

README.md

文件

附件是一份PRD文档，若把它交付给codex进行python代码的实现，请你对此进行客观评估，文档是否已经足够清晰，文档内容是否存在前后不一致的情况

已思考 2m 5s >

整体来看，这份 PRD 已经可以支撑 Codex 直接写出一版可用的 Python 计算模块，结构、输入输出、核心公式、阈值规则都说明得比较清楚，而且分了 4.1 / 4.2 / 4.3 三个子模块，依赖关系也写明了。 ◻ README

不过从“交给大模型写代码”的角度看，还有少量地方容易让模型自行脑补、或者存在轻微前后不一致 / 示例不严谨的点，我下面拆开说：

一、从 Codex 的角度看，哪些已经足够清晰？

1. 模块拆分与职责

- 4.1 父体整体指标：输入参数 + 数据源（`view_return_snapshot`）、过滤条件、聚合维度、指标公式都非常明确。
- 4.2 子 ASIN 结构：显式依赖 4.1 的 `parent_summary`，同时再次说明自己也读 `view_return_snapshot`，并给出 A 类 / B 类 / 观察名单的判定规则和中文展示字段。
- 4.3 问题 ASIN 核心原因：
 - 清楚写了“只对 `problem_class=A/B` 的 ASIN 做原因拆解”；
 - 对 `view_return_fact_details` 的筛选条件、计数逻辑（`distinct review_id`）、样本置信度规则、核心原因选择（ $\text{coverage} \geq 0.8$, 最多 3 个标签）都写得很细。

这三个模块的输入、输出 JSON 示例都给了清晰结构，对 Codex 来说非常友好，可以直接按示例去拼装 dataclass / Pydantic model 或普通 dict。

2. 数据结构与关键字段

- `view_return_snapshot`：主键、字段含义、粒度都讲得清楚。
- `view_return_fact_details`：包含了 `tag_code`、`tag_name_cn`、`sentiment`、`evidence` 等，已经足以支撑原因统计。

- 标签维表 `return_dim_tag`：定义了标签的层级、中文名、定义、边界说明等，虽然当前 Python 逻辑没有强依赖，但未来如果要根据 `version / effective_from` 做查询，也有基础。

总体上，**数据模型不会让 Codex 迷路**：每张“表”的主键、关键字段、示例 JSON 都在。

3. 规则与公式

- 所有关键指标的计算公式都有明确表达：
 - 父体退货率：`total_units_returned_parent / total_units_sold_parent`
 - 子 ASIN 的 share：`sales_share / returns_share`
 - 退货率阈值：`R_warn = 0.10, R_high_B = max(R_parent, R_warn) + 0.02`
 - 文本样本覆盖率、标签覆盖率、置信度等级 (high/medium/low) 的条件。
- 中文描述和公式是一致的，没有明显“文字说 A、公式写 B”的情况。

这些对代码生成来说非常关键——Codex 不需要自己猜“`high_return_watchlist` 到底是什么条件”，PRD 已经给出了完整布尔表达式。

二、文档里容易让 Codex 犹豫或脑补的点

下面这些不算“硬冲突”，但从“让模型少发挥一点想象力”的角度，建议在 PRD 里补一句/两句，会更稳：

1. 输入输出的“顶层形态”略有模糊

现在 4.1 / 4.2 / 4.3 分别给了各自的 JSON 示例，但**没有明确写出整个 Python 模块最终对外暴露的接口形态**：

- 是一个总入口函数，比如：

`python` □ 复制代码

```
def run_return_analysis(country, fasin, start_date, end_date) -> dict:
    return {
        "parent_summary": {...},
        "asin_structure": [...],
        "problem_asin_reasons": [...]
    }
```

- 还是 3 个独立函数，各自输入/输出 JSON，由调用方 orchestration？

Codex 有可能自己设计一个“看起来合理”的接口，而不一定符合你们的预期（比如它只实现了三个独立函数，没有统一封装一个总入口）。

👉 建议：在 PRD 里加一小节“4.0 模块总入口/接口设计”，显式给出函数签名和整体返回 JSON 的形态。

2. 示例数据时间范围不严谨（易误会）

在 4.2 的输入示例中：

- `parent_summary` 的时间是： 2025-08-01 ~ 2025-10-31
- 但 `view_return_snapshot` 里的 `snapshot_date` 却是 2025-11-02 和 2025-11-04。

同一个示例 JSON 中，时间范围不一致，对人来说可以理解为“只是示意”，但 Codex 可能会困惑：

“到底是 `parent_summary` 自己带时间区间，还是 `snapshot` 再过滤？二者是否必须一致？”

👉 建议：

- 要么让示例数据的日期也落在 2025-08-01 ~ 2025-10-31 内；
- 要么在示例上方加一句“以下示例仅为字段结构示例，日期不代表真实过滤关系”。

3. `review_source` 的定义与示例不太对得上

在 4.3 的逻辑里你写：

在打标事实表中筛选： `review_source = 0` （仅退货留言）

但 `view_return_fact_details` 的示例里 `review_source = 2`，旁边文案仍然在讲“退货原因”。

对人类来说可以理解为“示例随手写了个 2”，但对 Codex 来说就会出现分叉：

- 它可能会机械地按照规则写 `where review_source == 0`；
- 但又看到示例里退货样本是 `review_source = 2`，不确定 0/2 各代表什么。

👉 建议：

- 补充一行对 `review_source` 的枚举说明，比如：
 - 0 = 退货原因留言
 - 1 = 普通商品评价
 - 2 = 差评（星级 ≤ 2）
- 并把示例改成跟“本逻辑使用的是哪一类 source”保持一致（比如用 0）。

4. 除零 / 缺失值边界没写死，Codex 需要脑补

目前只有在 `text_coverage` 一处提到“若分母为 0 则置为 0”。其他地方涉及除法时，未明确边界处理，比如：

- `return_rate_parent = total_units_returned_parent / total_units_sold_parent`
如果父体在该时间段没卖出去 (`units_sold = 0`)，怎么处理？
- `return_rate_asin = units_returned_asin / units_sold_asin`
对零销量 ASIN 怎么定义？跳过 / 置 0 / 置 null？

Codex 很可能会：

- 要么直接写 Python 除法，产生 `ZeroDivisionError`；
- 要么自作聪明地把 0 销量的退货率设为 0，但这跟你们业务预期不一定一致。

👉 建议：统一补一句通用规则，例如：

若分母为 0，则对应比率字段统一置为 null，并在输出 JSON 中保留字段但值为 null；下游在展示时可特殊处理。

或：

若分母为 0，则退货率等比率统一置 0，视为“无法评估 / 可忽略”，但不参与 A/B 判定。

写清一版即可。

5. 阈值配置是“硬编码常量”还是“可配置项”

PRD 中给出的阈值很多 (`R_warn`、`+0.02`、`sales_share ≥ 0.10`、`units_returned ≥ 10`、`text_sample_count ≥ 30` 等)，都明确标了“默认值”。

但没有明确说明：

- 这些阈值是：
 - 写死在代码里？
 - 还是通过配置文件 / 环境变量传入？
 - 还是在函数参数里给一个 `config dict`？

Codex 通常会倾向于写死在代码里（常量），这样未来调节阈值就得改代码。

👉 建议：加一句统一约定，比如：

所有带“默认值”的阈值，需在代码中抽象为可配置参数（例如 `config.yaml` 或函数参数），不要在逻辑中直接写死数字。

6. 多站点 / 多父体的扩展边界

PRD 多处强调“本期聚焦单父体、单站点”，但同时也提到：

框架对父 ASIN / 站点 / 品类透明，只要输入数据结构满足约定即可复用。

对 Codex 来说，这里有两个可能的理解：

1. 只实现单个 (country, fasin) 的入口，批量由调用方循环；
2. 直接实现对多个 (country, fasin) 的批量处理。

目前文字里更偏向 1，但没明确写“**本模块的接口只支持一个国家 + 一个父 ASIN**”。

👉 **建议：**在接口设计里明确一句：

本期 Python 模块仅接受单一 country + fasin + 时间范围 作为输入，批量分析由调度系统在上游循环调用完成。

这样 Codex 不会擅自扩展成“输入 list 的国家和父体”。

三、文档内部是否有“前后矛盾”的硬问题？

从规则逻辑上看：没有明显的硬矛盾，主要是上面提到的示例/细节不严谨。再具体说几个你可能关心的点：

1. A/B 类定义 vs 2.2 里对分析对象的描述

- 2.2 里写：“原因分析只针对主战场款和高退货问题款，其他长尾 ASIN 仅做结构展示”；
 - 4.3 的步骤 1 的确只对 problem_class in ("A", "B") 做原因拆解，high_return_watchlist 不会被纳入。
- 二者是对得上的，没有矛盾。

2. 小体量高退货观察对象的处理

- 4.2 中说明这类 ASIN：high_return_watchlist = true 且 problem_class = null；
 - 4.3 中构建问题 ASIN 清单只看 problem_class，也就不会把观察名单拉进严肃的原因拆解。
- 逻辑一致，符合“观察但不作为正式问题 ASIN”的定义。

3. 标签维表 vs 事实表的使用

- 3.2.1 定义了 return_dim_tag，3.2.2 的事实表示例里已经带有 tag_name_cn；
- 4.3 的逻辑完全可以只用事实表中的 tag_name_cn 字段，无需在 Python 模块里 join dim 表。

- 这不算冲突，只是 dim 表目前更偏向数据仓库层面的设计，Python 这层暂时不必使用。

四、如果要让 Codex “一遍就写得比较对”，可以加的几个小增强

这些不是必要条件，但会明显减少“模型发挥”的空间：

1. 显式给出 1–2 个端到端输入/输出示例

- 比如给一段小数据（2 个子 ASIN、各 3 天的 snapshot，加上几条打标结果），再给出最终 `asin_structure` 和 `problem_asin_reasons` 的完整 JSON 输出。
- 这样 Codex 可以对照例子写单元测试，生成的代码更贴近你的预期。

2. 补一个整体的函数签名与模块结构建议

- 如：`core.py` 中有 `compute_parent_summary` / `compute_asin_structure` / `compute_problem_reasons` 三个函数，以及一个组合的 `run_pipeline`。
- 这样它不会随意拆文件结构。

3. 说明日期字段的类型约定

- `start_date` / `end_date` 用 YYYY-MM-DD 字符串；
- `review_date` 是 datetime 字符串（保留时分秒），过滤时按日期还是精确到秒；
- 这些对 Python 解析和比较会有影响。

总结一句

- **结论：**这份 PRD 的业务逻辑和计算规则已经相当清晰，给人类工程师或者 Codex 写一版 Python 模块是可行的，不存在关键逻辑上的自相矛盾。
- **风险点**主要集中在：少量示例字段（时间范围、`review_source`）、边界条件（除零）和整体接口形态未完全写死，这些都会让大模型有一定“自由发挥空间”。

如果你愿意，我可以按现在这份 PRD，直接帮你起一版「建议的 Python 模块接口与代码骨架」，顺手把这些边界条件都固化进去。

来源



请你帮我设计ETL流程，“3. 视图/表结构”从DORIS获取，4.1, 4.2, 4.3的核心计算逻辑分成3个py实现，4.1, 4.2,

4.3输出的JSON要入库DORIS，此外，参数配置统一在 config.py进行，最后需要有把所有步骤串联到一起的 pipeline.py，可能需要有日期处理函数之类的工具函数，统一放到calculator.py

已思考 53s >

我直接帮你把这一套 ETL 的“蓝图”和代码骨架都设计出来，你可以按需细化实现细节。

一、整体 ETL 流程概览

数据流向：

1. Extract:

- 从 Doris 读取 3.x 定义的视图 / 表：
 - view_return_snapshot
 - view_return_fact_details
 - return_dim_tag (如果 4.3 需要拉中文定义/边界)

2. Transform (Python)：拆成三个核心模块：

- 4.1 → parent_summary.py
- 4.2 → asin_structure.py
- 4.3 → problem_reasons.py

所有阈值、开关统一从 config.py 读取；共用工具函数从 calculator.py 引用。

3. Load:

- 4.1、4.2、4.3 的 JSON 结果写回 Doris (各一张结果表，或共用一张分 type 存储)。

4. Pipeline 编排：

- pipeline.py 串联全部步骤：解析参数 → 计算日期区间 → 调 Doris → 调各 4.x 模块 → 写回 Doris → 日志 & 监控。
-

二、Doris 侧表设计 (输入 & 输出)

1. 输入视图 (简化说明)

你们 PRD 里已经定义，这里只把关键字段列一下，方便和代码对齐：

- view_return_snapshot
 - country_code

- fasin (父体)
- asin
- snapshot_date
- units_sold
- units_returned
- ... (其他维度/指标)
- view_return_fact_details
 - country_code
 - fasin
 - asin
 - review_id
 - review_source
 - review_date
 - tag_code
 - tag_name_cn
 - sentiment
 - evidence
 - ... (其他字段)
- return_dim_tag
 - tag_code
 - tag_name_cn
 - tag_level1_name
 - tag_level2_name
 - ...

2. 输出结果表 (存 JSON)

假设用 Doris 的 JSON 或 STRING 存储, 给一个参考 DDL (可按你们规范调整) :

sql

复制代码

-- 4.1 父体整体指标

```
CREATE TABLE IF NOT EXISTS dwd_return_parent_summary_json (
    country_code  VARCHAR(8),
    fasin         VARCHAR(32),
    start_date    DATE,
    end_date      DATE,
    run_date      DATE,
    result_json   JSON -- 或 STRING, 按你们 Doris 版本决定
)
```

```
UNIQUE KEY(country_code, fasin, start_date, end_date)
DISTRIBUTED BY HASH(fasin) BUCKETS 10;
```

-- 4.2 子 ASIN 结构

```
CREATE TABLE IF NOT EXISTS dwd_return_asin_structure_json (
    country_code  VARCHAR(8),
    fasin         VARCHAR(32),
    start_date    DATE,
    end_date      DATE,
    run_date      DATE,
    result_json   JSON
)
UNIQUE KEY(country_code, fasin, start_date, end_date)
DISTRIBUTED BY HASH(fasin) BUCKETS 10;
```

-- 4.3 问题 ASIN 核心原因

```
CREATE TABLE IF NOT EXISTS dwd_return_problem_reasons_json (
    country_code  VARCHAR(8),
    fasin         VARCHAR(32),
    start_date    DATE,
    end_date      DATE,
    run_date      DATE,
    result_json   JSON
)
UNIQUE KEY(country_code, fasin, start_date, end_date)
DISTRIBUTED BY HASH(fasin) BUCKETS 10;
```

Load 端建议用 REPLACE INTO 或 INSERT ... ON DUPLICATE KEY UPDATE , 保证同一
(country, fasin, start_date, end_date) 的结果可重复跑/覆盖。

三、Python 项目结构设计

建议工程目录结构参考：

text

 复制代码

```
etl_return_analysis/
├── config.py          # 全局配置（DB、阈值等）
├── calculator.py      # 公共工具函数（日期、安全除法、coverage等）
├── db.py               # Doris 连接 & 通用查询/写入
├── parent_summary.py  # 4.1 核心逻辑
└── asin_structure.py  # 4.2 核心逻辑
```

```

└── problem_reasons.py      # 4.3 核心逻辑
└── pipeline.py              # 串联入口
└── requirements.txt         # 依赖

```

下面给出每个文件的职责 + 代码骨架。

四、 config.py – 配置中心

python

 复制代码

```

# config.py

from dataclasses import dataclass

@dataclass
class DorisConfig:
    host: str = "doris-host"
    port: int = 9030
    user: str = "etl_user"
    password: str = "*****"
    database: str = "return_db"

@dataclass
class ThresholdConfig:
    # 4.1 / 4.2 阈值（按你的 PRD 填）
    RETURN_RATE_WARN: float = 0.10
    RETURN_RATE_GAP_B: float = 0.02
    SALES_SHARE_MAIN: float = 0.10
    RETURN_UNITS_MIN: int = 10
    # R_warn
    # R_high_B = max(R_parent, R_warn) + GAP
    # 主战场销量占比阈值
    # 高退货问题款最小退货件数

    # 4.3 文本样本 & 覆盖率阈值
    TEXT_SAMPLE_MIN: int = 30
    TEXT_COVERAGE_HIGH: float = 0.80
    TEXT_COVERAGE_MEDIUM: float = 0.50
    # 最小文本样本数
    # 高置信度覆盖率阈值
    # 中置信度覆盖率阈值

@dataclass
class JobConfig:
    default_window_days: int = 90
    enable_debug_log: bool = True
    # 默认分析窗口，如最近90天

doris_config = DorisConfig()

```

```
threshold_config = ThresholdConfig()
job_config = JobConfig()
```

后续所有模块都通过 `from config import doris_config, threshold_config` 来拿配置，避免 hard-code 常量。

五、calculator.py – 通用工具函数

python

 复制代码

```
# calculator.py

from datetime import date, datetime, timedelta
from typing import Optional, Tuple

from config import job_config

def parse_date(date_str: str) -> date:
    """'2025-11-01' -> date对象"""
    return datetime.strptime(date_str, "%Y-%m-%d").date()

def get_date_range(biz_date_str: str,
                  window_days: Optional[int] = None) -> Tuple[date, date]:
    """
    业务日期 + 分析窗口天数 -> [start_date, end_date]
    默认 end_date = biz_date, start_date = biz_date - window_days + 1
    """
    window_days = window_days or job_config.default_window_days
    biz_date = parse_date(biz_date_str)
    start_date = biz_date - timedelta(days=window_days-1)
    end_date = biz_date
    return start_date, end_date

def safe_div(numerator: float,
            denominator: float,
            default: Optional[float] = None) -> Optional[float]:
    """统一除零逻辑"""
    if denominator is None or denominator == 0:
        return default
    return numerator / denominator
```

```
def percent(value: Optional[float]) -> Optional[float]:
    """把比例*100，方便输出，如果是None直接返回None"""
    if value is None:
        return None
    return value * 100.0
```

以后所有退货率、覆盖率的计算都走 `safe_div`，边界行为统一。

六、db.py – Doris 连接与通用读写

python

 复制代码

```
# db.py

import pymysql
import json
from typing import Any, Dict, List, Tuple
from contextlib import contextmanager

from config import doris_config

@contextmanager
def get_conn():
    conn = pymysql.connect(
        host=doris_config.host,
        port=doris_config.port,
        user=doris_config.user,
        password=doris_config.password,
        database=doris_config.database,
        cursorclass=pymysql.cursors.DictCursor,
    )
    try:
        yield conn
    finally:
        conn.close()

def query(sql: str, params: Tuple = ()) -> List[Dict[str, Any]]:
    with get_conn() as conn:
        with conn.cursor() as cursor:
            cursor.execute(sql, params)
    return cursor.fetchall()
```

```

def execute(sql: str, params: Tuple = ()) -> None:
    with get_conn() as conn:
        with conn.cursor() as cursor:
            cursor.execute(sql, params)
    conn.commit()

def insert_json(table: str,
               country_code: str,
               fasin: str,
               start_date: str,
               end_date: str,
               run_date: str,
               result: Dict[str, Any]) -> None:
    """统一写 JSON 到 Doris"""
    json_str = json.dumps(result, ensure_ascii=False)
    sql = f"""
        REPLACE INTO {table}
        (country_code, fasin, start_date, end_date, run_date, result_json)
        VALUES (%s, %s, %s, %s, %s, %s)
    """
    execute(sql, (country_code, fasin, start_date, end_date, run_date, json_str))

```

七、parent_summary.py (4.1 模块)

python

复制代码

```

# parent_summary.py

from typing import Dict, Any
from datetime import date

from config import threshold_config
from calculator import safe_div
from db import query

def compute_parent_summary(country_code: str,
                           fasin: str,
                           start_date: date,
                           end_date: date) -> Dict[str, Any]:
    """

```

4.1 父体整体指标计算:

- 从 `view_return_snapshot` 聚合
- 按 PRD 公式计算退货率等

返回一个 `dict`, 对应 4.1 输出 JSON 的结构

```

"""
sql = """
SELECT
    SUM(units_sold)      AS total_units_sold_parent,
    SUM(units_returned) AS total_units_returned_parent
FROM view_return_snapshot
WHERE country_code = %
    AND fasin = %
    AND snapshot_date BETWEEN %s AND %s
"""

rows = query(sql, (country_code, fasin, start_date, end_date))
agg = rows[0] if rows else {"total_units_sold_parent": 0, "total_units_returned_parent": 0}

units_sold = agg["total_units_sold_parent"] or 0
units_returned = agg["total_units_returned_parent"] or 0
return_rate_parent = safe_div(units_returned, units_sold, default=None)

# 根据 PRD 中 R_warn / R_high_B 等规则, 可以计算出父体状态(这里仅示意)
R_warn = threshold_config.RETURN_RATE_WARN
R_high_B = max(return_rate_parent or 0.0, R_warn) + threshold_config.RETURN_RATE_HIGH_B

result = {
    "country_code": country_code,
    "fasin": fasin,
    "start_date": str(start_date),
    "end_date": str(end_date),
    "total_units_sold_parent": units_sold,
    "total_units_returned_parent": units_returned,
    "return_rate_parent": return_rate_parent,
    "R_warn": R_warn,
    "R_high_B": R_high_B,
    # ... 其他在 4.1 中定义的字段
}
return result

```

八、asin_structure.py (4.2 模块)

python

 复制代码

```
# asin_structure.py

from typing import Dict, Any, List
from datetime import date

from config import threshold_config
from calculator import safe_div
from db import query

def compute_asin_structure(country_code: str,
                           fasin: str,
                           start_date: date,
                           end_date: date,
                           parent_summary: Dict[str, Any]) -> Dict[str, Any]:
    """
    4.2 子 ASIN 结构:
    - 从 view_return_snapshot 按 asin 维度聚合
    - 计算销量份额、退货份额、退货率
    - 根据 PRD 规则标注 problem_class / high_return_watchlist
    返回一个 dict, 对应 4.2 输出 JSON (包含 asin 列表)
    """

    sql = """
        SELECT
            asin,
            SUM(units_sold) AS units_sold_asin,
            SUM(units_returned) AS units_returned_asin
        FROM view_return_snapshot
        WHERE country_code = %s
            AND fasin = %s
            AND snapshot_date BETWEEN %s AND %s
        GROUP BY asin
    """
    rows = query(sql, (country_code, fasin, start_date, end_date))

    total_units_sold_parent = parent_summary["total_units_sold_parent"] or 0
    total_units_returned_parent = parent_summary["total_units_returned_parent"] or 0
    R_parent = parent_summary["return_rate_parent"] or 0.0

    asin_list: List[Dict[str, Any]] = []
    for r in rows:
        units_sold = r["units_sold_asin"] or 0
        units_returned = r["units_returned_asin"] or 0
        return_rate_asin = safe_div(units_returned, units_sold, default=None)
```

```

sales_share = safe_div(units_sold, total_units_sold_parent, default=0.0)
returns_share = safe_div(units_returned, total_units_returned_parent, defi

# === 按 PRD 设定 problem_class / high_return_watchlist ====
problem_class = None
high_return_watchlist = False

# 主战场 + 高退货问题款等逻辑（这里只写示意，细节按 PRD 规则补全）
if units_returned >= threshold_config.RETURN_UNITS_MIN and \
    (return_rate_asin or 0.0) >= (R_parent + threshold_config.RETURN_RATE_
        problem_class = "A"
# ... B 类、观察名单判定逻辑

asin_list.append({
    "asin": r["asin"],
    "units_sold": units_sold,
    "units_returned": units_returned,
    "return_rate_asin": return_rate_asin,
    "sales_share": sales_share,
    "returns_share": returns_share,
    "problem_class": problem_class,
    "high_return_watchlist": high_return_watchlist,
})

return {
    "country_code": country_code,
    "fasin": fasin,
    "start_date": str(start_date),
    "end_date": str(end_date),
    "parent_summary": parent_summary,
    "asin_list": asin_list,
}

```

九、problem_reasons.py (4.3 模块)

python

复制代码

```

# problem_reasons.py

from typing import Dict, Any, List
from datetime import date

```

```

from config import threshold_config
from calculator import safe_div
from db import query

def compute_problem_reasons(country_code: str,
                            fasin: str,
                            start_date: date,
                            end_date: date,
                            asin_structure: Dict[str, Any]) -> Dict[str, Any]:
    """
    4.3 问题 ASIN 核心原因:
    - 从 asin_structure 中取出 problem_class in (A,B) 的 ASIN 列表
    - 到 view_return_fact_details 取文本打标样本
    - 统计 review_id 去重数、标签 coverage, 计算 high/medium/low 置信度
    - 每个 ASIN 输出 0~3 个核心原因标签
    """
    problem_asins = [
        a["asin"] for a in asin_structure["asin_list"]
        if a["problem_class"] in ("A", "B")
    ]
    if not problem_asins:
        return {
            "country_code": country_code,
            "fasin": fasin,
            "start_date": str(start_date),
            "end_date": str(end_date),
            "problem_asins": [],
        }
    # 拉打标明细
    sql = f"""
        SELECT
            asin,
            review_id,
            tag_code,
            tag_name_cn
        FROM view_return_fact_details
        WHERE country_code = %s
            AND fasin = %s
            AND review_source = 0 -- 仅退货留言, 具体值按你们定义
            AND review_date BETWEEN %s AND %
            AND asin IN ({",".join(["%s"] * len(problem_asins))})
    """

```

```
"""
params = [country_code, fasin, start_date, end_date] + problem_asins
rows = query(sql, tuple(params))

# 组装: asin -> {review_ids, tag -> review_ids}
asin_to_reviews: Dict[str, set] = {}
asin_tag_reviews: Dict[str, Dict[str, set]] = {}

for r in rows:
    asin = r["asin"]
    review_id = r["review_id"]
    tag_code = r["tag_code"]
    tag_name_cn = r["tag_name_cn"]

    asin_to_reviews.setdefault(asin, set()).add(review_id)
    tag_map = asin_tag_reviews.setdefault(asin, {})
    tag_map.setdefault(tag_code, set()).add(review_id)

problem_asin_results: List[Dict[str, Any]] = []
for asin in problem_asins:
    review_ids = asin_to_reviews.get(asin, set())
    text_sample_cnt = len(review_ids)

    if text_sample_cnt == 0:
        problem_asin_results.append({
            "asin": asin,
            "text_sample_count": 0,
            "text_coverage": 0.0,
            "confidence_level": "low",
            "core_reasons": [],
        })
        continue

    # 计算标签覆盖率
    tag_map = asin_tag_reviews.get(asin, {})
    tag_coverages: List[Dict[str, Any]] = []
    for tag_code, tag_review_ids in tag_map.items():
        coverage = safe_div(len(tag_review_ids), text_sample_cnt, default=0.0)
        tag_coverages.append({
            "tag_code": tag_code,
            "tag_name_cn": next(
                (r["tag_name_cn"] for r in rows
                 if r["asin"] == asin and r["tag_code"] == tag_code),
            )
        })

problem_asin_results.extend(tag_coverages)
```

```

        None,
    ),
    "coverage": coverage,
})

# 按 coverage 排序, 选择前3, 且 >= 阈值
tag_coverages.sort(key=lambda x: x["coverage"], reverse=True)
core_reasons = [
    t for t in tag_coverages
    if t["coverage"] >= threshold_config.TEXT_COVERAGE_HIGH
][:3]

# 样本覆盖率 & 置信度 (示意, 可按 PRD 详细规则拆 high/medium/Low)
text_coverage = safe_div(len(review_ids), text_sample_cnt, default=1.0) :
if text_sample_cnt >= threshold_config.TEXT_SAMPLE_MIN and \
    text_coverage >= threshold_config.TEXT_COVERAGE_HIGH:
    confidence = "high"
elif text_sample_cnt >= threshold_config.TEXT_SAMPLE_MIN and \
    text_coverage >= threshold_config.TEXT_COVERAGE_MEDIUM:
    confidence = "medium"
else:
    confidence = "low"

problem_asin_results.append({
    "asin": asin,
    "text_sample_count": text_sample_cnt,
    "text_coverage": text_coverage,
    "confidence_level": confidence,
    "core_reasons": core_reasons,
})

return {
    "country_code": country_code,
    "fasin": fasin,
    "start_date": str(start_date),
    "end_date": str(end_date),
    "problem_asins": problem_asin_results,
}

```

这里 core_reasons / confidence 的具体规则, 你可以按 PRD 精细化 (比如 high/medium/low 三档的边界、标签 coverage ≥ 0.8 才进入核心原因等)。

十、 pipeline.py – 串联所有步骤

python

复制代码

```
# pipeline.py

import argparse
from datetime import date

from calculator import get_date_range
from parent_summary import compute_parent_summary
from asin_structure import compute_asin_structure
from problem_reasons import compute_problem_reasons
from db import insert_json
from datetime import datetime

def parse_args():
    parser = argparse.ArgumentParser(description="Return analysis ETL pipeline")
    parser.add_argument("--country", required=True, help="国家/站点, 如 US/JP/DE")
    parser.add_argument("--fasin", required=True, help="父 ASIN")
    parser.add_argument("--biz-date", required=True,
                        help="业务日期, 格式 YYYY-MM-DD, 通常为批次日期/昨日")
    parser.add_argument("--window-days", type=int, default=None,
                        help="分析窗口天数, 默认使用 config 中配置")
    return parser.parse_args()

def main():
    args = parse_args()
    country_code = args.country
    fasin = args.fasin
    start_date, end_date = get_date_range(args.biz_date, args.window_days)
    run_date = datetime.now().date()

    # Step 1: 4.1 父体整体指标
    parent_summary = compute_parent_summary(country_code, fasin, start_date, end_
    insert_json(
        table="dwd_return_parent_summary_json",
        country_code=country_code,
        fasin=fasin,
        start_date=str(start_date),
        end_date=str(end_date),
        run_date=str(run_date),
        result=parent_summary,
    )
```

```

# Step 2: 4.2 子 ASIN 结构
asin_structure = compute_asin_structure(
    country_code, fasin, start_date, end_date, parent_summary
)
insert_json(
    table="dwd_return_asin_structure_json",
    country_code=country_code,
    fasin=fasin,
    start_date=str(start_date),
    end_date=str(end_date),
    run_date=str(run_date),
    result=asin_structure,
)
print("Pipeline done",
      country_code, fasin, start_date, end_date, "run_date=", run_date)

if __name__ == "__main__":
    main()

```

外部调度系统 (Airflow、Shell cron 等) 只要调用:

python pipeline.py --country US --fasin XXX --biz-date 2025-11-26 即可。

十一、总结一下设计要点

- **Doris 只做存储 & 聚合：**

视图/表 (3.x) 在 Doris 中事先建好，Python 负责读取和业务规则计算。

- **业务逻辑模块化：**

- parent_summary.py : 4.1 父体聚合与指标
- asin_structure.py : 4.2 子 ASIN 结构与 problem_class
- problem_reasons.py : 4.3 文本原因拆解

- **配置收口：**所有阈值和 DB 信息在 config.py 统一维护。

- **工具函数归一：**日期、除零等通用细节集中在 calculator.py，保证一致性。

- **输出 JSON 入库 Doris：**通过 db.insert_json 统一写入，方便下游 BI / API 直接消费结果表。

如果你愿意，下一步我可以帮你：

- 按你 PRD 的具体字段补全 4.1 / 4.2 / 4.3 的字段清单；
- 或者把上面的骨架进一步改写成“可直接跑通的 demo”，比如只针对一个国家 + 一个父体做端到端测试。



↳ 一、整体 ETL 流程概览 ----- **数据流向：** 1. **Extract**： * 从 Doris 读取 3.x 定义的视图 / 表： * `view_return_snapshot` * `view_return_fact_details` * `return_dim_tag` (如果 4.3 需要拉中文定义/边界) 2. **Transform (Python)**：拆成三个核心模块： * `4.1` → ...

把整体ETL流程概览写到README中，归属到5.ETL流程

已思考 1m 0s >

我已经把“整体 ETL 流程概览”写进 README 里了，现在新增了一节：

markdown

复制代码

5. ETL 流程

5.1 整体流程概览

数据流向：

1. **Extract** (抽取) :

- 从 Doris 读取第 3 章定义的视图 / 表：
 - `view_return_snapshot`
 - `view_return_fact_details`

- `return_dim_tag` (如果 4.3 需要拉中文定义/边界)

2. **Transform (Python 计算):

- 将核心计算逻辑拆分为三个独立的 Python 模块:
 - `4.1` → `parent_summary.py` (父体整体指标计算)
 - `4.2` → `asin_structure.py` (子 ASIN 结构与问题 ASIN 识别)
 - `4.3` → `problem_reasons.py` (问题 ASIN 核心原因拆解)
- 所有阈值、开关与运行参数统一在 `config.py` 中配置;
- 日期处理、安全除零、覆盖率计算等通用工具函数统一收敛在 `calculator.py` 中复用。

3. **Load** (入库) :

- 将 4.1、4.2、4.3 的 **JSON** 结果**写回 Doris:
 - 每个模块对应一张结果表 (或统一表, 按 `result_type` 区分), 方便下游 BI / 应用端查询
 - 采用主键覆盖写入 (如 `REPLACE INTO` 或 `INSERT ... ON DUPLICATE KEY UPDATE`)

4. **Pipeline 编排**:

- 通过 `pipeline.py` 串联上述所有步骤, 形成可调度的批处理作业:
 - 解析外部传入参数 (如 `country`、`fasin`、`biz_date`、`window_days` 等);
 - 调用 `calculator.py` 计算本次任务的分析时间窗口 (`start_date`、`end_date`);
 - 按顺序调用 `parent_summary.py`、`asin_structure.py`、`problem_reasons.py` 完成各阶段逻辑;
 - 将各阶段输出 JSON 统一写回 Doris 对应结果表;
 - 预留日志记录、告警与简单监控能力, 方便后续接入 Airflow / Shell Cron 等调度系统。

位置：插在原文最后一段内容之前，也就是现在结构是：

- 1 背景
- 2 本期范围
- 3 视图 / 表结构
- 4 Python 计算模块
- **5 ETL 流程**
- 最后的“(完) ”

如果你想在 5 下面再加 5.2 「Doris 结果表设计」、5.3 「调度示例 (Airflow/Shell) 」之类，也可以在这个节下面继续往下扩展。



↳ 建议工程目录结构参考：

把这个写成md格式

已思考 8s >

markdown

复制代码

5.2 工程目录结构

项目建议目录结构如下：

```
```text
etl_return_analysis/
├── config.py # 全局配置（DB、阈值、运行参数等）
├── calculator.py # 公共工具函数（日期处理、安全除法、coverage 计算等）
├── db.py # Doris 连接 & 通用查询/写入封装
├── parent_summary.py # 4.1 父体整体指标计算逻辑
├── asin_structure.py # 4.2 子 ASIN 结构 & problem_class 判定
├── problem_reasons.py # 4.3 问题 ASIN 核心原因拆解
├── pipeline.py # ETL 编排入口（串联 4.1/4.2/4.3 并写回 Doris）
└── requirements.txt # Python 依赖列表
```

各文件职责说明：

- `config.py`：集中维护 Doris 连接信息、阈值配置（如退货率阈值、样本数阈值等）及任务级配置，供其他模块引用。
- `calculator.py`：存放通用工具函数，例如日期解析与时间窗口计算、安全除零函数、比例/覆盖率计算等，避免在各模块中重复实现。
- `db.py`：封装与 Doris 的交互，包括获取连接、执行查询以及将 JSON 结果写入 Doris 结果表的通用方法。
- `parent_summary.py`：实现 4.1 父体整体指标计算，从 `view_return_snapshot` 聚合父体层面销量/退货量等，并输出父体 summary JSON。
- `asin_structure.py`：实现 4.2 子 ASIN 结构分析，从 `view_return_snapshot` 计算子 ASIN 的销量/退货份额及退货率，并依据配置规则打标 `problem_class` 与 `high_return_watchlist`。
- `problem_reasons.py`：实现 4.3 问题 ASIN 核心原因分析，从 `view_return_fact_details` 读取文本打标样本，按 ASIN 统计标签覆盖率、样本充足性与置信度，并输出核心原因列表。
- `pipeline.py`：整体 ETL 流程入口，解析任务参数（国家、父 ASIN、业务日期等），调用 `calculator.py` 生成时间窗口，依次执行 4.1/4.2/4.3 计算并通过 `db.py` 将各阶段 JSON 结果写入 Doris。

- `requirements.txt`：记录运行所需第三方依赖（如 `pymysql` 等），便于环境初始化与部署