☑ Small-Data Leaderboard Tricks – TODO for Copilot AI Agent

> You are Copilot in VS Code, working on the Novartis Datathon forecasting project.
> Goal: Implement advanced tricks for **target encoding**, **data augmentation**, **training/ensembling**, and **cross-validation/meta-game** to squeeze more performance from a small dataset.

---

1. Target Encoding Done Properly (Leakage-Safe)

1.1 Identify Categorical Columns for Target Encoding

- [ ] Inspect the dataset and confirm key categorical features:
  - [ ] `country`
  - [ ] `therapeutic_area`
  - [ ] `main_package` (or packaging-related feature)
- [ ] Add a configuration section (e.g. `configs/features.yaml`) specifying:
  - [ ] Which columns should use target encoding.
  - [ ] Which target to encode (e.g. `MGE`, early erosion ratio, or `vol_norm` at a given month).

1.2 Implement Out-of-Fold Target Encoding

- [ ] Create a utility module, e.g. `src/features/target_encoding.py`, with a class:

```
class OOFTargetEncoder:
    def __init__(self, cols, alpha=10.0, target_col="target"):
        self.cols = cols
        self.alpha = alpha
        self.target_col = target_col
        self.global_mean_ = None
        self.encodings_ = {}  # {col: dict(category -> encoding)}
```

- [ ] In `.fit_transform()` for training:
  - [ ] Use **GroupKFold** or your main CV splitter.
  - [ ] For each fold:
    - [ ] Compute target means per category **only on the train folds**.
    - [ ] Apply encodings to the validation fold.
  - [ ] Concatenate all folds back together to get OOF-encoded features.
- [ ] In `.transform()` for test / holdout:
  - [ ] Recompute encodings on **full training data** (using smoothing) and apply to new data.

1.3 Add Smoothed Encoding Formula

- [ ] Implement smoothing:

```
enc = (sum_y + alpha * global_mean) / (count + alpha)
```

- [ ] Store `global_mean` as the overall target mean on the training set.
- [ ] For unseen categories in test data:
  - [ ] Use `global_mean` as fallback.

1.4 Integrate into Feature Pipeline

- [ ] In `src/features/build_features.py` (or similar):
  - [ ] Call `OOFTargetEncoder` for configured categorical columns.
  - [ ] Replace or augment raw categorical columns with encoded versions.
- [ ] Ensure:
  - [ ] No leakage: validation rows never see their own targets when encodings are calculated.
  - [ ] Encoded features are used by all main models (CatBoost/LGBM/XGB/etc.).

---

2. Synthetic / Augmentation Techniques (Training-Only)

2.1 Noise-Perturbed Series for GBMs

- [ ] Implement a function `augment_with_noise(df, sigma)` that:

  - [ ] Takes training panel data with normalized volume `vol_norm`.
  - [ ] For each row (per brand, per month):
    - [ ] Samples $\varepsilon \sim N(0, \sigma^2)$, with σ in [0.03, 0.07].
    - [ ] Creates `vol_norm_aug = vol_norm * (1 + ε)`.
  - [ ] Clones all other features unchanged (`months_postgx` , `n_gxs` , etc.).
  - [ ] Optionally marks augmented rows with a flag `is_augmented = 1`.

- [ ] Apply this augmentation:

  - [ ] **Only** on training folds when fitting models.
  - [ ] **Never** when generating OOF validation predictions (to keep metrics honest).
  - [ ] **Never** for test inference.

2.2 Curve-Shape Augmentation via Residual Transfer

- [ ] Implement a function to compute simple baseline decay curves per brand:

  - [ ] For each brand, fit `simple_decay(t)` (e.g., exponential or linear in log-space) on `vol_norm`.

- [ ] Compute residuals per brand:

  - [ ] `residual_j(t) = vol_norm_j(t) - simple_decay_j(t)`.

- [ ] Implement a mapping from brand j to a set of “analog” brands k with similar:

  - [ ] `avg_vol` ,

    - [ ] `therapeutic_area`,
    - [ ] `country`,
    - [ ] `n_gxs` trajectory.

- [ ] For curve-shape augmentation:
    - [ ] For target brand B, sample residual curves from similar brand A:
        - [ ] `residual_A(t)`.
    - [ ] Create synthetic series:
        - [ ] `vol_norm_synth_B(t) = simple_decay_B(t) + λ * residual_A(t)` with λ small (e.g. 0.5–0.8).
    - [ ] Treat these synthetic rows as additional training data with lower sample weight.

- [ ] Ensure this augmentation is:
    - [ ] Training-only.
    - [ ] Not used for validation OOF metrics or test prediction.

2.3 Bootstrapped Residuals

- [ ] After fitting baseline curves for all brands:
    - [ ] Collect a pool of residuals grouped by:
        - [ ] bucket (1 or 2),
        - [ ] therapeutic area,
        - [ ] country (optional).

- [ ] For each real brand j:
    - [ ] Build synthetic series:
        - [ ] Sample residuals from brands in the same group (or similar group).
        - [ ] Set:
            - [ ] `vol_norm_synth_j(t) = baseline_j(t) + residual_bootstrap(t)`.
    - [ ] Add these synthetic rows into training folds with smaller sample weights.

---

3. Training & Ensembling Tricks

3.1 Metric-Shaped Sample Weights

- [ ] Define a global `compute_sample_weight(months_postgx, bucket, scenario)` function that:
    - [ ] Gives higher weights to:
        - [ ] Months 0–5 (Scenario 1),
        - [ ] Months 6–11 (both scenarios),
        - [ ] Bucket 1 brands (e.g., ×2).
- [ ] Use these weights in **all** main models (CatBoost, LGBM, XGB):
    - [ ] Pass as `sample_weight` / `weight` during fit.
- [ ] Optionally expose weight parameters in config files for easy tuning.

3.2 Multi-Task / Auxiliary Targets

- [ ] Choose auxiliary targets, for example:
  - [ ] Cumulative erosion up to month t,
  - [ ] Bucket probability (B1 vs B2),
  - [ ] Early-erosion slope (months 0–5 or 6–11).
- [ ] Implement auxiliary models:
  - [ ] A classifier that predicts bucket from pre-LOE features.
  - [ ] A regressor that predicts cumulative erosion from pre-LOE or early-post features.
- [ ] Pipeline integration:
  - [ ] Train auxiliary models on training data.
  - [ ] Generate out-of-fold predictions for auxiliary targets.
  - [ ] Use those predictions as **features** in the main volume-forecasting model.

3.3 Teacher–Student Distillation

- [ ] Identify your **best ensemble** of models (teacher):
  - [ ] e.g. CatBoost + LGBM + Hybrid.
- [ ] Generate OOF teacher predictions on train:
  - [ ] `y_teacher_pred` as average or weighted ensemble of base models.
- [ ] Train a **student model** (single, simpler model) with loss:

```
loss = alpha * MSE(y_true, y_pred_student) + (1 - alpha) * MSE(y_teacher_pred, y_pred_student)
```

- [ ] Choose α in [0.3, 0.7] via validation.
- [ ] Apply stronger regularization to the student (fewer trees, higher regularization).
- [ ] Use student for:
  - [ ] More stable generalization,
  - [ ] Possibly as one of the stacked models in a final ensemble.

3.4 Monotonic Constraints in GBMs

- [ ] In LGBM/XGB configs, consider monotone constraints:
  - [ ] Enforce that predictions **do not increase** when:
    - [ ] `months_postgx` increases (on average),
    - [ ] `n_gxs` increases (more competitors → lower share).
- [ ] Example (LightGBM):
  - [ ] Identify feature index for `months_postgx` and `n_gxs`.

  - [ ] Set monotone constraints array, e.g.:
    - [ ] `[-1, -1, 0, 0, ...]` (negative for those features, 0 for others).

- [ ] Validate empirically that:
  - [ ] Monotonic constraints don't severely break fit for brands with atypical behavior.
  - [ ] They reduce overfitting and crazy upward spikes in forecast.

---

4. Cross-Validation & Leaderboard Meta-Game

4.1 Multiple CV Schemes

- [ ] Implement at least **two** CV split functions in `src/validation.py`:
  - [ ] `create_group_kfold(seed)` – GroupKFold by brand, stratified by bucket, with seed for shuffle.
  - [ ] `create_group_kfold_alt(seed)` – a slightly different fold assignment (different seed or fold count).

- [ ] For important models:
  - [ ] Train using CV scheme A and scheme B.
  - [ ] Get two sets of OOF predictions.
  - [ ] Compare metrics across both schemes.

- [ ] Mark models that perform **consistently well** across both CV schemes as more trustworthy.

4.2 Use OOF Predictions as First-Class Citizens

- [ ] Ensure every trained model:
  - [ ] Always outputs OOF predictions on train.
  - [ ] Stores them in `artifacts/oof/` with standardized naming.

- [ ] Build a stacking meta-model (even a simple linear regressor or GBM) that:
  - [ ] Takes multiple OOF predictions as input features.
  - [ ] Minimizes an approximation of the official metric (e.g. weighted MSE with time & bucket weights).

- [ ] Apply the same stacked meta-model to test predictions to produce ensemble submissions.

4.3 Submission Diversification Strategy

- [ ] Near deadline, prepare **3–5 different submission "flavors"**, e.g.:
  - [ ] **Baseline ensemble**:
    - [ ] Your best GBM ensemble with normal regularization and standard weights.
  - [ ] **Underfit submission**:
    - [ ] Models with stronger regularization (fewer trees, higher L2) to avoid overfitting.
  - [ ] **Overfit-ish submission**:

        - [ ] Models with more trees / lower regularization (within reason) to exploit any underfitting.
    - [ ] **Bucket 1–focused submission**:
        - [ ] Higher sample weights for Bucket 1 during training and/or stacking.
    - [ ] **Conservative submission**:
        - [ ] Predictions closer to simple baseline decay curves (less aggressive shifts).

- [ ] For each flavor:

    - [ ] Train/configure models accordingly.
    - [ ] Generate Scenario 1 and Scenario 2 submissions.
    - [ ] Check them with a submission sanity script (no negatives, reasonable ranges).

- [ ] Use public leaderboard feedback + local CV metrics to **select 1–2 strongest variants** for final locked submission, while keeping others as backups if allowed.

End State:
You have a small-data-optimized pipeline where:

- Target encodings are powerful but leak-safe.
- Synthetic augmentation enriches erosion shapes without cheating.
- Training emphasizes what the metric and business truly care about.
- Ensembles, distillation, and monotonic constraints stabilize forecasts.
- Multiple CV schemes and diversified submissions hedge against leaderboard noise.

All implemented with Copilot’s help, step by step, inside your existing project structure.