Exchange-Like Return Rate Predictor

Goal: Prove what *doesn't* work for labeling "returns", then define a **clean proxy label** ("exchange-like") and build **time-safe** features.

Section 1: Data Ingestion & Dataset Scale Awareness

Before thinking about "returns," I needed to understand **what one row in the data actually represents** and whether the dataset scale would constrain design choices.

- Loaded the **transactions table** and parsed t_dat as a datetime.
- Loaded the **articles table** to understand product metadata.
- Checked dataset size and basic structure.

Key observation

- The transactions table contains **~31.8 million rows**.
- Each row corresponds to **one purchased article**, not an order or basket.

- Because each row is **article-level**, any return or exchange signal must also be inferred at the **article level**, not order level.
- The scale (31.8M rows) means:
    - Full pairwise comparisons or naïve window joins are expensive.
    - EDA must be **sampled**, and feature engineering must be **linear or cumulative**, not quadratic.

Design implication

- This immediately pushed me toward:
    - Sampling for exploratory analysis
    - Using **groupby + shift / cumulative logic** instead of complex joins
    - Thinking carefully about how a "return" could even be defined without order IDs

Section 2: Transaction → Article Mapping Integrity

Does every transaction map cleanly to exactly one article?

- Treated each row in the transactions table as:
    - one customer_id
    - one article_id
    - one purchase timestamp (t_dat)

- Performed sanity checks by:
    - Inspecting joins between transactions and articles
    - Sampling transactions and verifying article metadata availability

Key observations

- Each transaction row maps to **exactly one article_id**.

- There are **no multi-article rows** or bundled purchases.
- Article metadata joins succeed for the overwhelming majority of rows.

Why this matters
- This confirms that:
    - Repeat purchases of the same article_id by the same customer are **meaningful signals**
    - I can safely define labels like “same article purchased again within X days”

Design implication
- I can model **exchange-like behavior** as:
  “The same customer purchasing the same article again shortly after a previous purchase”
- This validation is what made the later **article-level label definition** defensible.

Section 3: Purchase Clustering & Category-Level Behavior

Do customers buy many items close together in time — especially within the same product group?

This question is critical because many naïve “return” proxies assume:
“If a customer buys something similar shortly after, it must be a return or exchange.”
This grouping tests whether that assumption holds.

What I did
- Sampled **1,000,000 transactions** to keep analysis tractable.
- Analyzed the distribution of:
    - number of items purchased per customer per day
    - number of items purchased per product group per customer per day

Key findings — Same-day purchasing is common
From the 1M-row sample:
- **~84%** of customer-days include **1 item**
- **~6.7%** include **2 items**
- **Thousands** of cases include **3–5 items on the same day**
- A long tail exists with **10+ items in a single day**

Implication
- Same-day multi-item shopping is normal behavior.
- If same-day repurchases were labeled as “returns,” the model would **massively overlabel**.

Key findings — Same-day, same-product-group shopping is also common
This was the most important insight:
- **~92%** of customer-days per product group include **1 item**

- **~3.7%** include **2 items**
- Thousands of customers buy **2–4 items from the same product group on the same day**

Examples of normal behavior
- Buying the same top in multiple colors
- Buying multiple basics
- Sale or promotion-driven shopping

Why this breaks product-group-based return logic These patterns explain why earlier ideas like: "Same product group purchased again within 14 days" dramatically **inflates return rates**.
Even with a longer window:
- Category similarity ≠ return
- Short-term similarity ≠ exchange

Core conclusion
H&M customers naturally buy many similar items in short bursts — often on the same or next day. Therefore, a valid "return" proxy must be exchange-shaped, not just "similar purchase soon."

Design implication
- Product or garment group similarity **cannot** be used as a return label.
- Any label must be:
    - higher precision
    - resistant to burst-shopping behavior
    - based on **exact article repetition**, not category similarity

Section 4: Reframing the Target - From "Returns" to "Exchange-Like" Behavior

The dataset contains **no explicit return events**.
So the core challenge is:

> How do we define a target that is behaviorally meaningful, low-noise, and defensible?

Based on the earlier EDA, many intuitive proxies (same-day repurchase, same product group, short window similarity) were shown to overlabel normal shopping behavior.

Key insight from EDA
- Customers frequently buy:
    - multiple items
    - from the same product group
    - in short time bursts

- Therefore:
    - Category similarity ≠ return

    - Temporal proximity alone ≠ exchange

So instead of trying to infer *returns*, the target was reframed as: **“Exchange-like behavior”**
Final target definition

A transaction is labeled **exchange-like = 1** if:

- The **same customer**
- Purchases the **exact same article (article_id)**
- Again within **1–14 days**
- **Excluding same-day repurchases**

All other transactions are labeled 0.

Why this definition works

- Re-buying the *same SKU* shortly after purchase is far more likely to represent:
    - size exchange
    - color exchange
    - defect replacement
      than general browsing or shopping behavior.

- Matches how exchanges typically occur in fashion retail:
    - try → return/exchange → reorder same item

Tradeoff

- This proxy under-captures:
    - exchanges to different SKUs
    - exchanges across sizes/colors if encoded as different article_ids

- This is intentional:
    - Precision is prioritized over recall
    - Cleaner labels → better learning signal

I intentionally chose a conservative label that undercounts exchanges but avoids noise. This makes model behavior easier to interpret and improves trust in predictions.

Resulting label characteristics

- Exchange-like rate: ~2–3%
- Strong class imbalance
- Realistic for downstream modeling decisions (thresholding, cost tradeoffs)

Design implication

- The project is no longer framed as:
  “Return prediction”

- It is framed as:
  “Predicting exchange-prone purchases using behavioral signals”

This framing is much more defensible in an interview.

Section 5: Time-Safe Label Construction (No Leakage)

When labels depend on *future behavior* (e.g., “did the customer repurchase this article later?”), there is a serious risk of **data leakage**.

The key question here is:

> How do we label transactions using future information **without leaking it into features**?

For each (customer_id, article_id) pair:

1. Sort purchases chronologically by t_dat
2. For each purchase:
    - Look at the **next purchase date** of the same article by the same customer
3. Compute the day difference:
    - days_to_next_purchase = next_t_dat − current_t_dat
4. Assign label:
    - exchange_like = 1 if 1 ≤ days_to_next_purchase ≤ 14
    - exchange_like = 0 otherwise

Same-day repurchases are explicitly excluded.

Why this is time-safe

- The label uses **future behavior only to define the target**
- No feature is allowed to see:
    - the next purchase
    - the label itself
- All features are constructed using **shifted or cumulative past-only logic**

This separation ensures:

- Labels may look forward but Features **never do**

Why the 1–14 day window

- **Lower bound (≥1 day)**:
    - Prevents same-day bulk shopping from being mislabeled as exchanges
- **Upper bound (≤14 days)**:
    - Captures realistic exchange timelines in fashion retail
    - Limits false positives from long-term re-buying or seasonal behavior

Edge cases handled

- Final purchase of an article by a customer:
    - No future purchase → labeled 0
- Multiple repeat purchases:
    - Each transaction only considers the **next** purchase

- Customers with frequent buying:
    - Sorting + shift logic prevents cross-contamination across time

The label itself is future-aware by definition, but all predictive features are strictly past-only. I explicitly used shift() logic so that the model never has access to information that wouldn't exist at prediction time.

Section 6: Time-Safe Feature Engineering (Customer + Article History)

Now that the label is defined ("exchange-like" = same customer re-buys the same article within 1–14 days), the goal becomes:

> **Create features that could realistically be known at purchase time and help predict exchange-like behavior.**

I created two feature families:

1) Customer history features

These capture whether a customer has a pattern of exchange-like behavior.

- c_orders: how many purchases the customer has made *before* this one
- c_label_cnt: how many exchange-like events the customer had *before*
- c_label_rate: prior exchange-like rate
- c_days_since: days since the customer's last purchase

Why these matter

- Some customers consistently buy in ways that lead to exchanges (e.g., sizing uncertainty, bulk ordering)
- Recency features capture shopping bursts vs spaced-out purchases

2) Article history features

These capture whether the SKU itself tends to trigger exchange-like behavior.
Typical features:

- a_orders: how many times the article has been bought *before* this transaction
- a_label_cnt: how many times this article has appeared in exchange-like events *before*
- a_label_rate: historical exchange-like rate for the article

Why these matter

- Some SKUs are inherently more exchange-prone (fit issues, inconsistent sizing, etc.)
- This helps the model generalize beyond customer patterns

Every feature is built using only information available strictly **before the transaction timestamp**, typically via cumulative counts and shifted cumulative sums.

These features are:

- scalable to 31.8M rows (mostly groupby + cumulative operations)
- interpretable
- deployment-realistic (they can be computed incrementally as new transactions arrive)

Grouping 7: Time-Based Train / Validation Split

How would this model behave if deployed in the future, not just on shuffled historical data?

A random train/test split would:

- Mix early and late transactions together
- Allow the model to learn patterns from future time periods
- Inflate performance metrics unrealistically

This is especially problematic in retail, where:

- customer behavior changes over time
- seasonality and promotions exist
- new articles are introduced

What I did

- Sorted transactions chronologically by t_dat
- Chose a time cutoff at approximately the 85th percentile of dates
- Defined:
    - Training set → all transactions *before* the cutoff
    - Validation set → all transactions *after* the cutoff

This ensures:

- The model only learns from the past
- Validation simulates a real deployment scenario

Why 85% was chosen

- Provides enough data to:
    - learn stable customer and article patterns
- Leaves a meaningful future window to:
    - test generalization
- Avoids a tiny validation set in an already imbalanced problem (~2–3% positives)

Section 8: Baseline Model Selection & Class Imbalance Handling

Why start with a baseline model

- The label is rare (~2–3% exchange-like rate)
- Performance gains from complex models are meaningless without a reference point
- Interpretability matters early to validate feature signal

Model choice: Logistic Regression

I chose logistic regression as the baseline because:

- fast to train on large datasets
- easy to interpret

    - well-suited for binary classification

Handling class imbalance
Because positive cases are rare:
- I used class_weight="balanced"
- This penalizes misclassification of exchange-like events more heavily
- Without class weighting:
    - The model would learn to predict “no exchange” almost everywhere
    - Accuracy would look high but be meaningless

Evaluation metrics
I deliberately avoided accuracy.
Primary metrics:
- ROC-AUC → ranking quality
- Precision / Recall → behavior under imbalance

Because the positive class is rare, I evaluated with AUC and precision–recall instead of accuracy, which would be misleading.

Section 9: Adding Product-Group Context as a Feature (Not a Label)

Earlier EDA showed that **product-group similarity is too noisy to define a return label**. However, that does *not* mean product-group behavior has no predictive value.

> **Can category-level shopping behavior help the model *without* redefining the ground truth?**

What I did
1. Joined minimal article metadata (product_group_name) to transactions
2. For each transaction, created a binary feature:
    - **pg_rebuy_14d** =
      “Did the customer buy *another item* from the same product group within the next 14 days?”
3. Ensured this feature was:
    - computed after labeling logic was finalized
    - treated as a **feature**, not a target

EDA showed that product-group repurchase is too noisy to use as a return label, but it still contains signal. I added it as a feature so the model can decide when it matters, instead of hard-coding it into the target.

Design implication
- This feature can improve recall without inflating false positives
- It also gives flexibility:
    - remove it later if it harms generalization

    - extend it to other windows or category hierarchies

Section 10: Threshold Selection & Business Trade-offs

After training a probabilistic model, the final decision is **not** “what model did best?” but:

At what probability should we act?

Because this is an imbalanced problem, the default 0.5 threshold is arbitrary and usually wrong.

What I did

- Generated predicted probabilities for the validation set
- Swept probability thresholds from **0.05 → 0.90**
- For each threshold, computed:
    - Precision
    - Recall
    - F1 score
- Collected results into a comparison table

Why threshold tuning matters here

- Positive class (~2–3%) is rare
- Acting on every predicted exchange is costly
- Missing all exchanges is also costly

This creates a **precision–recall trade-off**, not an accuracy problem.

Chosen operating point (example)

At a threshold around **0.6**:

- Recall ≈ **58%**
- Precision ≈ **5× higher than baseline rate**

This represents:

- Catching a meaningful portion of exchange-like events
- Without overwhelming the system with false positives

Business interpretation

Depending on use case:

- **Higher precision**:
    - Fewer but more confident interventions
    - E.g., targeted sizing recommendations

- **Higher recall**:
    - Broader coverage
    - E.g., proactive exchange handling

The model supports **multiple operating modes** without retraining.

Final project narrative (one-line summary)

I designed a time-safe, exchange-focused return rate predictor by carefully defining a low-noise proxy label, engineering past-only behavioral features, validating with a temporal split, and explicitly tuning decision thresholds based on business trade-offs.