PROGRAMMING

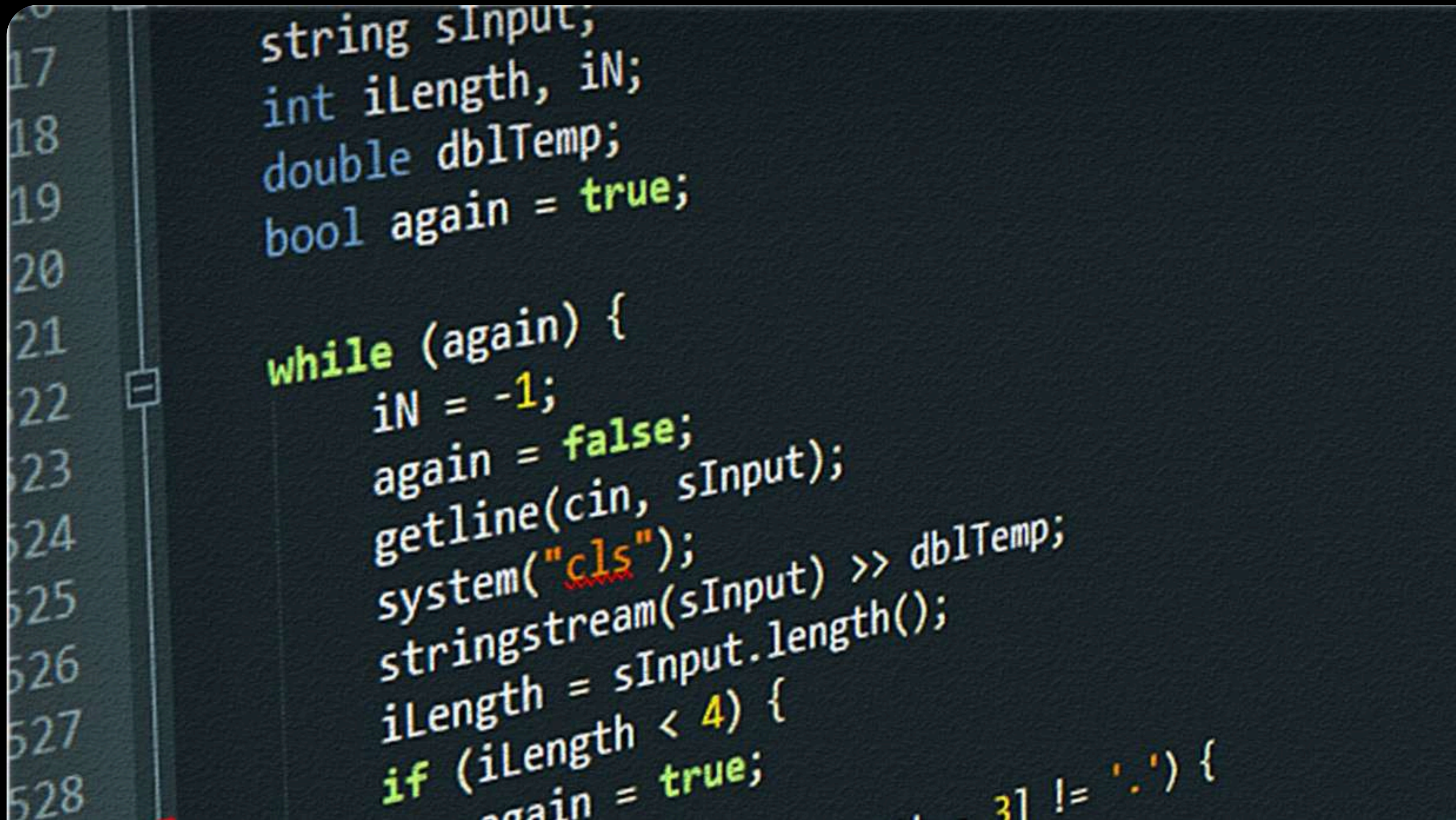
QRT DATA CHALLENGE

PROGRAMMING

Li Siqi

Content Outline

- Background
- Problem Understanding
- Exploratory Data Analysis
- Feature Engineering
- Baseline Model Selection
- Model Fine tuning
- Model Validation
- Final Thoughs



About me

LI SIQI

- Year 3 undergraduate from NUS School of Computing
- Enthusiastic about ML & AI
- Excited about competition and interact with like-minded people

$$\eta : \mathbb{R}^{100} \rightarrow \mathbb{R}^{100} \qquad \text{Mapping from illiquid to liquid assets}$$

$$X_t = (R_t^1, R_t^2, \ldots, R_t^{100}) \qquad \text{Returns of 100 illiquid assets at time } t$$

$$Y_t = (R_t^1, R_t^2, \ldots, R_t^{100}) \qquad \text{Returns of 100 liquid assets at time } t$$

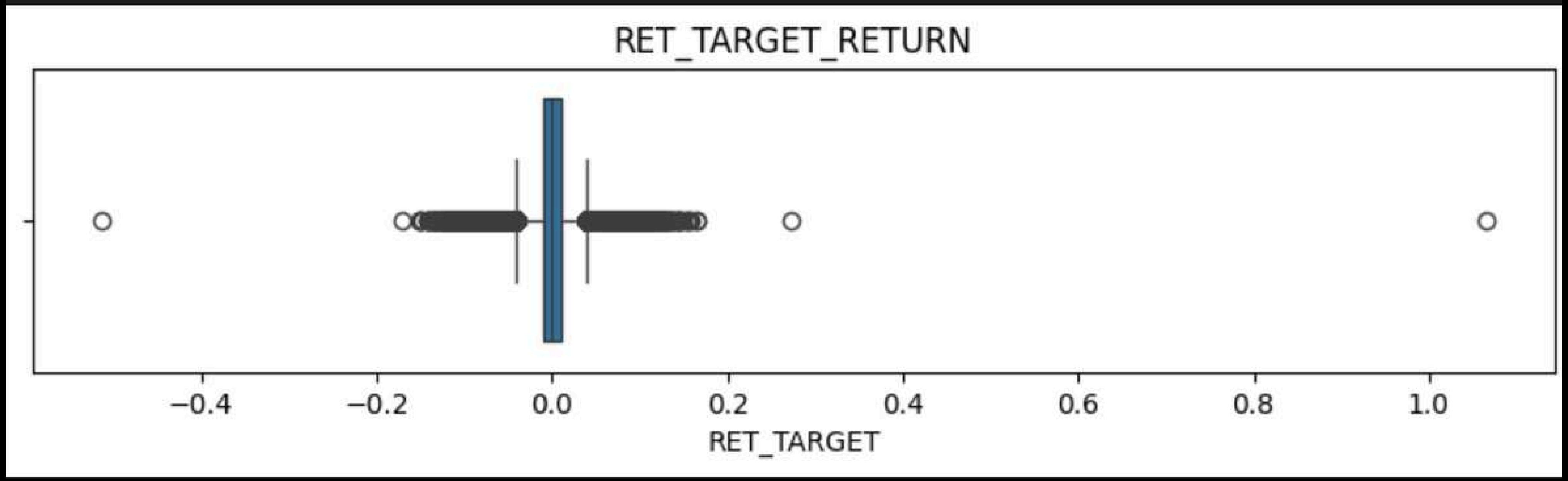
$$Y_t = \eta(X_t) \qquad \text{We project illiquid signals onto liquid assets}$$

Problem Statement

- In this challenge, we are tasked with predicting the returns of **100 liquid assets** by leveraging the historical returns of **100 illiquid assets**.
- Instead of predicting exact return values, the task is reframed as a **classification problem**
- Predict the sign of the return (+1 for positive, -1 for negative) for each liquid asset

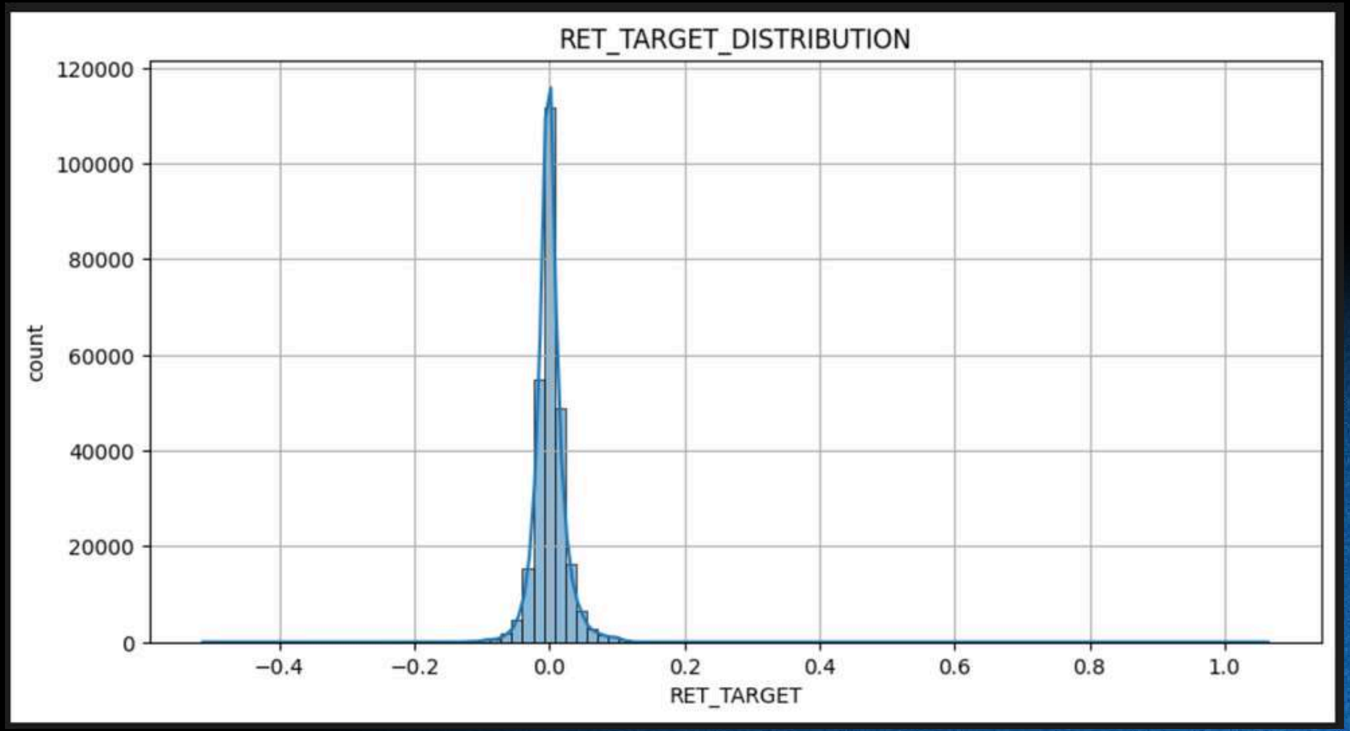
General Approach

- We need to predict 100 liquid assets, each with different sector behaviors.
- A single global model is too simplistic and fails to capture sector or asset-specific patterns.
- **First attempt**: group assets by CLASS_LEVEL (sector/industry) and train group models → too broad, underperformed.
- **Final approach**: train one dedicated model per liquid asset → 100 models for 100 assets, capturing unique dynamics and improving accuracy.
- Use **MultiOutputRegressor** from scikit-learn to fit one regressor per target.



Exploratory Data Analysis

Balanced classes: RET_SIGN = -1 (52.4%), +1 (47.6%) → no resampling required.
Distribution: Returns are tightly centered around 0 with high kurtosis → common in financial data (thin tails, many near-zero returns).
Outliers: A few extreme returns exist , but removing them gives no performance benefit.



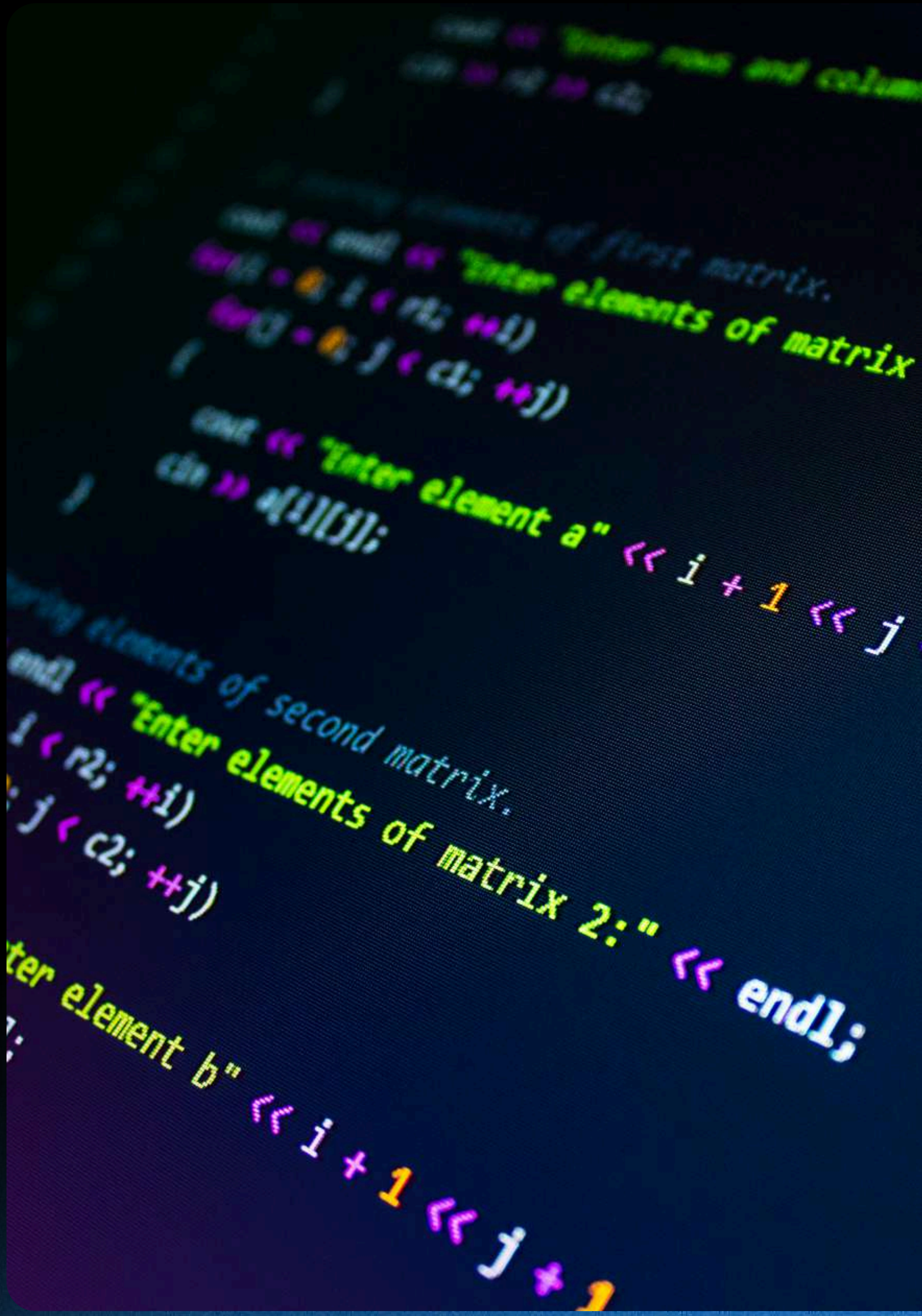
WEAK DATA FILTERING

Problem:
- Many returns are extremely small (near zero).
- These weak signals behave like noise to the sign prediction

What I did:
- Set **threshold**: RET_RETURN < 0.00001 → weak signal
- Label weak signals as 0 and exclude from training
- Keep only strong, meaningful signals for model training

Why it helps:
- Focus on real market moves
- Reduce noise and ambiguity around zero
- Improve robustness & generalization

Result: 183 data points were deleted
Public LB: 0.7481 → 0.7489



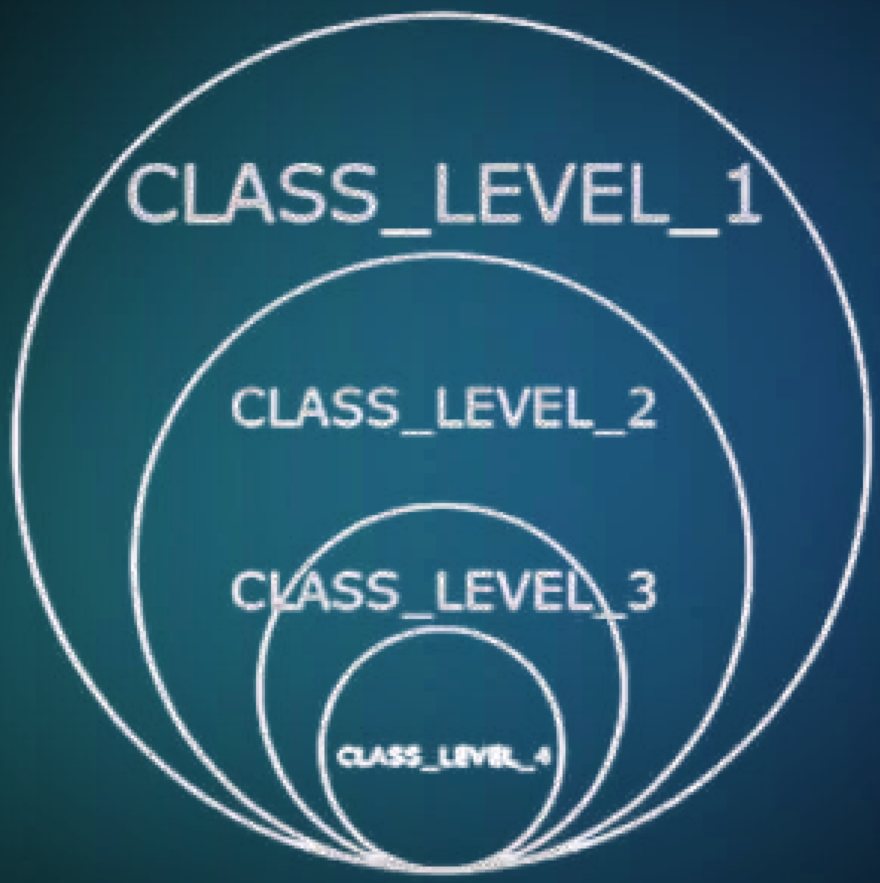
DATA PREPROCESSING

- Feature-level: up to **13%** missing values.
- Day-level: up to **18%** missing values.
- Such gaps can bias results → imputation is essential.

Data Imputation Methods – Comparison

Method	Pros	Cons	Performance
Row Deletion	Simple	Removes too much data	Poor
Column Mean	Stable, fast	Ignores sector info	Decent
KNN Imputer	Uses cross-asset similarity	Slow on large data, cost-sensitive	Decent
Class-Based Weighted	Leverages sector/industry hierarchy	Heavier compute	Best

MISSING DATA



- Work day-by-day.
- Use **sector hierarchy** (CLASS_LEVEL_1 → CLASS_LEVEL_4).
- Compute class means from available (non-NaN) assets.
- Impute missing assets using **weighted combination** of their sector path.
- Fallback: day's cross-sectional average if no sector info is available.

MISSING DATA

- Weighting: CLASS_LEVEL_4 (most specific) has highest weight (0.4), decreasing to L3=0.3, L2=0.2, L1=0.1.
- Fallback: If a level’s mean is missing, redistribute weights across available levels.
- Example: If L4 is missing:

$$\text{Value} = \frac{0.1 \times L1 + 0.2 \times L2 + 0.3 \times L3}{0.1 + 0.2 + 0.3}$$

```
Input:
    df          = daily returns (some missing)
    supp_data   = sector/industry class info (L1–L4)
    weights     = [0.1, 0.2, 0.3, 0.4] for L1→L4

For each day in df:
    1. Split assets into:
        - known   = assets with values
        - unknown = assets with missing values

    2. For each industry level (L1–L4):
        - Compute mean return of known assets within each class

    3. For each missing asset:
        - Look up its class path (L1→L4)
        - Collect weighted averages from available class means
          (more weight for specific classes, e.g. L4 > L1)
        - If no class info available → fallback to overall daily mean

    4. Fill the missing asset with this weighted average

Return: fully imputed daily return matrix
```

FEATURE ENGINEERING

- Raw returns vary a lot across assets and days, with extreme outliers that can dominate the model.
- Different assets have different volatility levels, making direct comparison unfair.
- **Converting to percentiles** standardizes returns into a common scale between 0 and 1.
- This reduces the influence of outliers and highlights the relative ranking of each asset’s return.
- The model then focuses on capturing directional trends, which matches our goal of predicting the sign.

FEATURE ENGINEERING

- Percentile conversion is applied within each dataset separately (train and test).
- This ensures that the transformation uses only the distribution of the current set, avoiding leakage from train → test.
- The method does not create new information; it only rescales returns into a comparable range.
- By focusing on relative ranking instead of absolute values, we reduce overfitting risk and improve generalization.
- Improve my public leaderboard score from 0.7463 to 0.7478, ceteris paribus

```
# Step 3: Convert to percentiles
illiquid_returns = convert_to_percentiles(illiquid_returns_raw)
returns_to_predict = convert_to_percentiles(test_ret_imputed)
```

INPUT: RAW TRAINING FEATURES (X_TRAIN) AND TARGETS (Y_TRAIN).

- For each trading day: Collect illiquid returns (RET_i) from X_train and corresponding liquid returns (RET_TARGET) from y_train.
- Re-index liquid returns as RET_ so they align with assets.
- Output: i_returns and l_returns→ DataFrame of asset returns (days × assets)
- Purpose: transform row-wise data into day-wise return matrices for modeling.

```python
def daily_returns(X_train, y_train):
    idx_ret_features = np.where(X_train.columns.str.contains('RET'))[0]
    i_returns, l_returns = {}, {}
    for day in tqdm(X_train.ID_DAY.unique()):
        u = X_train[X_train.ID_DAY == day]
        a = u.iloc[0, idx_ret_features]
        b = y_train.loc[u.index, 'RET_TARGET']
        b.index = ['RET_' + str(t) for t in u.ID_TARGET]
        i_returns[day] = a
        l_returns[day] = b
    return pd.DataFrame(i_returns).T.astype(float), pd.DataFrame(l_returns).T.astype(float)
```

(function) ID_TARGET: Any

Daily Return Construction

Baseline Model Selection

Baseline Model	Public leaderboard Score
LASSO	0.743
Ridge	0.741
CatBoost	0.711
Elastic NetCV	0.746 (Selected as final model)

- Selected **Elastic NetCV** for its strong performance on the public leaderboard.
- Tested ensemble methods, but a single model outperformed in this case.
- Computationally efficient → feasible to scale across 100 liquid assets with limited resources.

Why Elastic Net Succeeded?

- Combines L1 (LASSO) and L2 (Ridge) → balances feature selection with coefficient stability.
- Handles highly correlated features better than pure LASSO.
- Adds robustness to noisy financial signals while shrinking irrelevant variables.
- Captures sparse but stable relationships, aligning with financial return structures.
- Scales well to our setup of 100 separate models, making feature selection manageable for each liquid asset.

Model Fine-Tuning

- Used ElasticNetCV wrapped in MultiOutputRegressor to handle 100 liquid assets.
- l1_ratio → balances LASSO vs Ridge and alphas → controls penalty strength.
- 5-fold cross-validation (cv=5) ensures robust parameter selection.
- max_iter=10000 → stability in convergence.
- n_jobs=-1 → parallelized for efficiency.

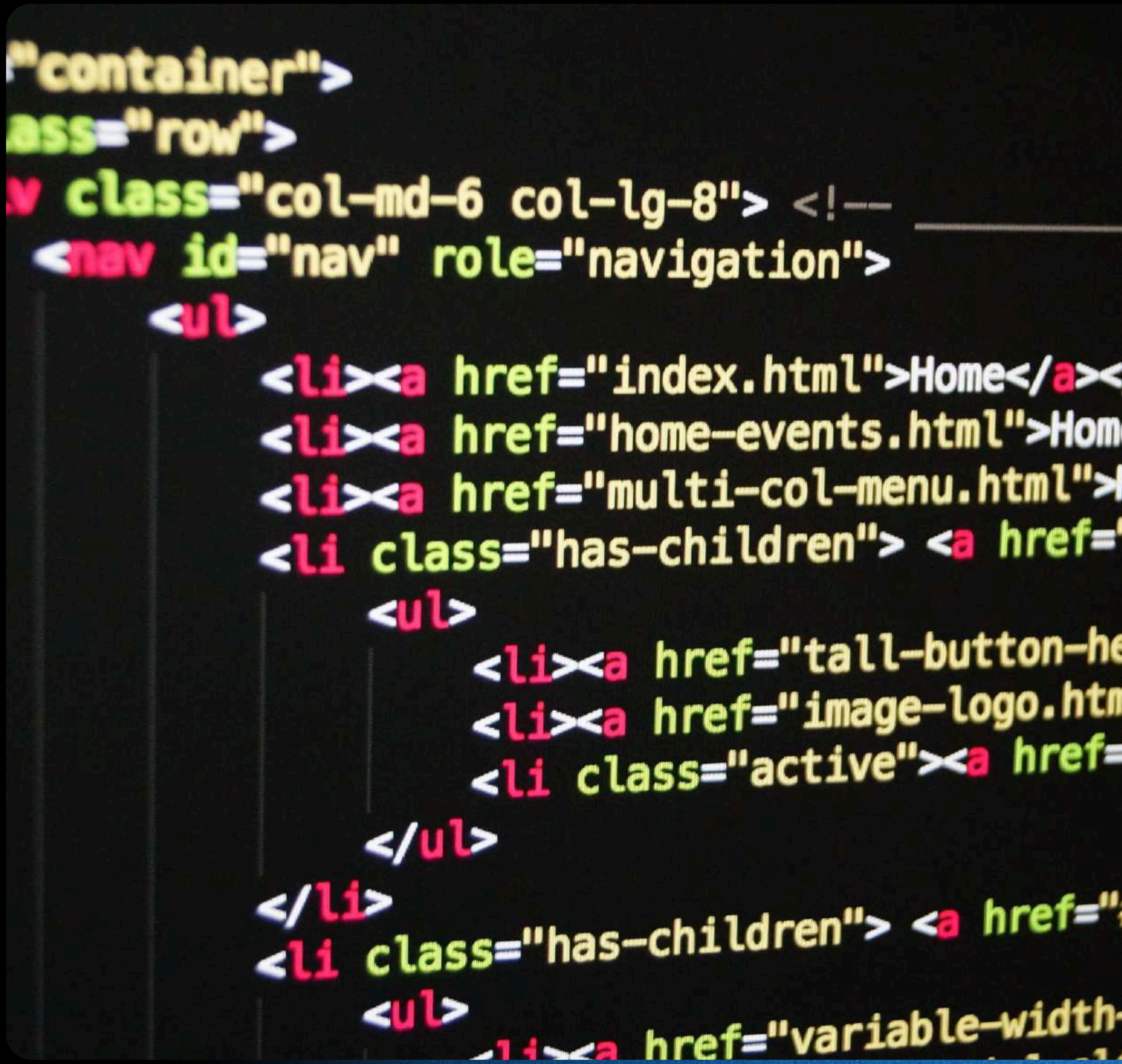
```
# Step 8: Train model
model = MultiOutputRegressor(ElasticNetCV(
    l1_ratio=[0.1, 0.3, 0.5, 0.7, 0.9, 1],
    alphas=[0.0001, 0.001, 0.01, 0.1, 1, 10],
    cv=5, max_iter=10000, n_jobs=-1
))
model.fit(illiquid_returns, l_returns)
```

Model Validation

- GroupKFold Cross-Validation (CV)
- Split by ID_DAY to avoid leakage across days.
- Ensures validation reflects true generalization (mimics competition split).
- Achieved stable CV scores across folds.
- Public Leaderboard Check
- Submitted predictions to verify CV alignment with leaderboard.
- Balance between CV and LB confirmed robustness of model.

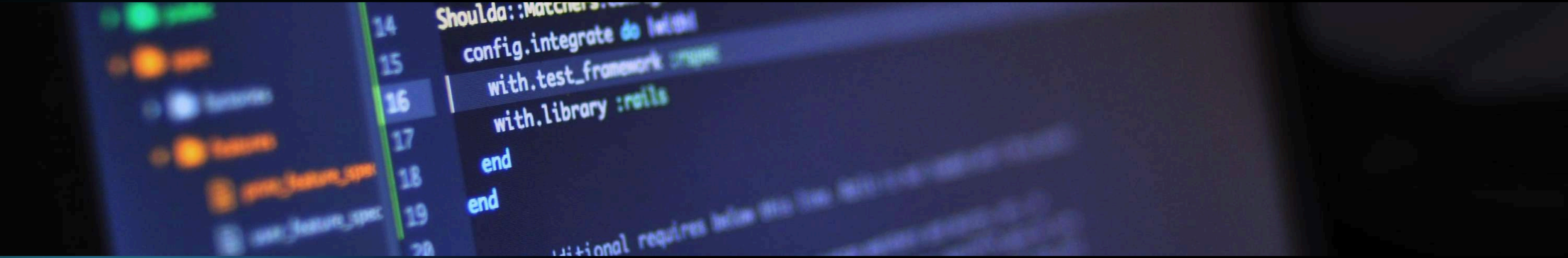
Challenges

- Cross-validation vs. Leaderboard Gap: Local CV results not always consistent with public leaderboard → hard to trust validation strategy.
- Accuracy Evaluation Limitations: True performance only testable on leaderboard → risk of overfitting to local CV.
- Anomalous Features: Some features behave unexpectedly → feature engineering becomes unreliable.
- Randomized Time Index: ID_DAY anonymized → impossible to exploit temporal continuity, limits sequence modeling.



Future Improvements

- Deeper EDA: Investigate hidden correlations between illiquid/liquid assets.
- Study feature distributions and sector-level dependencies.
- Advanced Feature Engineering: Incorporate sector/industry metadata more systematically. Try interaction terms, nonlinear transforms, or embedding-based representations.
- Modeling Enhancements: Explore ensemble methods (stacking/blending multiple models). Ensemble across data splits / parameter settings to capture diverse signals.
- Validation Strategy: Develop more robust CV schemes closer to leaderboard behavior.



Final thoughts

- Data processing > Model choice → preprocessing & feature engineering drove performance.
- Elastic Net worked best: balanced sparsity & stability, robust to noise.
- Simple > Complex → linear models outperformed boosting/NN under noisy returns.
- Scalable → efficient to train 100 models with limited resources.
- What I have learned → Patience is the key → No improvement for one week → Focus on the details of the model.

Q&A

THANK YOU

THANK YOU

THANK YOU

www.reallygreatsite.com