金融领域数据分析示例

1. 移动均线

移动均线(Moving Average，简称MA)是用统计分析的方法，将一定时期内的证券价格（指数）加以平均，并把不同时间的平均值连接起来，形成一根MA，用以观察证券价格变动趋势的一种技术指标。移动平均线是由著名的美国投资专家Joseph E.Granville（葛兰碧，又译为格兰威尔）于20世纪中期提出来的。均线理论是当今应用最普遍的技术指标之一，它帮助交易者确认现有趋势、判断将出现的趋势、发现过度延生即将反转的趋势。

移动均线常用线有5天、10天、30天、60天、120天和240天的指标。其中，5天和10天的短期移动平均线，是短线操作的参照指标，称做日均线指标；30天和60天的是中期均线指标，称做季均线指标；120天、240天的是长期均线指标，称做年均线指标。

收盘价5日均线：从第五天开始，每天计算最近五天的收盘价的平均值所构成的一条线。

移动均线算法：

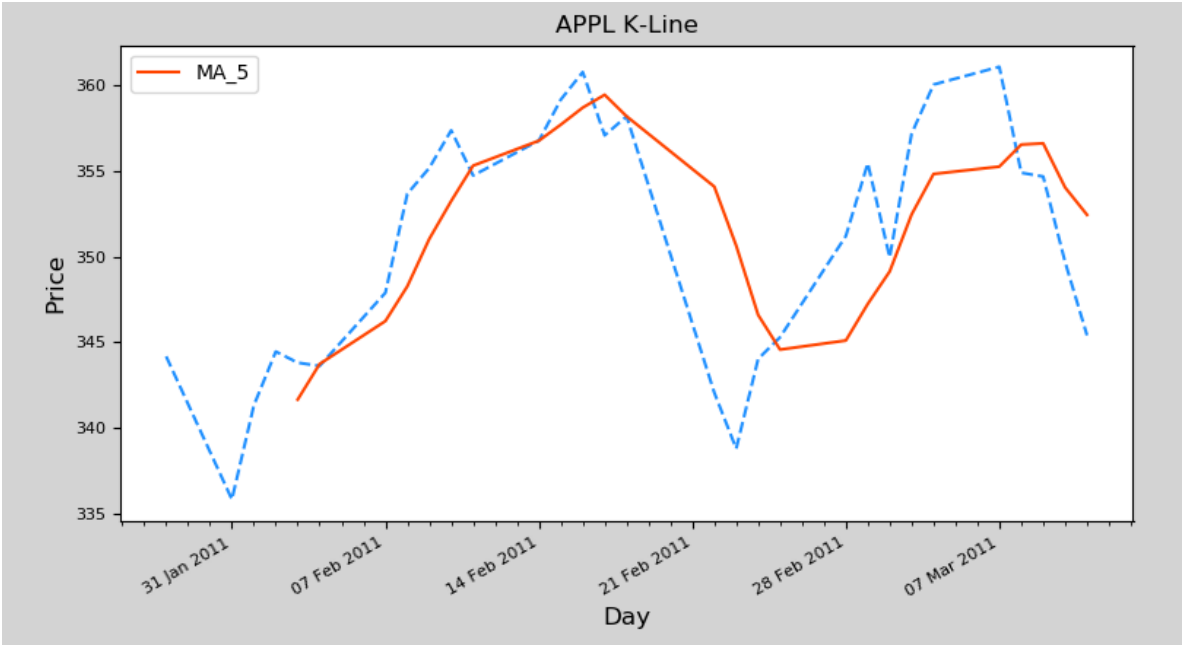
```
(a+b+c+d+e)/5
(b+c+d+e+f)/5
(c+d+e+f+g)/5
...
(f+g+h+i+j)/5
```

在K线图中绘制5日均线图

```
# 移动均线示例
import numpy as np
import datetime as dt


#### 1.读取数据
# 日期格式转换函数: 将日月年转换为年月日格式
def dmy2ymd(dmy):
    dmy = str(dmy, encoding="utf-8")
    # 从指定字符串返回一个日期时间对象
    dat = dt.datetime.strptime(dmy, "%d-%m-%Y").date()  # 字符串转日期
    tm = dat.strftime("%Y-%m-%d")  # 日期转字符串
    return tm


dates, open_prices, highest_prices, lowest_prices, close_prices = \
    np.loadtxt("../da_data/aapl.csv",  # 文件路径
               delimiter=",",  # 指定分隔符
               usecols=(1, 3, 4, 5, 6),  # 读取的列(下标从0开始)
               unpack=True,  # 拆分数据
               dtype="M8[D], f8, f8, f8, f8",  # 指定每一列的类型
               converters={1: dmy2ymd})  #

#### 2.绘制图像
import matplotlib.pyplot as mp
import matplotlib.dates as md

```

```
# 绘制k线图，x轴为日期
mp.figure("APPL K-Line", facecolor="lightgray")
mp.title("APPL K-Line")
mp.xlabel("Day", fontsize=12)
mp.ylabel("Price", fontsize=12)

# 获取坐标轴
ax = mp.gca()
# 设置主刻度定位器为周定位器(每周一显示刻度文本)
ax.xaxis.set_major_locator(md.WeekdayLocator(byweekday=md.MO))
ax.xaxis.set_major_formatter(md.DateFormatter("%d %b %Y"))  # %b表示月份简写
# 设置次刻度定位器为天定位器
ax.xaxis.set_minor_locator(md.DayLocator())
mp.tick_params(labelsize=8)
dates = dates.astype(md.datetime.datetime)

mp.plot(dates, open_prices, color="dodgerblue", linestyle="--")
mp.gcf().autofmt_xdate()  # 旋转、共享日期显示

# 绘制5日均值线
ma_5 = np.zeros(close_prices.size - 4)  # 均值数组
for i in range(ma_5.size):
    ma_5[i] = close_prices[i: i + 5].mean()  # 切片，求均值，并存入均值数组

mp.plot(dates[4:],  # 从第五天开始绘制
        ma_5,  # 数据
        color="orangered",
        label="MA_5")

mp.legend()
mp.show()
```

执行结果：



2. 布林带

1）什么是布林带

布林带（Bollinger Band）是美国股市分析家约翰·布林根据统计学中的标准差原理设计出来的一种非常实用的技术指标，它由三条线组成：

中轨：移动平均线（图中白色线条）

上轨：中轨+2x5日收盘价标准差　（图中黄色线条，顶部的压力）

下轨：中轨-2x5日收盘价标准差　（图中紫色线条，底部的支撑力）

布林带收窄代表稳定的趋势，布林带张开代表有较大的波动空间的趋势。

2）布林带的业务指标与含义

利用股价与布林带上轨线、下轨线进行比较，以及结合变化趋势，判断股票买入、卖出的时机。

(1)股价由下向上穿越下轨线（Down）时，可视为买进信号。
(2)股价由下向上穿越中轨时，股价将加速上扬，是加仓买进的信号。
(3)股价在中轨与上轨(UPER）之间波动运行时为多头市场，可持股观望。
(4)股价长时间在中轨与上轨（UPER）间运行后，由上向下跌破中轨为卖出信号。
(5)股价在中轨与下轨（Down）之间向下波动运行时为空头市场，此时投资者应持币观望。
(6)布林中轨经长期大幅下跌后转平，出现向上的拐点，且股价在2～3日内均在中轨之上。此时，若股价回调，其回档低点往往是适量低吸的中短线切入点。
(7)对于在布林中轨与上轨之间运作的强势股，不妨以回抽中轨作为低吸买点，并以中轨作为其重要的止盈、止损线。
(8)飚升股往往股价会短期冲出布林线上轨运行，一旦冲出上轨过多，而成交量又无法持续放出，注意短线高抛了结，如果由上轨外回落跌破上轨，此时也是一个卖点。

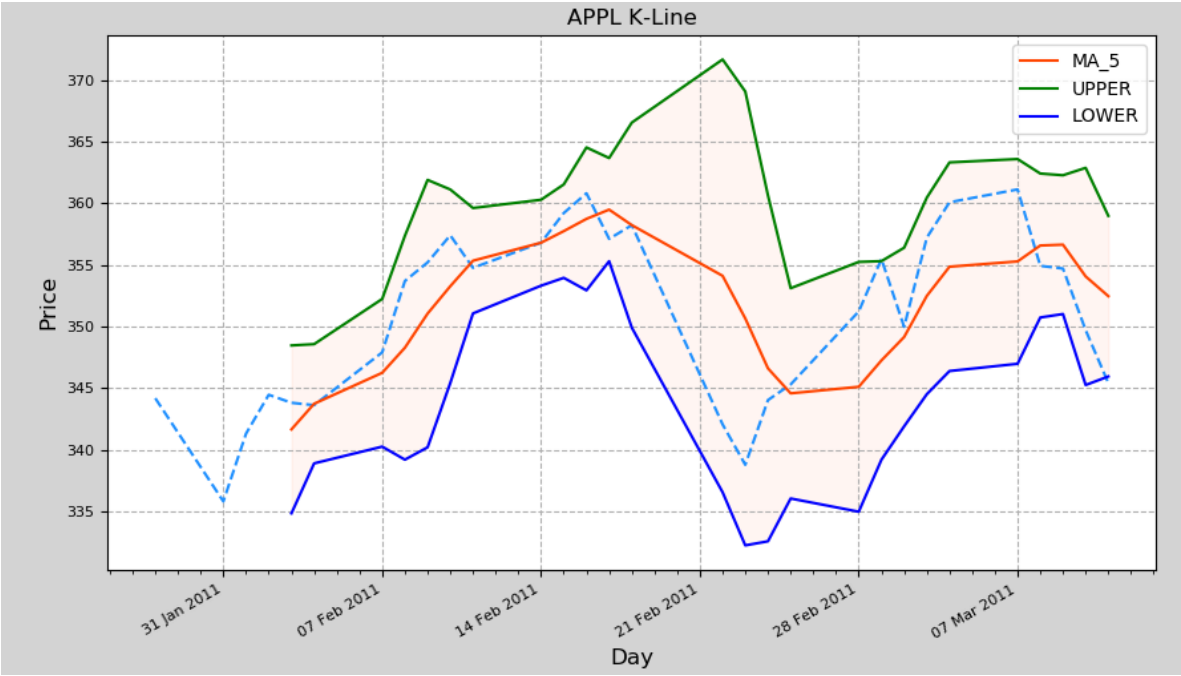
3）绘制布林带

以下是一个绘制布林带的示例。

```
# 布林带示例
import numpy as np
import datetime as dt

#### 1.读取数据
# 日期格式转换函数：将日月年转换为年月日格式
def dmy2ymd(dmy):
    dmy = str(dmy, encoding="utf-8")
    # 从指定字符串返回一个日期时间对象
```

```
    dat = dt.datetime.strptime(dmy, "%d-%m-%Y").date()  # 字符串转日期
    tm = dat.strftime("%Y-%m-%d")  # 日期转字符串
    return tm

dates, open_prices, highest_prices, lowest_prices, close_prices = \
    np.loadtxt("../da_data/aapl.csv",  # 文件路径
               delimiter=",",  # 指定分隔符
               usecols=(1, 3, 4, 5, 6),  # 读取的列(下标从0开始)
               unpack=True,  # 拆分数据
               dtype="M8[D], f8, f8, f8, f8",  # 指定每一列的类型
               converters={1: dmy2ymd})  #

#### 2.绘制图像
import matplotlib.pyplot as mp
import matplotlib.dates as md

# 绘制k线图，x轴为日期
mp.figure("APPL K-Line", facecolor="lightgray")
mp.title("APPL K-Line")
mp.xlabel("Day", fontsize=12)
mp.ylabel("Price", fontsize=12)

# 获取坐标轴
ax = mp.gca()
# 设置主刻度定位器为周定位器(每周一显示刻度文本)
ax.xaxis.set_major_locator(md.WeekdayLocator(byweekday=md.MO))
ax.xaxis.set_major_formatter(md.DateFormatter("%d %b %Y"))  # %b表示月份简写
# 设置次刻度定位器为天定位器
ax.xaxis.set_minor_locator(md.DayLocator())
mp.tick_params(labelsize=8)
dates = dates.astype(md.datetime.datetime)

mp.plot(dates, open_prices, color="dodgerblue", linestyle="--")
mp.gcf().autofmt_xdate()  # 旋转、共享日期显示

# 绘制5日均值线
ma_5 = np.zeros(close_prices.size - 4)  # 均值数组
for i in range(ma_5.size):
    ma_5[i] = close_prices[i: i + 5].mean()  # 切片，求均值，并存入均值数组

mp.plot(dates[4:],  # 从第五天开始绘制
        ma_5,  # 数据
        color="orangered",
        label="MA_5")

# 计算上轨、下轨线
stds = np.zeros(ma_5.size)
for i in range(stds.size):
    stds[i] = close_prices[i: i + 5].std()  # 计算标准差
upper = ma_5 + 2 * stds  # 计算上轨
lower = ma_5 - 2 * stds  # 计算下轨
# 绘制线
mp.plot(dates[4:], upper, color="green", label="UPPER")
mp.plot(dates[4:], lower, color="blue", label="LOWER")
# 填充布林带
mp.fill_between(dates[4:], upper, lower, lower < upper, color="orangered",
alpha=0.05)

mp.grid(linestyle="--")
mp.legend()
mp.show()
```

执行结果：



3. 基于函数矢量化的股票回测模型

函数的矢量化指通过一个只能处理标量数据的函数得到一个可以处理矢量数据的函数，从而可以对大量数据进行矢量化计算。

numpy提供了vectorize函数，可以把处理标量的函数矢量化，返回的函数可以直接处理ndarray数组。

```
# vectorize函数矢量化示例
import math
import numpy as np


def func(x, y):
    return math.sqrt(x ** 2 + y ** 2)


# 标量计算
x, y = 3, 4
print(func(x, y))

# 矢量化计算
X = np.array([3, 6, 9])
Y = np.array([4, 8, 12])

vect_func = np.vectorize(func)  # 创建函数矢量化对象

print(vect_func(X, Y))
```

执行结果：

```
5.0
[ 5. 10. 15.]
```

numpy还提供了frompyfuc函数，也可以完成与vectorize相同的功能：

```
# 把foo转换成矢量函数
# 原型：frompyfunc(func, nin, nout)
# func: 要矢量化的函数
# nin:输入参数个数
# nout:返回值个数
fun = np.frompyfunc(foo, 2, 1)
fun(X, Y)
```

案例：定义一种投资策略，传入某日期，基于均线理论返回是否应该按收盘价购买，并通过历史数据判断这种策略是否值得实施。

```
import numpy as np
import matplotlib.pyplot as mp
import datetime as dt
import matplotlib.dates as md

# 日期格式转换函数：将日月年转换为年月日格式
def dmy2ymd(dmy):
    dmy = str(dmy, encoding="utf-8")
    # 从指定字符串返回一个日期时间对象
    dat = dt.datetime.strptime(dmy, "%Y/%m/%d").date()
    tm = dat.strftime("%Y-%m-%d")  # 格式化
    return tm

dates, opening_prices, highest_prices, lowest_prices, closing_prices, ma5, ma10= \
    np.loadtxt("../data/pfyh.csv", #文件路径
               delimiter=",", #指定分隔符
               usecols=(0,1,2,3,4,5,6), #读取的列(下标从0开始)
               unpack=True, #拆分数据
               dtype="M8[D], f8, f8, f8, f8, f8, f8") #
mp.plot(dates, closing_prices)
mp.show()
# 定义一种投资策略，传入日期，基于均线理论返回是否应该按收盘价购买 1:应买入  0:应持有现状 -1:应卖出
def profit(m8date):
    mma5 = ma5[dates < m8date]
    mma10 = ma10[dates < m8date]
    # 至少两天数据才可以进行预测
    if mma5.size < 2:
        return 0
    # 出现金叉，则建议买入
    if (mma5[-2] <= mma10[-2]) and (mma5[-1] >= mma10[-1]):
        return 1
    # 出现死叉，则建议卖出
    if (mma5[-2] >= mma10[-2]) and (mma5[-1] <= mma10[-1]):
        return -1
    return 0

# 矢量化投资函数
vec_func = np.vectorize(profit)
# 使用适量换函数计算收益
profits = vec_func(dates)
print(profits)

```

```
# 定义资产
assets = 1000000
stocks = 0
payment_price = 0
status = 0
for index, profit in enumerate(profits):
    current_price = closing_prices[index]
    # 如果是买入并且赔了的状态，若已经跌出5%，则强制卖出
    if status == 1:
        payment_assets = payment_price * stocks
        current_assets = current_price * stocks
        if (payment_assets > current_assets) and ((payment_assets-current_assets) > payment_assets *0.05):
            payment_price = current_price
            assets = assets + stocks * payment_price
            stocks = 0
            status = -1
            print('止损：dates:{}, curr price:{:.2f}, assets:{:.2f}, stocks:{:d}'.format(dates[index], current_price, assets, stocks))
    if (profit == 1) and (status != 1): # 买入
        payment_price = current_price
        stocks = int(assets / payment_price)
        assets = assets - stocks * payment_price
        status = 1
        print('买入：dates:{}, curr price:{:.2f}, assets:{:.2f}, stocks:{:d}'.format(dates[index], current_price, assets, stocks))
    if (profit == -1) and (status != -1): # 卖出
        payment_price = current_price
        assets = assets + stocks * payment_price
        stocks = 0
        status = -1
        print('卖出：dates:{}, curr price:{:.2f}, assets:{:.2f}, stocks:{:d}'.format(dates[index], current_price, assets, stocks))
    print('持有：dates:{}, curr price:{:.2f}, assets:{:.2f}, stocks:{:d}'.format(dates[index], current_price, assets, stocks))
```