week08

1. Fork 第08周打卡 仓库至你的名下，然后将你名下的这个仓库 Clone 到你的本地计算机，在项目目录下，新建 `environment.yml` 文件

```yaml
name: week08
channels:
  - conda-forge
dependencies:
  - python=3.12
  - wat-inspector
  - xlrd
  - openpyxl
  - fastexcel
  - xlsxwriter
  - pandas
  - pyarrow
  - polars
  - jupyterlab
  - ipywidgets
  - jupyter-ruff
  - pip
  - pip:
    - perspective-python
    - tushare
```

然后运行 `conda env create` 命令创建 Conda 环境，安装完毕后运行 `conda activate week08` 命令激活

2. 在项目目录下运行下面的命令 (一行一行分别运行)，从我们开源的 课程仓库 下载案例数据到你的本地，并解压出文件夹

```
curl -O https://raw.gitcode.com/cueb-fintech/courses/blobs/8e70be13d8672dd685672f6624896ad5320d1110
unzip stock_trades.zip
```

里面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，时间为 2022 年 7 月至 2023 年 10 月，每月导出一个文件，扩展名为 `.xls` 和 `.xlsx`，已进行过数据脱敏 (即移除了个人可识别信息 PII)，可以双击用 Excel 打开查看

3. 在项目目录下，运行 `jupyter lab`，在 JupyterLab 页面里，新建 Notebook，改名为 `data-build.ipynb`，在里面按照下面的要求完成 **数据清洗** 操作：

    - 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
    - 用 VS Code 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 (选 “Open Anyway” 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 VS Code 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：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，VS Code 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，应该是使用了 `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 VS Code) 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

- 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 (seporator) 不是逗号 (`,`)，而是 TAB (`\t`)
- 尝试使用 `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 (可以在 Notebook 右键菜单里选择 “Show Contextual Help” 方便查看内置文档)，反复尝试，最终返回正确的 `polars.DataFrame` 对象，命名为 `df`
- 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()`
    - 数据模式/架构/类型：`df.schema`、`df.columns`、`df.dtypes`
    - 数据提取/切片：`df[...]` (取行/取列/取多行/取多列)、`df[..., ...]` (行列双向限制)、`df.row()`、`df.rows()`、`df.get_column()`、`df.to_series()`
    - 数据概览/描述：`df.glimpse()`、`df.head()`、`df.tail()`、`df.sample()`、`df.describe()`、`df.null_count()`
    - 转换/导出：`df.to_pandas()`、`df.to_arrow()`、`df.to_dicts()`
- `polars.DataFrame` 单独的一列数据是 `polars.Series`；*检查* `polars.Series` 对象 (命名为 `s`) 有以下常用的属性 (attributes) / 方法 (methods)：
    - 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)
    - 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()`

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

    浏览 Expressions 文档，寻找并组合恰当的 Expressions，对 `df` 进行以下清洗操作：

    - 把 `发生日期` 列转换为 `polars.Date` 类型
    - 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，如 `="000900"`，需要把这些多余的字符去掉
    - 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
    - 把 `成交时间` 列转换为 `polars.Time` 类型
    - 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

- 接下来开始处理另外一个券商的数据。尝试使用 `polars.read_excel()` 函数读取名称为 `202305-海通普通.xlsx` 的文件，观察报错 —— 应该没有报错，因为确实是 Excel 文件
- 命名为 `df`，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，读取后可以进一步转换类型
    - `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，对应的业务是红利、扣税、划转等，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 (可以基于 `证券代码` 的编码规律 (问大模型) 来进行过滤)
- 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`，每行的值都填 `"海通普通"`
- 利用 `pathlib.Path.glob()` 遍历所有 `*-海通两融.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 `d3`，再添加一个新列 `券商`，每行的值都填 `"海通两融"`
- 最后，我们希望把 `d1`、`d2`、`d3` 合并在一起 (纵向)，但他们的列名称和列类型 (即架构) 不一致，我们统一选择/保留/命名为以下几列，适当转换类型，然后合并：

- `券商`
- `交易日期`
- `交易时间`
- `证券代码`
- `证券名称`
- `买卖标志` (`买入` / `卖出`)
- `成交价格`
- `成交数量` (取绝对值)
- `成交金额` (验算是否等于 `成交价格` × `成交数量`)
- `手续费`
- `印花税`
- `过户费`
- `其他费`
- `发生金额` (验算与 `成交金额` 的关系，相差是不是各种税费)

合并后命名为 `df` (请检查，总共应该有 363 行)，保存为 `stock_trades.parquet` 文件

4. 在 JupyterLab 页面里，新建 Notebook，改名为 `data-query.ipynb` ，在里面按照下面的要求完成 **数据计算** 操作：

   - 首先，最简单地，我们可以通过计算和做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`
     - 为每笔交易生成一个序号 (index)
     - 用 Perspective 的 `X/Y Scatter` 视图，将序号 (index) 作为 X 坐标，某项费率作为 Y 坐标，不同券

   - 第三，我们可以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

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

- 第四，我们可以从 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

  - 调用 Tushare 的 `daily` 接口，需要指定股票代码和起止时间，但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`)
  - 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 `开盘价`、`收盘价`、`最高价`、`最低价`、`成交量` 数据，股票数量较多，需要循环调取，可以使用 `tqdm` 软件包显示进度条，全部获取后合并，保存为 `daily.parquet` 文件
  - 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
  - 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`，用 Perspective 的 `X/Y Line` 视图观察每日 `持股市值` 的动态变化
  - 要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。假设在期初，投资者的本金是 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
  - 调用 Tushare 的 `index_daily` 接口，获取 “沪深300指数” (`000300.SH`) 在起止时间之内的每日 `涨跌幅` 数据，这是每日的 **净收益率** (net rate of return)，除以 100 (因为单位是 %) 再加 1 转换为每

- 要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。假设在期初，投资者的本金是 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
- 调用 Tushare 的 `index_daily` 接口，获取 “沪深300指数” (`000300.SH`) 在起止时间之内的每日 `涨跌幅` 数据，这是每日的 **净收益率** (net rate of return)，除以 100 (因为单位是 %) 再加 1 转换为每日的 **总收益率** (gross rate of return)，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/Y Line` 视图观察每日 `沪深300` 的动态变化
- 最后，如果想在 Perspective 的 `X/Y Line` 视图里 *同时* 显示两种 (甚至更多种) 投资的动态变化，还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`