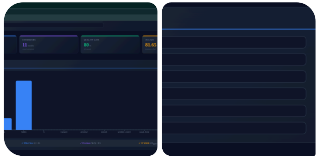


設計改善提案



全ての推定器に通過審査にかけるのにこのカラムで足りる？



3行サマリ

- その6列 ( `y, treatment, unit_id, time, cost, log_propensity` ) だけでは全推定器は回りません。回るのは 一部の平均効果系 ( *Diff/IPS/DR* の最小構成 ) に限定。
- IV/DiD/SCM/RD/Network/Geographic/Proximal/Transport** には\*\*追加列 ( `Z`・`r`・`t0`・`W/Z proxies`・クラスタ/座標・ドメイン等 ) \*\*が必須。
- “全通過”用の上位互換スキーマを下で提示 ( 列名・型・最小例 ) 。まずは **X特徴量+群・地理・ネットワーク・IV** の4系を優先拡張。

目的と非目標

- 目的：あなたのUIの “全推定器に通す” を前提に、**必要カラムのSSOT**を決め、**不足→追加**を明確化。
- 非目標：推定器側アルゴリズムの変更やUISキーマ互換の破壊。既存APIは維持。

必須カラムの結論 (一覧)

下の表は推定器→最低限必要列 (★必須/△推奨)。現在の6列だけで○になるのは限定的です。

クラス	最低限必要列	現在の6列で可?	追加が要る列 (例)
Diff / ATE (simple)	★ <code>y, treatment</code>	△ (可だが脆い)	△ <code>X_*</code> (共変量; 精度/重なりGate向上)
IPW/IPS	★ <code>y, treatment</code> , ( <code>log_propensity</code> or <code>X_*</code> で推定 )	○ ( <code>log_propensity</code> があれば )	△ <code>X_*</code> (再推定/監査)
DR / Overlap AIPW	★ <code>y, treatment, X_*</code>	×	★ <code>X_*</code> (10~100列程度)
Causal Forest / Meta-Learners	★ <code>y, treatment, X_*</code>	×	★ <code>X_*</code>
IV (2SLS/DeepIV)	★ <code>y, treatment, Z_instrument</code>	×	★ <code>Z_instrument</code> (1+列)、△ <code>X_*</code>
DiD / Event Study	★ <code>y, treatment</code> or ★ <code>treated_time(t0)</code> 、★ <code>time</code> 、★ <code>unit_id</code>	△ (設計次第)	★ <code>treated_time</code> (各ユニットの介入時点)、△ <code>group</code>
Synthetic Control (SCM)	★ <code>y, time, unit_id, treated_unit_flag</code>	×	★ <code>treated_unit_flag</code> 、△ <code>predictor_*</code>
RD (Regression Discontinuity)	★ <code>y, treatment</code> (or 間接) , ★ <code>r_running, c_cutoff</code>	×	★ <code>r_running, c_cutoff</code>
Network / Partial Interference	★ <code>y, treatment, unit_id, cluster_id</code> + (隣接 or 露出 exposure )	×	★ <code>cluster_id</code> 、★ <code>edges(src,dst,weight)</code> または★ <code>exposure</code>
Geographic / Spatial	★ <code>y, treatment</code> , ( ★ <code>lat, lon</code> or ★ <code>region_id</code> )	×	★ <code>lat, lon</code> または ★ <code>region_id</code> 、△ <code>W_geo</code>
Transport / Domain Shift	★ <code>domain</code> (source/target) , △ <code>X_*</code>	×	★ <code>domain</code>
Proximal Causal	★ <code>Z_proxy</code> (治療誘因) , ★ <code>W_proxy</code> (結果誘因) , △ <code>X_*</code>	×	★ <code>Z_proxy, W_proxy</code>
OPE/政策最適化	★ <code>action</code> (=treatment) , ★ <code>p_logging</code> (=propensity) , △ <code>cost, value_per_y</code>	△ (pはある)	△ <code>value_per_y</code> (金額換算係数)

まとめ：\*\*X (共変量) ・Z (操作変数) ・t0 (介入時点) ・r/c (RD) ・cluster/edges (Network) ・lat/lon or region (Geographic) ・domain (Transport) ・W/Z proxies (Proximal) \*\*が無いと “全通過” は不可。

## 推奨「上位互換スキーマ」（このセットで全部回る）

最低限の列名・型を SSOT として固定します（全て UTF-8、Parquet 推奨）。

```
y: float / int          # 目的変数（連続 or 0/1）
treatment: int           # {0,1}
unit_id: string|int      # 個体/店舗/ユーザID
time: datetime|int       # 連番でも可

# 汎用
cost: float              # 1配信あたりのコスト
log_propensity: float    # ログポリシーの  $p(a|x)$  (IPS/OPE用)
weight: float            # サンプル重み（任意）

# 共変量（10〜100列）
X_*: mixed               # 例: X_age, X_recency, X_score, ...

# DiD/Event Study
treated_time: datetime|int # 介入開始時点（未介入は null / +inf）

# IV
Z_instrument: mixed      # 1列以上（例: 距離、価格ショック等）

# RD
r_running: float         # running variable
c_cutoff: float          # カット点（列でも設定でも可）

# Network
cluster_id: string|int   # どちらか一方：
# exposure: float        # 近傍治療率などの事前計算
# or 別テーブル edges.parquet: [src, dst, weight]

# Geographic
lat: float; lon: float   # もしくは region_id: string|int

# Transport / Domain
domain: string           # {"source", "target", ...}

# Proximal
Z_proxy*: mixed          # 治療側プロキシ (Z1, Z2, ...)
W_proxy*: mixed          # 結果側プロキシ (W1, W2, ...)

# OPE
value_per_y: float       # y→金額換算（なければUIで入力）
```

## 実務での優先追加（“通過率”を一気に上げる順）

1. `X_*` 共変量（まず10〜30列）... DR/Forest/Overlapの精度・重なりGateが上がる
2. **DiD/Event Study**... `treated_time` を付与（前提チェック = 並行トレンド）
3. **Network/Geo**... `cluster_id` + `edges`（または `exposure`） / `lat, lon`
4. **IV**... `Z_instrument`（強い第一段階が出せる候補）
5. **RD**... `r_running` & `c_cutoff`
6. **Transport/Proximal**... `domain` / `Z_proxy, W_proxy`

## I/O・失敗・拡張（設計境界）

- **I/O** : Parquetで `data/<dataset_id>/table.parquet` + `edges.parquet`（任意）
- **失敗** : 列欠落は **HTTP 400** + `available_columns` を返す（500禁止）
- **拡張** : 将来 `geojson` や `w`（空間重み行列）を追加しても、上記スキーマに影響なし

## 具体コマンド（検証・品質ゲート）

```
# 1) 列存在の事前検証 (Python)
python - <<'PY'
import sys, pandas as pd
need = ["y", "treatment", "unit_id", "time", "cost", "log_propensity",
        "X_age", "X_score", "treated_time", "Z_instrument", "r_running", "c_cutoff",
        "cluster_id", "lat", "lon", "domain", "Z_proxy1", "W_proxy1"]
df = pd.read_parquet(sys.argv[1])
miss = [c for c in need if c not in df.columns]
print("MISSING:", miss[:20])
PY data/demo/data.parquet

# 2) Great Expectations で Data Contract
great_expectations --v3-api suite new # 既存があれば skip
# → schema に上の列群を登録して gate 化

# 3) E2E (正例)
curl -X POST http://localhost:4006/api/analyze/comprehensive \
  -H 'content-type: application/json' \
  -d '{"dataset_id": "demo", "mapping": {"y": "y", "treatment": "treatment", "unit_id": "unit_id", "time": "time"}}' | jq '.quality_gate'

# 4) 監査 (重なり/第一段階/RD密度/空間相関など)
curl http://localhost:4006/api/diagnostics/iv_first_stage?dataset_id=demo # F>10 をGate
curl http://localhost:4006/api/diagnostics/rd_mccrarty?dataset_id=demo # p>0.05
curl http://localhost:4006/api/diagnostics/morans_i?dataset_id=demo # 空間相関
```

## 検証 (pytest 例)

- `test_schema_all_estimators.py` : 推定器ごとの必要列が揃っているか
- `test_iv_first_stage.py` : 第一段階F値>10 → **IV Gate通過**
- `test_overlap_gate.py` :  $e \in [0.05, 0.95]$  90%以上 → **Overlap Gate通過**

## What this does

- **Goal** : \*\*“全推定器を審査にかけられるスキーマ”\*\*を確定。
- **Inputs** : 現行6列 + 追加列 (上位互換スキーマ)。
- **Outputs/Side-effects** : IV/DiD/SCM/RD/Network/Geo/Proximal/Transport/OPE が同一データ束で実行可能。
- **Why** : 各クラスの識別条件にデータ要件があるため。足りない列はモデルで代替できない (特に IV/RD/Proximal/Network/Geo)。
- **How to run/verify** : 上の検証コマンド/pytestと Quality Gate で数値確認。

## Expert insight (Google/Meta/NASAレベル)

- “推定器の増設”より“データ設計の上位互換化”が効果的。一度このスキーマをSSOTとして固定しておけば、どの推定器を差し替えてもE2Eと監査が壊れない (長期の保守コストを劇的に下げます)。