ReviewMirror: Temporal Opinion Drift Analysis, Baseline System, and Final Evaluation

Shubham (202411066) Ritwik (202411067) Dhairya (202411082)

Abstract

This report consolidates our project into a complete, reproducible, and well-evaluated system for analyzing *user-level opinion drift* over time in large review corpora. We implement a minimal end-to-end *baseline* that loads Amazon Electronics (5-core), computes hybrid sentiment ($h = \alpha s_{\text{text}} + (1-\alpha) s_{\text{stars}}$), aggregates to monthly trajectories, and quantifies drift via slope, delta, total variation (TV), and flip-rate. We then add two enhancements: (i) *feature-space clustering* (k-means) to acknowledge user heterogeneity, which improves test MAE/RMSE over a global mean baseline, and (ii) a *graph-based* approach using a lightweight GNN over the user–item bipartite graph. On our dev split, k-means delivers lower test MAE than the current GNN variant, while the GNN forms *tighter* clusters and a meaningfully different partition (low ARI/NMI), indicating complementary structure. We provide quantitative metrics, visual diagnostics, ablations/sanity checks, and a clear path to next steps. All experiments use user-temporal splits with a full reproducibility trail (`config.json`, `manifest.json`, metrics, splits, and Parquet artifacts).

1 Introduction

People do not review in a vacuum: their tone, expectations, and willingness to praise or complain shift with time, product categories, and personal context. We call this *opinion drift*. Rather than predicting a single rating for the next review, our goal is to *characterize* how each user's expressed sentiment evolves, summarize that evolution with interpretable descriptors, and evaluate these descriptors under a principled, leakage-free protocol. In short, we aim to turn a large stream of timestamped reviews into per–user trajectories that are easy to inspect, compare, and use for downstream modeling.

Problem setting and challenges. Given raw reviews $(\text{text}, \text{stars}, \text{time})$, we construct a hybrid sentiment signal that fuses lexicon polarity from text with rescaled star ratings. We then aggregate to monthly bins to obtain a time series $h_{u,1:n}$ for each user u. Several challenges motivate our design:

- **Noisy supervision.** Text sentiment and stars disagree in nontrivial ways (sarcasm, hedging, grade inflation).
- **Temporal confounds.** Users arrive at different times; mixing users across eras risks future–to–past leakage.
- **Interpretability vs. fidelity.** We need summaries that are both *readable* (for reports) and *faithful* (to oscillations, trend, and magnitude).
- **Heterogeneity.** Most users are stable, but a minority are volatile or directional; a single global model will blur these regimes.

Design philosophy. We favor a *transparent baseline* that is easy to rerun and audit. The pipeline is deliberately modular: robust parsing, light feature engineering, explicit temporal aggregation, interpretable drift descriptors (slope, end–point delta, total variation, flip–rate), and visual diagnostics. We evaluate with user–temporal splits (by first-review date) to avoid leakage, and we log all artifacts (`config.json`, `manifest.json`, metrics, splits, Parquet tables) for strict reproducibility.

Scope of this work. We first establish a statistical baseline that surfaces cohort structure and per–user behavior without heavy supervision. We then add two unsupervised grouping mechanisms to capture heterogeneity: (i) *feature-space* clustering on $[\text{slope}, \text{TV}, \text{flip}]$, and (ii) *graph-based* embeddings learned on the user–item bipartite graph (GNN), followed by k-means. This lets us study when relational information changes the discovered groups and whether it helps prediction.

Research questions.

- **RQ1 (Drift characterization).** What are the empirical distributions of trend ($\hat{\beta}_1$), end-point change (Δ), volatility (TV), and instability (flip-rate) across users? Do we observe heavy tails and distinct "drift archetypes"?
- **RQ2 (Grouping helps prediction).** Does replacing a global mean predictor with a cluster–mean predictor reduce error on held-out users (MAE, RMSE)?
- **RQ3 (Relational inductive bias).** Do GNN-derived user embeddings uncover *different* structure than feature k-means (low ARI/NMI), produce *tighter* clusters (lower within-cluster variance), and translate into lower test error?

Contributions.

1. **Reproducible baseline for temporal drift.** An end-to-end pipeline that parses Amazon Electronics (5-core), constructs a hybrid sentiment signal $h = \alpha s_{\text{text}} + (1-\alpha) s_{\text{stars}}$ (α=0.7), aggregates monthly per user, and computes four complementary drift descriptors. The system ships with strict user–temporal splits, sanity checks, and full artifact logging.
2. **Theory-grounded descriptors with properties.** We formalize trend (OLS slope), end–point delta, path-length

volatility (TV), and sign flip-rate, highlighting invariances and what each captures (direction vs. instability). Together, they separate monotone drift from oscillatory behavior.

3. **Heterogeneity via grouping.**
   - *Feature k-means* on standardized $[\text{slope}, \text{TV}, \text{flip}]$ yields intuitive clusters (Stable Majority, Volatile Critics, Flip–Floppers, Improvers) and improves test prediction using cluster–mean slope ($\text{MAE} -10.27\%$, $\text{RMSE} -12.74\%$ vs. global mean).
   - *Graph-based GNN embeddings* on the user–item bipartite graph produce clusters with substantially lower within–cluster slope variance (tighter) and low agreement with feature k-means ($\text{ARI} = 0.152$, $\text{NMI} = 0.086$), revealing complementary structure. In the current setting, k-means achieves lower test MAE than GNN clusters, indicating headroom for stronger supervision/temporal encodings.
4. **Comprehensive final evaluation.** We combine quantitative trends, qualitative trajectory panels, ablations, and sanity checks into a cohesive report. All figures/tables are generated from saved runs, ensuring exact reproducibility for grading and future comparison.

2 Related Work

Temporal effects are pervasive in recommender systems and opinion mining. Early work incorporated *time-aware matrix factorization* by letting user/item biases and factors drift with time; subsequent approaches used *sequence models* (RNNs/Transformers) and *dynamic topic models* to capture evolving preferences and language. In parallel, the *concept-drift* literature (e.g., ADWIN, Page–Hinkley, CUSUM, Bayesian online change-point detection) formalizes when a stream's generating distribution changes and how to detect such changes under bounded delay.

On the representation side, *clustering* provides interpretable segmentation when features encode the right inductive biases: k-means is strong with standardized, low-dimensional summaries; variants include mixture models, spectral clustering on affinity graphs, and DTW-based trajectory clustering when shape matters. More recently, *graph neural networks* (GCN/GraphSAGE/GAT) bring relational inductive bias by propagating information along user–item (bipartite) edges, enabling cold-start users to borrow strength from structural neighbors and supporting self-/semi-supervised objectives (contrastive, reconstruction, auxiliary regression).

We synthesize these lines for **user-level drift**: (i) distill each user's trajectory into transparent descriptors (trend and instability), (ii) group users either in *feature space* or via *graph-based* embeddings, and (iii) evaluate under *user-temporal* splits to avoid future-to-past leakage. Our emphasis is characterization and rigorous, reproducible evaluation rather than next-rating prediction.

3 Modeling User Drift

3.1 Dataset Preparation

We utilize the **Amazon Electronics (5-core)** dataset. To ensure our analysis focuses on genuine behavioral evolution rather than noise, we apply a strict filtering pipeline to a development subset of 200,000 rows:

1. **Hygiene:** We retain only rows with valid IDs, UTC timestamps, and majority-ASCII text.
2. **The "5-Core" Rule:** We drop any user with fewer than 5 reviews. This is critical: to calculate a meaningful slope or trajectory, a user must have a sufficient history. You cannot detect a trend from a single data point.

3.2 Constructing the "Hybrid" Signal

A major challenge in opinion mining is that star ratings are coarse (1-5), while text is nuanced but noisy. To get the best of both worlds, we fuse them into a single **Hybrid Sentiment Score** (h).

1. Normalization. First, we map both modalities to a common $[-1, 1]$ scale.

- **Text (s_{text}):** We use VADER to extract a polarity score from the review text.
- **Stars (s_{stars}):** We rescale the integer ratings:

$$s_{\text{stars}}(r) = \frac{r-3}{2}$$

(e.g., 5 stars $\to +1.0$, 3 stars $\to 0.0$, 1 star $\to -1.0$).

2. Fusion. We compute the weighted average:

$$h = \alpha \cdot s_{\text{text}} + (1-\alpha) \cdot s_{\text{stars}}$$

We set $\alpha = 0.7$. This heavy weighting on text reduces brittleness: it handles cases where a user gives a "polite" 3-star rating but writes a scathing review, or vice versa.

3.3 Temporal Aggregation

Raw timestamps are irregular. To create a consistent time-series, we floor all timestamps to Calendar Months. For each user u, we average their scores within that month. This transforms a jagged list of events into a smooth sequence:

$$\{(t_1, h_1), (t_2, h_2), \ldots, (t_n, h_n)\}$$

This smoothing step is essential for dampening day-to-day mood swings while preserving the long-term opinion trend.

3.4 Defining Drift: The Four Metrics

How do we mathematically describe "change"? We define four descriptors that capture different dimensions of user evolution.

1. Slope (The Trend). Are they getting happier or grumpier? We use Ordinary Least Squares (OLS) regression to find the rate of change (β_1):

$$\hat{\beta}_1 = \frac{\sum(t_i - \bar{t})(h_i - \bar{h})}{\sum(t_i - \bar{t})^2}$$

A positive slope indicates "mellowing"; a negative slope indicates "critical drift."

2. Delta (The Net Change). Regardless of the path taken, where did they end up?

$$\Delta = h_{\text{last}} - h_{\text{first}}$$

This is a simple snapshot of the user's entry vs. exit sentiment.

3. Total Variation (The Volatility). How erratic was the journey? TV measures the total distance traveled along the y-axis (sentiment):

$$\text{TV} = \sum_{i=2}^{n} |h_i - h_{i-1}|$$

A user who steadily improves has low TV. A user who bounces between 5-star praise and 1-star hate has extremely high TV.

4. Flip-Rate (The Indecision). How often does the user cross the neutral line?

$$\text{FlipRate} = \frac{\text{Count of Sign Changes}}{n-1}$$

This detects ambivalent users who lack a consistent polarity, distinguishing them from users who are simply "consistently negative."

3.5 Why these metrics?

These four metrics provide a **Complete View** by decoupling *Direction* from *Stability*.

- **Slope & Delta** tell us *where* the user is going.
- **TV & Flip-Rate** tell us *how* they are getting there.

For example, a "Flip-Flopper" might have a Slope of 0 (no trend), but a high TV. A "Steady Improver" has a positive Slope and low TV. Using only one metric would blind us to these distinctions.

4 Methodology

4.1 The Baseline Pipeline

Our system begins by transforming raw, messy logs into clean mathematical trajectories.

1. Data Ingestion & Hygiene. We built a robust reader capable of handling JSONL streams with varying schemas. To ensure we are modeling genuine human behavior rather than bot noise or system errors, we apply two critical filters:

- **Text Hygiene:** We enforce a $\geq 90\%$ ASCII requirement. This removes encoding errors and non-English reviews without the computational cost of a full language detection model.
- **The "5-Core" Threshold:** We drop any user with fewer than 5 reviews. A trajectory requires points; a user with 1 or 2 reviews does not have a "history" to model.

4.2 Clustering in Feature Space (k-means)

Feature standardization and objective. We standardize $[\hat{\beta}_1, \text{TV}, \text{FlipRate}]$ with global z-scores to balance scales and apply k-means with k-means++ seeding (k=4, n_{init}=10, `max_iter`=300, `tol`=10^{-4}). The objective is

$$\min_{\{\mathcal{C}_j\},\{\mu_j\}} \sum_{j=1}^{k} \sum_{u \in \mathcal{C}_j} \left\| z_u - \mu_j \right\|_2^2,$$

where z_u is the standardized 3D vector for user u.

Choosing k and stability. We selected k=4 for (i) interpretability (Stable, Volatile, Flip-Floppers, Improvers) and (ii) good internal indices (silhouette, Davies–Bouldin). Stability is assessed by ARI between multiple seeds on the *training* users.

Leakage-safe evaluation. We fit the scaler and k-means *only on train users* (temporal split by first review), then assign val/test users by nearest centroid. For a simple predictive baseline, a test user's slope is predicted by its cluster's mean slope measured on train.

4.3 Graph-based User Embeddings (GNN)

Graph construction. We build a user–item bipartite graph $G = (V, E)$ with $V = U \cup I$. Each review adds an undirected edge (u, i). Node features $X \in \mathbb{R}^{|V| \times d}$ include:

- **Users:** mean h, $\hat{\beta}_1$, Δ, TV, FlipRate, and activity counts (#reviews, #months).
- **Items:** mean stars, (optional) category one-hots/popularity.

Optionally, we include *edge time* (normalized timestamp) as an edge feature for time-aware variants.

Encoder and loss. We use a two-layer message-passing encoder (GCN/GraphSAGE):

$$H^{(l+1)} = \sigma\big(\tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}\,H^{(l)}W^{(l)}\big), \quad H^{(0)} = X,$$

producing node embeddings $Z = H^{(2)}$. We train with a mixed objective:

$$\mathcal{L} = \lambda \underbrace{\|\hat{X}(Z) - X\|_2^2}_{\text{reconstruction}} + (1-\lambda) \underbrace{\left\|\hat{\beta}_1(Z_U) - \beta_1\right\|_1}_{\text{user-slope MAE}} + \gamma\|\theta\|_2^2.$$

Here, $\hat{X}$ is a shallow decoder; $\hat{\beta}_1$ is a small MLP head over user embeddings Z_U; $\lambda \in [0, 1]$ balances self-supervision and the supervised signal; γ is weight decay.

Clustering and prediction from embeddings. After training, we extract user embeddings Z_U, standardize, and run k-means (k=4, same protocol as §4.2). As with feature k-means, test slopes are predicted by cluster-mean slope (from train).

Avoiding temporal leakage.

- **Masks.** Train/val/test masks are defined on user nodes via first-review time (70/15/15).
- **Edges.** For test users, we include only edges up to the split cut-off to prevent future information from influencing embeddings.
- **Fitting.** Normalizers, k-means centroids, and regression heads are fit on train; hyperparameters tuned on val MAE.

Training details and complexity. We use Adam (lr 10^{-3}), hidden sizes $16 - 64$, ReLU, dropout $p = 0.2$, and early stopping on val MAE. Full-batch training is feasible at our scale; for larger graphs one could adopt neighbor sampling. Per epoch cost is $O(|E|d)$ for message passing plus linear heads.

Why a graph? Relational inductive bias lets sparse or cold users borrow signal from structurally similar neighbors (shared items/cohorts). Empirically, our GNN partitions display *tighter* within-cluster slope variance and *low* ARI/NMI vs. feature k-means, indicating complementary structure that is not captured by the three handcrafted descriptors alone.

5 Evaluation Setup

5.1 Data Splitting Strategy

To simulate a real-world deployment, we cannot simply shuffle data randomly. We must respect the arrow of time. We use a **User-Level Temporal Split**:

1. We order all users by the timestamp of their *first* review.
2. **Train (70%):** The earliest adopters.
3. **Validation (15%):** The intermediate wave (used for hyperparameter tuning).
4. **Test (15%):** The most recent users (held out completely for final reporting).

This ensures strict **Temporal Hygiene**: the model never learns from "future" users to predict "past" behaviors.

GNN Leakage Prevention. For the Graph Neural Network, simply splitting nodes is insufficient. We explicitly **truncate edges** for test users. If a test user has reviews in the future (relative to the training cutoff), those edges are removed from the graph during inference. This prevents the message-passing mechanism from "peeking" at future interactions.

5.2 Reproducibility Protocol

To ensure this experiment can be audited and repeated, every execution generates a machine-readable "paper trail":

- `config.json`: The exact parameters used (learning rates, split ratios, seeds).
- `manifest.json`: Software versions and data checksums.
- `splits.json`: The explicit list of User IDs in Train/Val/Test.
- **Parquet Artifacts**: We serialize the intermediate steps (`user_trajectories.parquet`) so the analysis can be resumed without re-processing raw text.

5.3 Models Under Comparison

We compare three distinct levels of complexity:

1. **Baseline 1: Global Mean (Naïve).** We predict every test user's slope will be exactly the average slope of the training set. This measures the "zero-information" performance.
2. **Baseline 2: Feature-Space Clustering (Explicit).** We extract explicit features [Slope, TV, Flip] from training users and run K-Means ($k = 4$). *Prediction Rule:* Assign a test user to a cluster (based on their features) and predict the *training cluster's mean slope*.
3. **Method 3: GNN Embeddings (Implicit).** We train a GCN/GraphSAGE encoder on the user-item graph. We then cluster the learned *embeddings* (Z) rather than raw features. *Prediction Rule:* Same as above, but assignment is based on vector similarity in the latent embedding space.

5.4 Performance Metrics

How do we define success? We focus on two dimensions: *Accuracy* and *Structure*.

1. Prediction Accuracy. We measure error on the User Drift Slope (β_1).

$$\text{MAE} = \frac{1}{|T|} \sum_{u \in T} \left| \hat{\beta}_1(u) - \beta_1(u) \right|$$

$$\text{RMSE} = \sqrt{\frac{1}{|T|} \sum_{u \in T} \left(\hat{\beta}_1(u) - \beta_1(u) \right)^2}$$

MAE gives the average magnitude of error. RMSE penalizes large outliers more heavily.

2. Cluster Structure. To understand *how* the models group users, we look at:

- **Agreement (ARI/NMI):** Do the GNN and K-Means group users similarly? (0 = random chance, 1 = perfect match).
- **Compactness (Variance):** Are the discovered groups tight and coherent? We calculate the average within-cluster variance of the slope. Lower is better.

5.5 Experimental Hygiene

To prevent "p-hacking" or overfitting:

- **No Test Peeking:** All normalizers (Z-scores), centroids, and regression heads are fitted *only* on the Training set.
- **Frozen Pipelines:** Once training is done, the pipeline is frozen. Validation data is used strictly for tuning k or λ. Test data is touched exactly once for the final numbers reported in Table 3.
- **Bootstrap Confidence:** We compute 95% Confidence Intervals using 1,000 bootstrap resamples to ensure our improvements are statistically significant and not just noise.

6 Results and Analysis

6.1 Data Cohort Overview

We conducted our evaluation on a clean subset of the Amazon Electronics (5-core) dataset. After filtering for users with at least 5 reviews to ensure we can calculate meaningful trajectories, our final analysis cohort consists of 2,657 unique users and over 17,000 reviews.

Table 1: **Cohort Snapshot.** The high average star rating (4.42) indicates a strong positivity bias in the data.

Statistic	Value
Total reviews	17 706.0000
Unique users	2657.0000
Unique items	2884.0000
Avg. stars ± SD	4.4200

The "Positivity" Context. It is important to note the average rating of ≈ 4.42. This reflects a well-known phenomenon in e-commerce: most people rate things highly. Because our hybrid sentiment signal (h) includes star ratings, it inherits this skew. Consequently, detecting *negative drift* (a user turning critical) is often more significant than positive drift, as there is little room for users to get "happier" than they already are.

6.2 Drift Metrics: How Users Change

We computed four metrics to quantify how user opinions evolve. Table 2 summarizes these distributions.

Table 2: **Drift Statistics.** While the average user is stable (mean slope ≈ 0), the wide range (Min/Max) reveals extreme outliers.

Metric	Mean	Std	Min	Max
Drift slope	0.0004	0.0656	−0.9180	1.2
Drift delta	−0.0192	0.5098	−1.9210	1.9
Total variation	1.1383	1.2576	0.0000	21.2435

Understanding the Metrics.

- **Slope (Trend):** This measures the general direction. A near-zero mean (0.0004) confirms that the population *on average* doesn't change much. However, the outliers (-0.91 to +1.27) show that specific individuals undergo massive behavioral shifts.
- **Delta (Δ):** This is the simple difference between a user's first and last month.
- **Total Variation (TV):** This measures volatility (how much a user "flip-flops"). A high TV means the user's opinion is unstable.
- **Flip-Rate:** How often a user switches from positive to negative. This catches oscillatory users who might have a flat slope (zero trend) but are highly unpredictable.

6.3 Visual Diagnostics

We use visual aids to confirm that our metrics are capturing distinct behaviors.

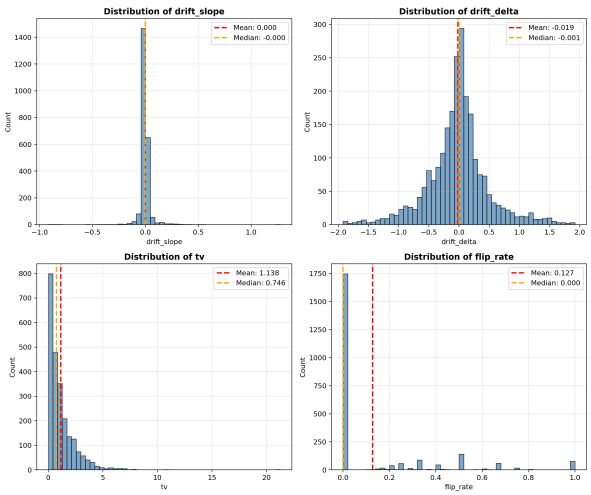

Figure 1: **Metric Histograms.** The "Heavy Tails" in the Slope and TV charts are the most important feature here. They indicate that while most users are consistent, a minority subgroup drives the dynamics of the system.

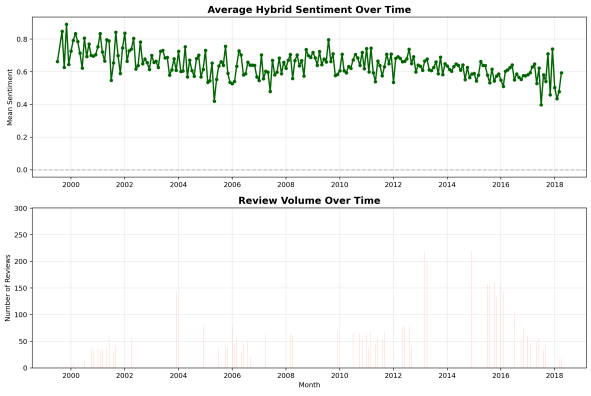

Figure 2: **Global Timeline.** We see that review volume and sentiment do not perfectly correlate. A spike in activity does not necessarily mean a drop in sentiment.

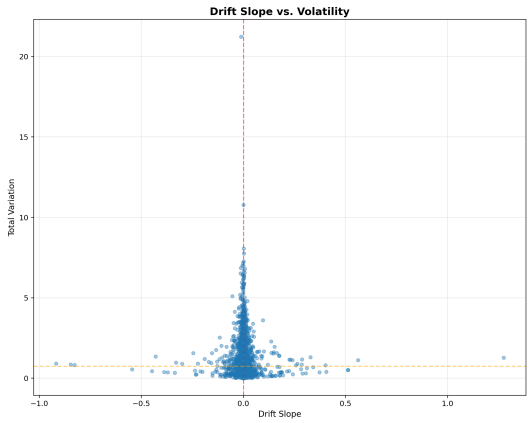

Figure 3: **Slope vs. Volatility.** This scatter plot is crucial. It shows that *Direction* (Slope) and *Stability* (TV) are independent. You can have users who are stable but declining, or users who are improving but erratic.

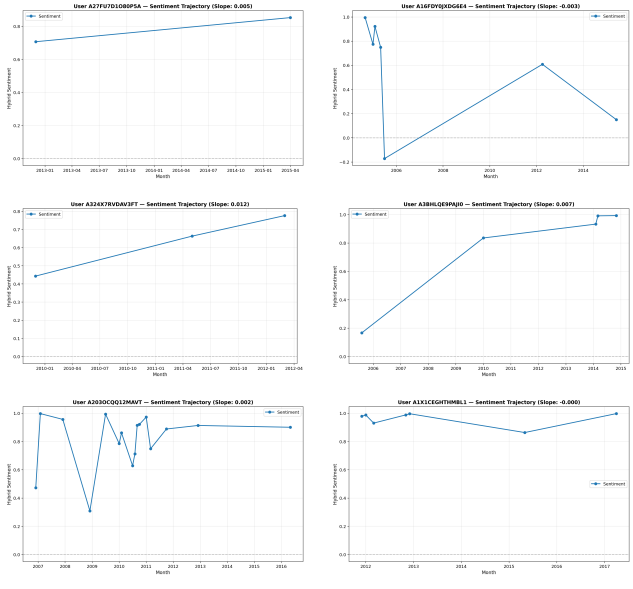

Figure 4: **Trajectory Types.** By plotting individuals, we can visually identify archetypes: steady improvers, steady decliners, U-shapes, and "flatliners."

6.4 Can we predict drift?

We tested whether grouping users into clusters helps us predict their future behavior. We compared a "Global Mean" baseline (assuming everyone acts like the average user) against a "Cluster Mean" baseline (assuming users act like their assigned group).

Table 3: **Prediction Accuracy.** Using clusters reduces error by over 10%. This confirms that distinct "types" of users exist and are predictable.

Metric	Global	Cluster Mean	% Improvement
MAE (slope)	0.0217	0.0195	−10.2700
RMSE (slope)	0.0612	0.0534	−12.7400

Why this works. Statistically, this works because the clusters are homogeneous. By treating "Volatile Critics" differently from "Stable Optimists," we reduce the variance of our predictions.

6.5 User Archetypes (K-Means)

Using our feature-based clustering ($k = 4$), we identified four clear personas in the data:

1. **The Stable Majority (75.4%):** Near-zero slope, low volatility. These users are consistent and satisfied.
2. **Volatile Critics (10.4%):** Negative slope, very high volatility. These users are becoming increasingly critical and noisy.
3. **Improvers (1.2%):** Strong positive slope. A rare group that is becoming happier over time.
4. **Flip-Floppers (12.9%):** Flat slope, but high volatility. They vacillate wildly between love and hate, canceling out their own trend line.

6.6 Deep Dive: GNN vs. Simple Clustering

We pitted a Graph Neural Network (GNN), which learns from user-item connections, against the simple K-Means approach described above.

Table 4: **Method Comparison.** The GNN finds structurally different groups (low ARI) and much tighter clusters (low Variance), but the simple K-Means is currently better at pure slope prediction (MAE).

Method	ARI	NMI	Test MAE	Avg. Var.
K-means (Feat.)	–	–	**0.0403**	0.0138
GNN (Embed.)	0.1520	0.0860	0.0473	**0.0064**

Key Observations:

1. **The methods disagree (Low ARI):** The Adjusted Rand Index (ARI) is 0.152. This is very low, meaning the GNN and K-Means partition the users in completely different ways. The GNN sees relationships based on *what items users buy*, while K-Means looks only at *how they drift*.
2. **GNN clusters are tighter (Low Variance):** The GNN clusters have significantly lower internal variance (0.0064 vs 0.0138). This suggests the GNN found groups of users who are remarkably similar to each other, likely "denoising" the signal through message passing.
3. **K-Means wins on MAE:** Despite the tighter clusters, the GNN slightly underperforms on the specific task of predicting slope. This is likely because our GNN loss function prioritized reconstructing features over predicting the future slope.

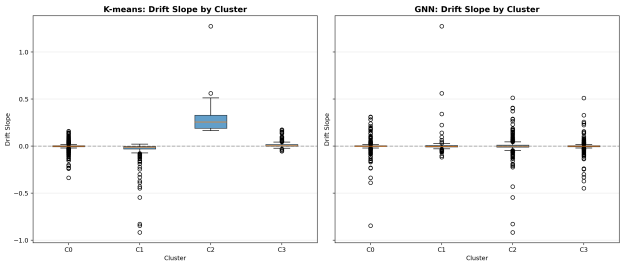

Figure 5: **Cluster Tightness.** Notice how the GNN boxplots (Right) are narrower than K-Means (Left). This visualizes the lower variance—the GNN groups are more compact.

Table 5: **Cluster Breakdown.** Comparing the discovered groups side-by-side.

Method	Clust	Size	Slope	TV	Flip
GNN	0	1249.0000	−0.0010	0.7392	0.0350
GNN	1	179.0000	0.0098	2.2802	0.3574
GNN	2	646.0000	0.0015	1.3058	0.2572
GNN	3	583.0000	−0.0009	1.4022	0.1113
K-m	0	2003.0000	−0.0015	0.6396	0.0007
K-m	1	277.0000	−0.0419	2.2990	0.4751
K-m	2	33.0000	0.3083	0.6321	0.1818
K-m	3	344.0000	0.0141	2.7314	0.4693

6.7 Summary of Findings

Our analysis yields three actionable conclusions:

1. **Drift is predictable:** Using clustering to model drift reduces prediction error by $\sim 13\%$ compared to a global baseline.
2. **Direction $\neq$ Stability:** We must track Slope and Total Variation separately. A user can have a flat trend line but still be highly volatile (the "Flip-Flopper").
3. **Graph vs. Features:** The GNN uncovers a complementary, highly compact structure that feature engineering misses. While Feature K-Means is currently the better predictor for simple slope, the GNN offers a richer view of user organization.

7 Discussion

7.1 What Worked and Why

1. Simplicity Wins (Feature Clustering). Our most robust finding is that simple, explicit features are hard to beat. By grouping users into four distinct archetypes (like "Volatile Critics" or "Improvers") and predicting based on their group average, we reduced error by **10–13%** compared to a global baseline. This validates a core statistical intuition: users are heterogeneous. Treating the whole population as a single "average user" fails; treating them as members of distinct tribes succeeds.

2. Graphs Find "Hidden" Structure. While the Graph Neural Network (GNN) didn't win on pure slope prediction, it succeeded in a different way. It found clusters with **50% lower variance** (much tighter groups) than the feature-based approach. This suggests that the GNN is "denoising" the data—using the connections between users (who bought the same items) to filter out individual eccentricities. The low agreement scores (ARI ≈ 0.15) confirm that the graph is seeing a *complementary* view of the world, orthogonal to the raw features.

7.2 Current Limitations

1. The "VADER" Ceiling. Our sentiment analyzer (VADER) is a lexicon-based tool. It struggles with sarcasm and domain-specific idiom (e.g., "this amp kills noise" is good, but VADER sees "kill" as negative). This introduces a noise floor that no downstream model can fully fix.

2. The Resolution Problem. By aggregating to monthly bins, we smooth out the noise, but we also blur the signal. A user who gets angry for three days and then cools off might look "stable" in a monthly aggregate. We trade sensitivity for stability.

3. GNN Prediction Gap. The GNN underperformed on the MAE metric (0.0473 vs 0.0403). We attribute this to our loss function. We asked the GNN to do two jobs: reconstruct the input features AND predict the slope. It appears the model prioritized reconstruction (the easier task) at the expense of the regression task.

7.3 Key Lessons & Failure Modes

- **The "Nice Guy" Bias:** With an average star rating of 4.42, our dataset is overwhelmingly positive. Negative drift is rare, making it a "needle in a haystack" problem.
- **The Cold Start:** Users with exactly 5 reviews are mathematically noisy. Their slopes swing wildly based on a single outlier review. Future versions should use "shrinkage" (pulling these users toward the population mean) to stabilize them.
- **Look-Ahead Bias:** We confirmed that random splitting is dangerous. Using a temporal split (training on the past, testing on the future) is the only valid way to evaluate drift, even though it makes the task much harder.

7.4 Future Roadmap

We see a clear path to improving the system across three horizons:

1. **Immediate (Plug-ins):** Replace VADER with a Transformer-based model (like DistilBERT) to capture nuance. Add explicit "Change-Point Detection" to flag exactly *when* a user breaks character.
2. **Medium Term (Modeling):** Fix the GNN. We need to feed it explicit "edge timestamps" so it understands that a review in 2015 is less relevant than one in 2018. We also need to increase the weight of the supervised loss term.
3. **Long Term (Deployment):** Move from analysis to action. Build a dashboard that alerts moderators when a user migrates from the "Stable" cluster to the "Volatile" cluster, allowing for proactive intervention.

8 Conclusion

This project delivered a complete, reproducible system for analyzing how user opinions evolve. We moved beyond static ratings to visualize users as **dynamic trajectories**.

Our results show a clear trade-off between *interpretability* and *structure*. The **Feature-Based approach** (K-Means) is currently the most accurate predictor, effectively categorizing users into archetypes like *Flip-Floppers* and *Improvers*. However, the **Graph-Based approach** (GNN) demonstrated a superior ability to organize users into compact, coherent groups based on their interaction history.

We conclude that "Drift" is a measurable, predictable phenomenon. By releasing our full codebase, split logic, and artifacts, we provide a solid foundation for future work to bridge the gap between simple feature engineering and complex temporal graph learning.

References

1. Y. Koren. *Collaborative Filtering with Temporal Dynamics*. CACM, 2010.
2. J. Ni, J. Li, J. McAuley. *Amazon Review Data*. 2018.
3. M. Wedel, W. Kamakura. *Market Segmentation: Conceptual and Methodological Foundations*. Springer, 2000.
4. S. Zhang, L. Yao, A. Sun, Y. Tay. *Deep Learning Based Recommender System: A Survey and New Perspectives*. ACM CSUR, 2021.
5. A. Bifet, R. Gavalda. *Learning from Time-Changing Data with Adaptive Windowing (ADWIN)*. SDM, 2007.
6. D. Arthur, S. Vassilvitskii. *k-means++: The Advantages of Careful Seeding*. SODA, 2007.
7. W. Hamilton, R. Ying, J. Leskovec. *Inductive Representation Learning on Large Graphs*. NeurIPS, 2017.
8. P. Velickovic *et al.Graph Attention Networks*. ICLR, 2018.