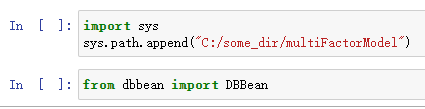
# 前置须知：python的系统路径设置（重要）

因为大部分代码都是自写的包，所以在使用时，无论是py文件，还是Jupyter Notebook中，都需要在最前面加两行代码。

假设包名叫做multiFactorModel（即所有代码放在一个名为multiFactorModel的文件夹下），所在路径为C盘下的some\_dir文件夹下，那么我们做的第一步是将这个路径加入到python的“系统路径”路径中，如下图第一个格子。

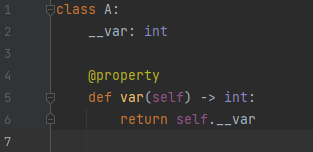


如果不运行第一个格子将包路径添加入系统路径，那么第二个格子import自写包dbbean就会失败。

# 前置须知：面向对象的程序设计细节

成员变量有三类：以字母开头的为public，以单下划线开头的为protected，以双下划线开头的为private

为了程序的安全，很多成员变量都设置为protected甚至private，如：实现了类A的对象aa，类A有成员变量\_\_var，在A的定义代码里，可以直接self.\_\_ var读取和修改\_\_ var，但是在外部，无法对对象aa使用aa.\_\_ var读取、修改\_\_ var，所以为了可以随时读取成员变量，有如下定义：



此时在外部，可以对对象aa使用aa.var读取aa中\_\_var的值（注意，property修饰的函数在被调用时不能加括号）。

这种结构在代码中经常出现，希望使用者们注意

# 与数据库交互

数据库交互采用的是经典的java bean思路。即所有与数据库直接交互的部分都有一个统计一接口解决。

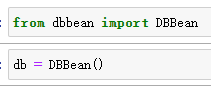
DBBean类：

成员：

|  |  |  |
| --- | --- | --- |
| 成员名 | 类型 | 作用 |
| \_\_host | str | 主机名 |
| \_\_port | str | 端口 |
| \_\_username | str | 用户名 |
| \_\_password | str | 密码 |
| \_\_db\_engine | str | 数据库引擎 |
| \_\_schema | str | Schema名 |
| \_\_conn | sqlalchemy.engine | 数据库连接对象 |

初始化：

一般来说，如下图：



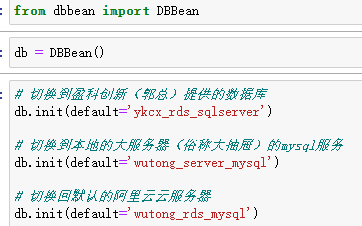
就会创建一个db，db是帮助我们连接到某个数据库Schema的专有连接。

看构造函数\_\_init\_\_的源代码：

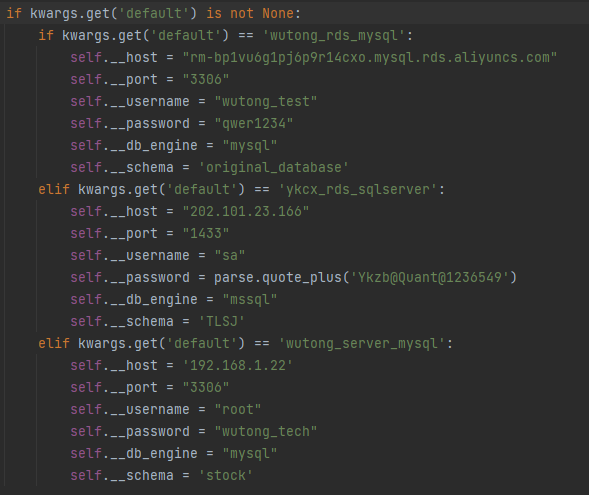


可以得知，db=DBBean()在不指定参数时，是用wutong\_test的身份登录阿里云云数据库中的original\_database这个schema中的连接，而schema时可以后天调整的，不需要在这一步就指定。

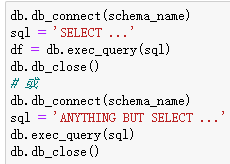
同时，为了方便使用者使用，尤其是要指定其他host、username、password的前提下，设置了一个简便的、不带下划线的init函数：



如果需要再加入其他数据库的连接，可以直接在下图的位置进行添加：



在正式使用时，使用模板可表示为下述形式：



schema\_name是字符串，即现在想要去连接的schema（需要先确认是不是在正确对应的host、username、password下）。

sql是sql语句，有select开头的查询语句和非查询语句（增删改）。

db.exec\_query(sql)会识别sql语句是增删改查的哪一种，如果是查询，则将查询结果返回为dataframe；如果是增删改，则将sql语句执行，返回一个空的dataframe。

最后db.db\_close()，是一个良好的习惯，否则如果一个大的系统开了过多的connect却没有及时关掉，可能会造成数据库的崩溃。

以上就是DBBean的核心功能。其他大部分模块都会直接引用DBBean处理数据库交互的部分，而不是直接在功能代码里直接引入sqlalchemy进行繁琐的配置。

而为了方便日更等其他功能，设置了一个保存表格功能。



# 数据的读取和基本操作

由于金融数据的特殊性，大部分的金融数据都是时序数据或截面数据，所以对每种数据都封装成了类，而这些类都继承了面板数据类Panel。

所有继承Panel的子类，同时继承了包括离群值处理、log大数、标准化、描述性统计等模块：也就是说这些子类不必再次编写相应的代码，即可继承这些功能。

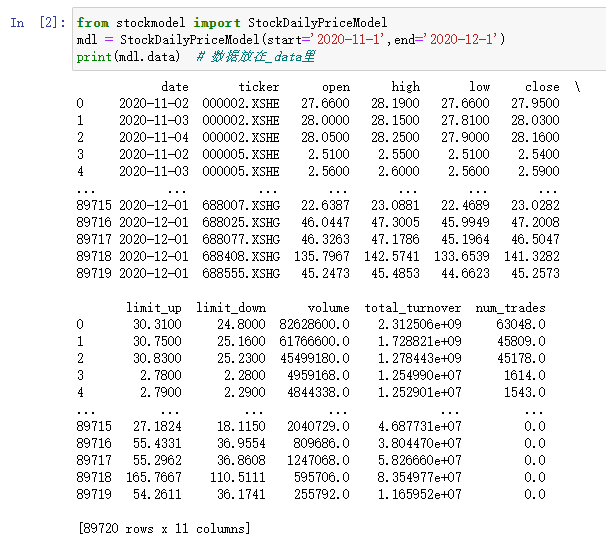
目前属于面板数据的类有StockDailyPriceModel、StockMinutePriceModel、StockFloatingSharesModel、StockTurnoverRateModel、SpecialTreatModel、MarketCapModel、IndustryModel、IndexModel、IndexComponentsWeightsModel、BarraModel、FactorModel

除了FactorModel，其他几种数据一般都会直接对应某张数据表，其功能大同小异，请直接参照代码的注释，在此只拿StockDailyPriceModel举例。

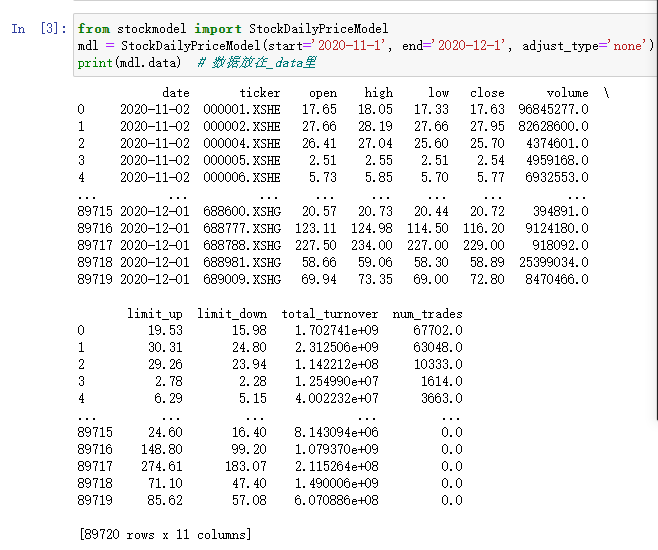
如下图，没有手动设置任何参数时，取出的数据是数据库里所有股票的所有日期的数据：



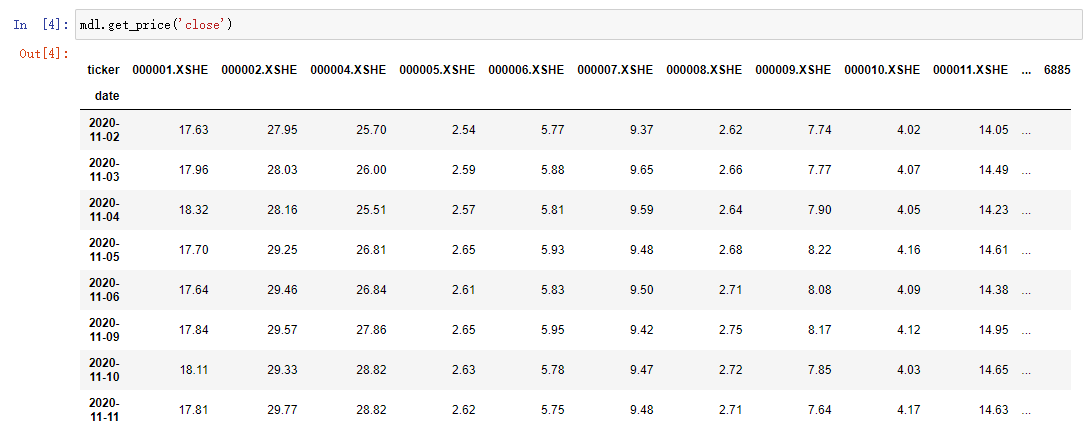
如下图，设置日期用start和end，如果不设start，默认从最前的数据开始；不设end，默认到最后的数据结束：



如下图，如果adjust\_type不设置，默认是前复权数据，可以设置为none（特别注意，不是python自带的空类型None，而是none四个字母组成的字符串），表示不复权。

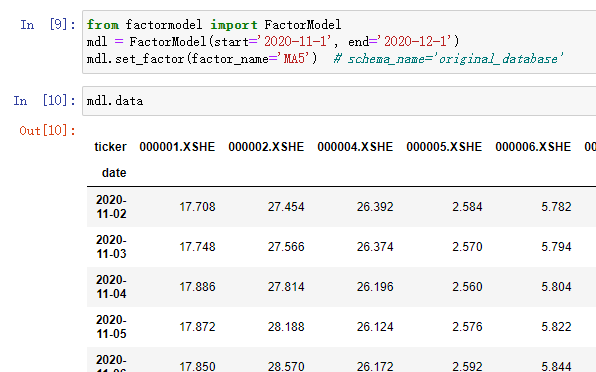


对于StockDailyPriceModel，有get\_price函数可以从mdl.data里快速获得某个特别值的矩阵面板形式：



至此，大部分表的基本读取功能已经介绍完毕。

下面介绍FactorModel的用法，如下图：



在实现对象mdl时，构造函数仅指定了开始日期和结束日期（可以不指定，如其他面板数据），在调用set\_factor函数后，才会根据factor\_name参数读取数据。

set\_factor函数内在读取逻辑是：

在cn\_factor\_info表中，判断MA5是否唯一存在

判断成立后，再从cn\_factor\_info中找到MA5所在的表，是cn\_technical\_ma

在cn\_technical\_ma中读取date、ticker、MA5三列

把三列的长标变成面板矩阵，放置到mdl的\_data中（用mdl.data读取）

所以，要读取一个因子，必须保证实现录入到cn\_factor\_info当中，且因子名不可重复。

应该注意到，set\_factor其实预留了schema\_name这个参数，默认为original\_database，而事实上，其他表中并没有规范的cn\_factor\_info表，所以FactorModel暂时不能用在其他schema中。

# 数据库日更

数据库日更有专门的notebook文件操作。

先阐述下无法做到完全自动化的理由：

网络问题

Windows自带更新问题

数据商服务崩溃问题

所以现阶段以现有人力无法部署一个全自动日更环境。

日更思路：

读取cn\_update\_info，每一行读取update\_date，即上次更新日期

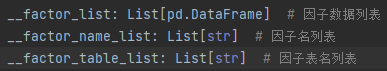
判断今天是否等于上次更新日期，如果等于，这一行所记录的表不更新

否则，将update\_date至今的所有数据补全并填入，同时将update\_date修改为今天

顺便，顺便把合成好的因子入库

# 因子合成逻辑

FactorComposition类包含三个与众不同的成员



我们将原始数据（价量、原始因子）直接合成的因子称为一级合成因子；

成分包含一级及以上的合成因子称为二级因子。

在合成二级因子时，都有一个参数middle\_is\_saved，默认否，如果是，则不会生成中间因子，而是直接从数据库调取因子进行合成；如果否，则先生成所有中间因子，再把所有中间因子的数据、因子名、欲存储的因子表名按顺序放入这三个列表中。

在这三个列表不为空时，调用save\_middle\_factor，会将所有中间因子保存；在调用任意一个gen开头的生成合成因子的函数后，调用save\_composition\_factor，将合成好的因子存入数据库。