因子研究框架

一、介绍

本因子研发框架与实盘环境不同，只是由于因子的挖掘等研究阶段，因此整个框架的开发并不直接使用 C++语言，而是基于 DolphinDB（之后简称为 DDB）的脚本语言。在此基础上，研发框架依赖于 DDB 的因子开发管理平台，将重复性的工作流程整合在这个研究框架内，使得量化研究员能够将精力专注于利用特征来挖掘因子本身。重复性的工作流程应包括数据导入，因子评价以及因子的回测等方面。项目的代码整理在 GitHub 上，见 git@github.com:wzrwisdom/DDB_factor_framework.git。在启动 DDB 引擎之前，可以将 GitHub 项目中“DDB_related”一级路径下 DDB 配置文件(“dolphindb.cfg”)以及 DDB 启动导入函数文件(“dolphindb.dos”)替换掉原有的文件。

二、插件部署

因子管理平台的使用需要进行安装插件，插件文件置于 Git 项目的一级目录中。安装步骤如下：

1) 将插件文件 DolphinDBFactorLab_alpha2.3.zip 解压后的文件放置到 DolphinDB 服务端对应的路径下 <DolphinDBInstallDir>/server/DolphinDBFactorLab。可运行下面指令

```
    unzip DolphinDBFactorLab_alpha2.3.zip -d
<DolphinDBInstallDir>/server/DolphinDBFactorLab
```

2) 将解压后的文件更新到对应目录（单节点情况）

```
    1. cd <DolphinDBInstallDir>/server/DolphinDBFactorLab/
```

```
2. cp -r ./web/* ../web/
3. cp -r ./modules/* ../modules/
4. cp -r ./factorLab/ ../
```

3) 脚本部署，指令需传入单节点端口，下面以端口为 8848 作为例子

```
sh updateSingle.sh 8848
```

4) 初始化，创建因子管理平台所需的各种库表，可在 GUI、VSCode 中执行

```
run(getHomeDir()+ '/factorLab/init/init.dos')
```

5) 访问因子管理平台的 Web 界面。在初始化后，通过在浏览器中输入网址 <ip>:<port>/factor-platform/index.html 进行登陆。登陆页面如下，用户名：admin，初始密码：123456

因子管理平台登陆后的界面如下图所示,总共包括 10 个模块，我们主要使用的模块将集中在 “因子库”、“因子评价”以及“数据导入”这三个模块中。

二、数据库表结构

在进行数据导入工作时，我们需要将服务器上关于逐笔数据以及快照数据对应的 csv 文件导入到 DDB 的库表中，这一步称为原始数据落库。由于因子挖掘中我们需要涉及到需要先得到一些特征值，由特征值构造因子的过程，部分特征值的使用频率较高，因此有必要先将原始数据进行整理，将使用频率高的特征值整理输出为新的库表。比如针对股票市场，我们会根据原始数据，聚合整理出不同时间频率的数据，频率可取 1s，1min 以及 5min 维度的数据，这些数据将按照时间和标的代码进行落库。

存放原始信息的表主要有 3 个，分别是原始逐笔委托信息，原始逐笔成交信息，原始 3s 快照信息，他们与 DDB 中的库和表名对应关系如表 一。此处的库表以及后面出现的库表结构和字段描述可以在‘因子研究框架 DDB 表结构.xlsx’文件中查询。

表 一

	数据库名	表名
原始逐笔委托	dfs://LEVEL2_Tick	OrderRaw
原始逐笔成交	dfs://LEVEL2_Tick	TradeRaw
原始 3s 快照	dfs://LEVEL2_Snapshot_ArrayVector	SnapRaw

关于聚合整理出来的 1min 股票数据，1min 股票额外数据和 5min 股票数据，他们与 DDB 中的库和表名对应关系如表 二。

表 二

	数据库名	表名
1min 股票数据	dfs://K_Minute_Level	OneMinute
5min 股票数据	dfs://K_Minute_Level	FiveMinute

以上介绍的库表只是目前阶段所考虑需要的，后续有需要可以进行添加!

三、数据导入

我们将在 DDB 因子管理平台上中的 ‘数据导入’模块中集中整理与数据的落库相关功能的函数。“数据导入”模块中包含“导入函数库”，“数据导入模版”两个子模块/在“导入函数库”中我们可以创建关于数据导入相关的函数库，里面存放着一些可复用的函数，比如库表创建，路径匹配等函数。“数据导入模版”中可以创建自定义的导入模版，它将调用函数库中相关函数来提交数据导入任务进行异步执行。

将原始数据对应的 csv 文件落库。首先我们需要将在“导入函数库”中创建相关的函数库，点击“新建函数库”按钮，名称设为“CSVDataModule”，具体代码实现存放在 GitHub 项目的“data_loader /loader_library/CSVDataModule.dos”文件中，只需将其复制粘贴到新建好的函数库中并保存。

下面我们将在“数据导入模板”中创建一系列的导入模板，如 loadEntrust、loadTrade、loadSnapshot。创建方式相同，在“数据导入模板”子模块下点击“新建模版”，名字按照上面的填写。相关的代码实现存放在 GitHub 项目的“http://data_loader/data_load_template/{loadEntrust/loadTrade/loadSnapshot}.dos”文件中。这些导入模版的功能是将存放在服务器上的原始逐笔委托，逐笔成交，3s 快照的 csv 数据落入 DDB 数据库中。对于需要导入的逐笔委托 csv 文件的命名需满足“order_yyyyMMdd”的格式。另外，逐笔成交 csv 文件的命名需满足“execution_yyyyMMdd”的格式，3s 快照 csv 文件的命名需满足“snapshot_yyyyMMdd”。

具体数据导入通过选择对应的导入模板，点击运行并配置模版所需参数，最后点击确定即可。下面以导入逐笔成交信息为例，我们选择“loadTrade”模板，相关的参数配置如下表所示，其中也包括了配置参数的含义和类型。

参数名称	数据类型	数据样例	参数含义
rootPath	String	/data3/csv_data	csv 文件所在路径
dbName	String	dfs://LEVEL2_Tick	用于数据输出的库名
tbName	String	TradeRaw	用于数据输出的表名
startDate	Date	2023-04-20	由于 csv 文件名包含日期信息。需要导入数据的起始日期
endDate	Date	2023-04-20	需要导入数据的截止日期
threadNum	Int	1	提交导入任务的所需线程数
initDb	Boolean	False	是否将库初始化
initTb	Boolean	False	是否将表初始化

运行逐笔委托数据的数据导入模板“loadEntrust”的配置参数如下：

参数名称	数据类型	数据样例
rootPath	String	/data3/csv_data
dbName	String	dfs://LEVEL2_Tick
tbName	String	OrderRaw
startDate	Date	2023-04-20
endDate	Date	2023-04-20
threadNum	Int	1
initDb	Boolean	False
initTb	Boolean	False

运行 3s 快照数据的数据导入模板“loadSnapshot”的配置参数如下：

参数名称	数据类型	数据样例
rootPath	String	/data3/csv_data
dbName	String	dfs://LEVEL2_Snapshot_ArrayVector
tbName	String	SnapRaw
startDate	Date	2023-04-20
endDate	Date	2023-04-20
threadNum	Int	1
initDb	Boolean	False
initTb	Boolean	False

至此我们已经将原始数据对应的 csv 文件全部导入到 DDB 的库表中了。

四、构建特征，因子或者算子

在量化研究领域，通常包括了特征，因子以及算子三个方面。

特征：一般是用来描述某个金融资产的属性或者特点的数据指标。特征可以是基本面数据（如收益、市盈率、市值等）也可以是内含价量关系的技术指标。在量化研究中，选择合适的特征是构建量化模型的第一步，好的特征能够捕捉到市场的规律和变化。

因子：一种用来解释金融资产收益的变量。因子通常是基于一系列特征的组合构建而成。通过因子分析，可以发现资产收益与这些因子之间的相关性，进而用于构建量化模型。

算子：是在因子挖掘过程中，我们对于一些常用运算过程的提炼。

在 DDB 的因子管理平台上，特征的提取我们将使用上一节中介绍到的“数据导入”模块来实现。对于因子挖掘，这方面的功能实现依赖于“因子库”模块。而算子的本质是一些常用函数，我们将这部分保存在 DDB 软件启动会加载的 dolphindb.dos 文件中，这样在 DDB 平台启动后，无论是在 dos 文件中还是因子平台中我们都可以方便地使用这些算子。后续会用到的算子放在 GitHub 项目中的“DDB_related/dolphindb.dos”文件中，在启动 DolphinDb 前先将文件中的内容覆盖原有的 dolphindb.dos 文件。

1. 特征构建及落库

前面提到我们将依赖因子管理平台的“数据导入”模块来实现 。与 csv 文件落库相同，依然是在“数据导入”模块的“导入函数库”子模块中我们新建一个名称为“ExtractFeatureModule”的函数库，将特征的计算实现过程整理在函数库中，具体的代码实现见 GitHub 项目中的“data_loader/loader_library/ExtractFeatureModule.dos”文件。

随后在“数据导入模板”子模块下创建新的模板来调用“ExtractFeatureModule”函数库。需要创建的模板共有四个，分别是 createCancelOrderCrossSec，createEntrustCrossSec，createTradeCrossSec 以及 createSnapCrossSec。具体代码实现见 GitHub 项目中的“data_loader/data_load_template/{createCancelOrderCrossSec/createEntrustCrossSec/createTradeCrossSec/createSnapCrossSec}.dos”文件。

创建好模板后，选择对应的导入模板，点击运行并配置模版所需参数，最后点击确定即可。对于提取撤单相关信息的 createCancelOrderCrossSec 模板，其配置参数见下表。

参数名称	数据类型	数据样例	参数含义

dbName	String	dfs://CROSS_SECTION_1sec	输出库名
tbName	String	cancelOrder_info	输出表名
startDate	Date	2023-04-20	处理数据的起始日期
endDate	Date	2023-04-20	处理数据的截止日期
window_size	String	1s	滚动时间窗口大小
threadNum	Int	1	线程数目
fromDbNameStr	String	{'entrust':'dfs:l/LEVEL2_Tick', 'trade': 'dfs://LEVEL2_Tick'}	委托和成交的库名
fromTbNameStr	String	{'entrust':'OrderRaw','trade': 'TradeRaw'}	委托和成交的表名
initDb	Boolean	false	是否覆盖输出库
initTb	Boolean	false	是否覆盖输出表

点击“确定”按钮运行后，输出的库表数据如下图所示，其中展示了部分关于撤单的特征数据。

	security_code	trade_time	cancel_buy_num	cancel_sell_num	cancel_buy_vol	cancel_sell_vol	cancel_buy_money	cancel_sell_money	cancel_buy_time_range	cancel_sell_time_range
2	000014	2023.04.20T09:30:06.000	4	2	10500	1000	89172	9350	562840	210
3	000014	2023.04.20T09:30:12.000	1	0	1000	0	9350	0	0	
4	000014	2023.04.20T09:30:18.000	0	7	0	3000	0	28080		3910
5	000014	2023.04.20T09:30:24.000	1	6	9000	9900	81900	92597	0	3540

对于 createEntrustCrossSec 模板，其配置参数见下表。

参数名称	数据类型	数据样例
dbName	String	dfs://CROSS_SECTION_1sec
tbName	String	entrust_info
startDate	Date	2023-04-20
endDate	Date	2023-04-20

window_size	String	1s
threadNum	Int	1
entrustDbName	String	dfs:l/LEVEL2_Tick
entrustTbName	String	OrderRaw
initDb	Boolean	false
initTb	Boolean	false

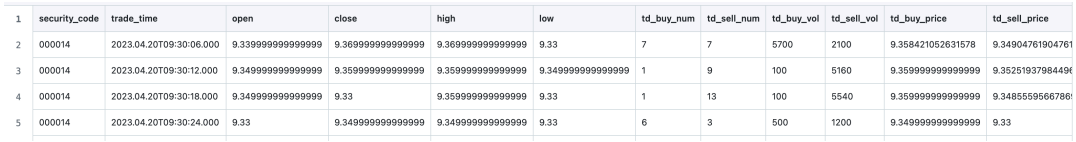
运行后的输出如下：

	security_code	trade_time	en_buy_num	en_sell_num	en_buy_vol	en_sell_vol	en_buy_price	en_sell_price	en_buy_price_std	en_sell_price_std	en_price_std
2	000014	2023.04.20T09:30:06.000	39	33	69700	54700	9.207288837876614	9.666051188299817	0.153141639254039	0.293787312266008	0.280361939109802
3	000014	2023.04.20T09:30:12.000	3	21	2400	11560	9.3475	9.39022491349481	0.017320508075688	0.205968097390973	0.192918094807999
4	000014	2023.04.20T09:30:18.000	2	33	700	14200	9.198571428571428	9.353098591549295	0.141421356237309	0.0082686888657895	0.031826736817817
5	000014	2023.04.20T09:30:24.000	1	7	500	13500	9.349999999999999	9.343925925925926		0.01	0.009910312089651
6	000014	2023.04.20T09:30:30.000	0	6	0	4900		9.329183673469387		0.017224014243685	0.017224014243685
7	000014	2023.04.20T09:30:36.000	4	6	6200	3300	9.245161290322581	9.412121212121212	0.07675719293113	0.059329587896765	0.070435155363718

对于 createTradeCrossSec 模板，其配置参数见下表。

参数名称	数据类型	数据样例
dbName	String	dfs://CROSS_SECTION_1sec
tbName	String	cancelOrder_info
startDate	Date	2023-04-20
endDate	Date	2023-04-20
window_size	String	1s
threadNum	Int	1
fromDbNameStr	String	{'entrust':'dfs:l/LEVEL2_Tick', 'trade': 'dfs://LEVEL2_Tick'}
fromTbNameStr	String	{'entrust':'OrderRaw','trade': 'TradeRaw'}
initDb	Boolean	false
initTb	Boolean	false

运行后的输出如下：

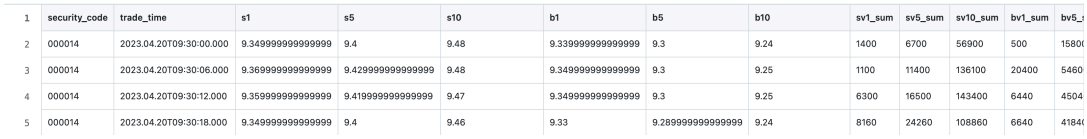


	security_code	trade_time	open	close	high	low	td_buy_num	td_sell_num	td_buy_vol	td_sell_vol	td_buy_price	td_sell_price
2	000014	2023.04.20T09:30:06.000	9.339999999999999	9.369999999999999	9.369999999999999	9.33	7	7	5700	2100	9.358421052631578	9.34904761904761
3	000014	2023.04.20T09:30:12.000	9.349999999999999	9.359999999999999	9.359999999999999	9.349999999999999	1	9	100	5160	9.359999999999999	9.35251937984496
4	000014	2023.04.20T09:30:18.000	9.349999999999999	9.33	9.359999999999999	9.33	1	13	100	5540	9.359999999999999	9.34855595667869
5	000014	2023.04.20T09:30:24.000	9.33	9.349999999999999	9.349999999999999	9.33	6	3	500	1200	9.349999999999999	9.33

对于 createSnapshotCrossSec 模板，其配置参数见下表。

参数名称	数据类型	数据样例
dbName	String	dfs://CROSS_SECTION_1sec
tbName	String	snap_info
startDate	Date	2023-04-20
endDate	Date	2023-04-20
window_size	String	1s
threadNum	Int	1
fromDbNameStr	String	dfs://LEVEL2_Snapshot_ArrayVector
fromTbNameStr	String	SnapRaw
initDb	Boolean	false
initTb	Boolean	false

运行后的输出如下：



	security_code	trade_time	s1	s5	s10	b1	b5	b10	sv1_sum	sv5_sum	sv10_sum	bv1_sum	bv5_s
2	000014	2023.04.20T09:30:00.000	9.349999999999999	9.4	9.48	9.339999999999999	9.3	9.24	1400	6700	56900	500	1580
3	000014	2023.04.20T09:30:06.000	9.369999999999999	9.429999999999999	9.48	9.349999999999999	9.3	9.25	1100	11400	136100	20400	5460
4	000014	2023.04.20T09:30:12.000	9.359999999999999	9.419999999999999	9.47	9.349999999999999	9.3	9.25	6300	16500	143400	6440	4504
5	000014	2023.04.20T09:30:18.000	9.349999999999999	9.4	9.46	9.33	9.289999999999999	9.24	8160	24260	108860	6640	4184

关于上面这些模板构造的这些特征的描述，我们整理在 GitHub 项目中的 data_loader 二级路径下的 README.md 中。

2. 因子挖掘及落库

因子挖掘相关工作在 DDB 因子管理平台上中的 ‘因子库’ 模块完成。其中包括“我的草稿”，“公开因子模块”以及“因子计算模板”子模块。

因子的计算逻辑在“我的草稿”子模块中书写，而具体调用因子计算逻辑完成实际计算任务以及落库需要在“因子计算模板”中完成。

下面以书写 “动量&反转”因子为例，首先在“我的草稿”子模块中点击“新建因子模块草稿”，因子模块名称为“Momentum_Revert”，属性可选填为高频和面板。具体代码见 GitHub 项目的“factor/factor_draft /mmt_revert.dos”文件，将其复制粘贴在草稿中即可。随后在“因子计算模板”中点击“新建因子计算模板”，名称设置为“secFreq_panel_call”，属性设置为“面板”。具体代码见 GitHub 项目的“factor/factor_cal_template/secFreq_panel_call.dos”文件，将其复制粘贴过来即可。

关于运行草稿中写好的因子，只需在“我的草稿”子模块中选中需要运行的因子模块，点击“测试”按钮，在“选择因子”处可选择单个或多个因子。“选择计算模板 1”中选中刚刚创建好的“secFreq_panel_call”计算模板，之后需要对选择的计算模板进行参数配置，配置与其文字描述如下表。配置好参数后，点击“确定”运行。

参数名称	数据类型	数据样例	参数含义

dbName	String	dfs://CROSS_SECTION_1sec	输入库名
tbName	String	trade_info	输入表名
startDate	Date	2023-04-20	处理数据的起始日期
endDate	Date	2023-04-20	处理数据的结束日期
factorName	因子参数入参		因子参数
securitidName	String	security_code	输入库表中股票代码对应列名
TradetimeName	String	trade_time	输入库表中交易时间对应列名

挖掘的因子不仅会使用构建好的特征，还会使用之前已经构建好的其他因子作为输入。对于这类因子的计算，“secFreq_panel_call”计算模板不再满足我们的需求，为此我们需要新的因子计算模板。还是以“动量&反转”因子为例，我们构建了一个“close_ret_1m_rank”因子，意义是获取股票当前收益率在过去某一时间段收益率的排名，其中收益率是“factor/factor_draft /mmt_revert.dos”中已经计算好的因子“close_ret”。我们新创建了因子模块“Mmt_Rev_TimeSeries”用于存放此类因子，具体代码见 GitHub 项目的“factor/factor_draft / mmt_revert_TimeSeries.dos”文件。对于这类需要特征以及其他因子作为参数输入的因子，需要创建一个新的“secFreq_feat_factor_sql”计算模板，将 GitHub 项目中“factor/factor_cal_template/secFreq_feat_factor_sql.dos”文件内容复制过来即可。

相应的因子计算运行过程一样，在“我的草稿”子模块中选中需要运行的因子模块，点击“测试”按钮，在“选择因子”处可选择单个或多个因子。“选择计算模板 1”中选中刚刚创建好的“secFreq_feat_factor_sql”计算模板，之后需要对选择的计算模板进行参数配置，配置与其文字描述如下表。配置好参数后，点击“确定”运行。

参数名称	数 据 类型	数据样例	参数含义
fromDbName	String	{'trade':'dfs://CROSS_SECTION_1sec','secFac':'dfs://secondLevelDB' }	输入库名
fromTbName	String	{'trade':'trade_info','secFac':'freq_1sec' }	输入表名
featureStr	String	{'trade':['td_buy_price','td_sell_price','td_vol'],'secFac':{ 'Momentum_Revert': ['close_ret']} }	处理数据的起始日期
startDate	Date	2023-04-20	处理数据的起始日期
endDate	Date	2023-04-20	处理数据的结束日期
factorName	因子参数入参		因子参数
oriName	String	tradetime,close_ret,td_vol,td_buy_price,td_sell_price	因子函数中所需参数名称
tarName	String	trade_time,close_ret,td_vol,td_buy_price,td_sell_price	库表中上述参数对应列名

securitidName	String	security_code	输入库表中股票代码对应列名
TradetimeName	String	trade_time	输入库表中交易时间对应列名

五、因子评价

在做好因子构建的相关工作后，如何评价因子的好坏需要一些指标来检验，这一步我们称为因子评价。在 DDB 的因子管理平台上，相关功能实现集中在“因子评价”模板中，其中包括“评价函数库”、“评价模板”以及“评价记录”三个子模块。

与“数据导入”模块类似，我们在“评价函数库”子模块中将一些评价过程中可以复用的函数整理形成函数库。在“评价模板”中创建调用评价函数库中函数进行评价的模板，并且可以为模板配置可视化视图。

GitHub 项目中的“factor_evaluate/evaluate_library/secFreq_eva_bt_lib.dos”文件是一个已经实现计算因子 IC 和 IR 指标，以及多个因子之间相关性，以及计算因子分组回测的 pnl 曲线功能的评价函数库。只需在“评价函数库”子模块下点击“新建函

数库”，名称设置为“secFreq_eva_bt_lib”后进行创建。然后将 GitHub 下的文件中代码复制到创建好的函数库中。之后我们在“评价模板”中新建一个名为“secFreq_evaluate”的模板，将 GitHub 项目中的“factor_evaluate/evaluate_template/secFreq_evaluate.dos”文件里的代码复制到创建好的模板中。

DDB 的因子管理平台对于因子评价的功能针对的是单个因子，所以进行因子评价时需要先选定一个之前已经计算的因子进行后续评价，即进入“因子库-我的草稿”子模块中然后点击某一个因子的评价记录数（见下图红色方框）。

页面跳转之后，点击“生成评价报告”按钮，其中“评价模板”选择“secFreq_evaluate”，配置参数以及描述如下表所示，之后点击“确认“运行。

参数名称	数据类型	数据样例	参数含义
dbNameStr	String	{'price':'dfs://CROSS_SECTION_1sec','factor':'dfs://secondLevelDB'}	价格和因子的库名
tbNameStr	String	{'price':'trade_info','factor':'freq_1sec'}	价格和因子的表名
factorNamesStr	String	{'Mmt_Rev_TimeSeries': ['QRS', 'close_ret_10m_kurt']}	需要评价的因子列表

winsorFactorNamesStr	String	{}	需要进行截断的因子列表
startDate	String	2023-04-20	评价的起始日期
endDate	String	2023-04-20	评价的结束日期
startTime	String	09:40:00	评价的每日起始时刻
endTime	String	10:00:00	评价的每日结束时刻
securityidColStr	String	{'price':'security_code','factor': 'securityid'}	股票代码对应的列名
tradetimeColStr	String	{'price':'trade_time','factor': 'tradetime'}	交易时间对应的列名
factorColName	String	factorname	因子名称对应的列名
priceCol	String	close	价格对应的列名
facValColName	String	value	因子值对应的列名
group	Int	5	分组回测的组数
window	Int	60	预测多少个时间间隔后的股票收益率
zscore_flag	Boolean	False	是否进行 zscore 归一化
long_short	Boolean	True	是否构建多空组合

运行结构可进行可视化展示，相关教程(GitHub 项目中的“DolphindbFactorLab_alpha2.3 /factor_management_platform.pdf”文件)可参考 DDB 官方提供的因子管理平台的“可视化模板”部分内容。

六、平台数据的外部运用

并非量化研究的所有工作都能便捷地在因子管理平台中实现。比如当我们有使用遗传算法挖掘有效因子、利用更复杂的深度学习模型进行因子组合等需求时，因子管理平台目前来说无能为力。考虑到因子平台中保存着的各种特征以及因子的数据，我们可以利用 DDB 的各种编程语言的 API 来简化我们的数据准备过程。鉴于目前 Python 的生态环境中提供有大量 Package 支持遗传算法以及深度学习模型的使用，下面将以 Python 语言为例，展示如何从因子管理平台获取数据，到运行一个简单的遗传算法。

DDB 官方提供了 dolphindb 的 Python 包，其封装好函数来调用 DDB 的脚本语言在 DDB 服务器上运行。如图表 1 中所示，首先通过 s=ddb.session()来与 DDB 服务器建立会话，注意程序结束前要调用 s.close()来手动关闭会话，避免闲置会话过多达到配置的上限。在建立好会话后，我们可以调用类似于 DDB 脚本语言的 s.loadTable()来读取库表，该函数是将整个库表数据全部传输到本地，如果读取时需要增加筛选条件，则可以通过 s.execute(str)来完成,其中的 str 是 sql 语句。读取得到的库表可以通过 toDF()函数转化成 pandas 的 DataFrame 类型，调用 pandas 保存文件的相关函数，即可在本地进行后续数据分析和处理了。

```python
import dolphindb as ddb
import torch
import torch.nn as nn
import dolphindb_tools
import os


# create DDB session
s = ddb.session()
s.connect("127.0.0.1", 11282, "wangzr", "wzr123456")

# load data
dbPath = "dfs://K_Minute_Level"
tbName = "OneMinute"
t1 = s.loadTable(dbPath=dbPath, tableName=tbName)
df1 = t1.toDF()

# save it into csv file
os.makedirs("./csv", exist_ok=True)
df1.to_csv("./csv/OneMin_KLine.csv")
s.close()
```

图表 1

接着我们将利用从 DDB 的因子管理平台中保存到本地数据进行后续遗传算法的研究。如图表 2 所示，加载之前保存的 csv 文件，将时间和标的代码的类型和格式还原，并对空值进行处理，输出后续用于研究的 data。简单起见，我们只使用 open, close, high, low 用于遗传算法的原始因子，而这些原始因子需要转换为面板格式，便于后续被算子进行运算操作，处理后的原始因子保存在 infos 变量中。

```python
random.seed(10)
# Load stock data
data = pd.read_csv('../DownloadData/csv/OneMin_KLine.csv', index_col=0)
data['trade_time'] = pd.to_datetime(data['trade_time'])
data['security_code'] = data['security_code'].astype(str).str.zfill(6)

# Obtain  the return for the next n minutes
n = 5
grouped = data.groupby(['security_code'])
data['new_ret'] = grouped['close'].apply(lambda x: x.shift(-n)/x.shift(-1)-1).droplevel(0)
# data['new_ret'] = data.groupby('security_code')['ret'].shift(-n)
data = data[~data.isna().any(axis=1)]

infos = {}
cols = ['open', 'close', 'high', 'low']
for col in cols:
    infos[col] = pd.pivot_table(data, values=col, index=["trade_time"], columns=["security_code"])
    infos[col] = infos[col]
```

图表 2

前面准备好的 infos 将用于后续的评估函数中，当遗传算法产生一个种群后，我们需要调用评估函数对其进行排序。这儿我们以种群中每个因子和预期收益率之间的信息系数作为评价指标。评估函数的实现如图表 3。

```python
def evaluate_factor(individual, pset):
    # print(individual)
    func = gp.compile(individual, pset)
    try:
        factor_values = func(*[infos[i] for i in cols])
    except Warning as w:
        if issubclass(type(w), RuntimeWarning):
            # handle ConstantInputWarning
            return (-1,)
        else:
            # other warnings
            print(individual)
            print(w)
            pass

    if not isinstance(factor_values, pd.DataFrame):
        return (-1,)

    df1 = pd.melt(factor_values.reset_index(), id_vars='trade_time', value_vars=factor_values.columns.tolist(), var_name='security_
    df1.sort_values(by=['security_code', 'trade_time'], ascending=[True, True])
    df1 = df1[~df1['value'].isna()]
    if df1.empty:
        return (-1,)
    df1 = pd.merge(df1, data[['trade_time', 'security_code', 'new_ret']], on=['trade_time', 'security_code'], how='inner')
    try:
        result = pearsonr(df1['value'], df1['new_ret'])
    except Warning as w:
        if issubclass(type(w), ConstantInputWarning):
            # handle ConstantInputWarning
            return (-1,)
        else:
            # other warnings
            print(individual)
            print(w)
            pass

    return (abs(result[0]),)
```

图表 3

关于遗传算法中算子的实现以及训练不是本文档的重点，因此这儿不再介绍。深度学习模型的研究中，数据获取环节跟上面提到的方式一样，按需从因子管理平台上读取库表。