

CQOx_gen 拡張案②



もっと超詳細に文章もコードも書かないと伝わらない。



Thought for 37s >

了解。「どういう前提でどう生成されたデータか」がコードと文章で完全に共有されていないと、因果や分析の“射程”が持てないので、そこまで全部出します。

0. 全体像（何を渡している CSV か）

3行サマリ

- emotion_cqox_sample_5000.csv は「1行=1エピソード（面接/1on1/話し合い etc.）」の 5,000 行データで、準備の強度 → 当日のストレス・涙・伝達度 → 関係への影響までが一貫した生成モデルで結びついています。
- 欠損・キャンセル・未来予定・ユーザーごとの頻度差・準備の自己選択バイアスなど、現実に近い“汚さ”を意図的に入れてあり、因果推論・探索アルゴリズムの PoC にそのまま使える難易度になっています。
- 下に、列仕様・生成ロジック（日本語）・完全な Python スクリプト・CLI 手順・検証のための EDA コードまで載せるので、そのまま Claude に渡して「このモデルに沿って実装 or 改造して」と指示できるレベルにしています。

1. CSV の列仕様（超詳細）

1.1 基本情報

- ファイル名: emotion_cqox_sample_5000.csv
- 行数: 5,000（ヘッダ 1 行 + データ 4,999 行ではなく、4,999 ではなくちゃんと 5,000 行のデータ）
- 文字コード: UTF-8
- 区切り: カンマ
- 日付形式: YYYY-MM-DDTHH:MM:SS（タイムゾーンなしの ISO 8601）

1.2 列ごとの詳細定義

列名	型 (CSV 上)	意味 / 使い方	値の範囲・欠損
episode_id	int	1エピソードに1つ振られるID。単なる連番（1〜5000）。	欠損なし
user_id	int	ユーザー識別子。1〜30。1〜5番は「ヘビーユーザー」で出現頻度高め。	欠損なし
status	string	エピソードの状態。completed / planned / cancelled のいずれか。	欠損なし
scenario_type	string	エピソード種別。interview / one_on_one / partner / family / friend / client / other	欠損なし
topic	string (日本語)	話題カテゴリ。例:「転職理由」「別れ話」「介護の相談」など。scenario_type ごとに候補が異なる。	欠損なし
scheduled_at	string	予定日時（あるいは実施日時に近い時点）。today=2025-11-28 を基準に 過去2年〜未来3か月 の一様分布。	欠損なし
location	string	online, office, home, cafe, coworking, client_site, park のいずれか。	欠損なし
pre_anxiety	int	開催前の「不安度」（0〜10）。scenario_type に応じて平均値が違う（interview が最も高い）。	欠損なし
pre_crying_risk	int	開催前の「泣きそう度」（0〜10）。topic が「過去の退職理由」「別れ話」「距離を置きたい」「介護の相談」の場合は +1 シフト。	欠損なし
pre_speech_block_risk	int	開催前の「言葉が詰まりそう度」（0〜10）。pre_anxiety をベースにしたノイズ付き。	欠損なし
prep_journaling_10m_intensity	int or ""	準備1：10分の書き出し（ジャーナリング）をどれくらい真面目にやったか（0〜10）。0 の場合は 空文字 "" にしてある（＝「未実施」を欠損で表現）。	欠損あり（未実施は空）
prep_three_messages_intensity	int or ""	準備2：「伝えたいメッセージを3つ決める」強度。ロジックは journaling と同様。	欠損あり

列名	型 (CSV上)	意味 / 使い方	値の範囲・欠損
prep_breathing_4_7_8_intensity	int or ""	準備3：呼吸法。	欠損あり
prep_roleplay_self_qa_intensity	int or ""	準備4：自分で Q&A ロールプレイ。	欠損あり
prep_safe_word_plan_intensity	int or ""	準備5：セーフワード（つらくなった時に出す合図）を決める。partner / family / friend で少し実施率高め。	欠損あり
stress_during	int or ""	当日の「話している最中のつらさ」（0～10）。completed のみ値あり、それ以外は空。	欠損あり
stress_after	int or ""	当日の「終わった直後のつらさ」（0～10）。completed のみ。	欠損あり
crying_level	int or ""	涙の出方（0～10）。	欠損あり
speech_block_level	int or ""	言葉の詰まり具合（0～10）。	欠損あり
expression_score	int or ""	「本当に言いたかったことをどれくらい伝えられたか」（0～10）。準備の影響を強めに受ける。	欠損あり
relationship_impact	int or ""	関係への影響（-5～+5）。expression_score 高いとプラス、stress と泣きすぎが大きいとマイナス側に寄る。	欠損あり
partner_reaction	string or ""	相手の反応。very_positive / positive / neutral / negative / very_negative / unknown。relationship_impact にゆるく依存。	欠損あり
days_after_reflection	int or ""	振り返りを書いた日数（1～14）。反省を残していない Episode は空。	欠損あり
would_repeat_preparation	int or ""	「今回の準備をまたやりたいか」（0～10）。expression_score が高く、かつ泣きすぎていない時に高くなりやすい。	欠損あり
reflection_short	string or ""	日本語の短い振り返り文。テンプレ + 乱択。約70%のcompleted Episodeだけ埋まる。	欠損あり

ポイント：

- 数値でも「未実施」「未記録」は **0 ではなく空文字** にしてあります。
→ 現実の CSV でよくある「空セル=欠損」として扱う必要がある（前処理の難易度を上げるため）。
- status によってアウトカム列が系統的に欠損するので、**MNAR (Not Missing At Random)** っぽい欠損も含まれます。

2. 生成ロジックの中身（日本語で全部出す）

ここから、上の CSV を生成するための **確率モデル / 擬似因果構造** を文章で説明します。

2.1 ステップ0: グローバル設定

- today = 2025-11-28
- 行数 n = 5000
- ユーザー数 30 (user_id 1～30)
- 乱数シード random.seed(42) で再現可能。

2.2 ステップ1: エピソードごとの基本情報

1. user_id

- user_id ∈ {1, ..., 30} を重み付き乱択。
- 1～5番は重み 5、6～30番は重み 1 → ヘビーユーザー 5 人 + ロー freq ユーザー 25 人 という構造。

2. scenario_type

- 候補と重み：
 - interview : 0.25
 - one_on_one : 0.20
 - partner : 0.15
 - family : 0.10
 - friend : 0.10
 - client : 0.10
 - other : 0.10

3. topic

- `scenario_type` に応じて、上記の `topics_by_type` から一様ランダムに選択。
(interview なら「転職理由」「キャリアの方向性」「過去の退職理由」「評価面談」など)

4. `scheduled_at`

- `today` を基準に、
 - 日数オフセット: -730～+90 日 (過去2年～未来3か月)
 - 時刻: 8～22時台のどこか + 分0～59
- `scheduled_at = today + timedelta(days_offset, hours, minutes)` → `.isoformat()`。

5. `status`

- `scheduled_at > today` のとき (未来の予定) :
 - `planned`: 80%
 - `cancelled`: 20%
- `scheduled_at <= today` (過去・今日) :
 - `completed`: 85%
 - `cancelled`: 15%

2.3 ステップ2: 事前状態 (`pre_*`)

`scenario_type` ごとに「平均不安度」を設定:

```
interview: 7.5
one_on_one: 6.0
partner: 7.0
family: 5.5
friend: 4.5
client: 6.5
other: 5.0
```

1. `pre_anxiety`

- `base_anxiety` を平均とする正規分布 ($\sigma=2.0$) を切り詰めて 0～10 にクリップ → 四捨五入→int。

2. `pre_crying_risk`

- 平均 = `base_anxiety - 1`
- ただし `topic` が以下のときは +1 シフト:
 - 「過去の退職理由」「別れ話」「距離を置きたい」「介護の相談」
- そこに $\sigma=2.0$ の正規ノイズ→0～10 クリップ→int。

3. `pre_speech_block_risk`

- 平均 = `base_anxiety - 0.5`
- $\sigma=2.0$ の正規ノイズ→0～10 クリップ→int。

→ 結果として、

- interview / partner / client が高不安・高ブロックリスク
- friend / other は低め
- topic 依存で「泣きそう度」だけ少し別カーブ

という構造になります。

2.4 ステップ3: 準備 (`prep_*_intensity`)

準備テンプレ5種類:

1. `journaling_10m`
2. `three_messages`
3. `breathing_4_7_8`
4. `roleplay_self_qa`
5. `safe_word_plan`

実施確率は `scenario_type` 依存。例:

- `journaling_10m`:
 - interview / partner / family なら 55%
 - その他 35%
- `three_messages`:
 - interview / client / one_on_one なら 65%

- その他 40%
- breathing_4_7_8: 一律 50%
- roleplay_self_qa:
 - interview / client なら 40%
 - その他 25%
- safe_word_plan:
 - partner / family / friend なら 25%
 - その他 10%

実施した場合：

- intensity ~ Normal(7, 2) を 3~10 にクリップ → int
(= 3~10 のそこそこガチな実施強度)

実施しなかった場合：

- 95% の確率で「完全未実施」 → intensity=0 → CSV上は空文字 ""
- 5% の確率で「少しだけやった」 → 1~3 のランダム値

2.5 ステップ4: アウトカム (completed のみ)

status == "completed" のエピソードのみ、以下の生成を行う。

それ以外 (planned / cancelled) は 全アウトカム列を空文字 にしている。

2.5.1 準備の合成効果

まず「合計準備効果」のスカラーを計算：

```
total_prep_effect =
  (0.25 * journaling_intensity
   + 0.35 * three_messages_intensity
   + 0.20 * breathing_intensity
   + 0.25 * roleplay_intensity) / 10
```

- journaling / three_messages / roleplay に重み高め
- 0~? の実数 (だいたい 0~2.5 くらいに収まる)

2.5.2 stress_during, stress_after

- stress_during (話している最中のつらさ)
 - 平均 \approx `pre_anxiety + 0.5 - 0.3 * total_prep_effect`
 - $\sigma=1.8$ の正規ノイズ → 0~10 クリップ → int。

→ 準備が効くと、**pre_anxiety** より少し下がるか、同程度になる。
- stress_after (終わった直後のつらさ)
 - 平均 \approx `max(pre_anxiety - 1.0 - 1.5 * total_prep_effect, 0)`
 - $\sigma=2.0$ → 0~10 クリップ → int。

→ 準備が効くと、「終わった後の楽さ」がかなり上がるようになっている。

2.5.3 crying_level

- ベース値：

```
crying_base =
  pre_crying_risk
  + 0.5 * (stress_during - pre_anxiety)
  - 0.3 * journaling_intensity / 2
  - 0.2 * breathing_intensity / 2
```

- $\sigma=2.0$ → 0~10 クリップ → int。

→

- 話している最中のストレスが高いと泣きやすい。
- journaling と呼吸は「涙レベル」を少し下げる。

2.5.4 speech_block_level

- ベース値：

```
speech_block_base =
  pre_speech_block_risk
```

```
+ 0.4 * (stress_during - pre_anxiety)
- 0.3 * three_messages_intensity / 2
```

- $\sigma=2.0 \rightarrow 0\sim 10$ クリップ $\rightarrow \text{int}$ 。

→

- 話している最中のストレスが高いと詰まりやすい。
- `three_messages` (伝えたい3つのメッセージ) は「詰まり」を減らす方向に効く。

2.5.5 expression_score

- ベース :

```
expr_base =
    5.0
    + 0.4 * three_messages_intensity / 2
    + 0.3 * roleplay_intensity / 2
    - 0.25 * speech_block_level
    - 0.15 * crying_level
```

- $\sigma=2.5 \rightarrow 0\sim 10$ クリップ $\rightarrow \text{int}$ 。

→

- `three_messages / roleplay` は「伝えられた感」を上げる。
- 泣きすぎ・詰まりすぎると下がる。
- ただし 5.0 を基準にしているので、「準備がそこそこあれば平均以上」になるケースも多い。

2.5.6 relationship_impact

- ベース :

```
rel_base =
    -1
    + 0.4 * (expression_score - 5) / 2
    - 0.25 * max(stress_during - 6, 0)
```

- $\sigma=1.8 \rightarrow -5\sim +5$ クリップ $\rightarrow \text{int}$ 。

→

- 伝えられた感が高いほど関係にプラス。
- 話している最中のストレスが極端に高すぎると、関係にマイナス影響。

2.5.7 partner_reaction

- `relationship_impact` に応じて、以下の重みでカテゴリをサンプリング :
 - `very_positive` : `max(0.1, 0.3 + 0.05 * relationship_impact)`
 - `positive` : `max(0.1, 0.3 + 0.04 * relationship_impact)`
 - `neutral` : 0.3 固定
 - `negative` : `max(0.05, 0.2 - 0.04 * relationship_impact)`
 - `very_negative` : `max(0.02, 0.1 - 0.05 * relationship_impact)`
 - `unknown` : 0.2

→ 関係影響が高いほど positive 側に寄るが、neutral / unknown は常にそれなりに出る。

2.5.8 Reflection (days_after_reflection, would_repeat_preparation, reflection_short)

- reflection を持つかどうか :
 - 70% の確率で「振り返りを書く」
 - 30% は完全には書かない (全列空文字)

書く場合 :

- `days_after_reflection` ~ Uniform(1, 14)
- `would_repeat_preparation` の平均 :

```
base = max(5, expression_score - max(crying_level - 4, 0))
```

→ `expression_score` 高く、かつ `crying_level` が 4 をあまり超えていないほど「またこの準備をしたい」が高くなる。

- $\sigma=2.0 \rightarrow 0\sim 10$ クリップ $\rightarrow \text{int}$ 。
- `reflection_short` は `reflection_templates` から日本語文の一つランダム選択。

3. 実際の生成コード (完全版)

3.1 スクリプト全体

ファイル名例: `generate_emotion_cqox_sample.py`

```
#!/usr/bin/env python
"""
Emotion CQOx サンプル CSV 生成スクリプト

1行 = 1エピソード (Episode)
各行には:
- 事前状態 (pre_anxiety / pre_crying_risk / pre_speech_block_risk)
- 準備の強度 (prep_*_intensity)
- 当日のアウトカム (stress, crying, expression, relationship)
- 振り返り (reflection)

が入る。生成ロジックは仕様書の 2.章に対応。
"""

import argparse
import random
from datetime import datetime, timedelta

import pandas as pd

def clipped_normal(mu: float, sigma: float, low: float, high: float) -> float:
    """正規分布を low~high の範囲にクリップして返す。"""
    x = random.gauss(mu, sigma)
    return max(low, min(high, x))

def generate_rows(n_rows: int, seed: int = 42) -> pd.DataFrame:
    random.seed(seed)

    today = datetime(2025, 11, 28)

    scenario_types = [
        "interview",
        "one_on_one",
        "partner",
        "family",
        "friend",
        "client",
        "other",
    ]

    scenario_weights = [0.25, 0.20, 0.15, 0.10, 0.10, 0.10, 0.10]

    topics_by_type = {
        "interview": ["転職理由", "キャリアの方向性", "過去の退職理由", "評価面談"],
        "one_on_one": ["評価フィードバック", "キャリア相談", "業務負荷の相談", "人間関係の摩擦"],
        "partner": ["将来の暮らし", "お金の話", "結婚について", "別れ話", "距離を置きたい"],
        "family": ["親への近況報告", "進路の話", "介護の相談", "家族との距離感"],
        "friend": ["久しぶりの再会", "価値観のズレ", "謝罪", "疎遠になっている理由"],
        "client": ["トラブルの謝罪", "値上げ交渉", "契約更新", "納期遅延の相談"],
        "other": ["自己開示の練習", "セラピーではない雑談", "将来への漠然とした不安"],
    }

    locations = ["online", "office", "home", "cafe", "coworking", "client_site", "park"]

    partner_reactions = [
        "very_positive",
        "positive",
        "neutral",
        "negative",
        "very_negative",
        "unknown",
    ]
```

```

reflection_templates = [
    "少し泣いたけど言いたいことは伝えられた。",
    "ほとんど話せずに終わってしまった。次は準備を変えたい。",
    "かなり落ち着いて話せた。準備が効いた感じがある。",
    "相手の反応が予想外で混乱した。振り返りが必要。",
    "泣かなかったが本音をあまり出せなかった。",
    "かなりつらかったが、終わってみると少し楽になった。",
    "正直、今回はタイミングを間違えたかもしれない。",
]

base_anxiety_by_type = {
    "interview": 7.5,
    "one_on_one": 6.0,
    "partner": 7.0,
    "family": 5.5,
    "friend": 4.5,
    "client": 6.5,
    "other": 5.0,
}

rows: list[dict] = []

for episode_id in range(1, n_rows + 1):
    # --- user & scenario ---
    # ヘビーユーザー（1〜5）に重みを付ける
    user_id = random.choices(
        population=list(range(1, 31)),
        weights=[5] * 5 + [1] * 25,
        k=1,
    )[0]

    scenario_type = random.choices(
        population=scenario_types,
        weights=scenario_weights,
        k=1,
    )[0]
    topic = random.choice(topics_by_type[scenario_type])

    # --- scheduled_at ---
    days_offset = random.randint(-730, 90) # 過去2年〜未来3か月
    time_offset = timedelta(
        days=days_offset,
        hours=random.randint(8, 22),
        minutes=random.randint(0, 59),
    )
    scheduled_at = today + time_offset
    location = random.choice(locations)

    # --- status ---
    if scheduled_at > today:
        status = random.choices(["planned", "cancelled"], weights=[0.8, 0.2], k=1)[0]
    else:
        status = random.choices(
            ["completed", "cancelled"], weights=[0.85, 0.15], k=1
        )[0]

    # --- pre state ---
    base_anxiety = base_anxiety_by_type[scenario_type]

    pre_anxiety = int(round(clipped_normal(base_anxiety, 2.0, 0, 10)))

    crying_topic_bonus = 1 if topic in ["過去の退職理由", "別れ話", "距離を置きたい", "介護の相談"] else 0
    pre_crying_risk = int(
        round(
            clipped_normal(base_anxiety - 1 + crying_topic_bonus, 2.0, 0, 10)
        )
    )
    pre_speech_block_risk = int(
        round(clipped_normal(base_anxiety - 0.5, 2.0, 0, 10))
    )

```

```

)

# --- preparations ---
def prep_prob(key: str) -> float:
    if key == "journaling_10m":
        return 0.55 if scenario_type in ["interview", "partner", "family"] else 0.35
    if key == "three_messages":
        return 0.65 if scenario_type in ["interview", "client", "one_on_one"] else 0.40
    if key == "breathing_4_7_8":
        return 0.50
    if key == "roleplay_self_qa":
        return 0.40 if scenario_type in ["interview", "client"] else 0.25
    if key == "safe_word_plan":
        return 0.25 if scenario_type in ["partner", "family", "friend"] else 0.10
    return 0.0

prep_keys = [
    "journaling_10m",
    "three_messages",
    "breathing_4_7_8",
    "roleplay_self_qa",
    "safe_word_plan",
]

prep_intensities: dict[str, int] = {}
for key in prep_keys:
    if random.random() < prep_prob(key):
        # しっかり実施したケース (3~10)
        intensity = int(round(clipped_normal(7, 2, 3, 10)))
    else:
        # やろうとして少しだけやったケース (1~3) を 5% 混ぜる
        if random.random() < 0.05:
            intensity = random.randint(1, 3)
        else:
            intensity = 0
    prep_intensities[key] = intensity

# --- outcomes (completed only) ---
if status == "completed":
    total_prep_effect = (
        0.25 * prep_intensities["journaling_10m"]
        + 0.35 * prep_intensities["three_messages"]
        + 0.20 * prep_intensities["breathing_4_7_8"]
        + 0.25 * prep_intensities["roleplay_self_qa"]
    ) / 10.0

    stress_during = int(
        round(
            clipped_normal(
                pre_anxiety + 0.5 - 0.3 * total_prep_effect,
                1.8,
                0,
                10,
            )
        )
    )

    stress_after = int(
        round(
            clipped_normal(
                max(pre_anxiety - 1.0 - 1.5 * total_prep_effect, 0),
                2.0,
                0,
                10,
            )
        )
    )

    crying_base = (
        pre_crying_risk

```



```

        + 0.5 * (stress_during - pre_anxiety)
        - 0.3 * prep_intensities["journaling_10m"] / 2
        - 0.2 * prep_intensities["breathing_4_7_8"] / 2
    )
    crying_level = int(round(clipped_normal(crying_base, 2.0, 0, 10)))

    speech_block_base = (
        pre_speech_block_risk
        + 0.4 * (stress_during - pre_anxiety)
        - 0.3 * prep_intensities["three_messages"] / 2
    )
    speech_block_level = int(
        round(clipped_normal(speech_block_base, 2.0, 0, 10))
    )

    expr_base = (
        5.0
        + 0.4 * prep_intensities["three_messages"] / 2
        + 0.3 * prep_intensities["roleplay_self_qa"] / 2
        - 0.25 * speech_block_level
        - 0.15 * crying_level
    )
    expression_score = int(
        round(clipped_normal(expr_base, 2.5, 0, 10))
    )

    rel_base = (
        -1
        + 0.4 * (expression_score - 5) / 2
        - 0.25 * max(stress_during - 6, 0)
    )
    relationship_impact = int(
        round(clipped_normal(rel_base, 1.8, -5, 5))
    )

    partner_reaction = random.choices(
        population=partner_reactions,
        weights=[
            max(0.1, 0.3 + 0.05 * relationship_impact), # very_positive
            max(0.1, 0.3 + 0.04 * relationship_impact), # positive
            0.3, # neutral
            max(0.05, 0.2 - 0.04 * relationship_impact), # negative
            max(0.02, 0.1 - 0.05 * relationship_impact), # very_negative
            0.2, # unknown
        ],
        k=1,
    )[0]

    if random.random() < 0.7:
        days_after = random.randint(1, 14)
        would_repeat = int(
            round(
                clipped_normal(
                    max(
                        5,
                        expression_score
                        - max(crying_level - 4, 0),
                    ),
                    2.0,
                    0,
                    10,
                )
            )
        )
        reflection_short = random.choice(reflection_templates)
    else:
        days_after = ""
        would_repeat = ""
        reflection_short = ""

```

```

else:
    # planned / cancelled の場合はアウトカム列を欠損扱いにする
    stress_during = ""
    stress_after = ""
    crying_level = ""
    speech_block_level = ""
    expression_score = ""
    relationship_impact = ""
    partner_reaction = ""
    days_after = ""
    would_repeat = ""
    reflection_short = ""

# 0 は CSV 上では "" にして「未実施・欠損」として扱う
def intensity_or_empty(key: str) -> str | int:
    return prep_intensities[key] if prep_intensities[key] > 0 else ""

row = {
    "episode_id": episode_id,
    "user_id": user_id,
    "status": status,
    "scenario_type": scenario_type,
    "topic": topic,
    "scheduled_at": scheduled_at.isoformat(),
    "location": location,
    "pre_anxiety": pre_anxiety,
    "pre_crying_risk": pre_crying_risk,
    "pre_speech_block_risk": pre_speech_block_risk,
    "prep_journaling_10m_intensity": intensity_or_empty("journaling_10m"),
    "prep_three_messages_intensity": intensity_or_empty("three_messages"),
    "prep_breathing_4_7_8_intensity": intensity_or_empty("breathing_4_7_8"),
    "prep_roleplay_self_qa_intensity": intensity_or_empty("roleplay_self_qa"),
    "prep_safe_word_plan_intensity": intensity_or_empty("safe_word_plan"),
    "stress_during": stress_during,
    "stress_after": stress_after,
    "crying_level": crying_level,
    "speech_block_level": speech_block_level,
    "expression_score": expression_score,
    "relationship_impact": relationship_impact,
    "partner_reaction": partner_reaction,
    "days_after_reflection": days_after,
    "would_repeat_preparation": would_repeat,
    "reflection_short": reflection_short,
}
rows.append(row)

return pd.DataFrame(rows)

def main() -> None:
    parser = argparse.ArgumentParser()
    parser.add_argument(
        "--n-rows", type=int, default=5000, help="生成する行数 (default: 5000)"
    )
    parser.add_argument(
        "--seed", type=int, default=42, help="乱数シード (default: 42)"
    )
    parser.add_argument(
        "--output",
        type=str,
        default="emotion_cqox_sample_5000.csv",
        help="出力 CSV パス",
    )
    args = parser.parse_args()

    df = generate_rows(args.n_rows, seed=args.seed)
    df.to_csv(args.output, index=False)
    print(f"wrote: {args.output}")

```

```
if __name__ == "__main__":
    main()
```

What this does (このスクリプトで何をするか)

- **Goal:** Emotion CQOx 用のサンプルデータを、同じ生成ロジックで何回でも再現可能にしつつ、行数やシードを変えられる形でスクリプト化します。
- **Inputs:** 乱数シード、行数 `n_rows` (デフォルト 5000) 、出力パス。
- **Outputs / side-effects:** 指定されたパスに UTF-8 / カンマ区切りの CSV ファイルを生成します。1行=1Episode で、仕様どおりの列構造・欠損・擬似因果構造を持ちます。
- **Why this approach:**
 - 仕様書レベルの文章と、完全に対応したコードを残しておくことで、Claude / LLM に「ここを改造して」「ここを RCT っぽく」と指示しやすくするため。
 - 行数・シードを変えて複数セットを生成すれば、開発用・検証用・ベンチマーク用のデータを簡単に分けられる。
- **How to run / verify:**

```
# 1. 生成
python generate_emotion_cqox_sample.py \
  --n-rows 5000 \
  --seed 42 \
  --output emotion_cqox_sample_5000.csv

# 2. Python で確認
python - << 'PY'
import pandas as pd
df = pd.read_csv("emotion_cqox_sample_5000.csv")
print(df.shape)
print(df.head(3))
print(df.isna().mean())
PY
```

4. 難易度チェック用の簡易 EDA コード (検証用)

Claude や自分のノートで「本当に因果っぽい関係が入っているか」を確認するためのスニペット。

```
import pandas as pd

df = pd.read_csv("emotion_cqox_sample_5000.csv")

# completed のみ
d = df[df["status"] == "completed"].copy()

# 準備の有無フラグ
for col in [
    "prep_journaling_10m_intensity",
    "prep_three_messages_intensity",
    "prep_breathing_4_7_8_intensity",
    "prep_roleplay_self_qa_intensity",
]:
    d[col + "_used"] = d[col].fillna(0).astype(float) > 0

# pre_anxiety と stress_after の差分
d["delta_stress"] = d["pre_anxiety"] - d["stress_after"]
print(d[["delta_stress"]].describe())

# three_messages の有無で expression_score を比較
print(
    d.groupby("prep_three_messages_intensity_used")["expression_score"].describe()
)

# crying_level と expression_score の相関
print(d[["crying_level", "expression_score"]].corr())
```

Expert insight (Google/Meta/NASA level)

サンプルデータ設計で一番効いてくるのは「どれだけ現実の“歪み”を埋め込むか」です。

- 完全ランダムな RCT 風データだと、モデル検証は楽ですが、本番ログとは全く性質が違う。
- 逆に、
 - 自己選択バイアス（つらい時ほど準備する）
 - 欠損（キャンセル・未来・未記録）
 - ユーザごとの頻度差を最初から入れておくと、
「どうやってバイアスに耐えるか」「どこまで因果っぽく解釈するか」という設計そのものを磨けます。

この CSV / スクリプトは、その“歪み”まで含めて Claude に渡せる粒度にしてあるので、

「この生成モデルを前提に推定モデル作って」「ここを RCT に寄せた別バージョン作って」など、次のステップに進みやすいはず。



辛い気持ちを抱えた人や悩んでいる人は、普段気持ちをあかせない傾向にあるから
スライダーのような形で何かしら自ら変数等いじれるといいと思う。
それも続きとして上手く組み込める？



Thought for 1m 7s >

結論から言うと、「自分で変数をいじれるスライダー」を、今までの設計の“続き”としてかなりきれいに組み込めます。

しかもそれを「ただのUI」ではなく、①入力負荷↓②本人の目的関数の明示③因果・バンディット側の重み設定、の3レイヤーで効かせられます。

1. そもそもスライダーで何をやりたいのかを分解する

「辛い人ほど普段気持ちをあかせない」という前提からすると、スライダーには少なくとも3つ役割があります：

- “言葉にする前段階”としての表現
 - テキストや長文は無理でも、「今どれくらいしんどいか」を1本のバーでなら動かせる。
- “自分でコントロールしている感覚”の付与
 - 一方的に問診されるのではなく、「自分で重みや許容ラインを決めている」感覚を持てる。
- アルゴリズムの目的関数を、本人が明示的にチューニングする手段
 - 「楽さ重視か」「伝達度重視か」「関係維持重視か」をスライダーで決めれば、
因果・バンディット側で報酬関数にそのまま使える。

この3つを満たすなら、「スライダー＝単なる入力UI」ではなく、**Emotion CQOx** の中核概念にできます。

2. どこにスライダーを入れるか（3レイヤー構成）

既に作った仕様・CSV を壊さず、「上にレイヤーとして乗せる」形が安全です。

2-1. レイヤーA：Episode 直前の「状態スライダー」

目的

- 「テキストをほぼ書かなくても、1エピソードをログできる」
- `pre_anxiety`, `pre_crying_risk`, `pre_speech_block_risk` を“数字”として自分で決められる

UIイメージ（最小構成）

- コンポーネント名（例）：`EpisodeQuickSliders`
- スライダー3本だけ：
 - 今のしんどさ：`pre_anxiety` (0-10)
 - 泣きそう度：`pre_crying_risk` (0-10)
 - 言葉が詰まりそう度：`pre_speech_block_risk` (0-10)
- ラベルは重くし過ぎない（例：絵文字+短い語：😓〜😞）

データ面

- そのまま既存の列にマッピングするだけなので、DB 変更は不要。
- すでに CSV に入っている3指標が、「数字だけで入れられる」UIになる。

挑戦ポイント：

「全部スライダーにするとかえって負荷が増える」ので、
最初はこの3本だけに絞り、他は「詳細モード」に隠すのが妥当です。

2-2. レイヤーB：本人の“目的関数”スライダー（Preference Profile）

ここが本質的に効いてくるところ。

やりたいこと

- 人によってゴールが違う：
 - Aさん：「とにかく楽になりたい（涙が出てもいい）」
 - Bさん：「涙が出てもいいから、本音をきちんと言いたい」
 - Cさん：「関係を壊したくないのが最優先」
- これを「重み w として明示」し、報酬関数に組み込む。

新しいエンティティ（DB）

EmotionPreferenceProfile

- `user_id`
- `weight_relief`（楽さ）：0～1
- `weight_expression`（伝達度）：0～1
- `weight_relationship`（関係維持）：0～1
- できれば正規化制約： $w_relief + w_expression + w_relationship = 1$ になるようUI側で調整。

UI スライダー例

- 1つの「配分スライダー」だと分かりにくいので、
 - 「楽でいたい vs 多少きつくても伝えたい」
 - 「自分の本音 vs 関係を壊さない」といった2軸をスライダーで決め、内部で ($w_relief, w_expression, w_relationship$) に変換する。

報酬関数への組み込み（因果・バンディット側）

たとえば1 Episode ごとの報酬を：

- 楽さ (relief) :
 - $R_relief = pre_anxiety - stress_after$ （プラスが良い）
- 伝達度 (expression) :
 - $R_expr = expression_score$ （0～10）
- 関係維持 (relationship) :
 - $R_rel = relationship_impact$ （-5～+5）

として、

$$R_{total} = w_{relief} \cdot \tilde{R}_{relief} + w_{expression} \cdot \tilde{R}_{expr} + w_{relationship} \cdot \tilde{R}_{rel}$$

ここで \tilde{R} は適度にスケールリング・標準化したもの。

→ スライダーで決めた重みが、そのまま「どの準備を推すか」の判断に効く。

こうすると、「何を最適化するか」をツール側が勝手に決めるのではなく、
ユーザー自身がスライダーで目的関数を定義する、という構造にできます。

2-3. レイヤーC：事前シミュレーション用スライダー（What-if）

「本番前に、準備スライダーを動かして結果の“予想”を見る」というレイヤー。

動き

- Episode 作成画面で：
 - 状態スライダー（レイヤーA）を入力
 - 準備スライダー（journaling, three_messages, breathing, roleplay）を 0～10 で動かす
- 裏側では、さっきの生成モデルや、将来は学習済み回帰モデルを使って：
 - `stress_after`, `crying_level`, `expression_score`, `relationship_impact` の「期待値」を計算
- UI 上で：
 - 「この準備セットなら、たぶんこのくらいの楽さ / 伝達度 / 関係影響になりそう」というバーを出す
 - 明示的に「これは予測であり、保証ではない」と表示

既存仕様との整合

- すでに CSV 生成コードで「準備 → アウトカム」の数式モデルは書いたので、それを「逆方向（シミュレーター）」としてフロントから呼べるようにすればいい。
- バックエンドに `/api/emotion/simulate` を追加して、
 - JSON で (pre系 + prep_intensity + preference) を送り、
 - JSON で予測アウトカムを返す。

3. 具体的な実装イメージ（コード+コマンド）

3-1. Preference Profile API（バックエンド側）

新テーブル

- Alembic マイグレーションに、`emotion_preference_profile` テーブル追加：

```
user_id (FK -> user)
weight_relief          NUMERIC(4,3)
weight_expression      NUMERIC(4,3)
weight_relationship    NUMERIC(4,3)
updated_at             TIMESTAMP
PRIMARY KEY (user_id)
```

最低限の FastAPI スキーマ

```
# backend/cqox/emotion/schemas.py

from pydantic import BaseModel, condecimal, root_validator

class EmotionPreferenceProfile(BaseModel):
    weight_relief: condecimal(gt=0, le=1)
    weight_expression: condecimal(gt=0, le=1)
    weight_relationship: condecimal(gt=0, le=1)

    @root_validator
    def normalize_weights(cls, values):
        total = (
            float(values["weight_relief"])
            + float(values["weight_expression"])
            + float(values["weight_relationship"])
        )
        if total == 0:
            # デフォルトは等分
            values["weight_relief"] = 1/3
            values["weight_expression"] = 1/3
            values["weight_relationship"] = 1/3
        else:
            values["weight_relief"] = float(values["weight_relief"]) / total
            values["weight_expression"] = float(values["weight_expression"]) / total
            values["weight_relationship"] = float(values["weight_relationship"]) / total
        return values
```

API 例

```
# backend/cqox/api/emotion.py (一部)

from fastapi import APIRouter, Depends
from .schemas import EmotionPreferenceProfile
from . import service # DB 操作を隠蔽

router = APIRouter(prefix="/api/emotion", tags=["emotion"])

@router.get("/preferences/me", response_model=EmotionPreferenceProfile)
def get_my_preferences(current_user=Depends(get_current_user)):
    return service.get_preferences_for_user(current_user.id)

@router.post("/preferences/me", response_model=EmotionPreferenceProfile)
def update_my_preferences(
    prefs: EmotionPreferenceProfile,
```

```
current_user=Depends(get_current_user),
):
    return service.set_preferences_for_user(current_user.id, prefs)
```

3-2. フロント側：状態スライダー+目的関数スライダー

ファイル構成 (例)

```
cd frontend
mkdir -p src/features/emotion/components
touch src/features/emotion/components/EpisodeQuickSliders.tsx
touch src/features/emotion/components/PreferenceSliders.tsx
```

EpisodeQuickSliders (状態スライダー)

```
// src/features/emotion/components/EpisodeQuickSliders.tsx
import React from "react";

type Props = {
  preAnxiety: number;
  preCryingRisk: number;
  preSpeechBlockRisk: number;
  onChange: (values: {
    preAnxiety: number;
    preCryingRisk: number;
    preSpeechBlockRisk: number;
  }) => void;
};

export const EpisodeQuickSliders: React.FC<Props> = ({
  preAnxiety,
  preCryingRisk,
  preSpeechBlockRisk,
  onChange,
}) => {
  return (
    <div className="space-y-4">
      <SliderRow
        label="今のしんどさ"
        emojiLow="😓"
        emojiHigh="😞"
        value={preAnxiety}
        onChange={(v) => onChange({ preAnxiety: v, preCryingRisk, preSpeechBlockRisk })}
      />
      <SliderRow
        label="泣きそう度"
        emojiLow="😓"
        emojiHigh="😭"
        value={preCryingRisk}
        onChange={(v) => onChange({ preAnxiety, preCryingRisk: v, preSpeechBlockRisk })}
      />
      <SliderRow
        label="言葉が詰まりそう"
        emojiLow="🧠"
        emojiHigh="..."
        value={preSpeechBlockRisk}
        onChange={(v) => onChange({ preAnxiety, preCryingRisk, preSpeechBlockRisk: v })}
      />
    </div>
  );
};

type SliderRowProps = {
  label: string;
  emojiLow: string;
  emojiHigh: string;
  value: number;
  onChange: (v: number) => void;
```

```

};

const SliderRow: React.FC<SliderRowProps> = ({
  label,
  emojiLow,
  emojiHigh,
  value,
  onChange,
}) => {
  return (
    <div className="flex flex-col space-y-1">
      <div className="flex items-center justify-between text-sm">
        <span>{label}</span>
        <span className="text-xs text-gray-500">{value}</span>
      </div>
      <div className="flex items-center space-x-2">
        <span>{emojiLow}</span>
        <input
          type="range"
          min={0}
          max={10}
          step={1}
          value={value}
          onChange={(e) => onChange(Number(e.target.value))}
          className="flex-1"
        />
        <span>{emojiHigh}</span>
      </div>
    </div>
  );
};

```

PreferenceSliders (目的関数スライダー)

```

// src/features/emotion/components/PreferenceSliders.tsx
import React, { useMemo } from "react";

type Props = {
  relief: number; // 0~10
  expression: number; // 0~10
  relationship: number; // 0~10
  onChange: (values: {
    relief: number;
    expression: number;
    relationship: number;
  }) => void;
};

export const PreferenceSliders: React.FC<Props> = ({
  relief,
  expression,
  relationship,
  onChange,
}) => {
  const total = useMemo(
    () => Math.max(relief + expression + relationship, 1),
    [relief, expression, relationship]
  );

  const normalized = {
    relief: relief / total,
    expression: expression / total,
    relationship: relationship / total,
  };

  return (
    <div className="space-y-4">
      <p className="text-sm text-gray-600">
        何を優先したかをざっくり決めます（合計のバランスは自動で調整されます）。
      </p>
    </div>
  );
};

```



```

</p>

<PrefSliderRow
  label="楽でいたい"
  description="しんどさをへらしたい"
  value={relief}
  share={normalized.relief}
  onChange={(v) => onChange({ relief: v, expression, relationship })}
/>
<PrefSliderRow
  label="ちゃんと伝えたい"
  description="本音や背景を伝えたい"
  value={expression}
  share={normalized.expression}
  onChange={(v) => onChange({ relief, expression: v, relationship })}
/>
<PrefSliderRow
  label="関係をこわしたくない"
  description="できるだけ関係を大事にしたい"
  value={relationship}
  share={normalized.relationship}
  onChange={(v) => onChange({ relief, expression, relationship: v })}
/>
</div>
);
};

type PrefSliderRowProps = {
  label: string;
  description: string;
  value: number;
  share: number; // 0~1
  onChange: (v: number) => void;
};

const PrefSliderRow: React.FC<PrefSliderRowProps> = ({
  label,
  description,
  value,
  share,
  onChange,
}) => {
  return (
    <div className="flex flex-col space-y-1">
      <div className="flex items-center justify-between text-sm">
        <div>
          <div>{label}</div>
          <div className="text-xs text-gray-500">{description}</div>
        </div>
        <div className="text-xs text-gray-500">
          {Math.round(share * 100)}%
        </div>
      </div>
      <input
        type="range"
        min={0}
        max={10}
        step={1}
        value={value}
        onChange={(e) => onChange(Number(e.target.value))}
        className="w-full"
      />
    </div>
  );
};

```

簡単な組み込み例

```
// src/pages/emotion/EpisodeCreatePage.tsx (抜粋)

const EpisodeCreatePage: React.FC = () => {
  const [pre, setPre] = useState({
    preAnxiety: 5,
    preCryingRisk: 5,
    preSpeechBlockRisk: 5,
  });

  const [prefs, setPrefs] = useState({
    relief: 5,
    expression: 5,
    relationship: 5,
  });

  // TanStack Query の mutation で /api/emotion/scenarios に送る...

  return (
    <div className="space-y-6">
      <h1 className="text-lg font-semibold">今回のエピソード</h1>

      <EpisodeQuickSliders
        preAnxiety={pre.preAnxiety}
        preCryingRisk={pre.preCryingRisk}
        preSpeechBlockRisk={pre.preSpeechBlockRisk}
        onChange={setPre}
      />

      <h2 className="text-md font-semibold">どこをいちばん大事にしたい?</h2>
      <PreferenceSliders
        relief={prefs.relief}
        expression={prefs.expression}
        relationship={prefs.relationship}
        onChange={setPrefs}
      />

      { /* ここに「準備パターンのおすすめ」や「What-if シミュレーション」を載せていく */ }
    </div>
  );
};
```

What this does (この変更がすること)

- **Goal:**
「辛いのに話せない人」が、長文を書かなくても、スライダーだけで自分の状態と“何を大事にしたいか”をシステムに渡せるようにしつつ、その情報を因果・探索アルゴリズム側の目的関数に直結させる。
- **Inputs:**
 - Episode 直前の状態 (しんどさ・泣きそう度・詰まりそう度)
 - 本人の優先度 (楽さ／伝えたい／関係維持)
- **Outputs / side-effects:**
 - 既存の `pre_anxiety` 等にきれいにマッピングされるため、CSV・DB スキーマを壊さない。
 - 新しい `EmotionPreferenceProfile` でユーザーごとの重みが保存され、レコメンド時の報酬関数 R_{total} の係数として直接利用できる。
 - フロントで「スライダーをいじるだけ」でログと目的関数設定が完了し、テキスト入力を強制しないインタラクションになる。
- **How to run / verify (ざっくり):**
 1. バックエンドで preference テーブル追加 → `/api/emotion/preferences/me` 実装 → マイグレーション。
 2. フロントで `EpisodeQuickSliders` / `PreferenceSliders` を組み込み、既存 Episode 作成フォームと接続。
 3. ダミー Episode を数十件作り、pre スライダー値とアウトカム・報酬の分布を確認。
 4. 好みスライダーを動かしたときに、将来の「おすすめ準備パターン」が変わるようにバンディットの報酬定義を差し替える。

Expert insight (Google/Meta/NASA level)

“感情ツールにスライダーを足す”だけだと、ただのアンケートUIで終わります。

本当に効かせるには、「スライダー = ユーザーが自分の目的関数を設定するインターフェース」として設計する必要があります。

- 目的関数をプロダクト側が勝手に決めると、ユーザーは「何かよく分からない基準でジャッジされている」と感じて離脱しやすい。
- 一方で、「楽しさ」「伝えたい度」「関係維持」の重みを本人に決めてもらうと、同じ因果・バンディットモデルでも“その人の人生観”に沿った最適化になる。
- これは、単なる UX の話ではなく、**強化学習・バンディットにおける報酬設計**そのものを人間側に開示するという意味で、世界トップレベルの実装がやっていることに近いです。