✅ 1. They’re Redundant (Derived Columns Cover Them)

We engineered two new features:

- `orig_diff = oldbalanceOrg - newbalanceOrig`
- `dest_diff = newbalanceDest - oldbalanceDest`

These capture the **actual transfer behavior**:

- `orig_diff` tells us **how much money left** the sender’s account.
- `dest_diff` tells us **how much money entered** the receiver’s account.

Once these are created and properly transformed (log1p, clipped), the original columns are **no longer necessary** because:

- They **don’t add new information**
- They could **confuse the model** or add noise due to multicollinearity

---

✅ 2. Some Original Columns Can Be Noisy or Misleading

For example:

- A balance might be zero **before or after** due to temporary internal processing — that doesn’t always mean fraud.
- `oldbalanceOrg` and `newbalanceOrig` often contain 0 for system-generated “CASH_IN” or bot-driven transactions.

Using engineered deltas (differences) focuses the model on **actual movement** of funds rather than raw values, which can vary widely.

✅ 3. Model Simplicity & Reduced Dimensionality

Including all original and derived features would result in:

- More input dimensions (risk of overfitting)
- Possible **correlated features** (bad for linear models like Logistic Regression)

By using `amount_log`, `orig_diff_log`, and `dest_diff_log`, we:

- Normalize the skewed distributions
- Keep the most **predictive and interpretable** columns
- Improve generalization

🔍 Summary Table

Column	Kept?	Why?
`amount`	✅	Key feature for fraud, log-transformed
`oldbalanceOrg`	❌	Replaced by `orig_diff`
`newbalanceOrig`	❌	Replaced by `orig_diff`
`oldbalanceDest`	❌	Replaced by `dest_diff`
`newbalanceDest`	❌	Replaced by `dest_diff`
`orig_diff` (new)	✅	Captures sender behavior
`dest_diff` (new)	✅	Captures receiver behavior
`type`	✅	Important categorical feature

👉 So what now?

- **If you're using tree-based models** (like Random Forest or XGBoost):
    - **No need to scale** at all! Trees don’t care about feature scale.
    - But we often keep scaling in a pipeline for compatibility across models.
- **If you're using Logistic Regression or SVM**:
    - Scaling **is important**, because these models are sensitive to the scale of features.
    - Since `log1p()` helps, but doesn’t perfectly normalize, you can:
        - Use `StandardScaler` if your `log1p`-transformed features look roughly normal (check histograms!)
        - Use `MinMaxScaler` if you want everything **strictly in [0,1]**, e.g. for neural nets or interpretability