設計改善提案

ないものをある様に見せて、偽推定はしていない？

Thought for 35s >

結論：**偽推定はしない設計にします（“ないものは使わない＝400で返す”）**。
現状のあなたの環境で500が出ていた主因は「列不一致・可視化失敗」でしたが、**自動補完で“ある様に見せる”挙動は排除**し、下の“厳格モード”で監査可能にします。

3行テイクアウェイ

1. **Strict Data Contract**を既定にして、必須列が無ければ**即400**（推定は走らない）。
2. 「導出した値（exposure等）」は**導出台帳**に必ず記録し、**推定入力と区別**する。
3. CIで**偽推定を検知するpytest**を追加（fail時にPRブロック）。

目的 / 非目標

- **目的**：データが無いのに“ある体”で推定する挙動（暗黙の推測・自動補完）を**ゼロ**にする。
- **非目標**：推定器の理論変更やAPI破壊。既存UI・APIは維持しつつ**ガードを強化**。

“偽推定”に該当し得る典型と対処

- ❌ **treatment/IV/DiD/RD などのキー列の暗黙補完**（例：`received_discount_code` が無いので似た列を勝手に使う）
  → **即400**＋`available_columns` 返却。
- ❌ **propensityの暗黙学習**（`log_propensity`が無いときに黙って推定）
  → **明示フラグなしでは禁止**。`ALLOW_ESTIMATE_PROPENSITY=1` の時だけ実行し、**導出台帳に記録**。
- ❌ **IVの自動探索**、**RDカットオフ自動推定**、**treated_timeの類推**
  → 既定では**禁止**。与えられない限り**該当推定器はスキップ**。
- ⭕ **合法的な導出（例）**
  - `exposure` を `edges` から計算、`distance_km` を `lat/lon` から計算等。
  - これらは**導出**であり**捏造ではない**。ただし**導出台帳**に式・入力・出力列を記録して**可視化**。

設計境界（I/O・失敗・拡張）

- **I/O**：CSV/Parquet → **契約検証** → OKなら推定、NGなら**HTTP 400**。
- **失敗**：可視化やWolfram失敗は**非致死**（200＋`warnings`）、**入力不足は致死**（400）。
- **拡張**：`ALLOW_*` 環境変数で“明示許可の導出だけ”を解禁可能。

即運用できる運用規約（環境変数）

```
# 既定は全て 0 = 禁止（無ければ推定しない）
STRICT_DATA_CONTRACT=1
ALLOW_ESTIMATE_PROPENSITY=0      # ←1 にすると推定可（導出台帳必須）
ALLOW_DERIVE_EXPOSURE_FROM_EDGES=1
ALLOW_DERIVE_DISTANCE_FROM_GEO=1
REQUIRE_IV_Z=1
REQUIRE_RD_CUTOFF=1
REQUIRE_DID_T0=1
```

開発者向けチェック（コピペ可）

“偽推定の芽”の静的検査：

```
# 暗黙補完/推測を疑う語をgrep（例）
rg -n "(fallback|guess|infer|auto|default.*column|imput(e|ation))" backend/ -S

# 関数境界で 400 を返しているか（例：FastAPI）
rg -n "HTTPException\\(status_code=400" backend/ -S

# 台帳に記録しているか
rg -n "(derivation_ledger|provenance|audit)" backend/ -S
```

検証（pytest/メトリクス）— 偽推定防止テスト

```
# 1) 必須列欠落 → 400（推定は走らない）
pytest -q tests/test_contract_strict_400.py

# 2) propensity 未提供かつ ALLOW=0 → 推定しない（400/skip）
pytest -q tests/test_propensity_not_estimated_by_default.py

# 3) ALLOW=1 の時だけ導出し、導出台帳に記録される
pytest -q tests/test_propensity_derivation_ledger.py

# 4) IV/DiD/RD の必須列チェック（Z / treated_time / r,c）
pytest -q tests/test_iv_did_rd_requirements.py
```

“導出台帳（Derivation Ledger）”の中身（JSON例）

```
{
  "dataset_id": "demo",
  "generated_at": "2025-11-04T16:59:00Z",
  "derivations": [
    {"out":"exposure", "fn":"mean_treatment_neighborhood(k=3)",
     "inputs":["edges.parquet","treatment"], "rows": 8800},
    {"out":"log_propensity", "fn":"logit(t~X_*)",
     "inputs":["X_*"], "rows": 8800, "enabled_by":"ALLOW_ESTIMATE_PROPENSITY=1"}
  ]
}
```

- 推定で**直接食わせた列**と**導出列**を**可視化カード**にも表示します（監査性）。

What this does

Goal：**“ないものは使わない”をシステムに強制し、偽推定を不可能にする。**
Inputs：データ列、環境フラグ。
Outputs/Side-effects：必須列不足は400、導出は**台帳**付き、図失敗は**非致死**。
Why this approach：識別条件を破る暗黙補完は**統計的正当性を欠く**ため、**明示許可＋監査ログ**以外を禁止。
How to run/verify：上の `rg` / `pytest` と、APIの**HTTPコード・Decision Cardの証跡**で機械的に検証。

Expert insight (Google/Meta/NASAレベル)

- **“No data, no model” をプラットフォームで強制**すると、**現場のショートカットを制度的に防止**できます。
  導出は**台帳**で透明化し、**入力不足＝400**を徹底することで、**再現性・監査性・信頼性**が飛躍的に上がります。