# 【Python】No.5 Python在股票分析中的综合应用——金叉死叉交易策略与可视化

NAU Analysts 2020-03-11 21:41

以下文章来源于心予欣愿 ， 作者原禹欣

**心予欣愿**

记录一下欣欣小兔叽的daily life~

上一篇介绍了Python的四大基本容器、循环判断、异常机制、类和函数等内容，这一次我们就来介绍一些Python在股票上的应用

## 1. 绘制均线

首先我们导入tushare、pandas：

```
import tushare as ts
import pandas as pd
```

获取股票数据

```
df = ts.get_hist_data('600848')
df
```

我们也可以输出看一下数据是什么样子的

| date | open | high | close | low | volume | price_change | p_change | ma5 | ma10 | ma20 | v_ma5 | v_ma10 | v_ma20 | MA5 | pchange | change | MA15 | MA20 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 2020-03-06 | 21.60 | 21.61 | 21.46 | 21.38 | 44897.09 | -0.18 | -0.83 | 21.390 | 21.450 | 21.488 | 55024.54 | 71101.56 | 68821.30 | NaN | NaN | NaN | NaN | NaN |
| 2020-03-05 | 21.87 | 21.87 | 21.64 | 21.50 | 63113.81 | 0.17 | 0.79 | 21.180 | 21.566 | 21.459 | 64250.27 | 82154.50 | 69555.66 | NaN | 0.008388 | 0.18 | NaN | NaN |
| 2020-03-04 | 21.13 | 21.54 | 21.47 | 20.97 | 52519.09 | 0.17 | 0.80 | 21.130 | 21.585 | 21.418 | 62789.40 | 83717.25 | 69772.48 | NaN | -0.007856 | -0.17 | NaN | NaN |
| 2020-03-03 | 21.28 | 21.77 | 21.30 | 21.10 | 62690.06 | 0.22 | 1.04 | 21.146 | 21.598 | 21.384 | 70787.22 | 83961.03 | 71948.36 | NaN | -0.007918 | -0.17 | NaN | NaN |
| 2020-03-02 | 20.52 | 21.10 | 21.08 | 20.52 | 51902.65 | 0.67 | 3.28 | 21.240 | 21.644 | 21.318 | 80082.14 | 84316.46 | 74674.49 | 21.39 | -0.010329 | -0.22 | NaN | NaN |



接下来获取5日，10日，20日ma值以及close值

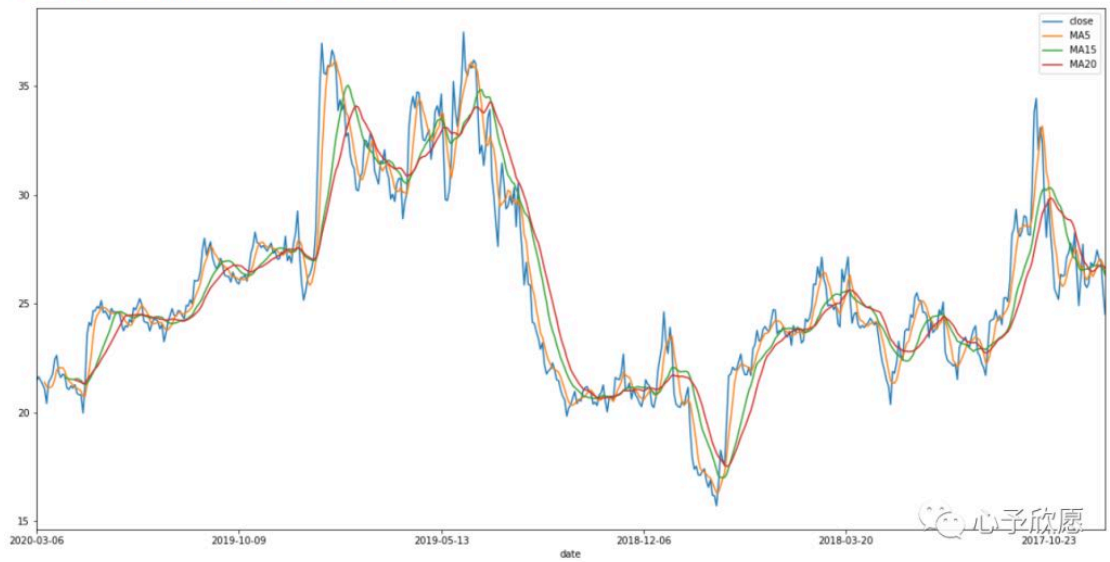
```
days = [5,15,20]
for ma in days:
    column_name = "MA{}".format(ma)
```

```
    df[column_name] = df['close'].rolling(ma).mean()
    df["pchange"] = df.close.pct_change()
    df["change"] = df.close.diff()
```

接下来可以用matplotlib画图啦

```
df[["close","MA5","MA15","MA20"]].plot( kind = 'line',figsize=(20, 10))
```

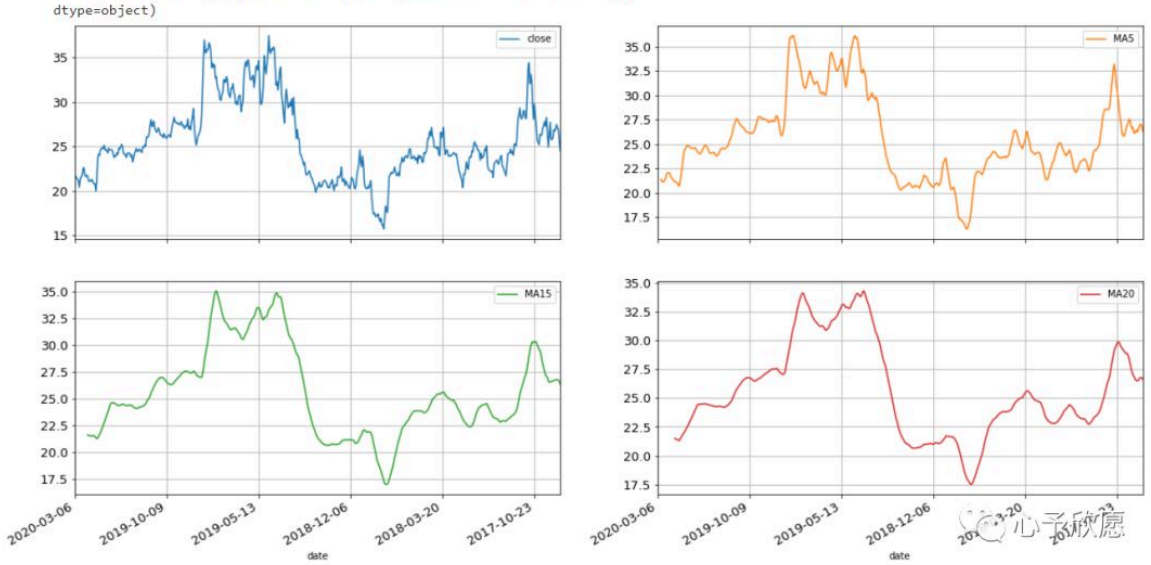
```
[51]: <matplotlib.axes._subplots.AxesSubplot at 0x1d2c1074358>
```




当然也可以分别画四张图

```
df[["close","MA5","MA15","MA20"]].plot(subplots=True, layout=(2,2), figsize=(20, 1
```

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x000001D2C1019DA0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001D2C0FB2FD0>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x000001D2C09FEFD0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001D2C10C1CC0>]],
      dtype=object)
```



不知道你是否有这样的经历：当一个股票涨上去的时候，你感觉到这个股票之后还可能涨，然后就及时买入，结果发现接连绿了几天，自己后悔不已。但是，其实如果让你再选择一次，你可能仍然拿不准自己是不是应该在那个时点进行买入，因为你缺少一个可以定量的指标策略。

下面介绍的3%策略和金叉死叉策略就是给投资者买卖股票提供的策略指标，其中金叉死叉更是被很多人在实战中使用。它既没有金融工程模型的得到的量化指标那么复杂，同时又可以给投资者一个买卖时点的考量。

## 2. 3%策略

接下来介绍3%策略，在股票单日涨幅超过3%的时候买入，跌幅超过3%的时候卖出，代码如下：

首先选定一支股票，并设定回撤值、突破值、账户资金、持有仓位手数、策略执行天数。案例中获取的股票数据是从2017-09-06开始获取的。

```
df = ts.get_hist_data('600848')
# 设定回撤值
withdraw = 0.03
# 设定突破值
breakthrough = 0.03
# 设定账户资金
account = 10000
# 持有仓位手数
position = 0
# 策略执行的天数
days = 20
```

定义买卖计算方式

```
def buy(bar):
    global account , position
    print("{}: buy {}".format(bar.date, bar.close))
    # 一手价格
    one = bar.close * 100
    position = account // one
    account = account - (position * one)

def sell(bar):
    global account , position
    # 一手价格
    print("{}: sell {}".format(bar.date, bar.close))
    one = bar.close * 100
    account = account + position * one
    position = 0
```

## 知识点1 global函数与全局变量

> 变量分为局部与全局，局部变量又可称之为内部变量。由某对象或某个函数所创建的变量通常都是局部变量，只能被内部引用，而无法被其它对象或函数引用。

举个例子，如果我们没有用global函数会出现以下报错，意思是说c为局部变量，在使用它之前没有被赋值。在函数内无法直接使用全局变量，所以我们要用global函数变为全局变量。

```
c = 1 # 全局变量
def add():
    c = c + 2 # 将 c 增加 2
    print(c)
add()
```

```
---------------------------------------------------------------------------
UnboundLocalError                         Traceback (most recent call last)
<ipython-input-13-95c9f4e4f027> in <module>
      3     c = c + 2 # 将 c 增加 2
      4     print(c)
----> 5 add()

<ipython-input-13-95c9f4e4f027> in add()
      1 c = 1 # 全局变量
      2 def add():
----> 3     c = c + 2 # 将 c 增加 2
      4     print(c)
      5 add()

UnboundLocalError: local variable 'c' referenced before assignment
```

正确操作：

[14]:
```
c = 1 # 全局变量
def add():
    global c
    c = c + 2 # 将 c 增加 2
    print(c)
add()
```

```
3
```

时间按升序调整，确定开始投资时间，写for循环和条件语句，得出结果。

```
# 调整到升序
data = df.sort_values(by='date').reset_index()
# data = data.iloc[?:?,:]

print('开始时间投资时间: ', data.iloc[0].date)
for i in range(days):
    bar = data.iloc[i, :]
    if bar.p_change > breakthrough and position == 0:
        buy(bar)
    elif bar.p_change < withdraw and position > 0:
```

```
        sell(bar)

print("最终可有现金: ", account)
print("最终持有市值: ", position * df.iloc[-1].close * 100)
```

```
开始时间投资时间:  2017-09-11
2017-09-11: buy 27.09
2017-09-13: sell 26.95
2017-09-15: buy 26.89
2017-09-18: sell 25.95
2017-09-20: buy 25.89
2017-09-22: sell 26.18
2017-09-26: buy 26.52
2017-09-28: sell 27.28
2017-09-29: buy 27.79
2017-10-09: sell 27.35
2017-10-13: buy 26.35
最终可有现金:  1954.0
最终持有市值:  8127.0
```

心予欣愿

## 知识点2 return在if语句中的用法

> 注意，写if语句时return必须在函数内使用。
> return的意义在于函数处理完逻辑之后，能够对外传出一个值。

错误示范：

```
for i in range(10):
    if i == 2:
        return 10
```

```
  File "<ipython-input-29-0087d3e860f2>", line 3
    return 10
    ^
SyntaxError: 'return' outside function
```

心予欣愿

正确用法：

```
def fun(a):
    return a**2
```

```
b = fun(2)
```

```
b
```

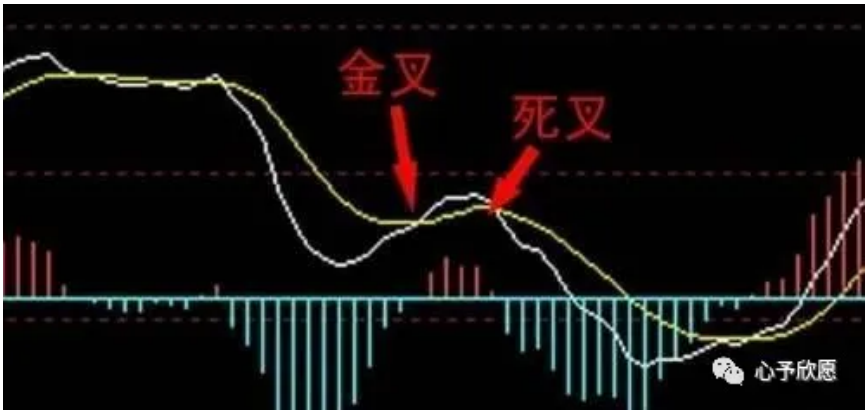
```
4
```

心予欣愿

## 3. 金叉死叉交易策略

首先介绍两个概念——金叉和死叉

> 金叉 ： 当较短周期的均线从长期均线的下方向上穿越较长周期的均线,形成的交点就是大家常说的金叉，为买入信号。
> 死叉 ： 当较短周期的均线从长期均线的上方方向下穿越较长周期的均线,形成的交点就是大家常说的死叉，为卖出信号。




（注：上图白色线为较短周期均线，黄色为较长周期均线）

## MACD策略

MACD（Moving Average Convergence and Divergence)是Geral Appel 于1979年提出的，利用收盘价的短期（常用为12日）指数移动平均线与长期（常用为26日）指数移动平均线之间的聚合与分离状况，对买进、卖出时机作出研判的技术指标。

由于MACD方法更复杂，所以我们用一种和MACD思想一样的方法，即金叉死叉的交易策略。金叉死叉方法给投资者一个交易时点的量化指标参考，可以初步判断买卖时机。

了解完金叉死叉的概念，那么假如我们选一支股票并从Tushare获取一定时期的股票数据，在金叉全部买入、死叉全部卖出，用这种策略进行投资的盈亏状况如何？

接下来正式介绍代码部分！
还是先导入数据包

```
import matplotlib.pyplot as plt
from pylab import mpl
import pandas as pd
import tushare as ts
mpl.rcParams['font.sans-serif']=['SimHei']
mpl.rcParams['axes.unicode_minus']=False
```

首先，我们要选定一支股票，以600000为例，即code = '600000'，然后按照日期进行升序排列。这里短周期和长周期分别取5日和20日均线，当然也可以算一下其他天数的均线。

```
df = ts.get_hist_data(code)
# 按日期升序
df = df.sort_values('date')
# 加两列，尽管有，我们要自己算，知道逻辑
```

```
df['ma5']=df['open'].rolling(5).mean()
df['ma20']=df['open'].rolling(20).mean()
```

要将前19行的数据去掉，因为ma在第20天才有值，不然会影响结果的！然后构建ma5和ma20大于小于的序列，而且这里用的是布尔值的series。用布尔形式来给pandas.Series取值的时候，会将True的那一行留下来，False的不显示。这样做的好处是方便我们判断是金叉还是死叉，否则还要写for循环。

```
#ma20从第20天才有值
df = df.iloc[19:, :]
#构建ma5和ma20大于小于的序列
sr1 = df['ma5'] < df['ma20'] # bool值的pandas.Series
sr2 = df['ma5'] >= df['ma20'] # bool值的pandas.Series
```

然后把第一天和第二天进行对比来判断金叉或死叉。以判断死叉为例，sr2.shift(1)把原本是今天的数据变成了昨天，将前后两天进行对位。和sr1取交集，出现True就代表着是死叉，这样可以避免每个日期都重复判断。写出死叉之后可以取反，将交集改并集，就可以得出金叉。

再将sr2通过shift(1)下移，在当前时间顺序的情况下，使得时间往后移了一天，这样就可以将前后两天进行比较。接下来由于shift(1)存在，所以会有NaN的干扰。要从第2个开始，提前的iloc。

```
# 接下来由于shift(1)存在，所以会有NaN的干扰。要从第2个开始，提前的iloc
df = df.iloc[1:,:]
# shift(1)在当前，时间顺序的情况下，使得时间往后移了一天，原本3.5号的值到了3.6号
death_cross = df[(sr1 & sr2.shift(1))[1:]].index #本日的ma5<ma20， 且昨天的ma5>=
golden_cross = df[(~(sr1 | sr2.shift(1)))[1:]].index #本日的ma5>=ma20， 且昨天的
```

## 知识点3 布尔值运算

1. “&”：与操作
2. “|” ： 或操作
3. “~” ： 取反操作

布尔值的与和或，同集合的交和并不是一个运算方法。用“1”表示Ture，“0”表示False，真值表如下：

| p | q | p∧q | p∨q |
|---|---|---|---|
| 1 | 1 | 1 | 1 |
| 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 |
| 0 | 0 | 0 | 0 |

例如：~(a|b) = (~a)&(~b)

现在来建立模型，假设本金为100000，其中money不参与计算，留作最后考评用。balance为余额。原始持有股票数为0。

```
money = 100000 #原始的钱，不参与计算，只是最后考评
balance = 100000
```

```
hold = 0#持有多少股
```

形成两个series是为了选出金叉和死叉点来直接交易，这样就不用每天都判断金叉死叉了

```
sr1 = pd.Series(1, index=golden_cross) # 全为1的金叉的Series， 日期对应了
sr2 = pd.Series(0, index=death_cross) #全为0的死叉的Series，日期对应了
#根据时间排序
sr = sr1.append(sr2).sort_index()
```

然后创建两个列表，用来画条形图

```
balist = []
srlist = []
```

前期准备做完之后就可以开始投资啦！
首先写个for循环，用来遍历数据；接着写if语句，出现金叉和死叉的时候依次算出买入数量、持有数量、余额及盈亏，就可以得到结果啦

```
#开始投资
for i in range(df.shape[0]):
    p = df['open'][i]
    date = df.index[i]
    srlist.append(date)
    if date in sr.index:
        if sr[date] == 1:
            #金叉
            buy = (balance // (100 * p))
            hold += buy*100 #hold = hold + buy*100
            balance -= buy*100*p
            balist.append(balance+hold*p-money)
            print('{}出现金叉，买入{}手，持有{}手，余额{}元，盈亏{}元'.format(dat
        else:
            print('{}出现死叉，卖出{}手，持有{}手，余额{}元，盈亏{}元'.format(dat
            balance += hold * p
            hold = 0
            balist.append(balance+hold*p-money)
    else:
        balist.append(balance+hold*p-money)
```

再用剩下的钱减去本金

```
p = df['open'][-1]
now_money = hold * p + balance
```

```
print('最终盈亏:', round(now_money - money, 1))
```

最后是这样的

```
2017-10-26出现死叉，卖出0手，持有0手，余额100000元，盈亏0.0元
2017-11-22出现金叉，买入76.0手，持有7600.0手，余额1200.0元，盈亏0.0元
2017-12-14出现死叉，卖出7600.0手，持有0手，余额1200.0元，盈亏-1596.0元
2018-01-09出现金叉，买入77.0手，持有7700.0手，余额768.0元，盈亏-1596.0元
2018-02-12出现死叉，卖出7700.0手，持有0手，余额768.0元，盈亏-1750.0元
2018-04-24出现金叉，买入84.0手，持有8400.0手，余额1146.0元，盈亏-1750.0元
2018-04-26出现死叉，卖出8400.0手，持有0手，余额1146.0元，盈亏-322.0元
2018-07-19出现金叉，买入103.0手，持有10300.0手，余额798.0元，盈亏-322.0元
2018-08-17出现死叉，卖出10300.0手，持有0手，余额798.0元，盈亏5446.0元
2018-08-20出现金叉，买入103.0手，持有10300.0手，余额695.0元，盈亏5446.0元
2018-09-12出现死叉，卖出10300.0手，持有0手，余额695.0元，盈亏3592.0元
2018-09-19出现金叉，买入100.0手，持有10000.0手，余额792.0元，盈亏3592.0元
2018-10-11出现死叉，卖出10000.0手，持有0手，余额792.0元，盈亏792.0元
2018-10-23出现金叉，买入94.0手，持有9400.0手，余额776.0元，盈亏792.0元
2018-11-15出现死叉，卖出9400.0手，持有0手，余额776.0元，盈亏1544.0元
2018-12-05出现金叉，买入92.0手，持有9200.0手，余额436.0元，盈亏1544.0元
2018-12-18出现死叉，卖出9200.0手，持有0手，余额436.0元，盈亏-1308.0元
2019-01-16出现金叉，买入97.0手，持有9700.0手，余额722.0元，盈亏-1308.0元
2019-03-15出现死叉，卖出9700.0手，持有0手，余额722.0元，盈亏11981.0元
2019-04-08出现金叉，买入94.0手，持有9400.0手，余额1155.0元，盈亏11981.0元
2019-04-29出现死叉，卖出9400.0手，持有0手，余额1155.0元，盈亏7845.0元
2019-05-09出现金叉，买入93.0手，持有9300.0手，余额1081.0元，盈亏7845.0元
2019-05-10出现死叉，卖出9300.0手，持有0手，余额1081.0元，盈亏5520.0元
2019-06-06出现金叉，买入91.0手，持有9100.0手，余额779.0元，盈亏5520.0元
2019-07-02出现死叉，卖出9100.0手，持有0手，余额779.0元，盈亏7431.0元
2019-07-26出现金叉，买入90.0手，持有9000.0手，余额961.0元，盈亏7431.0元
2019-08-06出现死叉，卖出9000.0手，持有0手，余额961.0元，盈亏861.0元
2019-08-27出现金叉，买入88.0手，持有8800.0手，余额805.0元，盈亏861.0元
2019-11-05出现死叉，卖出8800.0手，持有0手，余额805.0元，盈亏12917.0元
2019-12-17出现金叉，买入92.0手，持有9200.0手，余额953.0元，盈亏12917.0元
2020-01-16出现死叉，卖出9200.0手，持有0手，余额953.0元，盈亏13929.0元
2020-02-24出现金叉，买入101.0手，持有10100.0手，余额506.0元，盈亏[illegible]元
最终盈亏: 11606.0
```

## 知识点4 运算符号

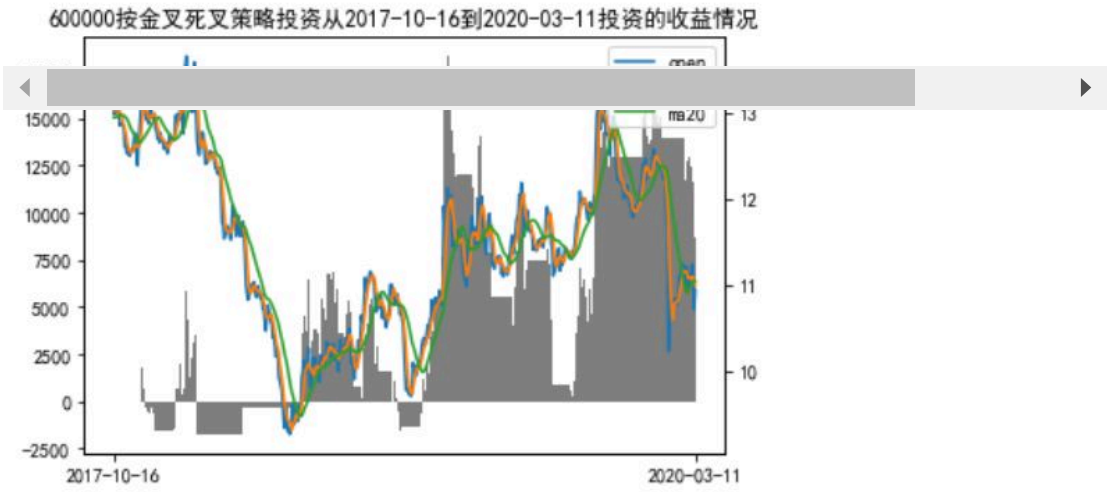
1. “//” ：整数除法。
2. “+=”：两个值相加，然后返回值给符号左侧的变量。也就是这里的hold += buy*100 相当于hold = hold + buy*100，“-=”同理。

最后一步，进行可视化！
由于交易价格达到10万元，而股票价格只有十几块，所以采用双坐标轴。
首先设定子图，先画条形图，这样就不会覆盖上面的线。twinx代表第二个坐标轴，legend显示图例。为了美观，设置了xticks，只选定了开始和结束日期，否则近3年的日期都要写在横轴上就会一团黑。最后设定图表名称

```
fig, ax1 = plt.subplots()
ax1.bar(srlist, balist, color='grey', width=1, label='net income')
ax2 = ax1.twinx()
```

```
ax2.plot(df[['open']], label='open')
ax2.plot(df[['ma5']], label='ma5')
ax2.plot(df[['ma20']], label='ma20')
plt.legend(loc= 'upper right')
plt.xticks([df.index[0],  df.index[-1]])
plt.title('{}按金叉死叉策略投资从{}到{}投资的收益情况'.format(code, df.index[0], d
```

```
[3]: Text(0.5, 1.0, '600000按金叉死叉策略投资从2017-10-16到2020-03-11投资的收益情况')
```




通过上图可以很直观的看出投资的收益情况~

本期的内容就到这里啦~用Python建立的股票分析模型不止这些，还有一些更好的方法等待我们探索。当然，这些模型有一些局限性，但是它可以当做我们股票投资策略的参考。心动不如行动，动动小手试一试吧！

学术指导：阿繁QVQ
本期撰写人：原禹欣
南京审计大学2018级CFA1班
邮箱：1033257046@qq.com