

## 数据清洗与计算

## 一、准备工作

- 1.clone 仓库、新建环境
- 2.运行命令从开源的课程仓库下载数据到本地，并解压出文件夹

- stock\_trades
- .gitignore
- environment.yml
- LICENSE
- README.md
- stock\_trades.zip

PS: 该数据是同一位投资者从两个券商的交易软件分别导出的许多“股票交割单”文件, 2022 年 7 月至 2023 年 10 月, 每月导出一个文件, 扩展名为.xls 和.xlsx,

- 3.运行 jupyter lab, 新建 Notebook, 改名为 data-build.ipynb, 用于后续数据清洗操作

## 二、读取 CSV

- 1.尝试使用 `polars.read_excel()`函数读取名称为 202207-湘财.xls 的文件，观察报错:

```
import polars as pl

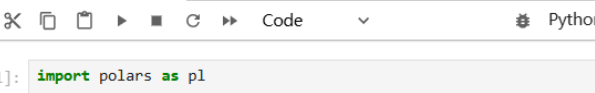
pl.read_excel("stock_trades/202207-湘财.xls")

-----
CalamineError                                Traceback (most recent call last)
Cell In[2], line 1
----> 1 pl.read_excel( )

File ~\anaconda3\envs\week08\Lib\site-packages\polars\
9, in deprecate_renamed_parameter.<locals>.decorate.<locals>
kwargs)
    114 @wraps(function)
    115 def wrapper(*args: P.args, **kwargs: P.kwargs) -> T:
    116     _rename_keyword_argument(
    117         old_name, new_name, kwargs, function, qualname, version

CalamineError: calamine error: Xls error: Cfb error: Invalid OLE signature (not an office document?)
Context:
  0: Could not open workbook at stock_trades/202207-湘财.xls
  1: could not load excel file at stock_trades/202207-湘财.xls
```

PS: shift tab 或右键选择 show contextual help 可查看帮助文档:



The screenshot shows a Jupyter Notebook window titled "data-build.ipynb". The interface includes a toolbar with icons for file operations, a code editor, and a console. The code in the notebook is as follows:

```
[1]: import polars as pl

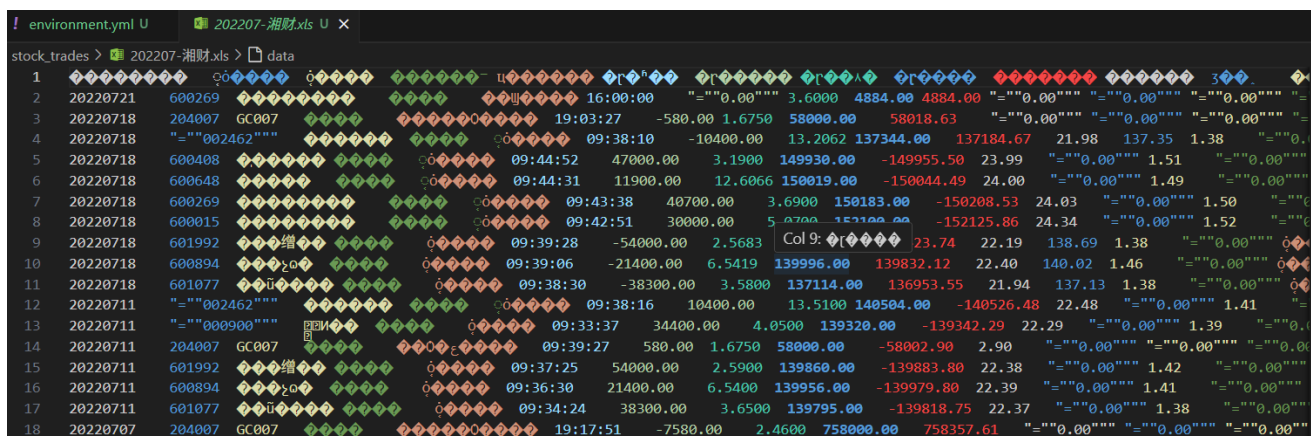
[ ]: pl.read_excel()
```

A tooltip is visible for the `pl.read_excel()` function, showing its signature and parameters:

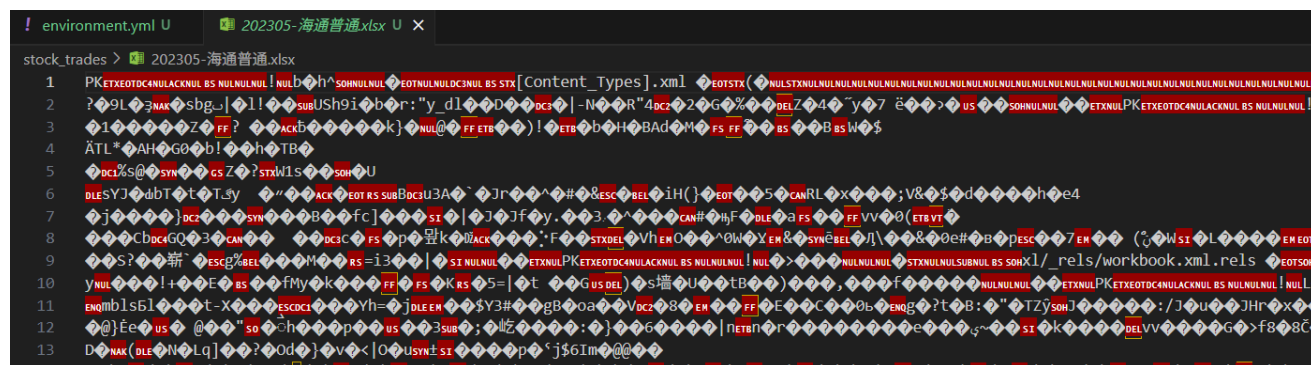
```
Signature:
pl.read_excel(
    source: 'FileSource',
    *,
    sheet_id: 'int | Sequence[int] | None' = None,
    sheet_name: 'str | list[str] | tuple[str] | None' = None,
    table_name: 'str | None' = None,
    engine: 'ExcelSpreadsheetEngine' = 'calamine',
    engine_options: 'dict[str, Any] | None' = None,
    read_options: 'dict[str, Any] | None' = None,
```

## 2.理解以上报错:

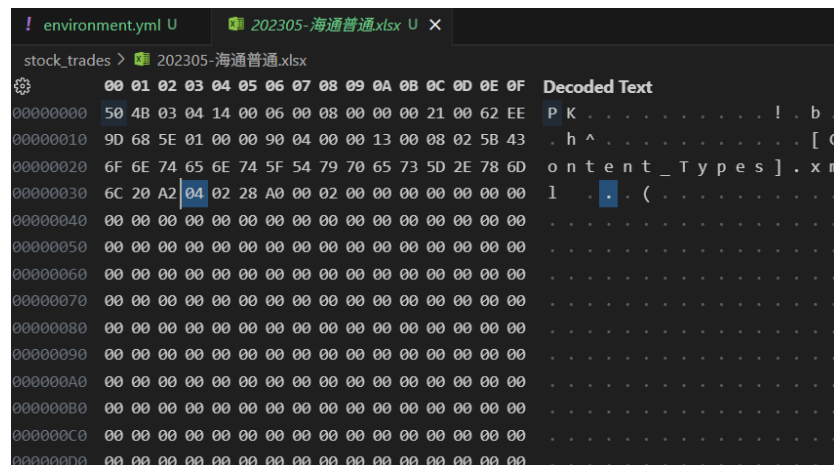
- 用 VSCode 以文本方式查看 202207-湘财.xls 文件的内容, 可以看到其并非二进制的 Excel 格式, 而是纯文本文件, 并且有乱码:



- 作为对比, 可以查看 202305-海通普通.xlsx 文件的内容(“OpenAnyway”强制打开), 可以看到全是二进制乱码, 这才是正确的 Excel 文件格式:



- 在 VSCode 扩展商店里安装 Hex Editor 扩展, 此十六进制编辑器可以更直观地查看/编辑底层的二进制字节:



PS: 每个比特(bit)可以表示两种状态(二进制, [01]), 连续的 4 个比特可以表示 16 ( $2^4=16$ )种状态 (十六进制, [0-9A-F]), 连续的 8 个比特构成一个字节(1 byte=8 bits), 可以用两个十六进制数字来表示

- 纯文本文件 202207-湘财.xls 有乱码的原因是, VSCode 将二进制字节解码(decode)为文本代码(Unicode, 对应着各国字符)时, 默认使用了错误的编解码器(encoding)。当初导出/保存这个文件所用的券商交易软件, 应该使用了 **GB18030 编解码器**(简体中文 Windows 操作系统的默认选

择)将文本代码(Unicode)编码(encode)为二进制字节。而现代软件(尤其是在 macOS、Ubuntu 等类 Unix 操作系统里, 以及 Windows 下的 VSCode)默认都使用的是 **UTF-8** 编解码器。解码所用的 **encoding** 如果与编码所用的 **encoding** 不匹配, 就会翻译错误, 显示出乱码。

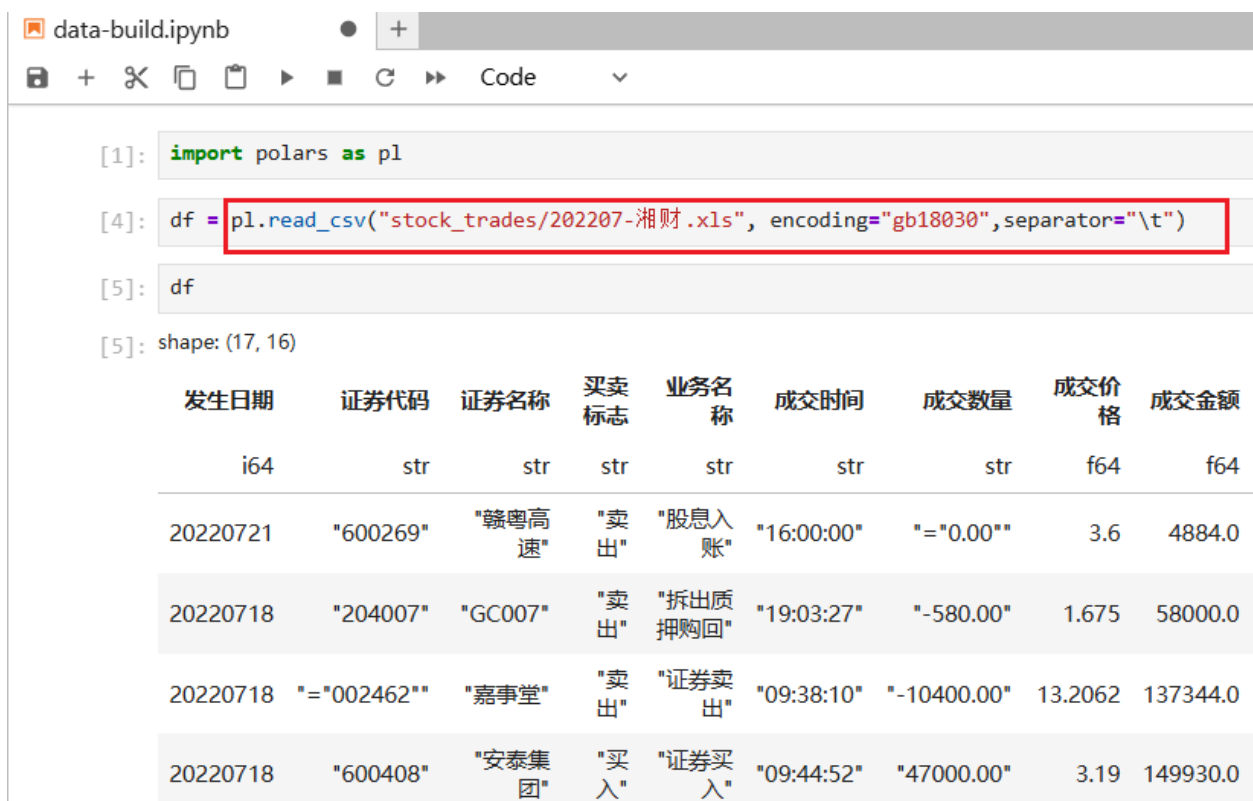
- 在 VS Code 界面右下角 UTF-8 处点击鼠标, 在菜单里选择 “Reopen with Encoding”, 进一步选择 GB18030 编解码器, 就能够正确地看到汉字了:



	发生日期	证券代码	证券名称	买卖标志	业务名称	成交时间	成交数量	成交价格	成交金额	发生金额
1	20220721	600269	赣粤高速	卖出	股息入账	16:00:00	"="0.00""	3.6000	4884.00	4884.00
2	20220718	204007	GC007	卖出	拆出质押购回	19:03:27	-580.00	1.6750	58000.00	58018.63
3	20220718	"="002462""	嘉事堂	卖出	证券卖出	09:38:10	-10400.00	13.2062	137344.00	137184.67
4	20220718	600408	安泰集团	买入	证券买入	09:44:52	47000.00	3.1900	149930.00	-149955.50
5	20220718	600648	外高桥	买入	证券买入	09:44:31	11900.00	12.6066	150019.00	-150044.49
6	20220718	600269	赣粤高速	买入	证券买入	09:43:38	40700.00	3.6900	150183.00	-150208.53
7	20220718	600015	华夏银行	买入	证券买入	09:42:51	30000.00	5.0700	152100.00	-152125.86
8	20220718	601992	金隅集团	卖出	证券卖出	09:39:28	-54000.00	2.5683	138686.00	138523.74
9	20220718	600894	广日股份	卖出	证券卖出	09:39:06	-21400.00	6.5419	139996.00	139832.12
10	20220718	601077	渝农商行	卖出	证券卖出	09:38:30	-38300.00	3.5800	137114.00	136953.55
11	20220711	"="002462""	嘉事堂	买入	证券买入	09:38:16	10400.00	13.5100	140504.00	-140526.48
12	20220711	"="000900""	现代投资	买入	证券买入	09:33:37	34400.00	4.0500	139320.00	-139342.11
13	20220711	204007	GC007	卖出	质押回购拆出	09:39:27	580.00	1.6750	58000.00	-58002.90
14	20220711	601992	金隅集团	买入	证券买入	09:37:25	54000.00	2.5900	139860.00	-139883.80
15	20220711	600894	广日股份	买入	证券买入	09:36:30	21400.00	6.5400	139956.00	-139979.80
16	20220711	601077	渝农商行	买入	证券买入	09:34:24	38300.00	3.6500	139795.00	-139818.75
17	20220707	204007	GC007	卖出	拆出质押购回	19:17:51	-7580.00	2.4600	758000.00	758357.61
18										
19										

PS: 在 VS Code 里可以看出, 202207-湘财.xls 文件实际上并不是 Excel 格式, 而是 CSV 格式, 而且分隔符 (separator) 不是逗号 (,), 而是 TAB (t)

3. 解决以上报错: 用 `polars.read_csv()` 函数重新读取 202207-湘财.xls 文件, 参照函数文档恰当指定参数, 反复尝试, 最终返回正确的 `polars.DataFrame` 对象, 命名为 `df`



```
[1]: import polars as pl
[4]: df = pl.read_csv("stock_trades/202207-湘财.xls", encoding="gb18030", separator="\t")
[5]: df
[5]: shape: (17, 16)
```

发生日期	证券代码	证券名称	买卖标志	业务名称	成交时间	成交数量	成交价格	成交金额
i64	str	str	str	str	str	str	f64	f64
20220721	"600269"	"赣粤高速"	"卖出"	"股息入账"	"16:00:00"	"="0.00""	3.6	4884.0
20220718	"204007"	"GC007"	"卖出"	"拆出质押购回"	"19:03:27"	"-580.00"	1.675	58000.0
20220718	"="002462""	"嘉事堂"	"卖出"	"证券卖出"	"09:38:10"	"-10400.00"	13.2062	137344.0
20220718	"600408"	"安泰集团"	"买入"	"证券买入"	"09:44:52"	"47000.00"	3.19	149930.0

### 三、检查 DataFrame

#### 1. 掌握以下几个检查 polars.DataFrame 对象时常用的属性 (attributes) / 方法 (methods)

- 形状/维度: df.shape、df.height、df.width、df.is\_empty()

```
[4]: df.shape
[4]: (17, 16)

[5]: df.height
[5]: 17

[6]: df.width
[6]: 16

[7]: df.is_empty() 要调用
[7]: False
```

- 数据架构/类型: df.schema、df.columns、df.dtypes

df.schema	df.columns	df.dtypes
Schema([('发生日期', Int64), ('证券代码', String), ('证券名称', String), ('买卖标志', String), ('业务名称', String), ('成交时间', String), ('成交数量', String), ('成交价格', Float64), ('成交金额', Float64), ('发生金额', Float64), ('手续费', String), ('印花税', String), ('过户费', String), ('其他费', String), ('备注', String), ('币种', String)])	['发生日期', '证券代码', '证券名称', '买卖标志', '业务名称', '成交时间', '成交数量', '成交价格', '成交金额', '发生金额', '手续费', '印花税', '过户费', '其他费', '备注', '币种']	[Int64, String, String, String, String, String, String, Float64, Float64, Float64, String, String, String, String, String, String]
list(df.schema.keys()) == df.columns True	list(df.schema.values()) == df.dtypes True	
isinstance(df.schema, dict) True	type(df.columns) list	type(df.dtypes) list

- 数据提取/切片: df[ ] (取行/列)、df.row()、df.rows()、df.get\_column()、df.to\_series()

```
df[3:5] 选行
```

shape: (2, 16)

发生日期	证券代码	证券名称	买卖标志	业务名称	成交时间	成交数量	成交价
i64	str	str	str	str	str	str	f64
20220718	"600408"	"安泰集团"	"买入"	"证券买入"	"09:44:52"	"47000.00"	3.1
20220718	"600648"	"外高桥"	"买入"	"证券买入"	"09:44:31"	"11900.00"	12.606

```
df[:, -3:]
```

```
df[:, ["证券名称", "成交金额"]]
```

```
df[:5, "证券名称": "成交金额"]
```

```
df[[0, 5, 6], "证券名称": "成交金额"]
```

选行和列





- 转换/导出: `df.to_pandas()`、`df.to_arrow()`、`df.to_dicts()`

```
type(df.to_pandas())
```

`pandas.core.frame.DataFrame`

pandas和polars非常兼容

```
df.to_pandas()
```

	发生日期	证券代码	证券名称	买卖标志	业务名称	成交时间	成交数量	成交价格	成交金额	发生金额	手续费
0	20220721	600269	赣粤高速	卖出	股息入账	16:00:00	= "0.00"	3.6000	4884.0	4884.00	= "0.00"
1	20220718	204007	GC007	卖出	拆出质押购回	19:03:27	-580.00	1.6750	58000.0	58018.63	= "0.00"
2	20220718	= "002462"	嘉事堂	卖出	证券卖出	09:38:10	-10400.00	13.2062	137344.0	137184.67	21.98
3	20220718	600408	安泰集团	买入	证券买入	09:44:52	47000.00	3.1900	149930.0	-149955.50	23.99
4	20220718	600648	外高桥	买入	证券买入	09:44:31	11900.00	12.6066	150019.0	-150044.49	24.00

## 2. `polars.DataFrame` 单独的一列数据是 `polars.Series`，检查 `polars.Series` 有以下常用的属性/方法

- 基本属性: `s.name`、`s.dtype`、`s.shape`、`s.len()`

- 数据提取/切片: `s[ ]` (取单值/取多值)

```
s = df.to_series()
```

```
s.name
```

'发生日期'

```
s.dtype
```

Int64

```
s.shape
```

(17,)

```
s.len()
```

17

```
s[:5]
```

shape: (5,)

发生日期

i64

20220721

20220718

20220718

20220718

20220718

```
s[2,8,9]
```

shape: (3,)

发生日期

i64

20220718

20220718

20220718

- 数据概览/描述: `s.unique()`、`s.value_counts()`、`s.describe()`、`s.null_count()`

- 转换/导出: `s.to_pandas()`、`s.to_arrow()`、`s.to_list()`

```
df.to_series(4).unique()
```

shape: (5,)

业务名称

str

"证券买入"

"拆出质押购回"

"证券卖出"

"股息入账"

"质押回购拆出"

```
df.to_series(4).value_counts()
```

shape: (5, 2)

业务名称 count

str u32

"证券卖出" 4

"股息入账" 1

"质押回购拆出" 1

```
df.to_series(9).describe()
```

shape: (9, 2)

statistic

value

str

f64

"count"

17.0

"null\_count"

0.0

"mean"

815.642353

"std"

230047.143813

"min"

-152125.86

"25%"

-140526.48

"50%"

-139342.29

"75%"

136953.55

"max"

758357.61

```
df[:, "发生金额"].to_list()
```

[4884.0,  
58018.63,  
137184.67,

3. 经过检查，我们发现有几个列的类型 (dtype) 存在错误，所以建议在 `polars.read_csv` 读取时指定参数 `infer_schema=False`，将所有列都先读取为字符串类型，再进行数据的清洗和类型转换

```
df.glimpse()
Rows: 17
Columns: 16
$ 发生日期 <i64> 20220721, 20220718, 20220718, 20220718, 20220718, 20220718, 20220718, 20220718, 20220718, 20220718, 20220718, 20220718, 20220718, 20220718, 20220718, 20220718
$ 证券代码 <str> '600269', '204007', '002462', '600408', '600648', '600269', '600015', '601992', '600894', '601077'
$ 证券名称 <str> '赣粤高速', 'GC007', '嘉事堂', '安泰集团', '外高桥', '赣粤高速', '华夏银行', '金隅集团', '广日股份', '渝农商行'
$ 买卖标志 <str> '卖出', '卖出', '卖出', '买入', '买入', '买入', '买入', '卖出', '卖出', '卖出'
$ 业务名称 <str> '股息入账', '拆出质押购回', '证券卖出', '证券买入', '证券买入', '证券买入', '证券买入', '证券买入', '证券卖出', '证券卖出', '证券卖出', '证券卖出'
$ 成交时间 <str> '16:00:00', '19:03:27', '09:38:10', '09:44:52', '09:44:31', '09:43:38', '09:42:51', '09:39:28', '09:39:06', '09:38:30'
$ 成交数量 <str> '0.00', '-580.00', '-10400.00', '47000.00', '11900.00', '40700.00', '30000.00', '-54000.00', '-21400.00', '-38300.00'
$ 成交价格 <f64> 3.6, 1.675, 13.2062, 3.19, 12.6066, 3.69, 5.07, 2.5683, 6.5419, 3.58
$ 成交金额 <f64> 4884.0, 58000.0, 137344.0, 149930.0, 150019.0, 150183.0, 152100.0, 138686.0, 139996.0, 137114.0
$ 发生金额 <f64> 4884.0, 58018.63, 137184.67, -149955.5, -150044.49, -150208.53, -152125.86, 138523.74, 139832.12, 136953.55
$ 手续费 <str> '0.00', '0.00', '21.98', '23.99', '24.00', '24.03', '24.34', '22.19', '22.40', '21.94'
$ 印花税 <str> '0.00', '0.00', '137.35', '0.00', '0.00', '0.00', '0.00', '0.00', '138.69', '140.02', '137.13'
$ 过户费 <str> '0.00', '0.00', '1.38', '1.51', '1.49', '1.50', '1.52', '1.38', '1.46', '1.38'
$ 其他费 <str> '0.00', '0.00', '0.00', '0.00', '0.00', '0.00', '0.00', '0.00', '0.00', '0.00', '0.00'
$ 备注 <str> '股息入账:赣粤高速600269; 权益股数:40700;', '融券购回:18.63实际占款天数: 7-888880', '证券卖出', '证券买入', '证券买入', '证券卖出', '证券卖出'
$ 币种 <str> '人民币', '人民币', '人民币', '人民币', '人民币', '人民币', '人民币', '人民币', '人民币', '人民币', '人民币', '人民币', '人民币', '人民币', '人民币', '人民币'
```

```
pl.read_csv("stock_trades/202207-湘财.xls", encoding="gb18030", separator="\t", infer_schema=False)
shape: (17, 16)
```

发生日期	证券代码	证券名称	买卖标志	业务名称	成交时间	成交数量	成交价格	成交金额	发生金额	手续费	印花税	过户费
str	str	str	str	str	str	str	str	str	str	str	str	str
"20220721"	"600269"	"赣粤高速"	"卖出"	"股息入账"	"16:00:00"	"0.00"	"3.6000"	"4884.00"	"4884.00"	"0.00"	"0.00"	"0.00"

## 四、数据清洗

1. `polars.DataFrame` 的计算，都是整列进行的向量化 (vectorized) 计算，利用 CPU 的 SIMD 指令能够极大地提升计算效率

### PS: 基本思想

- `DataFrame.with_columns()` 方法用来添加/修改列
  - `DataFrame.select()` 方法用来挑选/计算列
  - `DataFrame.filter()` 方法用来过滤行 (计算为 True 的行将被保留)
  - 她们接受的参数都是 Polars Expression —— 存储的是算法逻辑，而非具体数值
  - Polars 之所以功能强大，就是因为设计有大量的 Expressions，可以组合使用
- Expressions 文档: [Expressions — Polars documentation](#)
- 构建 Polars Expression 的起点，一般都是通过 `polars.col` 选择一列或多列，也可以通过 selectors 挑选符合条件的列，然后利用 `.` 运算符进行链式调用，或者用其他各种运算符组合计算出更进一步的、复杂的 Expression

## 2. 基于以上思想，对 df 进行清洗操作：

- 把发生日期列转换为 polars.Date 类型

```
df = pl.read_csv("stock_trades/202207-湘财.xls", encoding="gb18030", separator="\t", infer_schema=False)
df = df.with_columns(
    pl.col("发生日期").str.to_date("%Y%m%d")
)
df
```

shape: (17, 16)

发生日期	证券代码	证券名称	买卖标志
date	str	str	str
2022-07-21	"600269"	"赣粤高速"	"卖出"
2022-07-18	"204007"	"GC007"	"卖出"

- 在证券代码列 (以及其他许多列) 里，0 开头的字符串外面都包围了双引号，左边加了等号，需要把这些多余的字符去掉

```
df[:, "证券代码"].unique().to_list()
['600648',
'601077',
'"=002462"',
'600269',
'600408',
'601992',
'600894',
'"=000900"',
'600015',
'204007']

df = df.with_columns(
    pl.col("证券代码").str.strip_prefix("=").str.strip_chars('"')
)
df[:, "证券代码"].unique().to_list()
['600408',
'601992',
'601077',
'000900',
'600015',
'600648',
'600894',
'204007',
'002462',
'600269']
```

- 在业务名称列里，除了证券买入和证券卖出外还有其他几种业务，为简单起见，把其他业务的行全部删除 (filter)

```
df = df.filter(
    pl.col("业务名称").is_in(["证券买入", "证券卖出"]),
)
df
```

shape: (13, 16)

发生日期	证券代码	证券名称	买卖标志	业务名称	成交时间	成交数量	成交价格
date	str	str	str	str	str	str	str
2022-07-18	"002462"	"嘉事堂"	"卖出"	"证券卖出"	"09:38:10"	"-10400.00"	"13.2062"
2022-07-18	"600408"	"安泰集团"	"买入"	"证券买入"	"09:44:52"	"47000.00"	"3.1900"
2022-07-18	"600648"	"外高桥"	"买入"	"证券买入"	"09:44:31"	"11900.00"	"12.6066"
2022-07-18	"600269"	"赣粤高速"	"买入"	"证券买入"	"09:43:38"	"40700.00"	"3.6900"
2022-07-18	"600015"	"华夏银行"	"买入"	"证券买入"	"09:42:51"	"30000.00"	"5.0700"
...	...	...	...	...	...	...	...
2022-07-11	"002462"	"嘉事堂"	"买入"	"证券买入"	"09:38:16"	"10400.00"	"13.5100"



➤ 把成交时间列转换为 polars.Time 类型

```
df = df.with_columns(  
    pl.col("成交时间").str.to_time()  
)  
df
```

shape: (13, 16)

发生日期	证券代码	证券名称	买卖标志	业务名称	成交时间	成交数量	成交价格
date	str	str	str	str	time	str	str
2022-07-18	"002462"	"嘉事堂"	"卖出"	"证券卖出"	09:38:10	"-10400.00"	"13.2062"

➤ 把成交数量、成交价格等几个数值类型的列都转换为 polars.Float64 类型

```
df = pl.read_csv(  
    "stock_trades/202207-湘财.xls",  
    encoding="gb18030",  
    separator="\t",  
    infer_schema=False,  
)  
df = df.with_columns(  
    pl.selectors.all().str.strip_prefix("=").str.strip_chars(' '),  
)  
.with_columns(  
    pl.col("发生日期").str.to_date("%Y%m%d"),  
    pl.col("成交时间").str.to_time(),  
    pl.col(  
        "成交数量",  
        "成交金额",  
        "成交价格",  
        "发生金额",  
        "手续费",  
        "印花税",  
        "过户费",  
        "其他费",  
    ).cast(pl.Float64),  
)  
df = df.filter(  
    pl.col("业务名称").is_in(["证券买入", "证券卖出"]),  
)  
df
```

先借助selectors.all()将所有多余字符去掉

转成Float64

shape: (13, 16)

发生日期	证券代码	证券名称	买卖标志	业务名称	成交时间	成交数量	成交价格	成交金额	发生金额	手续费	印花税
date	str	str	str	str	time	f64	f64	f64	f64	f64	f64
2022-07-18	"002462"	"嘉事堂"	"卖出"	"证券卖出"	09:38:10	-10400.0	13.2062	137344.0	137184.67	21.98	137.35

TIPS: 安装 jupyter-ruff 软件包，可以在 jupyter lab 中设置：Settings/Jupyter Ruff/Format On Save，同时将 Autosave 关闭，这样 ctrl s 后 cell 可自动格式化

3. 在成功处理一个 .xls 文件的基础上，利用 `pathlib.Path.glob()` 遍历所有 \*-湘财.xls 文件，都进行以上处理 (列表推导式)

```
def read_df_湘财(f: str | Path) -> pl.DataFrame:
    df = pl.read_csv(
        f,
        encoding="gb18030",
        separator="\t",
        infer_schema=False,
    )
    df = df.with_columns(
        pl.selectors.all().str.strip_prefix("=").str.strip_chars(''),
    ).with_columns(
        pl.col("发生日期").str.to_date("%Y%m%d"),
        pl.col("成交时间").str.to_time(),
        pl.col(
            "成交数量",
            "成交金额",
            "成交价格",
            "发生金额",
            "手续费",
            "印花税",
            "过户费",
            "其他费",
        ).cast(pl.Float64),
    )
    df = df.filter(
        pl.col("业务名称").is_in(["证券买入", "证券卖出"]),
    )
    return df
```

定义了一个函数 `read_df_湘财`，  
参数 `f` 是文件路径，可以是字符串或者 `Path` 对象  
这个函数会返回一个 `polars.DataFrame`

```
from pathlib import Path
```

```
df = [read_df_湘财(f) for f in Path("stock_trades/").glob("*-湘财.xls")]
```

```
len(df)
```

13 `df` 含有13个数据列表

然后用 `polars.concat()` 合并成一个 `DataFrame`，命名为 `d1`，再添加一个新列：券商，每行的值都填 "湘财"

```
d1 = pl.concat(df)
```

```
d1.with_columns(
    券商=pl.lit("湘财"),
)
```

shape: (257, 17)

发生日期	证券代码	证券名称	买卖标志	业务名称	成交时间	成交数量	成交价格	成交金额	发生金额	手续费	印花税	过户费	其他费	备注	币种	券商
date	str	str	str	str	time	f64	f64	f64	f64	f64	f64	f64	f64	str	str	str
2022-07-18	"002462"	"嘉事堂"	"卖出"	"证券卖出"	09:38:10	-10400.0	13.2062	137344.0	137184.67	21.98	137.35	1.38	0.0	"证券卖出"	"人民币"	"湘财"
2022-07-18	"600408"	"安泰集团"	"买入"	"证券买入"	09:44:52	47000.0	3.19	149930.0	-149955.5	23.99	0.0	1.51	0.0	"证券买入"	"人民币"	"湘财"
2022-07-18	"600648"	"外高桥"	"买入"	"证券买入"	09:44:31	11900.0	12.6066	150019.0	-150044.49	24.0	0.0	1.49	0.0	"证券买入"	"人民币"	"湘财"

## 接下来处理另外一个券商的数据(读取、检查、清洗)

1. 尝试使用 `polars.read_excel()` 函数读取名称为 202305-海通普通.xlsx 的文件，命名为 df

```
df = pl.read_excel("stock_trades/202305-海通普通.xlsx")
```

df

shape: (10, 14)

证券代码	证券名称	成交日期	成交时间	成交数量	成交价格	成交金额	发生金额	操作
str	str	i64	str	i64	f64	f64	f64	str
"300107"	"建新股份"	20230529	"	0	0.0	600.0	600.0	"卖"
"131810"	"R-001"	20230524	"	1500	100.005	150007.66	150007.66	"买"

2. 检查行列数 (shape)，检查架构 (dtype)，逐列检查值 (value\_counts)，发现一些问题，进行清洗：

- 成交日期列的类型有错误，读取时可以用 `schema_overrides` 参数指定类型，读取后再进一步转换类型
- 成交时间列应该转换为 `polars.Time` 类型
- 成交时间列里有些值是空字符串，把这些行全部删除
- 操作列里，除了买和卖外还有其他几种操作，为简单起见，把其他操作的行全部删除
- 证券名称列里可以看到存在 R-001 之类的国债逆回购，为简单起见，把这些行都删除(基于证券代码的编码规律来进行过滤)

```
df = pl.read_excel(
    "stock_trades/202305-海通普通.xlsx",
    schema_overrides={
        "成交日期": pl.String,
        "成交时间": pl.String,
    },
)
df = df.filter(
    (pl.col("成交时间") != "")
    & (pl.col("操作").is_in(["买", "卖"]))
    & (~pl.col("证券代码").str.starts_with("1318"))
    & (~pl.col("证券代码").str.starts_with("204"))
).with_columns(
    pl.col("成交日期").str.to_date("%Y%m%d"),
    pl.col("成交时间").str.to_time("%H:%M:%S"),
)
return df
```

~表示取反，过滤掉1318和204开头的

shape: (5, 14)

证券代码	证券名称	成交日期	成交时间	成交数量	成交价格	成交金额
str	str	date	time	i64	f64	f64
"600626"	"申达股份"	2023-05-24	10:06:14	14800	3.37	49876.0
"600178"	"东安动力"	2023-05-24	09:59:17	16400	6.07	99548.0
"603002"	"宏昌电子"	2023-05-24	09:54:48	9800	5.06	49588.0
"300107"	"建新股份"	2023-05-23	10:01:57	10000	5.0	50000.0
"002224"	"三力士"	2023-05-23	09:58:07	10800	4.59	49572.0

3. 利用 `pathlib.Path.glob()` 遍历所有 \*-海通普通.xlsx 文件，都进行以上处理 (列表推导式)，然后用 `polars.concat()` 合并成一个 DataFrame，命名为 d2，再添加一个新列：券商，每行都填 "海通普通"

```
def read_df_海通普通(f: str | Path) -> pl.DataFrame:
    df = pl.read_excel(
        f,
        schema_overrides={
            "成交日期": pl.String,
            "成交时间": pl.String,
            "成交数量": pl.Float64,
            "成交金额": pl.Float64,
            "印花税": pl.Float64,
            "其他费": pl.Float64,
        },
    )
    df = df.filter(
        (pl.col("成交时间") != "")
        & (pl.col("操作").is_in(["买", "卖"]))
        & (~pl.col("证券代码").str.starts_with("1318"))
        & (~pl.col("证券代码").str.starts_with("204"))
    ).with_columns(
        pl.col("成交日期").str.to_date("%Y%m%d"),
        pl.col("成交时间").str.to_time("%H:%M:%S"),
    )
    return df
```

```
df = [read_df_海通普通(p) for p in Path("stock_trades/").glob("*-海通普通.xlsx")]
df = pl.concat(df)
d2 = df.with_columns(
    券商=pl.lit("海通普通"),
)
d2.sort("成交日期", "成交时间")
```

shape: (53, 15)

证券代码	证券名称	成交日期	成交时间	成交数量	成交价格	成交金额	发生金额	操作	手续费	印花税	过户费	其他费	备注	券商
str	str	date	time	f64	f64	f64	f64	str	f64	f64	f64	f64	str	str
"002224"	"三力士"	2023-05-23	09:58:07	10800.0	4.59	49572.0	-49576.96	"买"	4.96	0.0	0.0	0.0	"三力士证券买入"	"海通普通"

4.对海通两融执行同样的操作（存为 d3）

```
df = [read_df_海通普通(p) for p in Path("stock_trades/").glob("*-海通两融.xlsx")]
df = pl.concat(df)
d3 = df.with_columns(
    券商=pl.lit("海通两融"),
)
d3.sort("成交日期", "成交时间")
```

shape: (53, 15)

证券代码	证券名称	成交日期	成交时间	成交数量	成交价格	成交金额	发生金额	操作	手续费	印花税	过户费	其他费	备注	券商
str	str	date	time	f64	f64	f64	f64	str	f64	f64	f64	f64	str	str
"600638"	"新黄浦"	2023-07-26	09:33:56	9300.0	6.526	60696.0	60628.01	"卖"	6.68	60.7	0.61	0.0	"新黄浦证券卖出"	"海通两融"
"002492"	"恒基达鑫"	2023-07-27	09:30:42	8700.0	6.12	53244.0	53184.91	"卖"	5.86	53.23	0.0	0.0	"恒基达鑫证券卖出"	"海通两融"
"002136"	"安纳达"	2023-07-27	09:32:52	4400.0	11.881	52277.0	52218.97	"卖"	5.75	52.28	0.0	0.0	"安纳达证券卖出"	"海通两融"

## 最后把 d1、d2、d3 合并在一起 (纵向):

它们的列名称和列类型 (即架构) 不一致, 我们统一选择、保留、命名部分几列, 适当转换类型, 然后合并, 合并后命名为 df, 保存为 stock\_trades.parquet 文件

### 1.先操作 d1

```
d1.select(
    券商=p1.col("券商"),
    交易日期=p1.col("发生日期"),
    交易时间=p1.col("成交时间"),
    证券代码=p1.col("证券代码"),
    证券名称=p1.col("证券名称"),
    买卖标志=p1.col("业务名称").replace({"证券卖出":"卖出","证券买入":"买入"}), 改换标志
    成交价格=p1.col("成交价格"),
    成交数量=p1.col("成交数量").abs(), 取绝对值
    成交金额=p1.col("成交金额"),
    手续费=p1.col("手续费"),
    印花税=p1.col("印花税"),
    过户费=p1.col("过户费"),
    其他费=p1.col("其他费"),
    发生金额=p1.col("发生金额"),
)
```

shape: (257, 14)

券商	交易日期	交易时间	证券代码	证券名称	买卖标志	成交价格	成交数量	成交金额	手续费	印花税	过户费	其他费	发生金额
str	date	time	str	str	str	f64	f64	f64	f64	f64	f64	f64	f64
"湘财"	2022-07-18	09:38:10	"002462"	"嘉事堂"	"卖出"	13.2062	10400.0	137344.0	21.98	137.35	1.38	0.0	137184.67
"湘财"	2022-07-18	09:44:52	"600408"	"安泰集团"	"买入"	3.19	47000.0	149930.0	23.99	0.0	1.51	0.0	-149955.5
"湘财"	2022-07-18	09:44:31	"600648"	"外高桥"	"买入"	12.6066	11900.0	150019.0	24.0	0.0	1.49	0.0	-150044.49
"湘财"	2022-07-18	09:43:38	"600269"	"赣粤高速"	"买入"	3.69	40700.0	150183.0	24.03	0.0	1.5	0.0	-150208.53
"湘财"	2022-07-18	09:42:51	"600015"	"华夏银行"	"买入"	5.07	30000.0	152100.0	24.34	0.0	1.52	0.0	-152125.86

d1

shape: (257, 17)

发生日期	证券代码	证券名称	买卖标志	业务名称	成交时间	成交数量	成交价格	成交金额	发生金额	手续费	印花税	过户费	其他费	备注	币种	券商
date	str	str	str	str	time	f64	f64	f64	f64	f64	f64	f64	f64	str	str	str
2022-07-18	"002462"	"嘉事堂"	"卖出"	"证券卖出"	09:38:10	-10400.0	13.2062	137344.0	137184.67	21.98	137.35	1.38	0.0	"证券卖出"	"人民币"	"湘财"
2022-07-18	"600408"	"安泰集团"	"买入"	"证券买入"	09:44:52	47000.0	3.19	149930.0	-149955.5	23.99	0.0	1.51	0.0	"证券买入"	"人民币"	"湘财"

### 2.在 d1 的基础上稍作修改, 操作 d2 和 d3

```
d2 = d2.select(
    券商=p1.col("券商"),
    交易日期=p1.col("成交日期"),
    交易时间=p1.col("成交时间"),
    证券代码=p1.col("证券代码"),
    证券名称=p1.col("证券名称"),
    买卖标志=p1.col("操作").replace({"卖":"卖出","买":"买入"}),
    成交价格=p1.col("成交价格"),
    成交数量=p1.col("成交数量").abs(),
    成交金额=p1.col("成交金额"),
    手续费=p1.col("手续费"),
    印花税=p1.col("印花税"),
    过户费=p1.col("过户费"),
    其他费=p1.col("其他费"),
    发生金额=p1.col("发生金额"),
)
```



3.最终合并，命名为 df

df = pl.concat([d1,d2,d3])

df

shape: (363, 14)

券商	交易日期	交易时间	证券代码	证券名称	买卖标志	成交价格	成交数量	成交金额	手续费	印花税	过户费	其他费	发生金额
str	date	time	str	str	str	f64	f64	f64	f64	f64	f64	f64	f64
"湘财"	2022-07-18	09:38:10	"002462"	"嘉事堂"	"卖出"	13.2062	10400.0	137344.0	21.98	137.35	1.38	0.0	137184.67
"湘财"	2022-07-18	09:44:52	"600408"	"安泰集团"	"买入"	3.19	47000.0	149930.0	23.99	0.0	1.51	0.0	-149955.5
"湘财"	2022-07-18	09:44:31	"600648"	"外高桥"	"买入"	12.6066	11900.0	150019.0	24.0	0.0	1.49	0.0	-150044.49
"湘财"	2022-07-18	09:43:38	"600269"	"赣粤高速"	"买入"	3.69	40700.0	150183.0	24.03	0.0	1.5	0.0	-150208.53
"湘财"	2022-07-18	09:42:51	"600015"	"华夏银行"	"买入"	5.07	30000.0	152100.0	24.34	0.0	1.52	0.0	-152125.86
...	...	...	...	...	...	...	...	...	...	...	...	...	...
"海通两融"	2023-10-18	09:46:15	"300464"	"星徽股份"	"卖出"	5.74	16100.0	92414.0	8.82	46.21	0.0	0.0	92358.97
"海通两融"	2023-10-18	09:55:41	"002661"	"克明食品"	"买入"	9.42	8500.0	80072.0	7.64	0.0	0.0	0.0	-80079.64

4.验算（不保存）

df.with\_columns(  
 成交金额2=pl.col("成交价格")\*pl.col("成交数量"),  
).with\_columns(  
 成交金额D=pl.col("成交金额") - pl.col("成交金额2"),  
).with\_columns(  
 发生金额D=(  
 pl.col("发生金额")  
 - (  
 pl.when(pl.col("买卖标志") == "买入")  
 .then(-pl.col("成交金额"))  
 .when(pl.col("买卖标志")=="卖出")  
 .then(pl.col("成交金额"))  
 -pl.col("手续费")  
 -pl.col("印花税")  
 -pl.col("过户费")  
 -pl.col("其他费")  
 )  
 ).round(4)  
).sort("发生金额D")

shape: (363, 17)

券商	交易日期	交易时间	证券代码	证券名称	买卖标志	成交价格	成交数量	成交金额	手续费	印花税	过户费	其他费	发生金额	成交金额2	成交金额D	发生金额D
str	date	time	str	str	str	f64	f64	f64	f64	f64	f64	f64	f64	f64	f64	f64
"湘财"	2022-07-18	09:44:52	"600408"	"安泰集团"	"买入"	3.19	47000.0	149930.0	23.99	0.0	1.51	0.0	-149955.5	149930.0	0.0	0.0
"湘财"	2022-07-18	09:44:31	"600648"	"外高桥"	"买入"	12.6066	11900.0	150019.0	24.0	0.0	1.49	0.0	-150044.49	150018.54	0.46	0.0

5.最终保存

Name	Modified	Size
stock_trades	5 days ago	
data-build.ipynb	5 min. ago	178.6 KB
environment.yml	4 days ago	293 B
LICENSE	4 days ago	18.4 KB
README.md	4 days ago	2.2 KB
stock_trades.csv	10 min. ago	41 KB
stock_trades.parquet	11 min. ago	17.8 KB
stock_trades.xlsx	9 min. ago	35 KB
stock_trades.zip	4 days ago	75.2 KB

观察大小

[140]: df.write\_parquet("stock\_trades.parquet")

[141]: df.write\_csv("stock\_trades.csv")

[142]: df.write\_excel("stock\_trades.xlsx")

[142]: <xlsxwriter.workbook.Workbook at 0x213ac0e0fe0>

[143]: pl.read\_parquet("stock\_trades.parquet")

[143]: shape: (363, 14)

券商	交易日期	交易时间	证券代码	证券名称	买卖标志
str	date	time	str	str	str
"湘财"	2022-07-18	09:38:10	"002462"	"嘉事堂"	"卖出"
"湘财"	2022-07-18	09:44:52	"600408"	"安泰集团"	"买入"
"湘财"	2022-07-18	09:44:31	"600648"	"外高桥"	"买入"
"湘财"	2022-07-18	09:43:38	"600269"	"赣粤高速"	"买入"
"湘财"	2022-07-18	09:42:51	"600015"	"华夏银行"	"买入"

## 五、数据计算

在 JupyterLab 页面新建 Notebook，改名为 data-query.ipynb，读取上文处理过的数据，用于后续数据计算操作：

```
[1]: import polars as pl

[2]: df = pl.read_parquet("stock_trades.parquet")
```

### 1. 通过计算和作图检查每一笔交易的费率

➤ 分别计算每笔交易的手续费率、印花税率、过户费率

```
df.with_columns(
    手续费率=pl.col("手续费")/pl.col("成交金额"),
    印花税率=pl.col("印花税")/pl.col("成交金额"),
    过户费率=pl.col("过户费")/pl.col("成交金额"),
)
```

shape: (363, 17)

券商	交易日期	交易时间	证券代码	证券名称	买卖标志	成交价格	成交数量	成交金额	手续费	印花税	过户费	其他费	发生金额	手续费率	印花税率	过户费率
str	date	time	str	str	str	f64	f64	f64	f64	f64	f64	f64	f64	f64	f64	f64
"湘财"	2022-07-18	09:38:10	"002462"	"嘉事堂"	"卖出"	13.2062	10400.0	137344.0	21.98	137.35	1.38	0.0	137184.67	0.00016	0.001	0.00001

➤ 为每笔交易生成一个序号 (index)

```
df = df.with_row_index("序号", 1)
df
```

shape: (363, 18)


序号	券商	交易日期	交易时间	证券代码	证券名称
u32	str	date	time	str	str
1	"湘财"	2022-07-18	09:38:10	"002462"	"嘉事堂"
2	"湘财"	2022-07-18	09:44:52	"600408"	"安泰集团"

➤ 用 Perspective 的 X/Y Scatter 视图，将序号作为 X 坐标，某项费率作为 Y 坐标，不同券商区分颜色，比较哪个券商的费率更优惠，其他感兴趣的值可以显示在悬浮框 (tooltip) 里

```
[12]: import polars as pl
      from perspective.widget import PerspectiveWidget
```

```
[18]: df = pl.read_parquet("stock_trades.parquet")
      # 转成str, 因为PerspectiveWidget不支持time类型
      df = df.with_columns(
          pl.col("交易时间").cast(pl.String),
          手续费率=pl.col("手续费")/pl.col("成交金额"),
          印花税率=pl.col("印花税")/pl.col("成交金额"),
          过户费率=pl.col("过户费")/pl.col("成交金额"),
      )
      df = df.with_row_index("序号", 1)
```

```
[19]: PerspectiveWidget(df)
```

[19]:  untitled 363 x 18

序号	券商	交易日期	交易时间	证券代码	证券名称	买卖标志	成交价格	成交数量
Edit	Edit	Edit	Edit	Edit	Edit	Edit	Edit	Edit
1	湘财	2022/7/17	09:38:10	002462	嘉事堂	卖出	13.21	10,400.00
2	湘财	2022/7/17	09:44:52	600408	安泰集团	买入	3.19	47,000.00



2. 期间的交割单涉及多支股票，可以计算每支股票是否都已完成平仓 (首次买入算开仓，全部卖光算平仓)

①使用 `df.groupby().agg()` 进行分组汇总，`group_by()` 分组和 `agg()` 汇总都接受一个/多个 Expression 作为输入

➤ 按证券代码或/和证券名称分组

➤ 按成交数量汇总，首先需要根据买卖标志决定正负号，然后汇总求和，命名为结余数量

②最后，按照结余数量排序：结余数量为负的，是在交割单期初之前就有持仓；结余数量为正的，是在交割单期末之后仍有持仓

```
df.groupby("证券代码", "证券名称").agg(
    结余数量=(
        pl.when(pl.col("买卖标志") == ("卖出"))
        .then(-pl.col("成交数量"))
        .when(pl.col("买卖标志") == "买入")
        .then(pl.col("成交数量"))
        .sum()
    ),
).sort("结余数量")
```

shape: (152, 3)

证券代码	证券名称	结余数量
str	str	f64
"603167"	"渤海轮渡"	-13600.0
"300889"	"爱克股份"	-7000.0
"600333"	"长春燃气"	-3900.0
"300107"	"建新股份"	0.0
"600097"	"开创国际"	0.0
...	...	...
"002661"	"克明食品"	8500.0

③为简化起见，我们把结余数量为负的（交割单期初之前就有持仓的）股票，从 df 里剔除

- 使用 DataFrame.filter() 选择出准备剔除的股票
- 使用 DataFrame.join(how="anti") 进行基于匹配的剔除

```
d1 = df.join(  
    df.groupby("证券代码", "证券名称")  
    .agg(  
        结余数量=(  
            pl.when(pl.col("买卖标志") == ("卖出"))  
            .then(-pl.col("成交数量"))  
            .when(pl.col("买卖标志") == "买入")  
            .then(pl.col("成交数量"))  
            .sum()  
        ),  
    )  
    .filter(pl.col("结余数量") < 0),  
    on="证券代码",  
    how="anti",  
)
```

shape: (358, 18)

序号	券商	交易日期	交易时间	证券代码	证券名称	买卖标志	成交价格	成交数量	成交金额	手续费
u32	str	date	str	str	str	str	f64	f64	f64	f64
1	"湘财"	2022-07-11	"09:33:37"	"000900"	"现代投资"	"买入"	4.05	34400.0	139320.0	22.29
2	"湘财"	2022-07-11	"09:34:24"	"601077"	"渝农商行"	"买入"	3.65	38300.0	139795.0	22.37
3	"湘财"	2022-07-11	"09:36:30"	"600894"	"广日股份"	"买入"	6.54	21400.0	139956.0	22.39

3. 计算和做图观察这段期间累计的股票持仓数量变化情况 (注意，不同股票的持股数量相加是没有意义的，为简化起见，我们现在暂不考虑股价和市值)，以及每天持有股票数量 (支数) 的变化情况

①现在要考虑的是每一天、每一支股票的动态情况及其汇总，所以我们先计算时间范围 (k1)，再计算股票范围 (k2)，再计算二者的笛卡尔积 k (k1.join(k2, how="cross"))

```
[6]: start_date = df["交易日期"].min()  
start_date
```

```
[6]: datetime.date(2022, 7, 11)
```

```
[7]: end_date = df["交易日期"].max()  
end_date
```

```
[7]: datetime.date(2023, 10, 31)
```

```
[9]: k1 = pl.select(日期=pl.date_range(start_date, end_date))  
k1
```

```
[9]: shape: (478, 1)
```

日期

date

2022-07-11

2022-07-12

2022-07-13

2022-07-14

2022-07-15

```
k2 = df["证券代码"].unique().sort().to_frame()  
k2
```

```
shape: (152, 1)
```

证券代码

str

"000096"

"000532"

```
k = k1.join(k2, how="cross")
```

k

```
shape: (72_656, 2)
```

日期 证券代码

date

str

2022-07-11 "000096"

2022-07-11 "000532"

不转的话是series

②用得到的笛卡尔积 k 与交割单数据做左匹配 (left join)，即保留全部的 k，未匹配到的行赋空值

```
k.join(
    d1, left_on=["日期", "证券代码"], right_on=["交易日期", "证券代码"], how="left"
)
```

“日期”和“交易日期”不同

不能直接用on

shape: (72\_671, 18)

日期	证券代码	序号	券商	交易时间	证券名称	买卖标志	成交价格	成交数量	成交金额
date	str	u32	str	str	str	str	f64	f64	f64
2022-07-11	"000096"	null	null	null	null	null	null	null	null

③对于成交数量列，买入取正值，卖出取负值，空值(null)取值 0(when)，由此衍生计算一列结余数量；在每支股票范围内(over)，沿交易日期计算其累计的(cum\_sum)成交数量作为结余数量；把结余数量为 0 的行全部剔除，便于统计每天的持股

```
k.join(
    d1, left_on=["日期", "证券代码"], right_on=["交易日期", "证券代码"], how="left"
).sort("日期", "证券代码").with_columns(
    结余数量=(
        pl.when(pl.col("买卖标志") == "买入")
        .then(pl.col("成交数量"))
        .when(pl.col("买卖标志") == "卖出")
        .then(-pl.col("成交数量"))
        .otherwise(0)
        .cum_sum()
        .over("证券代码")
    ),
).filter(pl.col.结余数量 > 0)
```

分别累加

保留大于0的

④用 Perspective 的 Y Area 视图，横轴 Group By 设定为交易日期，纵轴 Y Axis 可以查看每日总的结余数量(注意，其实是不可加的)，也可以查看每日的持股数量(即持有的证券代码的数量)





4. 从 Tushare 获取行情数据，与每日持股数据匹配，由此计算每日持股的市值的动态变化，并能够由此计算每日投资收益率，并与股市指数的每日收益率 (基准收益) 相对照

①调用 Tushare 的 daily 接口，需要指定股票代码和起止时间（单只股票举例）：

```
[27]: import tushare as ts
[28]: pro = ts.pro_api()
[29]: start_date
[29]: datetime.date(2022, 7, 11)
[30]: f"{start_date:%Y%m%d}"
[30]: '20220711'
[31]: end_date
[31]: datetime.date(2023, 10, 31)
[32]: format(end_date, "%Y%m%d")
[32]: '20231031'
[33]: hq = pro.daily(
    ts_code="002462.SZ",
    start_date=format(start_date, "%Y%m%d"),
    end_date=format(end_date, "%Y%m%d"),
)
hq = pl.from_pandas(hq)
hq
```

必须先转格式，两种方式都可以

把上面调用得到pandas df 转成polars df

[33]: shape: (318, 11)

ts_code	trade_date	open	high	low	close	pre_close	change	pct_chg	vol	amount
str	str	f64	f64	f64	f64	f64	f64	f64	f64	f64
"002462.SZ"	"20231031"	14.75	14.9	14.59	14.7	14.75	-0.05	-0.339	65859.96	96984.271

注意，交割单里的证券代码 (如 002462) 不含交易所代码，与 Tushare 的编码不符，因此需要先根据沪深交易所的编码规律转换出含交易所代码的证券代码 (如 002462.SZ)

```
d1.select(证券代码=pl.col("证券代码").str.head(1)).to_series().value_counts()
```

shape: (3, 2)

证券代码	count
str	u32
"6"	166
"0"	106
"3"	86

```
d1.select(
    证券代码=(
        pl.when(pl.col("证券代码").str.head(1).is_in(["0", "3"]))
        .then(pl.format("{}SZ", pl.col("证券代码")))
        .when(pl.col("证券代码").str.head(1) == "6")
        .then(pl.format("{}SH", pl.col("证券代码")))
    ),
)
```

shape: (358, 1)

证券代码

str

"000900.SZ"
"601077.SH"
"600894.SH"
"601992.SH"
"002462.SZ"

添加了后缀，与tushare保持一致

②根据上文，以恰当的形式向 Tushare 接口传入参数，获取每支股票在时间范围内的每日数据：

➤ 股票数量较多，需要循环调用，可以使用 **tqdm** 软件包显示进度条

```
[47]: ts_codes = (
    d1.select(
        证券代码=(
            p1.when(p1.col("证券代码").str.head(1).is_in(["0", "3"]))
            .then(p1.format("{}SZ", p1.col("证券代码")))
            .when(p1.col("证券代码").str.head(1) == "6")
            .then(p1.format("{}SH", p1.col("证券代码")))
        ),
    )
    .to_series()
    .unique()
    .sort()
    .to_list()
)

[48]: from tqdm.notebook import tqdm

[51]: hq = [
    p1.from_pandas(
        pro.daily(
            ts_code=ts_code,
            start_date=format(start_date, "%Y%m%d"),
            end_date=format(end_date, "%Y%m%d"),
        )
    )
    for ts_code in tqdm(ts_codes)
]

100% 149/149 [00:10<00:00, 16.48it/s]
```

➤ 全部获取后合并，保存为 **daily.parquet** 文件

```
[52]: hq = p1.concat(hq) 合并

[53]: !pwd 加！运行
/c/Users/71970/repo/week08

[54]: hq.write_parquet("daily.parquet") 存储

[55]: hq = p1.read_parquet("daily.parquet") 读取

[56]: hq 查看

[56]: shape: (47, 355, 11)
```

ts_code	trade_date	open	high	low	close	pre_close	change	pct_chg	vol	amount
str	str	f64	f64	f64	f64	f64	f64	f64	f64	f64
"000096.SZ"	"20231031"	8.95	9.25	8.88	8.95	8.84	0.11	1.2443	45191.22	40697.722
"000096.SZ"	"20231030"	8.72	8.94	8.68	8.84	8.78	0.06	0.6834	33567.85	29716.156

③将行情数据与交割单数据匹配，检查每一行的成交价格是否落在最高价与最低价之间，检查每一行的交割单成交数量占股票成交量的比例，以防交割单数据造假

```
[43]: hq = p1.read_parquet("daily.parquet")
hq.head(2)

[43]: shape: (2, 11)
```

ts_code	trade_date	open	high	low	close	pre_close	change	pct_chg	vol	amount
str	str	f64	f64	f64	f64	f64	f64	f64	f64	f64
"000096.SZ"	"20231031"	8.95	9.25	8.88	8.95	8.84	0.11	1.2443	45191.22	40697.722
"000096.SZ"	"20231030"	8.72	8.94	8.68	8.84	8.78	0.06	0.6834	33567.85	29716.156

```
[44]: hq = hq.with_columns(
    p1.col("ts_code").str.head(6), p1.col("trade_date").str.to_date("%Y%m%d")
)
hq.head(2)

[44]: shape: (2, 11)
```

ts_code	trade_date	open	high	low	close	pre_close	change	pct_chg	vol	amount
str	date	f64	f64	f64	f64	f64	f64	f64	f64	f64
"000096"	2023-10-31	8.95	9.25	8.88	8.95	8.84	0.11	1.2443	45191.22	40697.722

```
d1.join(
    hq, left_on=["交易日期", "证券代码"], right_on=["trade_date", "ts_code"], how="left"
).filter(
    ~pl.col("成交价格").is_between(pl.col("low"), pl.col("high")),
)
~tilde取反, 检查价格区间
```

先合并

shape: (0, 27)

序号	券商	交易日期	交易时间	证券代码	证券名称	买卖标志	成交价格	成交数量	成交金额	手续费	印花税	过户费	其他费	发生金额	手续费率	印花税率	过户费率	open	high	low	close
u32	str	date	str	str	str	str	f64	f64	f64	f64	f64	f64	f64	f64	f64	f64	f64	f64	f64	f64	f64

```
d1.join(
    hq, left_on=["交易日期", "证券代码"], right_on=["trade_date", "ts_code"], how="left"
).with_columns(
    vratio=pl.col("成交数量") / 100 / pl.col("vol"),
).sort("vratio")
```

检查成交数量占比

代码	证券名称	买卖标志	成交价格	成交数量	成交金额	手续费	印花税	过户费	其他费	发生金额	手续费率	印花税率	过户费率	open	high	low	close	pre_close	change	pct_chg	vol	amount	vratio
str	str	str	f64	f64	f64	f64	f64	f64	f64	f64	f64	f64	f64	f64	f64	f64	f64	f64	f64	f64	f64	f64	f64
6*	"兴业银行"	"卖出"	17.75	3000.0	53250.0	8.52	53.25	0.53	0.0	53187.7	0.00016	0.001	0.00001	17.77	17.94	17.5	17.56	17.63	-0.07	-0.3971	1.6276e6	2.8810e6	0.000018
8*	"汇金股份"	"卖出"	6.2	100.0	620.0	0.1	0.62	0.01	0.0	619.28	0.000161	0.001	0.000016	6.18	6.25	5.95	6.06	6.2	-0.14	-2.2581	49219.99	29911.915	0.00002

④将行情数据与每日持股数据匹配, 将非交易日缺失的价格数据填充为最近数值 ( fill\_null().over() ), 按交易日期分组, 汇总计算每日总的持股市值

```
d3 = (
    k.join(
        d1, left_on=["日期", "证券代码"], right_on=["交易日期", "证券代码"], how="left"
    )
    .sort("日期", "证券代码")
    .with_columns(
        结余数量=(
            pl.when(pl.col("买卖标志") == "买入")
            .then(pl.col("成交数量"))
            .when(pl.col("买卖标志") == "卖出")
            .then(-pl.col("成交数量"))
            .otherwise(0)
            .cum_sum()
            .over("证券代码")
        ),
    )
)
d3
```

shape: (72\_671, 19)

日期	证券代码	序号	券商
date	str	u32	str
2022-07-11	"000096"	null	null

```
d4 = (
    d3.join(
        hq,
        left_on=["日期", "证券代码"],
        right_on=["trade_date", "ts_code"],
        how="left",
    )
    .sort("证券代码", "日期")
    .with_columns(close=pl.col("close").fill_null(strategy="forward").over("证券代码"))
    .with_columns(持股市值=pl.col("结余数量") * pl.col("close"))
    .group_by("日期")
    .agg(pl.col("持股市值").sum())
)
d4
```

不同证券内部填列null

分类汇总

shape: (478, 2)

日期	持股市值
date	f64
2022-10-07	538326.0
2023-07-11	659685.0
2023-03-02	820589.0

用 Perspective 的 X/YLine 视图观察每日持股市值的动态变化



⑤计算投资者的投资收益率

- 除了要知道股票交易情况外，还要知道总的资金情况，即本金（假设期初投资者本金是 100 万）
- 先根据交割单计算每日总的发生金额
- 再生成一个转账金额列，只有第一行取值 100 万，其余行全部为零
- 再把每日的发生金额和转账金额加在一起，沿日期做累加，就得到每日的现金余额
- 再把每日现金余额和每日持股市值加在一起，就得到每日的总资产
- 用 Perspective 的 X/Y Line 视图观察每日总资产的动态变化

```
d4 = (  
    d3.join(  
        hq,  
        left_on=["日期", "证券代码"],  
        right_on=["trade_date", "ts_code"],  
        how="left",  
    )  
    .sort("证券代码", "日期")  
    .with_columns(close=pl.col("close").fill_null(strategy="forward").over("证券代码"))  
    .with_columns(持股市值=pl.col("结余数量") * pl.col("close"))  
    .group_by("日期")  
    .agg(  
        pl.col("持股市值").sum(),  
        pl.col("发生金额").sum(),  
    )  
    .sort("日期")  
    .with_columns(  
        转账金额=pl.when(pl.int_range(0, pl.len()) == 0).then(1000000).otherwise(0),  
    )  
    .with_columns(  
        现金余额=(pl.col("转账金额") + pl.col("发生金额")).cum_sum(),  
    )  
    .with_columns(  
        总资产=pl.col("持股市值") + pl.col("现金余额"),  
    )  
)  
d4
```

shape: (478, 6)

日期	持股市值	发生金额	转账金额	现金余额	总资产
date	f64	f64	i32	f64	f64
2022-07-11	703040.0	-699551.12	1000000	300448.88	1.0035e6



⑥调用 Tushare 的 index\_daily 接口, 获取 “沪深 300 指数” (000300.SH)在期限内的每日涨跌幅数据:

- 获得的是每日的净收益率(net rate of return), 除以 100 再加 1 转换为每日的总收益率(gross)
- 沿日期做累乘, 就得到投资指数的每日累计收益率(cumulative return)
- 再与本金 100 万相乘, 就能够得到如果全部投资于 “沪深 300 指数” 每日的总资产变化情况, 命名为沪深 300;
- 用 Perspective 的 X/YLine 视图观察每日沪深 300 的动态变化

```
ihq = pro.index_daily(  
    ts_code="000300.SH",  
    start_date=format(start_date, "%Y%m%d"),  
    end_date=format(end_date, "%Y%m%d"),  
    fields="ts_code,trade_date,pct_chg",  
)
```

```
pl.from_pandas(ihq).write_parquet("index_daily.parquet")
```

```
ihq = pl.read_parquet("index_daily.parquet")  
ihq
```

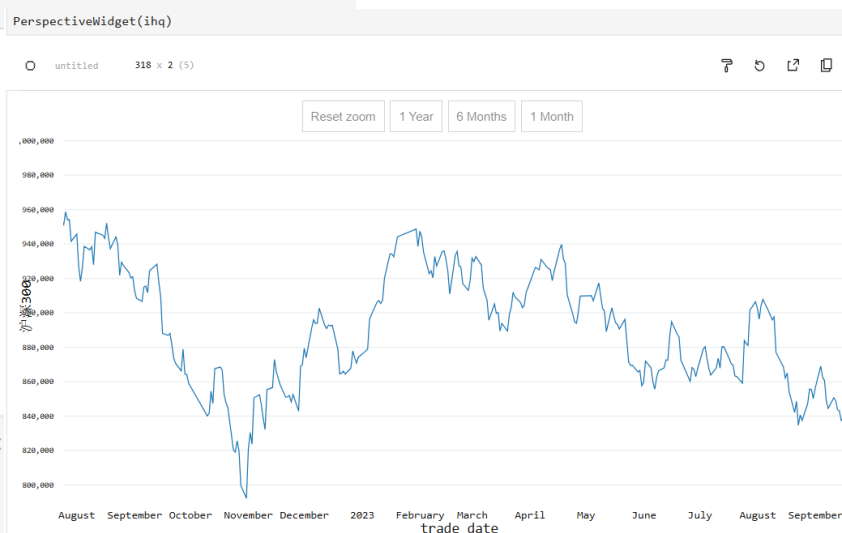
shape: (318, 3)

ts_code	trade_date	pct_chg
str	str	f64
"000300.SH"	"20231031"	-0.3144
"000300.SH"	"20231030"	0.6003

```
ihq = pl.read_parquet("index_daily.parquet")  
ihq = (  
    ihq.with_columns(  
        pl.col("pct_chg") / 100 + 1,  
    )  
    .sort("trade_date")  
    .with_columns(  
        car=pl.col("pct_chg").cum_prod(),  
    )  
    .with_columns(  
        沪深300=pl.col("car") * 1000000,  
    )  
)  
ihq
```

shape: (318, 5)

ts_code	trade_date	pct_chg	car	沪深300
str	str	f64	f64	f64
"000300.SH"	"20220711"	0.983254	0.983254	983254.0
"000300.SH"	"20220712"	0.990585	0.973997	973996.66359





## ⑦在 Perspective 的 X/YLine 视图里同时显示两种(甚至更多种)投资的动态变化

- 需要做一种数据变形, 因为现在这两个曲线的数值分别在两个列里(总资产和沪深 300), 属于宽形(wide form), 如果有更多曲线, 就要加更多的列进去(变宽), 会改变表格的架构(schema), 不利于存储和分析。
- 我们需要把数据变为长形(long form), 值都放在同一列(value column)(在 Perspective 里设置为 Y Axis), 列名放在另一列(variable column)用于区分(在 Perspective 里设置为 Split By)。
- 从“长形”变为“宽形”叫做 pivot, 从“宽形”变为“长形”叫做 unpivot/melt

```
[36]: d5 = d4.join(ihq, left_on="日期", right_on="trade_date")
      d5.head(3)
```

[36]: shape: (3, 10)

日期	持股市值	发生金额	转账金额	现金余额	总资产	ts_code	pct_chg	car	沪深300
date	f64	f64	i32	f64	f64	str	f64	f64	f64
2022-07-11	703040.0	-699551.12	1000000	300448.88	1.0035e6	"000300.SH"	0.983254	0.983254	983254.0
2022-07-12	707714.0	0.0	0	300448.88	1.0082e6	"000300.SH"	0.990585	0.973997	973996.66359
2022-07-13	713855.0	0.0	0	300448.88	1.0143e6	"000300.SH"	1.001818	0.975767	975767.389524

```
[39]: d5 = d5.unpivot(宽形变长形
on=["总资产", "沪深300"], index="日期", variable_name="资产类型", value_name="财富"
)
d5
```

[39]: shape: (636, 3)

日期	资产类型	财富
date	str	f64
2022-07-11	"总资产"	1.0035e6
2022-07-12	"总资产"	1.0082e6
2022-07-13	"总资产"	1.0143e6
2022-07-14	"总资产"	1.0105e6
2022-07-15	"总资产"	992825.88
...	...	...
2023-10-25	"沪深300"	791288.453778
2023-10-26	"沪深300"	793475.575065

