
    427
    428     # Create the parser.
--> 429     parser = TextFileReader(filepath_or_buffer, **kwds)
    430
    431     if chunksize or iterator:

/opt/conda/lib/python3.6/site-packages/pandas/io/parsers.py in __init__(self, f, engine, **kwds)
    893             self.options['has_index_names'] = kwds['has_index_names']
    894
--> 895         self._make_engine(self.engine)
    896
    897     def close(self):

/opt/conda/lib/python3.6/site-packages/pandas/io/parsers.py in _make_engine(self, engine)
   1120     def _make_engine(self, engine='c'):
   1121         if engine == 'c':
-> 1122             self._engine = CParserWrapper(self.f, **self.options)
   1123         else:
   1124             if engine == 'python':

/opt/conda/lib/python3.6/site-packages/pandas/io/parsers.py in __init__(self, src, **kwds)
   1851         kwds['usecols'] = self.usecols
   1852
-> 1853         self._reader = parsers.TextReader(src, **kwds)
   1854         self.unnamed_cols = self._reader.unnamed_cols
   1855

pandas/_libs/parsers.pyx in pandas._libs.parsers.TextReader.__cinit__()

pandas/_libs/parsers.pyx in pandas._libs.parsers.TextReader._setup_parser_source()

FileNotFoundError: [Errno 2] File b'data/data.csv' does not exist: b'data/data.csv'
```

In [6]:

```
wt_output_twap = WTable(output_twap) #注意：只能接收DataFrame参数
wt_output_twap.plot()
wt_output_vwap = WTable(output_vwap) #注意：只能接收DataFrame参数
wt_output_vwap.plot()
```



```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<pbl> in <module>


----> 1 wt_output_twap = WTable(output_twap) #注意：只能接收DataFrame参数
 2 wt_output_twap.plot()
 3 wt_output_vwap = WTable(output_vwap) #注意：只能接收DataFrame参数

NameError: name 'output_twap' is not defined
```

截取有效结果并分组计算其均值与标准差

In [34]:

```
output_twap_use = output_twap[pd.notnull(output_twap['oer'])]
output_vwap_use = output_vwap[pd.notnull(output_vwap['vdfa'])]
result_twap_p = output_twap_use.groupby('sec_code')[['vwap_market','twap_real','p_close_pre']].agg(['mean','std'])
result_vwap_p = output_vwap_use.groupby('sec_code')[['vwap_market','vwap_real','p_close_pre']].agg(['mean','std'])
result_twap_v = output_twap_use.groupby('sec_code')[['vdfa','oer','maper','pdcper']].agg(['mean','std'])
result_vwap_v = output_vwap_use.groupby('sec_code')[['vdfa','oer','maper','pdcper']].agg(['mean','std'])
```

In [35]:

```
result_twap_v
```

Out[35]:

	vdfa		oer		maper		pdcper	
	mean	std	mean	std	mean	std	mean	std
sec_code								
000002.SZ	0.003986	0.001976	0.973218	0.038004	-0.000555	0.002337	0.000087	0.019631
600029.SH	0.004766	0.002470	0.966220	0.034190	-0.000540	0.002898	-0.000968	0.018220
600036.SH	0.003415	0.001653	0.981893	0.025028	-0.000374	0.001482	0.000732	0.013375
600348.SH	0.005178	0.002416	0.940687	0.048266	-0.000535	0.002413	-0.000427	0.016370
601899.SH	0.005875	0.002934	0.962415	0.043178	-0.000537	0.001757	-0.001899	0.015683

In [36]:

```
result_vwap_v
```

Out[36]:

	vdfa		oer		maper		pdcper	
	mean	std	mean	std	mean	std	mean	std
sec_code								
000002.SZ	0.003152	0.001703	0.991581	0.030564	-0.000643	0.002977	6.593350e-07	0.019435
600029.SH	0.003990	0.002054	0.983943	0.020529	-0.000278	0.002554	-1.415525e-03	0.018384
600036.SH	0.002735	0.001203	0.992515	0.016252	-0.000328	0.001788	6.092165e-04	0.013712
600348.SH	0.004372	0.002230	0.966487	0.030076	-0.000509	0.002905	-4.614998e-04	0.016337
601899.SH	0.004767	0.002581	0.982484	0.024117	-0.000444	0.001964	-2.145880e-03	0.015814

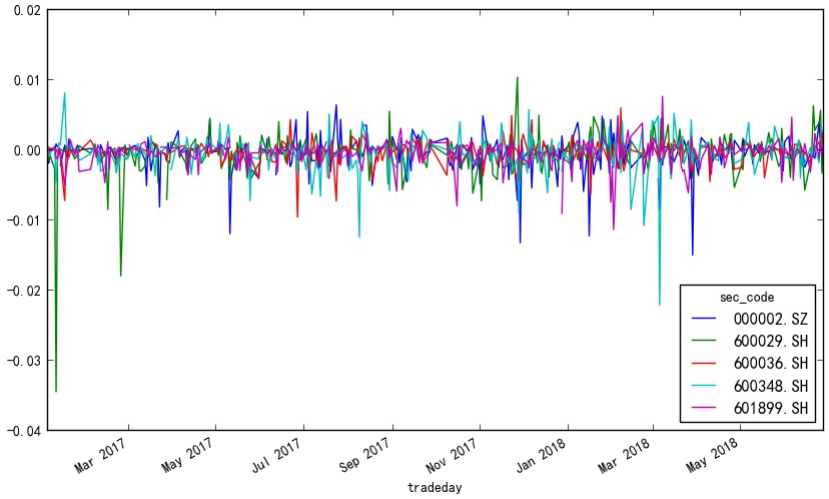
可以发现：

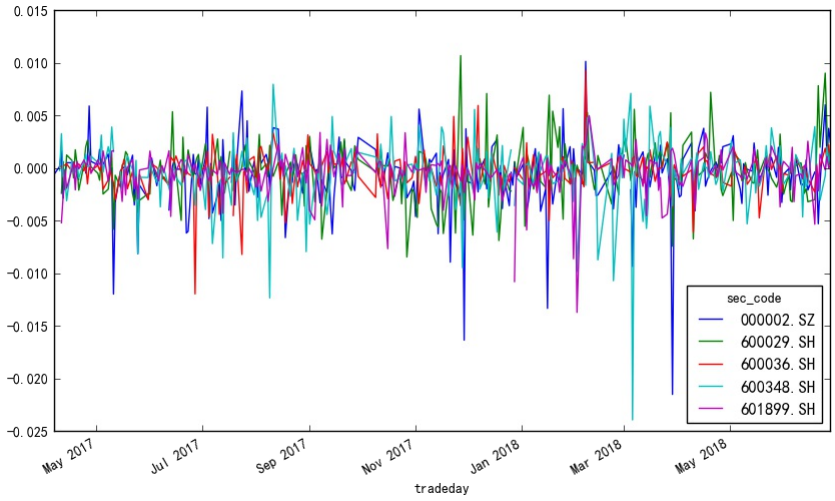
1. 在成交量分布预测准确度（vdfa，Volume distribution forecast accuracy）方面，vwap明显好于twap，这与逻辑直觉是吻合的；
2. 在订单执行率（oer，Order execution rate）方面，vwap明显好于twap，平均高出两个百分点，这是因为相比于死板的twap策略，vwap策略可以更好地捕捉日内交易量分布，从而优化订单分配效率和执行效率。
3. 在与市场均价误差比（maper，Market average price error ratio）和与前日收盘价误差比（pdcper，Previous day closing price error ratio）方面，vwap与twap均值都为负（通过了KS检验），这说明这两个策略都减小了交易成本，平均来看是千分之几的数量级，如果交易额有1000万的话，就可以节省几万元。但是这两个策略效果几乎无差，这可能是因为这些标的流动性较好，价格日内波动较小。值得注意的是，vwap的标准差较大，代表其波动较大，如下图，vwap有着更多更小的负值，这意味着vwap有着很大的改进空间，使得其保留更多这些负值，在正值交易日准确判断并不进行交易，从而使得交易成本进一步降低。

In [37]:

```
result_twap_maper = output_twap_use.pivot(index = 'tradeday', values = 'maper', columns = 'sec_code')
result_vwap_maper = output_vwap_use.pivot(index = 'tradeday', values = 'maper', columns = 'sec_code')
result_twap_maper.plot(figsize = (10,6))
result_vwap_maper.plot(figsize = (10,6))
```

Out[37]:





五、总结与展望

本文简单介绍了一下算法交易，解释了TWAP策略和VWAP策略，并进行实证测试，发现算法交易策略的确可以减小交易成本，优化订单执行。其中，vwap策略在订单执行率上有优势，但是交易成本的减少两策略差异却不大。其中，由于vwap策略负值更小更多，有着更大的改进空间。因此，接下来本系列的研究方向如下：

1. 基于历史VWAP策略，寻找VWAP适用条件，比如根据市场透明度、流通性、市值等指标来判断，对标的和交易日期进行选择性交易；
2. 对历史VWAP策略进行参数寻优，寻找不同标的和市场趋势下适用的参数范围；
3. 动态VWAP策略实施，比如根据日内价格对交易量进行实时调整等。

本文研究不足：写完这篇策略贴后，我又仔细思考了一下。无论是TWAP和VWAP，只要执行率不到100%，就说明在市场的某一时刻交易时存在饱和单，然而在实际交易中该阶段内的订单几乎不能完全执行，因此这与现实交易会有一定出入。同时，TWAP订单执行率更低，这说明与现实交易差异更大，因此本文只能说明TWAP在理论上交易成本和VWAP无差，但实践中还需证实！另外，由于下单方式也有讲究，究竟是市价订单更优还是限价订单更优，还需要进行实践测试或者人工股市模拟交易。

还有，在python中不会适用双样本ks检验，因此没有去比较maper与0是否具有显著性差异，还需继续学习！（希望有会的朋友可以留言教教我）

参考文献：

1. 广发证券算法交易系列研究之一——积小流以成江海——关于算法交易的一个综述
2. 广发证券算法交易系列研究之二——传统算法交易策略中的相关参数研究