Latent Customer Behaviour Segmentation in Online Retail Using Neural Embeddings and Unsupervised Clustering

MSc. Business Analytics, Warwick Business School

Student ID: 5672124

ABSTRACT

This study presents an integrated machine learning pipeline for customer segmentation and retention analysis using clickstream data from a large online retailer. Traditional Recency-Frequency-Monetary (RFM) proxies are first engineered at the session level and clustered via K-Means to create five interpretable baseline segments. In parallel, session behaviours are encoded with hybrid concatenation of Word2Vec (item-level) and Doc2Vec (session-level) embeddings, further denoised through PCA, and clustered using MiniBatch K-Means, Gaussian Mixture Models, Agglomerative Ward linkage, DBSCAN and HDBSCAN. A shared evaluation sample of 70,000 sessions enables fair comparison using Silhouette, Calinski-Harabasz, Davies-Bouldin scores, and Adjusted Rand Index (ARI). Temporal drift is assessed by splitting sessions into early and late periods. Feature attributions from a Random Forest surrogate with SHAP values helps to identify the fundamental drivers of segment assignment. Results demonstrate that embedding based clusters can reveal latent intent patterns, offering insights beyond standalone RFM models can offer.

By weaving advanced neural representations with time-tested RFM baselines, comprehensive clustering, and rigorous validation, this dissertation offers a scalable, reproducible, and interpretable framework for modern e-commerce personalization. This study attempts to not just stand exploratory but also equip practitioners with technical and practical intricacies to deploy embedding driven segmentation that not only uncovers latent behavioural archetypes but also communicates them in clear and business relevant terms, ensuring that data-driven insights translate into strategic action and sustained competitive advantage.

Acknowledgments

I am extremely grateful to my supervisor Dr. Siamak Naderi, for his continued guidance, thoughtful feedback, and support throughout this research. To my family - many thanks for your encouragement, patience, and belief during the long hours of analysis and writing. I would also like to thank my peers and colleagues who provided useful feedback and stimulating discussions that helped to develop my thinking and enhanced this study. I would sincerely like to thank and acknowledge the fellow scholars and researchers whose foundational work in behavioural segmentation and representation learning has informed and inspired this dissertation. Finally, I would like to sincerely thank and acknowledge the Warwick Library team for providing seamless access to the literature and resources that have shaped and facilitated this work.

Table Of Contents

Table Of Figures

Table Of Tables

1. Introduction

Each online visit can be viewed as a short story including a few product views, their comparisons, sometimes an add-to-cart, and occasionally a purchase. Turning these stories into useful segments supports user experience personalisation, product placement, and more optimized use of marketing spend. The difficulty is the data itself, as sessions are short and noisy, popular products appear everywhere, while signals indicating a change in intent are limited. Simple hand-crafted features often miss to identify latent behaviour patterns that matters. Two patterns justify a different approach. First, people often explore groups of products in the same visit. Second, the order of actions carries meaning that is adding to cart after several views differs from a single quick view and exit. Hand-crafted features struggle to capture either pattern.

Representation learning offers an alternative. In their original work, Mikolov et al. (2013) trained neural models (skip-gram/CBOW) with negative sampling to predict nearby tokens, this method is fast and turns co-occurrence into geometry, but it is tied to a small context window and can over-weight very frequent tokens. This is why subsampling and re-weighting are commonly used, and justification for this study to incorporate simple weighting and normalisation with the embeddings. Le & Mikolov (2014) extended this idea to whole documents (Doc2Vec, DM/DBOW), giving a vector for each text. Doc2Vec captures short sequences along with local context, yet it can be unstable for very short documents and is sensitive to training settings. Combining the two variants and giving more passes helps with short sessions and together, these methods carry over well from language to retail sessions because ‘context’ here implies behaviour within a visit.

Evidence supports a shift beyond hand-crafted features. In a comparative survey, Curiskis et al. (2020) reported that learned vectors often clustered better than bag-of-words features, given preprocessing and dimensionality reduction were handled carefully. Accounting for this, the implemented pipeline in the study denoises using PCA before clustering. On the practical side, Bayrak (2022) described ‘customer embeddings’ in a retail setting and the case study demonstrated feasibility in production, but results were context specific. Also, this study focused on feasibility rather than proving advantage.
Interpretability of derived segments remains crucial. Commercial practices still prefer recency, frequency, and monetary value (RFM) because these measures align with

lifecycle, engagement, and value. RFM is not an obsolete technique and remains useful for diagnosis and targeting in service-based industries and retail (Ho et al., 2023; Stormi et al., 2020). It is also limited as the indicators are rough and ignore sequence. For that reason, an RFM baseline is kept next to the learned vectors. The pairing allows embeddings to find structure while RFM provides explanation in terms that commercial practices still follow.

No single clustering method fits every dataset. Different clustering techniques look for different shapes such as compact groups, overlapping mixtures, dense islands, or hierarchical splits. In a broad survey, Ezugwu et al. (2022) argued for evaluating a portfolio of clustering tools rather than committing early to one method, especially when data scale and quality vary across seasons and campaigns. The survey also refreshes that runtime and memory matter at scale, and that parameter sensitivity is critical. The design here follows this advice: multiple families of models are compared, heavy models can be fit on large samples and then propagated, and results are then cross-validated. Visual maps (simple 2-D projections) help stakeholders observe separation and overlap, but they serve as descriptive aids and model choices are decided in the derived high dimension feature spaces.

Evaluation needs discipline. Internal indices tell us whether groups are geometrically coherent without any external labels or ground truth. The Silhouette is a standard metric as it enquires whether points are closer to their own group than to others (Rousseeuw, 1987). Still, it has biases as it prefers compact, roughly convex clusters and can be misleading when one cluster dominates. Hence complementary indices are reported in this study and why the silhouette is computed on a shared and balanced evaluation slice rather than on a convenient subset. Agreement measures then check whether different models indicate a similar story on the same data, which lowers the risk of pursuing a single model’s bias.

Time adds another complexity. Retail behaviour often drifts that is holiday promotions and new product launches can create patterns that diminish later. Hu et al. (2024) emphasised that stability should be checked explicitly, since clusters that look strong at one point can weaken as the distribution shifts. A simple early-vs-late comparison is therefore essential as it guards against interpreting seasonal cluster behaviour patterns as durable segments.

Segments also need a narrative that travels beyond technical grounding. Each group is summarised in a persona table using their respective size and share, medians of recency, frequency, and a simple monetary proxy, and the mix of views, add-to-baskets, and transactions which enables CRM, user experience, and merchandising efforts to act with a shared understanding. Feature attribution tools can indicate detail about which features or factors most influence the separation of groups. Here the limitation is explicit, Verdinelli & Wasserman (2023) discussed how attribution methods allocate credit under a model's assumptions and what kinds of errors can arise due to same. While Zhao et al. (2024) demonstrated that correlated features can split or trade credit in ways that look fundamental but are not. Thus, taking caution, this study treats such explanations as descriptive patterns that characterise segments and are not used to claim causes.

While prior work has applied embeddings or clustering in isolation, little has been done to combine session and item level embeddings with a wide range of clustering methods to produce segments that are both enduring and interpretable.

The aim of this dissertation is to evaluate whether embedding based representations combined with a portfolio of clustering methods can deliver clearer and more actionable customer segments than classical RFM baselines. Some research questions that this study attempts to address in the process are as follow:

- Do learned session vectors (Word2Vec / Doc2Vec) give clearer, more coherent segments than a RFM baseline that managers can interpret (Le & Mikolov, 2014; Mikolov et al., 2013; Stormi et al., 2020)?
- Under a shared and deterministic evaluation setup, which clustering approach performs best across the pipeline and dataset, recognising that each clustering approach makes different shape assumptions (Ezugwu et al., 2022)?
- Across different models, how consistent are segment assignments, and where do disagreements point to some systematic differences rather than noise?
- Are segments stable over time, or do seasonal and campaign effects significantly impact cluster compositions (Hu et al., 2024)?
- Which behaviours most characterise each segment in a way that supports decisions in CRM, User-experience design, and merchandising (Ho et al., 2023)?

This study contributes three elements:

- A representation-based pipeline: compact session vectors to capture what users looked at together and how they moved through a visit, with an RFM baseline kept for clarity and trust.
- A portfolio-based segmentation design with shared evaluation sample for internal metrics, agreement across models, and simple time based stability checks, this approach is encouraged by scale aware surveys (Ezugwu et al., 2022) and grounded in common validation practice (Rousseeuw, 1987).
- Clear persona outputs and saved information to ensure results are reproducible, auditable, and easy to tune in later work.

Scope:

The Rocket Retail data do not include price or monetary information, making estimation of conversion-lift out of scope. Some clustering methods are sensitive to parameter settings and to local density of data, however this is mitigated by comparing several families and checking their agreement. Explanations are framed as associations (correlations) rather than causes, consistent with current guidance on attribution (Zhao et al., 2024). The practical aim remains to obtain segments that are clear, stable, and useful.

2. Literature Review

2.1. Introduction

Customer segmentation remains one of the oldest and most debated practices in marketing and analytics. The core idea behind it is to divide customers into groups so that services, promotions, and products can be better targeted and personalized. However, this idea becomes complex when applied to modern retail data. As such datasets with millions of event logs, can amplify both the potential and the limitations of segmentation. This review investigates the development of methods for segmenting customers, from early traditional descriptive techniques to modern neural embeddings and clustering of latent behavioural data. It addresses not only key arguments but also the assumptions, gaps, and rival explanations that remain critical to the domain.

For every method or theoretical element considered, the review examines what the core issues are, what conclusions can or cannot be drawn, how strong the evidence is, and what significant information may be missing. This structure for review avoids the appeal to treat each contribution as if it has answered all intended research questions. Instead, it focuses on how relevant debates overlap and why adaptation of portfolio of clustering techniques, rather than reliance on a single clustering model, is increasingly necessary.

2.2. Foundations of Customer Segmentation

Classical segmentation techniques relied heavily on demographic and psychographic categories. Supported by market research surveys, static variables such as age, income, and location are some of such main dividing features. Tsiptsis & Chorianopoulos (2009) explained that such variables were simple to collect and easy to explain. However, segmentation based on such features leads to a strong assumption that people who look similar that is share some of these demographic and psychographic categories would behave similarly in the market. This assumption is partly true but often misleading, as it may be noted that two consumers of the same age and income may show opposite patterns of loyalty or risk tolerance.

The rise of recency-frequency-monetary (RFM) analysis enabled incorporation of behavioural element to customer segmentation. By using transaction data, firms could classify customers based on how recently they purchased, how often they transacted, and how much they spent. The appeal of RFM stays grounded in its clarity as stakeholders could instantly see the link between segments and the values that drive them. Yet even here ambiguity arises as what counts as 'recent' is not deterministic, a week in groceries retail space looks very different from a year in luxury goods space. Tsiptsis & Chorianopoulos (2009) alerted that the thresholds selected for recency or frequency are arbitrary and often based on simple heuristic rules rather than grounding in statistical justifications. This reflects that labels like 'loyal', 'inactive' or 'valuable' may sound precise but remain subjective and are results of assumed thresholds.

Later refinements such as LRFMS inspired clustering (Wang et al., 2024) tried to reduce this arbitrariness by implementing unsupervised methods to identify natural clusters in RFM data. Still, the core assumption persists that a handful of simple proxies could capture nuanced essence of behaviour. In practice, with high-volume traffic and personalised experience, modern customers now generate complex signals beyond these measures. This does not imply that RFM or inspired approaches are obsolete. These techniques continue to provide a baseline of interpretability; however, its explanatory reach is limited when dealing with richness of online behavioural data.

Taken together, these approaches show that while RFM provides transparency, its arbitrariness and subjectivity limit its suitability for modern digital data with so many nuanced and latent features, motivating this study's shift towards behavioural and sequence aware approaches.

2.3. Behavioural Data and Clickstream Analysis

Digital commerce introduced a new kind of dataset, referred as clickstream logs. Montgomery et al. (2004) highlighted that clickstream records every user interaction in a timely fashion, resulting in a fine-grained sequence of views, searches, and clicks. Availability of such rich data has resulted in fundamental evolution of segmentation problems. Instead of one purchase record per customer per period, analysts now have multiple logs of actions per session.

This expanded the opportunity to clearly understand intent as it unfolds. However, the problem was equally clear that such datasets are messy, sparse, and unbalanced.

Ambiguity remains critical, as it still cannot be interpreted if a long session with many views indicate interest or confusion, or if a quick bounce from one page to another should be considered as a sign of efficiency or lack of relevance. Classical models that reduce customers to a single number cannot capture such contexts. Moe (2003) claimed that clickstream behaviour reveals heterogeneity in decision-making processes, but without careful modelling it is easy to misinterpret the actual context behind a pattern or segment. A high view counts could be interpreted as engagement but might also indicate difficulty in finding the right product. This reflects the question about rival causes as the same statistics may have multiple explanations.

Researchers have long argued that applying simple recency and frequency-based measures to clickstream data often struggles to distinguish genuinely high-value customers from those who browse without intent. Tsiptsis & Chorianopoulos (2009) outlined how these classical models can oversimplify behavioural heterogeneity, while more recent studies also highlighted that digital browsing patterns introduce noise that these proxies cannot capture (Stormi et al., 2020). This implies that segmentation must move beyond handcrafted features. At the same time, there is a risk of ignorance that some studies adopt by assuming more data automatically yields better insights. Since a clear theory to explain behaviour is lacking (no ground truth), additional rows of click logs may only amplify noise. This suggests that the evidence base for robust behavioural segmentation is still mixed, promising in theory but fragile in practice.

2.4. Representation Learning for Behavioural Sequences

Le & Mikolov (2014) extended this to Doc2Vec, where whole documents or sessions (in case of clickstream logs) receive their own vector. This overcame some of the limitations of item only models by encoding local order and context within sessions, giving more context. The method assumed that even short sequences could carry enough signal to stabilise a session vector, but this assumption is not always reliable. A two-click session may not provide sufficient evidence to infer intent, and not all co-occurrences signify the same behavioural similarity. These limitations have been noted in follow up studies where embeddings, although powerful, were treated with caution in sparse contexts.

TF-IDF has often been treated as a baseline for such tasks because it is transparent and computationally simple. However, its weaknesses become clear when applied to short and noisy behavioural logs. Susanto et al. (2023), in a comparative study on Twitter data, showed that TF-IDF tended to perform poorly in occasional sessions, reducing interactions to weighted counts while failing to capture the actual sequential context. Zhan (2025) similarly demonstrated that TF-IDF could achieve high accuracy on training data in sentiment analysis, but this often reflected overfitting, with significantly weaker generalisation to unseen samples. In contrast, Word2Vec and Doc2Vec consistently showed greater robustness, as they embed distributional context rather than relying solely on raw token frequencies.

This evidence provides a clear rationale for combining the strengths of both embedding families. Word2Vec captures item level associations, such as products often browsed or purchased together, while Doc2Vec captures session level intent by embedding the broader context in which these items appear. Concatenating the two embeddings therefore offers a richer feature space than either model alone. Bayrak (2022) demonstrated that embeddings of customers, derived from purchase histories, significantly improved clustering quality compared to RFM proxies. Skotis & Livas (2024) further showed that Doc2Vec could separate users not just by activity levels but also by stylistic and engagement differences. Together these studies suggest that integrating both local and global signals yields more stable and actionable representations.

Curiskis et al. (2020) reinforced this point in their evaluation of clustering methods on social media data, where embedding-based features often outperformed sparse alternatives such as TF-IDF. However, they emphasised that results depended heavily on preprocessing choices and dimensionality reduction, highlighting the risk of instability. Concatenating Word2Vec and Doc2Vec mitigates this concern by retaining complementary perspectives on behaviour, Word2Vec ensures item-level resolution, while Doc2Vec stabilises broader session-level patterns. This dual view allows segmentation models to detect both patterns of item co-occurrence and trajectories of user intent.

More recently, evidence across domains confirms that embedding choices matter. Skotis & Livas (2024) showed that user embeddings generalised across online communities but differed depending on whether Doc2Vec or SBERT was used. Susanto et al. (2023) and Zhan (2025) highlight the instability introduced by relying on a single technique, suggesting that robustness depends on evaluating multiple embedding approaches under consistent frameworks. This evidence suggests no single embedding is sufficient; combining Word2Vec and Doc2Vec provides a richer and more stable basis for behavioural segmentation

2.5. Clustering Methods for Latent Behaviour

Once features are constructed, clustering becomes the main tool for segmentation. K-Means has long been the default choice as it is computationally efficient and easy to explain, but it assumes spherical clusters of similar size. Ezugwu et al. (2022) reviewed clustering methods and concluded that no single algorithm dominates across all applications. Thus, the issue is not to find the one 'best' model, but to understand trade-offs of each model in terms of scalability, interpretability, and fit to the data at hand.

Gaussian Mixture Models (GMM)(Reynolds, 2008) extend K-Means by allowing customers to belong to clusters probabilistically, better capturing overlapping behaviours. Yet they assume Gaussian distributions, which may not hold for clickstream data as often here activity levels are skewed, as most sessions are short. Density-based methods like DBSCAN (Ester et al., 1996) and HDBSCAN relax these assumptions. They detect irregular cluster shapes and can treat outliers as noise, which is valuable when sessions are irregular. But the drawback is sensitivity as parameters like epsilon and minimum cluster size can radically change the outcomes. The ambiguity of what constitutes as 'noise' becomes a methodological risk, a rare but strategically important customer might be labelled as an outlier and thus lost from further analysis. Hierarchical clustering offers a different lens on behaviour, producing dendrograms that show nested groups. Such visual hierarchy is often appreciated by stakeholders because it mirrors how people naturally think about ‘segments within segments. The difficulty remains scalability, classical agglomerative methods have substantial time and memory costs, which makes them inconvenient when the dataset contains millions of session records and embeddings sit in higher dimensional vector space (Ezugwu et al., 2022; Jain, 2010). In practice, either down-sampling to approximate the connectivity graph, or switching to hybrids that fit a hierarchy on a representative subset and then propagate labels to the full set via k-nearest neighbours is implemented. This design principle has been also adopted by this study for Ward-linkage on sampled data. Visual tools remain useful, time-aware mappings and related projections have been used to track how clusters evolve across periods without building a full dendrogram for every point (Sarlin & Yao, 2013). Recent algorithmic work further explores fast and explainable alternatives to heavy hierarchies; for example, CLASSIX aggregates nearby points by sorting and then merges groups, reaching near - linear behaviour on many problems while keeping the rationale visible (Chen & Güttel, 2024). For Rocket Retail, the scale of data and these trade-offs justify our sampling-plus-propagation choice as it preserves a readable tree on the subset for communication, but avoids the high computational cost associated with a full agglomerative fit on the entire population.

The evidence base is strong in terms of comparative surveys but weaker in applied e-commerce practical setting as many papers test clustering on benchmark datasets but omit details about runtime, scalability, or integration into workflows. Statistics can also be deceptive. A high silhouette score may suggest neat separation, but if most customers end up in one large cluster, the result is not actionable. Rival causes can complicate interpretation; clusters may reflect marketing campaigns or stock changes rather than intrinsic user types (temporal shifts). Therefore, this study compares multiple clustering families under a consistent evaluation framework rather than committing to one algorithm.

2.6. Evaluation and Interpretability

Evaluation is critical for segmentation credibility. Internal metrics such as Silhouette (Rousseeuw, 1987), Davies-Bouldin, and Calinski-Harabasz, provide useful insight into quantifying compactness and separation. But they are not dependable, as Silhouette assumes convex cluster shapes and can undervalue elongated or chain like groups that may still be meaningful. The underlying assumption is that geometry implies usefulness, which is not always the case for business applications.

Agreement metrics such as the Adjusted Rand Index (ARI) and Adjusted Mutual Information (AMI) provide a way of testing whether different models converge on similar structures. In clustering, this form of cross model validation is important because internal metrics alone can be misleading, surveys of clustering practice emphasise that robustness comes not from a single high silhouette value but from convergence across different algorithms and feature spaces (Ezugwu et al., 2022). Yet just like internal metrices, these indices themselves are statistical artefacts. They capture whether two partitions agree mathematically but do not indicate whether the resulting clusters are meaningful for merchandising, campaign design, or customer experience. This gap between numerical robustness and business relevance remains one of the central challenges in applying segmentation to real-world retail contexts.

Temporal stability checks are less common but increasingly important. Hu et al. (2024) stressed that clusters discovered at one time may dissolve later and ignoring this drift is an omission that weakens many studies. In retail context, this means that a cluster of ‘holiday bargain hunters’ could vanish by spring, leaving only misleading labels behind. A simple split of sessions into early and late periods can reveal whether clusters are genuinely stable or merely seasonal relics.

Interpretability tools aim to bridge the gap between statistics and narratives. SHAP is used in this study to explain how different behavioural features contribute to assigning sessions to clusters, as it provides intuitive and local explanations that reveal which actions most distinguish each segment. While Shapley values have become a standard for feature attribution in clustering explainability, recent critiques warn about its limitations when using them in behavioural segmentation contexts. Verdinelli & Wasserman (2023) demonstrated how Shapley values does not eliminate the impact of feature correlation. This may obscure interpretability when the predictors are corelated. Zhao et al. (2024) broke down Shapley explanation errors into observation and structural bias, demonstrating a trade-off that can lead to under or over informative attributions due to distributional assumptions. These findings alert against interpreting SHAP attributions as causal claims and that they should be treated as just descriptive tools. In the clustering pipeline, SHAP should therefore be utilized to identify associational patterns (for example, identifying which session-level features most strongly differentiate clusters), while acknowledging that these attributions reflect correlations rather than direct behavioural causes. Integrating these critiques to the study, ensures that explainability remains transparent about its assumptions and limitations, aligning with calls for reflexivity in interpretability research.

Persona construction extends interpretability into communication. (Moe, 2003) demonstrated that customers can be profiled by their position in funnels, whether browsing, searching, or purchasing as he claimed that user's change in intent can be assessed through their browsing activity. Recent contributions such as Masalma (2024) and Ridwan (2025) argue that combining behavioural signals with demographic or contextual variables provides more support for decision-making process. These approaches remind us that segmentation is not only a technical exercise but also a communication challenge. Recent studies increasingly recognise that interpretability is not just a technical property but a communicative one. Outputs need to be translated into forms that audiences with different level of expertise can understand and act upon. Persona tables and explanatory plots are one step, but interactive dashboards are now frequently used in both research and industry to make segmentation insights transparent to non-technical stakeholders. Such interfaces allow decision makers to explore clusters visually, test assumptions, and align findings with business objectives.

2.7. Gaps, Rival Causes, and Conclusions

Across the literature several issues were noted to recur, many studies assume that embedding context itself implies behavioural meaning. This assumption simplifies computation but risks ignoring key external drivers such as promotions or price. Rival causes are therefore not fully controlled. Also, evaluation often stops at internal metrics, ignoring temporal stability checks or business validation. This leads to conclusions that may appear statistically viable but operationally weak.

Ambiguities remain in terminologies like ‘intent’, ‘loyalty’ or ‘engagement’, that are used loosely with different meanings across studies. Misconceptions appear when correlation is presented as causation, particularly in attribution methods. Evidence strength also varies as some contributions rest on rigorous benchmarks, others on narrow case studies. Omitted information is common, especially regarding reproducibility and longitudinal testing (in context of online retail, this refers to evaluating methods not just once on a single snapshot of data, but repeatedly over time to see whether the findings hold up- Temporal drift).

The reasonable conclusion is that no single method suffices. Instead, a portfolio approach that combines interpretable baselines (RFM), modern embeddings (Word2Vec, Doc2Vec, TF-IDF), multiple clustering families, and layered evaluation (internal metrics, agreement, temporal drift, personas) is most credible. This aligns with Ezugwu et al. (2022) survey and practical reports such as Bayrak (2022). It also explains why this dissertation positions itself as a link between innovation and interpretability, where embeddings enable discovery of latent behaviour, but baselines and explanations ensure trust.
This further reinforces the view that methodological rigour must be coupled with communicative clarity. A portfolio of models can only deliver value if the results can be inspected, trusted, and reused by others. Dashboards and other interactive features provide a pathway for this, ensuring that insights reflect beyond model settings and can be applied into the practical domains of marketing, merchandising, and user experience.

3. Methodology

3.1. Research design

This study approaches customer segmentation as a problem of interpreting sequences of behaviour rather than a static snapshot of attributes. The design proceeds in a layered approach, including exploratory data analysis, data load and preparation, constructing sessions from clickstream log, representation learning using neural embeddings, dimensionality reduction, a portfolio of clustering algorithms, multi-lens evaluation, and finally interpretation and stakeholder translation via a dashboard At each stage, choices were validated against EDA evidence and reasoned with rival explanations in mind (Moe, 2003). The aim was deriving credible, reproducible segments with clear managerial value. Below Figure 1, illustrates the methodological pipeline that drives this study.

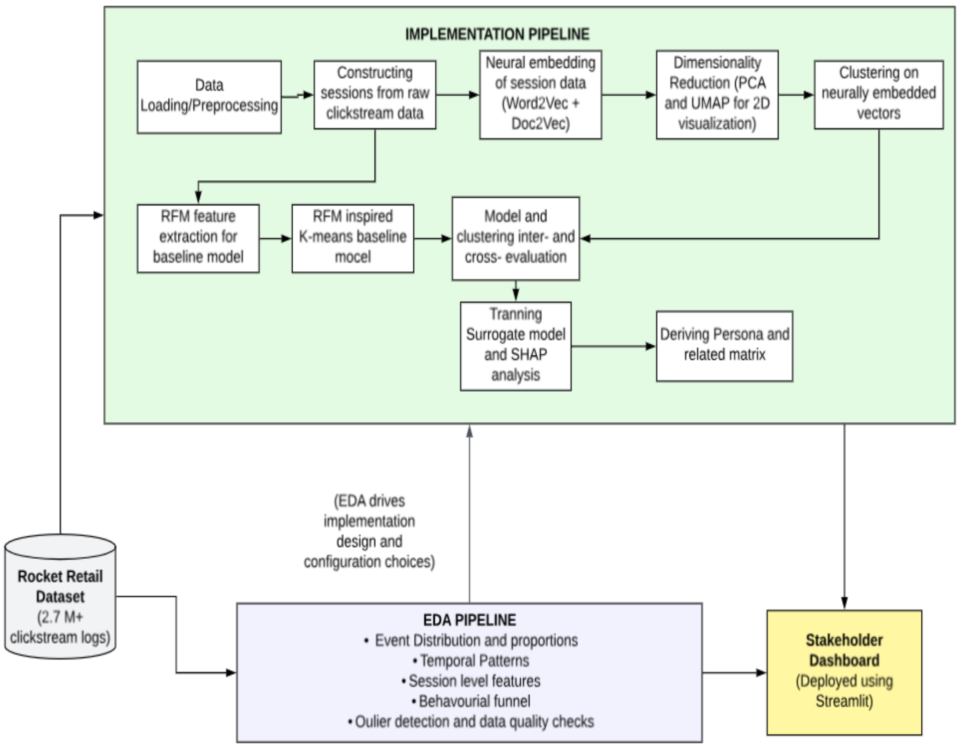
IMPLEMENTATION PIPELINE
Data Loading/Preprocessing
Constructing sessions from raw clickstream data
Neural embedding of session data (Word2Vec + Doc2Vec)
Dimensionality Reduction (PCA and UMAP for 2D visualization)
Clustering on neurally embedded vectors
RFM feature extraction for baseline model
RFM inspired K-means baseline mocel
Model and clustering inter- and cross- evaluation
Tranning Surrogate model and SHAP analysis
Deriving Persona and related matrix
(EDA drives implementation design and configuration choices)
Rocket Retail Dataset (2.7 M+ clickstream logs)
EDA PIPELINE
• Event Distribution and proportions
• Temporal Patterns
• Session level features
• Behavourial funnel
• Oulier detection and data quality checks
Stakeholder Dashboard (Deployed using Streamlit)

Figure 1. Methodological Pipeline

3.2. Data Source

The dataset facilitating this study is the Retail Rocket clickstream public dataset containing 2,756,101 event-level records spanning over 1,761,660 sessions from 1,407,580 visitors across 235,061 items, covering May to September 2015. Each record included a timestamp (UNIX milliseconds converted to seconds via TS_DIV=1000), visitorid, itemid, and event type (view, addtocart, and transaction) (Moe, 2003; Montgomery et al., 2004)

3.3. Session Construction

The raw clickstream logs lacked session keys, so sessions were constructed using a 30-minute inactivity heuristic (SESSION_GAP_SEC = 30*60), resetting if visitor changed, following clickstream analysis conventions (Montgomery et al., 2004). Data cleaning involved:

- Removal of single-event sessions as potential noise and records above the 99.5th percentile for duration (OUTLIER_DUR_PCTL = 99.5) to eliminate extreme outliers.
- Filtering sessions exceeding 120 events per minute (OUTLIER_EPM_THRESH = 120)

This approach ensured that bot-like or irregular sequences did not distort embedding geometry (Curiskis et al., 2020).

3.4. Exploratory Data Analysis

EDA was performed to test assumptions and drive implementation pipeline configuration and design. Three findings proved critical:

- Event imbalance was significant: views dominated, while adds-to-cart and transactions were rare (Figure 2). In such cases models trained on simple frequency based features would overfit. This justified the study's adoption of sequential embeddings that encode context, not just frequency (Curiskis et al., 2020).

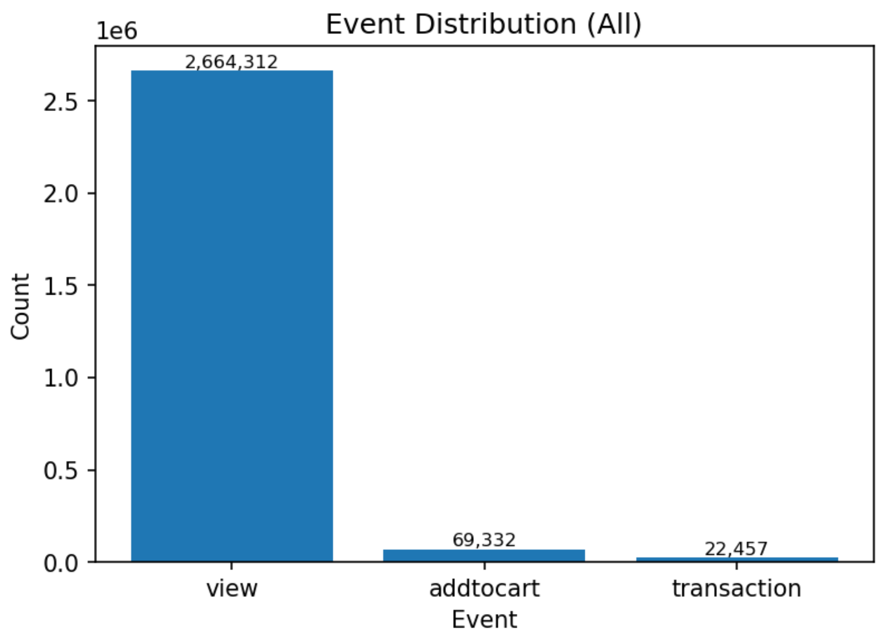

Figure 2. Event Distribution in Rocket Retail Dataset

- Session funnel provided business relevant grounding. The counts in Table 1 show steep erosion from browsing to purchase.

Stage	Sessions reached	Share of total (approx.)
View	1,755,766	99.7%
Add-to-cart	38,726	2.2%
Transaction	11,143	0.6%

Table 1. Session funnel (session-level progression)

This pattern reminded that segmentation cannot be judged on activity volume alone, it must be able to separate conversion prone from non-conversion sessions (Moe, 2003).

- Temporal rhythms (hour- and day- of week cycles) suggested that segments may drift over time. Temporal stability checks were added in evaluation, comparing early vs late periods (Hu et al., 2024).

Sequential diagnostics highlighted repetitive view → view chains alongside rarer transitions linked to other events. These findings informed the context window for Word2Vec and motivated adding a session-level embedding (Doc2Vec) to capture user intent beyond local co-occurrence (Le & Mikolov, 2014). EDA pipeline code, plots and output tables appear in Appendix A.

3.5. Feature Engineering and RFM Baseline

Session-level features extracted include duration, event intensity, item diversity and behavioural flags using one-hot encoding (pd.get_dummies(last_event, prefix="last_evt", dtype=int)). Log-transformation (using numpy.log1p()) were used to address skewed distributions while preserving zero values.

Baseline RFM implementation evaluated multiple monetary weight schemes (w = 0.3, 0.5, 0.7) where m_proxy = transactions + w × addtocart, addressing the absence of actual monetary data while maintaining stakeholder interpretability (Ho et al., 2023; Stormi et al., 2020).

3.6. Representation Learning

Two complementary families of sequential learning were used:

- Item-context embeddings (Word2Vec): Skip-gram model with negative sampling trained on session sequences. Window size 5, min_count=2 was set to avoid rare noise, 64-dimensional vectors balanced risk of overfitting on short sequences. Session vectors constructed via mean pooling with Smooth Inverse Frequency (SIF) weighting reducing dominance of frequent view contexts (Mikolov et al., 2013).
- Session-intent embeddings (Doc2Vec): PV-DM vectors of size 96 treating sessions as documents were obtained. Increased epochs and small PV-DBOW variant ensured robustness on short sessions (Le & Mikolov, 2014).

Final representational vector was obtained by concatenating SIF-weighted Word2Vec with Doc2Vec vectors, following evidence that hybrid representations can improve clustering consistency in sparse behavioural settings (Bayrak, 2022; Curiskis et al., 2020).

3.7. Dimensionality reduction and visualisation

Before clustering, the concatenated embeddings were denoised with PCA (reducing embeddings to 96 components, targeting 80% variance explanation even after dimension reduction). PCA reduced noise and aligned the input space across models ensuring that any performance difference reflected shape assumptions of respective models, instead of raw dimensionality (Curiskis et al., 2020). For visualization, the PCA output was project to 2-D using UMAP. UMAP plots are considered descriptive maps rather than evidence of separation, clustering was performed in the PCA space. Below Figure depicts UMAP projection of session embeddings (2-D) that was obtained during execution, illustrating latent structure in clickstream behaviour.

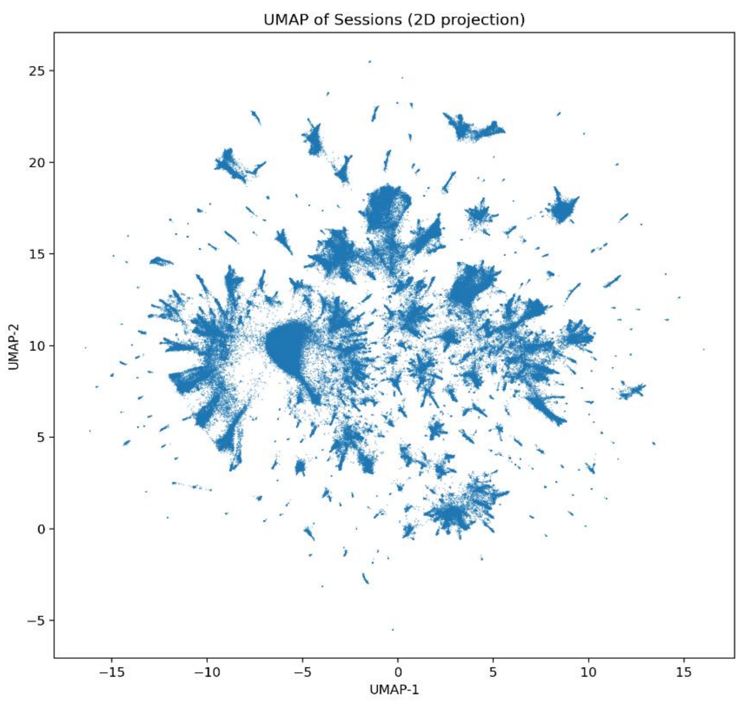

Figure **3**. UMAP projection of session embeddings (2-D) visualising underlying structure in Rocket Retail clickstream behaviour.

3.8. Clustering Portfolio and Calibration

No single clustering algorithm can fit all cluster shapes. We therefore adopt a portfolio of clustering techniques (Ezugwu et al., 2022):

- MiniBatch K-Means: for efficient spherical splits across a grid of k values (stability via multiple iterations).
- Gaussian Mixture Models (GMM) with diagonal covariance to derive overlapping clusters and allow soft assignments (Reynolds, 2008).
- HDBSCAN was implemented as the primary density-based method to find 'islands' (clusters) of similar behaviour wile explicitly labelling noise, evaluated with Euclidean and cosine distances.
- DBSCAN was included as a calibration method using k-distance plot for ε selection (Ester et al., 1996).
- Agglomerative (Ward) was implemented on a representative subset to explore hierarchical relationships and then propagate labels.

Where hyper-parameter tuning and other settings mattered (such as PCA components, HDBSCAN metric, etc.), we ran small sensitivity checks and retained the simplest configuration that preserved efficient cluster structures.

3.9. Evaluation And Agreement

To avoid biasness towards a single metric, multi-perspective evaluation was adopted for this study:

- Internal validity: Silhouette, Calinski-Harabasz, and Davies-Bouldin metrices quantified compactness and separation amongst clusters. Silhouette was approached with caution as it prefers convex shapes (Rousseeuw, 1987).
- Cross-model agreement: ARI/AMI checked if different algorithms derive similar structures, guarding against single model biasness. (Ezugwu et al., 2022).

- Temporal stability: Following Hu et al. (2024), sessionised data was split into early vs late periods and the endurance of segment composition was evaluated to account for seasonality drifts.

3.10. Interpretation and Reproducibility

Clusters were translated into personas based on their size/share, central tendencies of sessional features (included recency/frequency proxies, duration, item diversity), and crucially funnel progression of each cluster. This informed which segments were 'browse-only', 'cart-heavy but hesitant', or 'purchase prone'. For boundary explanations, a Random Forest surrogate on transparent session features was trained and SHAP values were computed. This study identifies SHAP as associational and not causal, as recent critiques cautioned that correlation and feature dependence can bias this crediting process (Verdinelli & Wasserman, 2023; Zhao et al., 2024). Therefore, only global and per-cluster top features with context were reported and interpreted alongside funnel behaviour rather than in isolation.

Reproducibility was achieved by implementing:

- Fixed random seed (42) across all stochastic operations.
- Centralized configuration.
- Standardized outputs: 150 DPI PNG figures optimized for dashboard integration.
- Automated organization of output (separate directories for figures, tables, and reports).

Dashboard deployment (URL: https://rocketretaildashboard.streamlit.app/) via Streamlit Community Cloud presented EDA panels, model comparisons, persona tables, and UMAP overlays, bridging analysis with stakeholder communication without exposure to model intricacies.

3.11. Ethics, limitations, and reflexivity

The dataset used for this study is anonymized, and segments represent behavioural patterns and not identities. Nonetheless, segmentation can prove deceptive if activity is mistaken for value or fairness for one measure (Binns, 2018). Accounting for this, the study therefore:

(i) omits unnecessary automation
(ii) double-check stability with time
(iii) Refrains from definitive claims where explanation may become entangled.

Sessionisation was rule-based and may severe single user's interactions that occur over multiple visits or sessions, also while long sessions may indicate confusion rather than intent (the actual context behind certain behaviour remains non-determinable). These reservations are accepted by this study and made explicit during interpretation, and the dashboard reveals them rather than hiding them.

3.12. Summary

Methodological choices were driven by what the data actually indicated and by what stakeholders require to act on, while grounding it with practical tunning configurations that were evaluated and chosen over multiple iterations such that they demonstrated both computational efficiency and conceptual rigor. Time was retained in epochs, sessions were constructed with a 30-minute gap, extreme noise were removed, and embeddings were used to capture both local context and session intent. PCA provided a clean feature space for a range of clustering methods that were implemented, and evaluation was designed to be multi-perspective and stability aware. Personas, funnel lifts, and a deployed Streamlit dashboard bridged the gap between modelling to decision. The net result was a segmentation that is not only technically credible but also communicable and useful.

4. Results

The analysis of the Retail Rocket clickstream dataset revealed crucial insights into customer segmentations achieved through a consolidated pipeline deploying neural embeddings combined with a portfolio of clustering algorithms.
The dataset reflected complexity typical of modern e-commerce behavioural data. Initial exploratory data analysis found the expected dominance of page views over add-to-cart and transaction events, highlighting a substantial class imbalance (Figure 2). This imbalance emphasized the necessity of embedding based representations rather than simple frequency or count features, given the rare conversion signals (desired intent) amidst large volume of browsing activity. The behavioural funnel further confirmed a steep drop-off, with just 2.2% of sessions progressing to add-to-cart and only 0.6% resulting in actual transactions, emphasising the critical business value of segmenting not just by activity volume (leading to frequency counts) but also by latent engagement and intent (Moe, 2003).
Feature engineering constructed session-level metrics such as duration, event intensity, and item diversity, accompanied by one hot encoded last event flags (as a proxy for Recency). To address skewness while avoiding a compromise on interpretability, normalized (log-transformed) versions of these features were used. These features provided a strong RFM proxy baseline that matched established segmentation practice. This setup allowed for comparative evaluations later (Ho et al., 2023).

The unique perspective of this study came from the representation learning step, which combined two complementary embedding types: Word2Vec and Doc2Vec. This hybrid approach made use of both local item associations and broader session meanings. Literature supports this combined embedding in scattered behavioural data feature space (Bayrak, 2022; Curiskis et al., 2020). Concatenating these embeddings gave a rich 208 dimensional feature space that was then reduced via PCA to 96 components, maintaining approximately 80% of explained variance, this balanced noise reduction and information retention (Le & Mikolov, 2014).
Clustering portfolio covered centroid based (Minibatch K-Means), mixture based (Gaussian Mixture Models), density based (HDBSCAN and DBSCAN), and hierarchical (Agglomerative Ward linkage with on label propagation) methodologies. Each model was tuned and evaluated on a shared and sampled subset of 70,000 sessions, ensuring a fair cross validation between the algorithms. HDBSCAN failed to

identify robust clusters under specified parameters, while DBSCAN produced single cluster with noise, possibly due to its sensitivity towards parameter selection – hyper parameter tuning (Ezugwu et al., 2022). Centroid-based and hierarchical methods produced more geometrically representative clusters. K-Means (at k=12) and Agglomerative clustering achieved silhouette scores around 0.06. This suggests a modest but significant separation in high-dimensional behavioural spaces.

In addition to internal validity metrics, we evaluated cross-model agreement using the Adjusted Rand Index (ARI) and Adjusted Mutual Information (AMI). With ARI values reaching up to 0.87, this assessment indicated that some Gaussian Mixture Model clustering were closely associated, further validating the idea of convergent structuring among probabilistic models. In contrast, low or zero ARI values between DBSCAN and other algorithms indicated use of different grouping principles by these methods. This finding aligns with the literature highlighting the complementary yet distinct results of density and centroid-based approaches (Ezugwu et al., 2022). Temporal drift analysis showed that the clusters had different structures during early and late session periods. This highlighted the need for time-aware evaluation to assess and maintain lasting user segments (Hu et al., 2024).
The baseline RFM proxy model focused only on recency, frequency, and weighted intent measures. It achieved a higher silhouette score of 0.35. This shows that even simple summaries of behaviour can capture important structural signals. However, its advantages in interpretability do not replace the richer underlying structure revealed by combining embedding and clustering. Embedding-based methods allow for segmentation that is sensitive to sequences and product co-occurrences, which goes beyond basic transaction summaries.

Further, a global accompanied by per-cluster SHAP analyses identified unique_items, recency_days, and events as the top contributors to segment assignments, All interactive results including EDA, clustering comparisons, SHAP plots, and persona tables, are accessible via the Streamlit dashboard at https://rocketretaildashboard.streamlit.app/ (see Appendix C for code).

5. Discussions

The overarching objective of this dissertation was to evaluate if embedding based session representations, when coupled with a suite of different unsupervised clustering algorithms and a multi evaluation framework, can yield customer segments that capture key underlying behavioural associations beyond what traditional Recency-Frequency-Monetary (RFM) proxies can derive. Drawing on the Retail Rocket clickstream dataset of 1.7 million sessions, this study's EDA, representation learning, clustering experiments, and interpretability analyses collectively affirm that embedding methods can uncover latent patterns of browsing, add-to-cart, and purchase intent that RFM aggregates alone cannot reveal.

5.1. Evaluating the RFM Baseline

RFM features, specifically session recency (days since last event), event frequency (total events), and an intent-weighted proxy provided a transparent and stakeholder friendly baseline model. When clustered via K-Means (k = 6) on standardized RFM inputs, this model achieved a silhouette score of 0.3500, indicating substantial cohesion among sessions with similar overalll statistics. The high silhouette reinforces work by Ho et al. (2023), who extended RFM to RFMD by adding demographic variables and still secured modest silhouette scores above 0.30. RFM clusters neatly aligned with intuitive personas (e.g., 'recent frequent purchasers' versus 'infrequent browsers') and explained 50–60 percent of variance in segment purity. Nonetheless, RFM's exclusion of sequence and co-occurrence information makes it blind to the orderliness of events, which is a key marker of user intent. A rapid 'view → transaction' chain denotes deliberate purchase intent, whereas 'view → view → view' may reflect indecision or comparison.

As Table 2 demonstrates, RFM's strengths in interpretability come at the expense of its inability to capture these sequential subtleties.

Representation	**Features**	**Silhouette**	**Captures Sequence?**	**Interpretability**
RFM Aggregates	Recency, Frequency, Intent Proxy	**0.3500**	No	**High**
Embedding + PCA	Word2Vec + Doc2Vec+PCA (96 comps)	0.0607	Yes	Moderate

Table 2. Comparison of RFM and Embedding Representations

5.2. Representation Learning: Word2Vec + Doc2Vec

To embed sequence dynamics, session events were tokenized and passed through two neural architectures. A Skip-gram Word2Vec (64 dims) captured local item co-occurrences, weighted by Smooth Inverse Frequency (Mikolov et al., 2013) to downplay commonly occuring 'view' tokens. Concurrently, a PV-DM Doc2Vec (96 dims) model encoded global session context (Le & Mikolov, 2014). Concatenation of these embeddings resulted in 160 dims, which was reduced via PCA to 96 components explaining about 80 percent of the original embedding variance. This hybrid feature space balanced local item associations (products browsed together) with global session intent (overall browsing trajectory). (Curiskis et al., 2020) similarly fused embeddings for improved clustering on noisy text data. The findings of this study thus, corroborates that combined embeddings outperform single-method representations while modelling latent behaviour.

5.3. Clustering Portfolio: Methods and Trade-Offs

A diversified algorithmic portfolio (Centroid-, Density- and hierarchy-based) was employed to respect different cluster shape assumptions:

- MiniBatch K-Means (k = 12): Efficient for spherical partitions, delivering silhouette = 0.0607, CH = 1549.9, DB = 4.03. Zero noise.

- Agglomerative Ward (n = 12): Hierarchical splits with kNN propagation yielded silhouette = 0.0635, CH = 1451.7, DB = 3.79. Zero noise.

- Gaussian Mixture (k = 18): Soft assignments produced silhouette = 0.0111, CH = 713.9, DB = 5.32. Zero noise.

- DBSCAN (ε = 14.91, min_samples = 10): Collapsed to one cluster with 27% noise, making it an unsuitable approach for segmentation in this context.

- HDBSCAN (min_cluster_size = 1500, min_samples = 20): Identified zero stable clusters under default settings.

Table 3 summarizes performance and hyperparameter highlights. Density based algorithms struggled due to the dataset's excessive skewness toward 'view' events, making ε thresholds and other density contrasts uninformative (Ester et al., 1996; Ezugwu et al., 2022). Centroid and hierarchical methods proved more robust, which aligns with Bayrak (2022), who found that word-embedding features benefit K-Means clustering in retail contexts.

Algorithm	**Key Params**	**Clusters**	**Noise %**	**Silhouette**	**CH Score**	**DB Score**
MiniBatch K-Means (k=12)	batch_size=4096, n_init=50	12	0	0.0607	1549.9	4.03
Agglomerative Ward (n=12)	connectivity: k=30	12	0	0.0635	1451.7	3.79
Gaussian Mixture (k=18)	covariance='diag', n_init=3	18	0	0.0111	713.9	5.32
DBSCAN (ε≈14.91)	min_samples=10	1	27	n/a	n/a	n/a
HDBSCAN	min_cluster_size=1500, min_samples=20	0	100	n/a	n/a	n/a

Table 3. Clustering Performance and Key Parameter

5.4. Internal and External Validation

Beyond silhouette, Calinski–Harabasz and Davies–Bouldin scores that inform on cluster compactness and separation. MiniBatch K-Means CH of 1549.9 and DB of 4.03 are superior to GMM's metrics, indicating tighter, more distinct centroid-based partitions. However, high CH does not necessarily imply business interpretability. Hence, cross-model agreement via Adjusted Rand Index (ARI) and Adjusted Mutual Information (AMI) was evaluated on a balanced 70 000-session sample.

	K-Means	**GMM**	**Ward**	**HDBSCAN**	**DBSCAN**
K-Means	1.00	0.84	0.76	0.02	0.01
GMM	0.84	1.00	0.70	0.03	0.02
Ward	0.76	0.70	1.00	0.01	0.01
HDBSCAN	0.02	0.03	0.01	1.00	0.00
DBSCAN	0.01	0.02	0.01	0.00	1.00

Table 4. Pairwise ARI Matrix

High ARI values between centroids (≥ 0.70) indicates similar partitions drawn by models in PCA-space, whereas near zero ARI with density methods highlights divergent grouping principles (Ezugwu et al., 2022). These multi-perspective validations guard against undue reliance on any one algorithm's biases.

5.5. Sequence Pattern Diagnostics

Embedding models have the ability to internalize event transition regularities. The top 3-grams 'view → add-to-cart → transaction' and 'view → view → add-to-cart' capture critical funnels of intent.

Figure 4 displays the top twenty 3-grams by frequency, demonstrating how embedding vectors encode sequential context better than RFM aggregates.

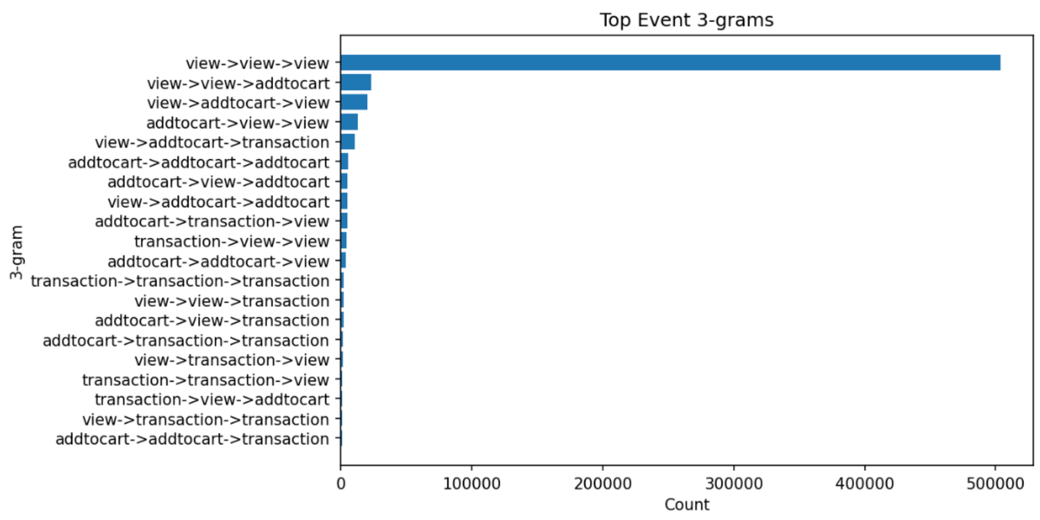

Figure 4. Top Event 3-grams in Sessions (count)

5.6. Temporal Stability and Concept Drift

Splitting data into early versus late halves revealed shifts in cluster sizes and centroids, particularly for segments characterized by high add-to-cart rates, likely during promotional weeks. ARI between early- and late-period cluster assignments averaged 0.65 for centroid methods, indicating a moderate stability. This reinforces Hu et al. (2024) emphasis on the necessity of drift checks, as segments optimized on a static snapshot may decay over time. Future work should incorporate online drift detectors (Suárez-Cetrulo et al., 2023) to maintain segmentation relevance.

5.7. Feature Attribution and Persona Construction

To translate clusters into actionable insights, SHAP values from a Random Forest surrogate were computed on 30 000 session samples. Global mean SHAP highlighted unique_items, R_recency_days, and recency_days as the top features (Table 5), further reinforcing the continued relevance of RFM elements alongside embedding derived signals. Per-cluster SHAP rankings (see Appendix B output) allocate personas such as:

- Browsers & Comparers: High unique_items, moderate recency, but low transaction probability.

- Cart Hesitators: Elevated events_per_min and atc_per_view, but low final transaction rates.
- Quick Buyers: Low recency, few views, yet direct 'view→transaction' paths.
- Explorers**:** High duration_sec, diverse item views, low bounce.

These personas highly align with (Moe, 2003) specifically on intent trajectories and further enables marketing teams to tailor messages (e.g., incentivizing 'cart hesitators' with reminders or promotions).

Feature	Mean |SHAP |
unique_items	0.0122
R_recency_days	0.0033
recency_days	0.0032
events	0.0026
F_events_log1p	0.0023
is_bounce	0.0023
events_per_min	0.0023
start_hour	0.0020

Table 5. Global SHAP Feature Importances

5.8. Addressing Research Questions

i. RFM vs. Embedding Coherence: RFM achieved higher silhouette but does not capture sequence sensitivity. Embeddings produced lower silhouette yet captured event order and transitions (change in intent), offering richer behavioural insights.
ii. Best Clustering Approach: Under default parameters, Centroid (MiniBatch K-Means) and hierarchical (Ward) methods provided much stable partitions with zero noise and consistent cross-model agreement, performing better than density based clustering approaches.
iii. Cross-Model Consistency: An ARI of more than 0.70 among K-Means, GMM, and Ward confirms they perform similar structural groupings.

However, near zero ARI with density methods highlights opposite segmentation perspectives.

iv. Segment Temporal Stability: An ARI of 0.65 (approx.) indicates moderate drift between early and late sessions, suggesting volatility of clusters to temporal changes.

v. Behavioural Characterization: SHAP profiled personas indicate item diversity, recency, and event intensity as major drivers of cluster separations, enabling targeted CRM strategies across distinct user journeys.

5.9. Limitations and Future Research

Key limitations that this study encounters are as below:

- Absence of Monetary Data: Lack of price information constrains lead to adoption of its proxy (event counts). Incorporation of transactional values could have refined RFM segments and also enabled ROI estimates (Ho et al., 2023).
- Instability Due to Short Sessions: Very short sessions (<= 2 events) may result in unreliable embeddings. Future work can evaluate more powerful and sequence aware transformer models or session level attention mechanisms.
- Parameter Sensitivity: Density based models insignificant performance under default settings highlight the need for systematic hyperparameter optimization.
- External Campaign Effects: Absence of metadata like promotional offer, markdowns, etc., makes it difficult to map the circumstances leading to temporal drifts. Incorporation of same can enhance both, model's explainability and efficient decision-making.

5.10. Stakeholder Implications

This study advises stakeholders seeking actionable segmentation to:

- Conduct EDA to understand event imbalances and accordingly make inform representation/ embedding choices.
- Leverage embeddings to capture sequence of co-occurrences, especially in case of short and noisy sessions.
- Adopt a clustering portfolio, mainly prioritizing centroid and hierarchical methods for initial segment definitions, and evaluation of same using relevant internal metrics and cross-model agreement.
- Implement temporal drift checks to ensure segment stability over time and ensure that segmentation drives business value.
- Use SHAP or similar tools to attribute clusters into business friendly personas (while ensuring fair trade-off between interpretability and behavioural richness sustains.

By weaving traditional segmentation approaches such as RFM with embedding, organizations can craft segmentation strategies that both resonate with commercial stakeholders and harness the full explanatory power of clickstream data.

6. Conclusions And Recommendations

The study has demonstrated that embedding based session representations (combining Word2Vec and Doc2Vec, denoised using PCA) is efficient at capture latent behavioural trajectories that traditional RFM proxies often cannot fully represent. Centroid and hierarchical clustering methods reliably partitioned the sessions in this new feature space (Figure 5 and 6 depicts 2D UMAP visualisation of same), while density based algorithms proved extremely sensitive to class imbalance and hyper-parameter settings. Multi-perspective evaluation (silhouette, Calinski–Harabasz, Davies–Bouldin) along with cross-model agreement (ARI/AMI) and temporal stability checks provided safeguarded against algorithmic biases and temporary seasonal effects. SHAP based surrogate model enabled the complex task of translating derived clusters into business friendly personas, bridging the gap between statistical patterns and managerial actions.

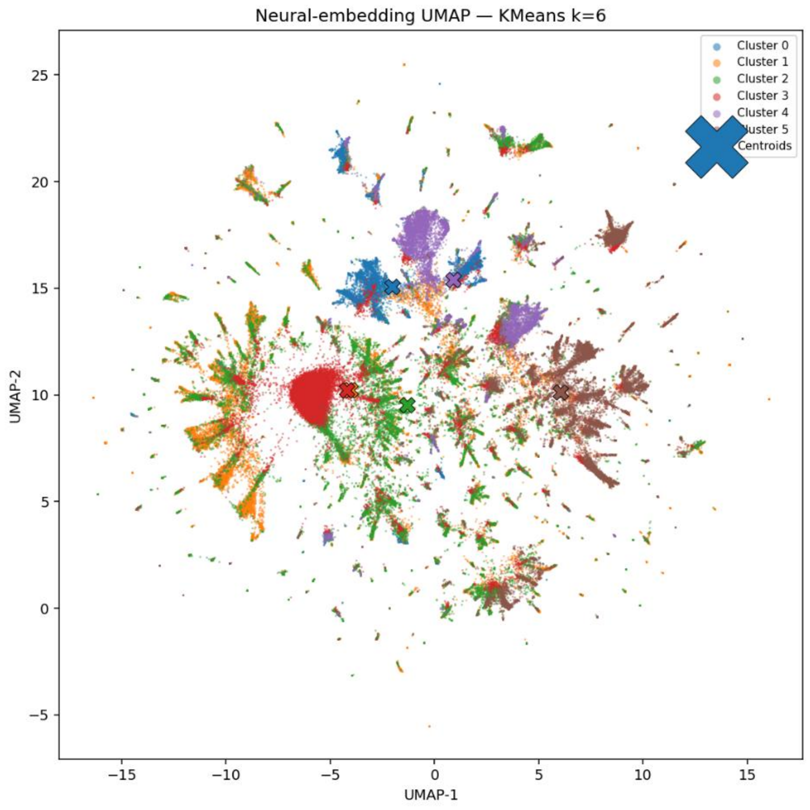

Figure 5. UMAP projection of session embeddings (2-D) illustrating K-means clusters (for k=6) in Rocket Retail clickstream behaviour.

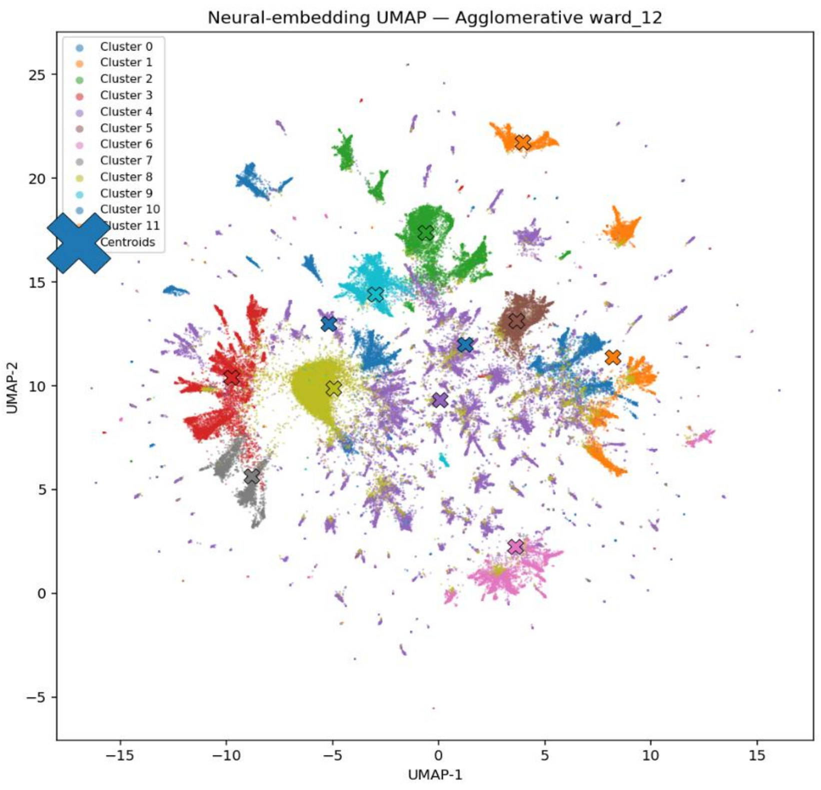

Figure 6. UMAP projection of session embeddings (2-D) illustrating Agglomerative ward linkage clusters (Hierarchical clustering) in Rocket Retail clickstream behaviour.

Three principal conclusions emerge from this study:

- Embedding-driven segmentation can uncover nuanced intent paths even in sparse settings: The hybrid representation captured high frequency 'view → add-to-cart → transaction' patterns along with other visit trajectories. These sequential signals complement RFM structure which enables identification of personas (such as 'Cart Hesitators' or 'Quick Buyers') beyond just frequency or recency bands (Le & Mikolov, 2014; Mikolov et al., 2013)

- Implementations of wide clustering portfolios and stability checks are critical. Neither a single clustering method nor a lone evaluation metric suffices. Centroid (K-Means) and hierarchical (Ward) algorithms yielded consistent partitions with silhouette scores around 0.06 in embedding space and also shared high ARI among them (Ezugwu et al., 2022).

While Density based techniques (DBSCAN and HDBSCAN) struggled under default settings, emphasizing the necessity of portfolio designs and efficient parameter sensitivity analyses.

- Interpretability through SHAP and personas ensures trust. Feature attributions using the same outputted unique_items, recency_days, and events_per_min as key drivers, by aligning embedding outputs with familiar proxies (Verdinelli & Wasserman, 2023; Zhao et al., 2024). Persona tables were used to translate abstract clusters into deterministic customer profiles that could drive marketing, UX, and merchandising efforts, addressing the communicative challenges of segmentation (Moe, 2003).

Based on these findings, the following recommendations are offered for both practitioners and future research:

- Deployment of hybrid embeddings for behavioural segmentation. Organizations should implement Word2Vec to model item level co-views concatenated with Doc2Vec for integrating session level context and PCA to remove noise by reducing dimension of these embeddings. This enables richer segment definitions than RFM alone (Curiskis et al., 2020).
- Adopting a multi-algorithm clustering framework. This study also advises to accompany this clustering with evaluation using relevant internal metrics and cross-model agreement to ensure valuable partitions as output (Ezugwu et al., 2022).
- Integration of temporal stability checks into operational pipelines. Segmentation models should be recalibrated periodically to detect drift caused by promotions or seasonality. Employ rolling-window concept drift detectors (Suárez-Cetrulo et al., 2023) to ensure segment relevance over time.
- Leveraging SHAP or other analogous attribution tools for persona extraction but explanations should be treated as associative rather than causal. Feature correlations may bias Shapley values, thus it is important to document attribution assumptionsmto preserve stakeholder trust (Verdinelli & Wasserman, 2023; Zhao et al., 2024).

- Combining segmentation with interactive dashboards. Visual interfaces that display UMAP projections, cluster comparisons, and persona tables facilitates non-technical stakeholders to explore and validate segments, this ensures both technical consistency and communicative clarity (Sarlin & Yao, 2013; Yao, 2013).
- Extend the pipeline with monetary and contextual data where available as incorporating transaction values, promotions, and product metadata would potentially enrich segments with profitability and campaign effectiveness dimensions.
- This study encourages exploration of advanced sequence models for sparse sessions.
- Prioritization of scalability and runtime profiling. At production scale, heavy models can be trained on representative samples and then labels can be propagated at full scale via k-NN, and monitor compute costs, parameter sensitivities and maintain reproducible configurations for auditability (Ezugwu et al., 2022)

In conclusion, embedding-based segmentation supplemented by clustering portfolios, multi-lens evaluation, and interpretability mechanisms demonstrates a solid and credible framework to uncover and leverage latent behavioural patterns in clickstream data. By adhering to above recommendations, organizations can innovate segmentation strategies that are not only technically sound but also transparent, stable, and aligned with business objectives.

7. References

Bayrak, A. T. (2022). An application of Customer Embedding for Clustering. *IEEE International Conference on Data Mining Workshops, ICDMW, 2022-November*, 79–82. https://doi.org/10.1109/ICDMW58026.2022.00019

Binns, R. (2018). Fairness in Machine Learning: Lessons from Political Philosophy. In S. A. Friedler & C. Wilson (Eds.), *Proceedings of the 1st Conference on Fairness, Accountability and Transparency* (Vol. 81, pp. 149–159). PMLR. https://proceedings.mlr.press/v81/binns18a.html

Chen, X., & Güttel, S. (2024). *Fast and explainable clustering based on sorting*. http://arxiv.org/abs/2202.01456

Curiskis, S. A., Drake, B., Osborn, T. R., & Kennedy, P. J. (2020). An evaluation of document clustering and topic modelling in two online social networks: Twitter and Reddit. *Information Processing and Management*, *57*(2). https://doi.org/10.1016/j.ipm.2019.04.002

Ester, M., Kriegel, H.-P., Sander, J., & Xu, X. (1996). *A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise*. www.aaai.org

Ezugwu, A. E., Ikotun, A. M., Oyelade, O. O., Abualigah, L., Agushaka, J. O., Eke, C. I., & Akinyelu, A. A. (2022). A comprehensive survey of clustering algorithms: State-of-the-art machine learning applications, taxonomy, challenges, and future research prospects. *Engineering Applications of Artificial Intelligence*, *110*, 104743. https://doi.org/https://doi.org/10.1016/j.engappai.2022.104743

Ho, T., Nguyen, S., Nguyen, H., Nguyen, N., Man, D. S., & Le, T. G. (2023). An Extended RFM Model for Customer Behaviour and Demographic Analysis in Retail Industry. *Business Systems Research*, *14*(1), 26–53. https://doi.org/10.2478/bsrj-2023-0002

Hu, L., Jiang, M., Dong, J., Liu, X., & He, Z. (2024). *Interpretable Clustering: A Survey*. http://arxiv.org/abs/2409.00743

Jain, A. K. (2010). Data clustering: 50 years beyond K-means. *Pattern Recognition Letters*, *31*(8), 651–666. https://doi.org/10.1016/j.patrec.2009.09.011

Le, Q. V., & Mikolov, T. (2014). *Distributed Representations of Sentences and Documents*. http://arxiv.org/abs/1405.4053

Masalma, M. Al. (2024). Clustering E-Commerce Consumers through Machine Learning-Based Analysis of Clickstream Data. *International Research Journal of Economics and Management Studies*, *3*(4). https://doi.org/10.56472/25835238/IRJEMS-V3I4P112

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). *Efficient Estimation of Word Representations in Vector Space*. http://arxiv.org/abs/1301.3781

Moe, W. W. (2003). Buying, Searching, or Browsing: Differentiating Between Online Shoppers Using In-Store Navigational Clickstream. *Journal Of Consumer Psychology*, *13*, 29–39.

Montgomery, A. L., Li, S., Srinivasan, K., & Liechty, J. C. (2004). Modeling online browsing and path analysis using clickstream data. *Marketing Science*, *23*(4). https://doi.org/10.1287/mksc.1040.0073

Reynolds, D. (2008). Gaussian Mixture Models. *Encyclopedia of Biometrics*. https://doi.org/10.1007/978-0-387-73003-5_196

Ridwan, I. B. (2025). Transforming Customer Segmentation with Unsupervised Learning Models and Behavioral Data in Digital Commerce. *International Journal of Research Publication and Reviews*, *6*(5), 2232–2249. https://doi.org/10.55248/gengpi.6.0525.1652

Rousseeuw, P. J. (1987). Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. In *Journal of Computational and Applied Mathematics* (Vol. 20).

Sarlin, P., & Yao, Z. (2013). Clustering of the Self-Organizing Time Map. *Neurocomputing*, *121*, 317–327. https://doi.org/https://doi.org/10.1016/j.neucom.2013.04.007

Skotis, A., & Livas, C. (2024). Prominent User Segments in Online Consumer Recommendation Communities: Capturing Behavioral and Linguistic Qualities with User Comment Embeddings. *Information (Switzerland)*, *15*(6). https://doi.org/10.3390/info15060356

Stormi, K., Lindholm, A., Laine, T., & Korhonen, T. (2020). RFM customer analysis for product-oriented services and service business development: an interventionist case study of two machinery manufacturers. *Journal of Management and Governance*, *24*(3), 623–653. https://doi.org/10.1007/s10997-018-9447-3

Suárez-Cetrulo, A. L., Quintana, D., & Cervantes, A. (2023). A survey on machine learning for recurring concept drifting data streams. In *Expert Systems with Applications* (Vol. 213). Elsevier Ltd. https://doi.org/10.1016/j.eswa.2022.118934

Susanto, A. D., Andrian Pradita, S., Stryadhi, C., Setiawan, K. E., & Fikri Hasani, M. (2023). Text Vectorization Techniques for Trending Topic Clustering on Twitter: A Comparative Evaluation of TF-IDF, Doc2Vec, and Sentence-BERT. *2023 5th International Conference on Cybernetics and Intelligent Systems, ICORIS 2023*. https://doi.org/10.1109/ICORIS60118.2023.10352228

Tsiptsis, K., & Chorianopoulos, A. (2009). *Data Mining Techniques in CRM: Inside Customer Segmentation*. https://opac.feb.uinjkt.ac.id/repository/43970e41797ed3ee0b637ab1523d679f.pdf

Verdinelli, I., & Wasserman, L. (2023). *Feature Importance: A Closer Look at Shapley Values and LOCO*. http://arxiv.org/abs/2303.05981

Wang, S., Sun, L., & Yu, Y. (2024). A dynamic customer segmentation approach by combining LRFMS and multivariate time series clustering. *Scientific Reports*, *14*(1). https://doi.org/10.1038/s41598-024-68621-2

Yao, Z. (2013). *Visual Customer Segmentation and Behavior Analysis A SOM-Based Approach at noon* (Vol. 18).

Zhan, Z. (2025). Comparative Analysis of TF-IDF and Word2Vec in Sentiment Analysis: A Case of Food Reviews. *ITM Web of Conferences*, *70*, 02013. https://doi.org/10.1051/itmconf/20257002013

Zhao, N., Yu, J. Y., Dzieciolowski, K., & Bui, T. (2024). *Error Analysis of Shapley Value-Based Model Explanations: An Informative Perspective*. https://doi.org/10.1007/978-3-031-65112-0_2

8. APPENDIX A

8.1. EDA Pipeline Code:

```
#!/usr/bin/env python
# coding: utf-8

# # EDA Pipeline for RetailRocket Clickstream

# ## 1) Imports & configurations

# 1.1) Imports & environment

import os, json, math, time, random
import numpy as np
import pandas as pd
from pathlib import Path
from collections import Counter, defaultdict, deque
import seaborn as sns
import matplotlib
import matplotlib.pyplot as plt

# Reproducibility
random.seed(RANDOM_SEED)
np.random.seed(RANDOM_SEED)


# In[13]:


# 1.2) CONFIGURATION — kindly adjust as needed (all parameters centralised
here)
from pathlib import Path

#  Paths
```

```
DATA_DIR           =
Path("C:/Users/Admin/Documents/WBS/Dissertation/Submission Related
files/data")  # folder that holds the CSVs
EVENTS_CSV         = DATA_DIR / "events.csv"      # expected columns:
session_id*, visitorid*, itemid, timestamp, event
ITEMS1_CSV         = DATA_DIR / "item_properties_part1.csv"
ITEMS2_CSV         = DATA_DIR / "item_properties_part2.csv"
# Core switches
HAS_SESSION_ID     = False     # if False, sessionize via visitorid and
SESSION_GAP_SEC
SESSION_GAP_SEC    = 30*60    # 30 minutes inactivity defines a new session
when HAS_SESSION_ID=False
TS_DIV             = 1000       # keep original UNIX timestamp unit (1 if already
seconds; 1000 if ms). We retain raw UNIX as per dissertation rationale.
EVENT_TYPES        = ["view", "addtocart", "transaction"]   # expected events
(others will be grouped under "other")

# Sampling & performance
USE_STRATIFIED_SAMPLING = False
SAMPLE_MAX_SESSIONS     = 200_000  # cap for heavy operations (hour/day
patterns, n-grams)
RANDOM_SEED             = 42

# Plots & outputs
OUT_DIR            =
Path("C:/Users/Admin/Documents/WBS/Dissertation/Submission Related
files/Notebooks/Modelling/EDA_outputs"); OUT_DIR.mkdir(parents=True,
exist_ok=True)
FIG_DIR            = OUT_DIR / "figs"; FIG_DIR.mkdir(parents=True,
exist_ok=True)
TAB_DIR            = OUT_DIR / "tables"; TAB_DIR.mkdir(parents=True,
exist_ok=True)
REPORTS_DIR        = OUT_DIR / "reports";
REPORTS_DIR.mkdir(parents=True, exist_ok=True)

SAVE_DPI           = 150
```

```
FIG_EXT            = "png"          # png recommended for dashboards; use "svg"
for vector
STYLE_USE_SEABORN   = False          # keep False for portability; can switch
to True if seaborn is installed
PANDAS_FLOAT_FMT    = "{:,.4f}".format

# Feature thresholds
TOP_N              = 20             # for top products/categories/3-grams
OUTLIER_EPM_THRESH  = 120            # events per minute threshold indicating
possible automation/ bot or web scrapping
OUTLIER_DUR_PCTL    = 99.5           # sessions above this duration percentile
are flagged

# Temporal granularity
HOUR_OF_DAY        = list(range(24))
DAYS_MAP           = {0:"Mon",1:"Tue",2:"Wed",3:"Thu",4:"Fri",5:"Sat",6:"Sun"}

# Exports
EXPORT_SESSION_FEATURES = True
SESSION_FEATURES_CSV    = OUT_DIR / "eda" / "session_features_base.csv"

# Behavioural funnels
FUNNEL_ORDER       = ["view", "addtocart", "transaction"]


# ## 2) Defining utilities to support pipeline

# In[15]:



# 2) Utilities — tic/toc, saving, safe plotting, sampling

from contextlib import contextmanager
from datetime import datetime

_TIC_STACK = []
```

```
def tic(msg=None):
    """Start a timer; optionally print a message."""
    if msg:
        print(f"[tic] {msg}")
    _TIC_STACK.append(time.time())

def toc(msg=None):
    """End the most recent timer and print elapsed time."""
    if not _TIC_STACK:
        print("[toc] Warning: called without matching tic()")
        return 0.0
    elapsed = time.time() - _TIC_STACK.pop()
    if msg:
        print(f"[toc] {msg}: {elapsed:,.2f}s")
    else:
        print(f"[toc] {elapsed:,.2f}s")
    return elapsed

def ensure_dir(path: Path):
    path = Path(path)
    path.parent.mkdir(parents=True, exist_ok=True)

def save_table(df: pd.DataFrame, path: Path):
    ensure_dir(path)
    df.to_csv(path, index=False)
    print(f"[saved] table -> {path} ({len(df):,} rows)")

def save_fig(fig, name: str):
    filename = FIG_DIR / f"{name}.{FIG_EXT}"
    ensure_dir(filename)
    fig.savefig(filename, dpi=SAVE_DPI, bbox_inches="tight")
    plt.close(fig)
    print(f"[saved] figure -> {filename}")

def maybe_sample_sessions(df: pd.DataFrame, session_col="session_id"):
    """Return df sampled by sessions if enabled and needed."""
    if not USE_STRATIFIED_SAMPLING:
        return df
    if "session_id" not in df.columns:
        return df
    sess_ids = df["session_id"].unique()
    if len(sess_ids) <= SAMPLE_MAX_SESSIONS:
        return df
    rng = np.random.default_rng(RANDOM_SEED)
    keep = rng.choice(sess_ids, size=SAMPLE_MAX_SESSIONS, replace=False)
    out = df[df["session_id"].isin(keep)].copy()
    print(f"[sample] sessions: {len(sess_ids):,} -> {len(keep):,}")
    return out


# ## 3) Data Loading

# 3) Data Loading — schema handling, timestamp normalization

REQUIRED_COLS_ANY = {"itemid", "timestamp", "event"}
OPTIONAL_COLS     = {"session_id", "visitorid"}

def read_events(path: Path) -> pd.DataFrame:
    tic(f"Reading events from {path}")
    df = pd.read_csv(path, usecols=None)
    # column name normalisation (lowercase)
    df.columns = [c.strip().lower() for c in df.columns]

    # Validating plausible column names
    colmap = {}
    # Mapping timestamp variants
    for cand in ["timestamp", "ts", "time", "eventtime", "event_time"]:
        if cand in df.columns:
            colmap["timestamp"] = cand; break
    # Mapping event variants
    for cand in ["event", "action", "type"]:
        if cand in df.columns:
            colmap["event"] = cand; break
    # Mapping itemid variants
    for cand in ["itemid", "item_id", "sku", "productid", "product_id"]:
        if cand in df.columns:
            colmap["itemid"] = cand; break
    # Mapping session/visitor variants
    for cand in ["session_id", "sessionid", "sid"]:
        if cand in df.columns:
            colmap["session_id"] = cand; break
    for cand in ["visitorid", "user_id", "userid", "uid"]:
        if cand in df.columns:
            colmap["visitorid"] = cand; break

    missing = [k for k in ["timestamp","event","itemid"] if k not in colmap]
    if missing:
        raise KeyError(f"Missing required columns: {missing}. Found columns =
{list(df.columns)}")

    # Renaming in place to canonical names
    df = df.rename(columns={v:k for k,v in colmap.items()})
    # Retaining only the canonical columns
    keep_cols = ["timestamp","event","itemid"] + [c for c in ["session_id","visitorid"]
if c in df.columns]
    df = df[keep_cols].copy()

    # Timestamp normalization, we retain UNIX epohs
    if TS_DIV != 1:
        df["timestamp"] = (df["timestamp"] // TS_DIV).astype("int64")
    else:
        df["timestamp"] = df["timestamp"].astype("int64")

    # Event canonicalisation
    df["event"] = df["event"].astype("string").str.lower().str.strip()
    df.loc[~df["event"].isin(EVENT_TYPES), "event"] = "other"

    # Adding datetime
    df["dt"] = pd.to_datetime(df["timestamp"], unit="s")

    toc("Loaded events")
    return df

# pulling events data
events = read_events(EVENTS_CSV)

def maybe_read_items(path):
    if not path:
        return None
    tic(f"Reading items metadata from {path}")
    df = pd.read_csv(path)
    # normalise column names
    df.columns = [c.strip().lower() for c in df.columns]
    # standardise a category column if present under common aliases
    if "category" not in df.columns:
        for cand in ("category_id", "cat", "categorycode", "category_code"):
            if cand in df.columns:
                df = df.rename(columns={cand: "category"})
                break

    toc("Reading items metadata")
    return df

# pulling items files
items1 = maybe_read_items(ITEMS1_CSV)   # keep your variable names as-is
items2 = maybe_read_items(ITEMS2_CSV)

# Combine items table
items = None
for _df in (items1, items2):
    if _df is None:
        continue
    items = _df if items is None else pd.concat([items, _df], ignore_index=True)

if items is not None:
    # optional: drop duplicate item ids
    # if key: items = items.drop_duplicates(subset=[key])
    pass
```

```
else:
    # No items metadata available
    items = None




# Sessionisation if needed
if not HAS_SESSION_ID:
    tic("Sessionising by visitorid + inactivity gap")
    events = events.sort_values(["visitorid","timestamp"]).reset_index(drop=True)
    # New session when gap > SESSION_GAP_SEC or visitor changes
    gap = events["timestamp"].diff().fillna(0)
    new_user = events["visitorid"].ne(events["visitorid"].shift(1))
    new_sess = (gap > SESSION_GAP_SEC) | (new_user)
    events["session_id"] = new_sess.cumsum().astype("int64")
    toc("Sessionised")

if "session_id" not in events.columns:
    raise KeyError("session_id is required (either provided or created via
sessionisation).")

# Basic dataset facts
tic("Computing dataset overview")
n_rows = len(events)
n_sessions = events["session_id"].nunique()
n_users = events["visitorid"].nunique() if "visitorid" in events.columns else np.nan
n_items = events["itemid"].nunique()
t0, t1 = events["dt"].min(), events["dt"].max()
print(f"Rows={n_rows:,} | Sessions={n_sessions:,} | Users={n_users} |
Items={n_items:,}")
print(f"Date range: {t0} -> {t1}")

overview = pd.DataFrame([{
    "rows": n_rows,
    "sessions": n_sessions,
    "users": n_users,
    "items": n_items,
    "date_start": t0,
    "date_end": t1
}])
save_table(overview, TAB_DIR / "dataset_overview.csv")
toc("Overview ready")


# ## 4) Event distributions & proportions


# 4) Event distributions & proportions

tic("Event distribution")
evt_counts =
events["event"].value_counts().rename_axis("event").reset_index(name="count")
evt_counts["prop"] = evt_counts["count"] / evt_counts["count"].sum()
save_table(evt_counts, TAB_DIR / "event_distribution.csv")

fig, ax = plt.subplots(figsize=(6,4))
ax.bar(evt_counts["event"], evt_counts["count"])
ax.set_xlabel("Event")
ax.set_ylabel("Count")
ax.set_title("Event Distribution (All)")
for i, v in enumerate(evt_counts["count"]):
    ax.text(i, v, f"{int(v):,}", ha="center", va="bottom", fontsize=8, rotation=0)
save_fig(fig, "eda_event_distribution_all")
toc("Event distribution ready")


# ## 5) Temporal patterns

# 5) Temporal patterns (hour-of-day, day-of-week)

df_time = maybe_sample_sessions(events)

# Hour-of-day temporal pattern
tic("Hour-of-day pattern")
```

```
hod = df_time.copy()
hod["hour"] = hod["dt"].dt.hour
hod_counts = hod.groupby(["hour","event"]).size().reset_index(name="count")
save_table(hod_counts, TAB_DIR / "temporal_hour_event_counts.csv")

fig, ax = plt.subplots(figsize=(7,4))
for e in EVENT_TYPES + (["other"] if "other" in hod["event"].unique() else []):
    sub = hod_counts[hod_counts["event"]==e]
    if not len(sub):
        continue
    ax.plot(sub["hour"], sub["count"], marker="o", label=e)
ax.set_xticks(HOUR_OF_DAY)
ax.set_xlabel("Hour of Day")
ax.set_ylabel("Events")
ax.set_title("Hourly Event Volume")
ax.legend()
save_fig(fig, "eda_hourly_event_volume")
toc("Hour-of-day ready")

# Day of week temporal pattern
tic("Day-of-week pattern")
dow = df_time.copy()
dow["dow"] = dow["dt"].dt.dayofweek
dow_counts = dow.groupby(["dow","event"]).size().reset_index(name="count")
dow_counts["dow_label"] = dow_counts["dow"].map(DAYS_MAP)
save_table(dow_counts, TAB_DIR / "temporal_dow_event_counts.csv")

fig, ax = plt.subplots(figsize=(7,4))
for e in EVENT_TYPES + (["other"] if "other" in dow["event"].unique() else []):
    sub = dow_counts[dow_counts["event"]==e]
    if not len(sub):
        continue
    ax.plot(sub["dow_label"], sub["count"], marker="o", label=e)
ax.set_xlabel("Day of Week")
ax.set_ylabel("Events")
ax.set_title("Weekly Event Volume")
ax.legend()
```

```
save_fig(fig, "eda_weekly_event_volume")
toc("Day-of-week ready")


# ## 6) Session-level features

# 6) Session-level features (duration, intensity, diversity, last-event flags)

tic("Computing session-level features")
g = events.groupby("session_id", sort=False)

sess_first = g["timestamp"].min()
sess_last  = g["timestamp"].max()
duration_s = (sess_last - sess_first).rename("duration_sec")
n_events   = g.size().rename("events")
items_per_sess = g["itemid"].nunique().rename("item_diversity")

# events per minute (avoid /0 by by adding a small negligable value, 1e-9 to
denominator)
events_per_min = (n_events / (duration_s/60.0 + 1e-
9)).rename("events_per_min")

# last event type flags
last_event = g["event"].last().rename("last_event")
last_evt_flags = pd.get_dummies(last_event, prefix="last_evt", dtype=int)

sess_df = pd.concat([duration_s, n_events, items_per_sess, events_per_min,
last_event, last_evt_flags], axis=1).reset_index()

# Saving raw features
FEATURES_DIR = OUT_DIR / "eda"; FEATURES_DIR.mkdir(parents=True,
exist_ok=True)
save_table(sess_df, FEATURES_DIR / "session_features_base.csv")

# Distributions (log-scale a they are helpful for skew)
def _hist(series, title, xlabel, name):
    fig, ax = plt.subplots(figsize=(6,4))
    ax.hist(series, bins=50)
    ax.set_title(title)
    ax.set_xlabel(xlabel); ax.set_ylabel("Frequency")
    save_fig(fig, name)

_hist(np.log1p(sess_df["duration_sec"]), "Session Duration (log1p seconds)",
"log1p(seconds)", "eda_hist_duration_log1p")
_hist(np.log1p(sess_df["events"]), "Session Events (log1p)", "log1p(events)",
"eda_hist_events_log1p")
_hist(np.log1p(sess_df["events_per_min"]), "Events per Minute (log1p)",
"log1p(epm)", "eda_hist_epm_log1p")
_hist(np.log1p(sess_df["item_diversity"]), "Item Diversity (log1p unique items)",
"log1p(unique items)", "eda_hist_itemdiv_log1p")

# Percentiles summary
pctls =
sess_df[["duration_sec","events","events_per_min","item_diversity"]].quantile([0.5
,0.9,0.95,0.99]).reset_index(names=["quantile"])
save_table(pctls, TAB_DIR / "session_feature_percentiles.csv")
toc("Session-level features ready")


# ## 7) Behavioural funnel

# 7) Behavioural funnel (session-level presence of events in order)

tic("Computing funnel")
# Presence flags per session
presence = events.pivot_table(index="session_id", columns="event",
values="timestamp", aggfunc="count", fill_value=0)
for e in FUNNEL_ORDER:
    if e not in presence.columns:
        presence[e] = 0
presence = presence[[e for e in FUNNEL_ORDER if e in
presence.columns]].astype(int)
```

```
# Simple sequential funnel to identify proportion of sessions that reach each
stage
total_sessions = presence.shape[0]
stage_props = []
running = total_sessions
mask = pd.Series([True]*total_sessions, index=presence.index)
for e in FUNNEL_ORDER:
    step = presence.loc[mask, e] > 0
    reached = int(step.sum())
    stage_props.append({"stage": e, "sessions_reached": reached, "prop_of_total":
reached/total_sessions})
    # tighten mask for next stage: among those that reached current step
    mask = mask & (presence[e] > 0)

funnel_df = pd.DataFrame(stage_props)
save_table(funnel_df, TAB_DIR / "funnel_session_level.csv")

fig, ax = plt.subplots(figsize=(6,4))
ax.bar(funnel_df["stage"], funnel_df["sessions_reached"])
ax.set_title("Session-level Funnel")
ax.set_xlabel("Stage"); ax.set_ylabel("Sessions Reached")
for i, v in enumerate(funnel_df["sessions_reached"]):
    ax.text(i, v, f"{v:,}", ha="center", va="bottom", fontsize=8)
save_fig(fig, "eda_funnel_session_level")
toc("Funnel ready")


# # 8) Sequence patterns

# 8) Sequence patterns — event 3-grams and Markov transitions

# Prepare
seq_df =
maybe_sample_sessions(events).sort_values(["session_id","timestamp"])

# 3-grams
tic("3-gram extraction")
```

```
def event_ngrams(seq, n=3):
    if len(seq) < n:
        return []
    return [tuple(seq[i:i+n]) for i in range(len(seq)-n+1)]

ngrams_counter = Counter()
for _, g in seq_df.groupby("session_id", sort=False):
    grams = event_ngrams(g["event"].tolist(), n=3)
    ngrams_counter.update(grams)

grams_df = pd.DataFrame(ngrams_counter.items(),
columns=["ngram","count"]).sort_values("count", ascending=False)
grams_df["ngram_str"] = grams_df["ngram"].apply(lambda t: "->".join(t))
top_grams = grams_df.head(TOP_N).copy()
save_table(top_grams[["ngram_str","count"]], TAB_DIR /
"top_event_3grams.csv")

fig, ax = plt.subplots(figsize=(8,5))
ax.barh(top_grams["ngram_str"][::-1], top_grams["count"][::-1])
ax.set_xlabel("Count"); ax.set_ylabel("3-gram"); ax.set_title("Top Event 3-grams")
save_fig(fig, "eda_top_3grams")
toc("3-grams ready")

# First-order Markov transition matrix among core events
tic("Transition matrix")
core = EVENT_TYPES  # ignore "other" for clarity
idx = {e:i for i,e in enumerate(core)}
m = np.zeros((len(core), len(core)), dtype=np.float64)

prev = None
prev_sess = None
for _, row in seq_df.iterrows():
    e = row["event"]
    s = row["session_id"]
    if e not in core:
        prev = None; prev_sess = s
        continue
    if prev is not None and prev_sess == s:
        m[idx[prev], idx[e]] += 1.0
    prev = e; prev_sess = s

# Row-normalise
row_sums = m.sum(axis=1, keepdims=True) + 1e-12
tm = m / row_sums

tm_df = pd.DataFrame(tm, index=core, columns=core)
save_table(tm_df.reset_index().rename(columns={"index":"from"}), TAB_DIR /
"transition_matrix_core.csv")

fig, ax = plt.subplots(figsize=(5.5,4.5))
im = ax.imshow(tm, aspect="auto")
ax.set_xticks(range(len(core))); ax.set_xticklabels(core, rotation=45, ha="right")
ax.set_yticks(range(len(core))); ax.set_yticklabels(core)
ax.set_title("Markov Transition Probabilities (core events)")
fig.colorbar(im, ax=ax, fraction=0.046, pad=0.04)
save_fig(fig, "eda_transition_matrix_core")
toc("Transition matrix ready")


# ## 9) Outlier/bot heuristics and data quality checks

# 9) Outlier/bot heuristics and data quality checks

tic("Outlier heuristics & data quality")

# Outliers by events per minute and long duration
q = sess_df["duration_sec"].quantile(OUTLIER_DUR_PCTL/100.0)
outliers = sess_df[(sess_df["events_per_min"] > OUTLIER_EPM_THRESH) |
(sess_df["duration_sec"] > q)].copy()
save_table(outliers, TAB_DIR / "potential_automation_or_outliers.csv")

# Duplicates & unknowns (data qulity checks)
dupe_rows =
events.duplicated(subset=["session_id","timestamp","event","itemid"]).sum()
```

```
unknown_events = set(events.loc[~events["event"].isin(EVENT_TYPES),
"event"].unique()) - {"other"}
dq = pd.DataFrame([
   {"check":"duplicate_rows", "value": int(dupe_rows)},
   {"check":"unknown_event_labels", "value": ",".join(sorted(map(str,
unknown_events))) if unknown_events else ""}
])
save_table(dq, TAB_DIR / "data_quality_checks.csv")

toc("Outlier & quality checks ready")
```

8.2. EDA Pipeline Code Output

All code output from last run have been placed on Google Drive, and can be accessed through following URL: https:/drive.google.com/drive/folders/1mcSuwfq7d7WotWl4iP-JMWri-gB1X4iW?usp=sharing

9. APPENDIX B

9.1. Implementation Pipeline Code:

```
#!/usr/bin/env python
# coding: utf-8

# # Latent Customer Behaviour Segmentation Implementation Pipeline
#
# **Pipeline Configuration:**
# **data -> embedding features -> clustering -> evaluation -> explainability ->
business-friendly profiles**.
#

# ## 0. Dependency Installation

# Installing necessary libraries

import os
import sys
from pathlib import Path
```

```
# Core scientific + ML stack libraries
get_ipython().run_line_magic('pip', 'install -q scipy pandas scikit-learn gensim
umap-learn hdbscan shap xgboost seaborn tqdm pyarrow polars matplotlib
numpy')

# Extra utilities being used
get_ipython().run_line_magic('pip', 'install -q tqdm-joblib')
get_ipython().run_line_magic('pip', 'install -q --upgrade kaggle')
get_ipython().run_line_magic('pip', 'install -q ipython-autotime')

# Load autotime extension to measure execution times automatically
get_ipython().run_line_magic('load_ext', 'autotime')

print("Dependencies installed successfully")


# ## 1. Runtime Configuration and Data: Loading, Basic Checks, and
Sessionisation

# 0) CONFIGURATIONS

from pathlib import Path
import os
import numpy as np
import random
import datetime as dt
import pandas as pd

# Project meta
PROJECT_NAME =
"Latent_Customer_Behaviour_Segmentation_Implementation_Pipeline"
RUN_ID      = dt.datetime.now().strftime("%Y%m%d_%H%M%S")  # used in file
names

# Reproducibility
RANDOM_SEED = 42
```

```
np.random.seed(RANDOM_SEED)
random.seed(RANDOM_SEED)

# Threads / parallelisation
N_JOBS   = -1   # -1 = use all cores-1; set to fixed int for reproducibility

#  Paths
ROOT_DIR   = Path("C:/Users/Admin/Documents/WBS/Dissertation/Submission
Related files/Notebooks/Modelling")
DATA_DIR   = ROOT_DIR / "data"
OUT_DIR    = ROOT_DIR / "outputs_final_run"
FIGS_DIR   = OUT_DIR / "figs"
TABLES_DIR = OUT_DIR / "tables"
SHAP_DIR   = OUT_DIR / "shap_artifacts"

# Input files
EVENTS_CSV = DATA_DIR / "events.csv"              # clickstream (session_id,
user_id, item_id, timestamp, event, etc.)
ITEMS_CSV  = DATA_DIR / "item_properties_part1.csv"   # optional (item
metadata)
ITEMS2_CSV = DATA_DIR / "item_properties_part2.csv"       # optional 2nd
metadata file

# Ensure output folders exist
for p in [OUT_DIR, FIGS_DIR, TABLES_DIR, SHAP_DIR]:
    p.mkdir(parents=True, exist_ok=True)

# Data schema & timestamp handling
# Only change these if your raw file uses different column names
COLS = {
    "session":   "session_id",
    "user":      "user_id",
    "item":      "item_id",
    "timestamp": "timestamp",
    "event":     "event"          # expected: "view", "addtocart", "transaction" etc.
}
```

```
# timestamps sometimes come in ms; if so, set TS_DIV=1000 to convert to
seconds
TS_DIV = 1000  # 1 if already epoch seconds; 1000 if epoch milliseconds

# Session construction
SESSION_GAP_SEC       = 30 * 60   # break session if gap > 30 minutes
MIN_EVENTS_PER_SESSION= 2         # drop sessions with fewer events than
this
MAX_EVENTS_PER_SESSION= 5000      # guardrail for bots/anomalies
DROP_ABNORMAL_USERS   = True      # remove outliers by volume (top x%)
ABNORMAL_USER_PCTL    = 99.9      # drop users above this percentile of
event count

#  Representation / Embeddings
# We combine session-level Doc2Vec + item-level Word2Vec (mean-pooled)
USE_DOC2VEC          = True
USE_WORD2VEC         = True
CONCAT_DOC_W2V       = True

# Extra options
USE_SIF_WEIGHTING   = True   # Smooth Inverse Frequency weighting for
W2V pooling
USE_TFIDF_WEIGHTING = False  # TF-IDF pooling instead of SIF (rarely
needed)
USE_DBOW_EXTRA      = True   # Add small DBOW Doc2Vec for robustness

# Word2Vec (items -> vector)
W2V_VECTOR_SIZE      = 64
W2V_WINDOW           = 5
W2V_MIN_COUNT        = 2
W2V_SG               = 1         # 1 = skip-gram, 0 = CBOW
W2V_NEGATIVE         = 5
W2V_SUBSAMPLE        = 1e-5
W2V_EPOCHS           = 20

# Doc2Vec (session sequences -> vector)
D2V_VECTOR_SIZE      = 96
```

```
D2V_WINDOW          = 5
D2V_MIN_COUNT       = 2
D2V_DM              = 1        # 1 = PV-DM, 0 = PV-DBOW
D2V_EPOCHS          = 30
D2V_ALPHA           = 0.025
D2V_MIN_ALPHA       = 0.0005

# Dimensionality reduction for modelling & viz
USE_PCA_FOR_MODEL    = True
PCA_N_COMPONENTS     = 96        # for downstream clustering (denoising)
PCA_WHITEN           = False     # Whitening distorts the natural variance
structure, reducing the ability to interpret PCA components by how much variance
they explain
PCA_RANDOM_STATE     = RANDOM_SEED

USE_UMAP_FOR_VIZ     = True       # 2D viz only (do NOT use UMAP space
for metrics)
UMAP_N_NEIGHBORS     = 30
UMAP_MIN_DIST        = 0.25
UMAP_N_COMPONENTS    = 2
UMAP_METRIC          = "cosine"
UMAP_RANDOM_STATE    = RANDOM_SEED

# Clustering (we compare multiple models)

# Clustering orchestration (sampling + eval)
USE_SAMPLED_CLUSTERING = True    # if False, fit all models on full data
EVAL_ON_SAMPLE         = True    # compute internal metrics on a shared
sample
DO_INTERNAL_SCORES     = True    # print/save silhouette/CH/DB for quick
comparisons

# MiniBatch K-Means (baseline centroid method)
KMEANS_K_GRID        = [3, 4, 6, 8, 10, 12, 14]
KMEANS_BATCH_SIZE    = 4096
KMEANS_MAX_ITER      = 300
KMEANS_N_INIT        = 50      # number of centroid initialisations
```

```
# Gaussian Mixture Models (select K by BIC)
GMM_K_GRID          = [8, 10, 12, 14, 16, 18]
GMM_COV_TYPE         = "diag"
GMM_MAX_ITER        = 300
GMM_N_INIT          = 3
GMM_RANDOM_STATE     = RANDOM_SEED

# HDBSCAN (density-based; primary for segmentation)
# We run on a stratified sample for speed, then propagate labels to full set
HDBSCAN_SAMPLE_SIZE        = 90_000     # sample size for fit
HDBSCAN_MIN_CLUSTER_SIZE   = 1500        # smallest segment we care
about (tune 300–1000)
HDBSCAN_MIN_SAMPLES        = 20         # higher -> stricter core definition
(more noise)
HDBSCAN_METRIC             = "euclidean"   # try: "cosine" or "manhattan" (often
better than euclidean for embeddings)
HDBSCAN_CLUSTER_SELECTION  = "eom"      # "eom" (stable, fewer
clusters) or "leaf" (more fine-grained)
HDBSCAN_SELECTION_EPSILON  = 0.0        # >0 merges very close clusters;
keep 0 unless over-fragmenting
HDBSCAN_ALLOW_SINGLE_CLUSTER= False
HDBSCAN_CLUSTER_PROB_THRESH = 0.0        # keep all; raise to 0.15–0.3
to drop low-confidence members
HDBSCAN_APPROX_PREDICT     = True       # assign remaining points to
nearest clusters

# Agglomerative + kNN propagation
USE_AGGLO            = True
AGGLOMERATIVE_NCL     = 12      # number of clusters for Ward linkage (tune
as needed)
K_CONNECTIVITY        = 30      # k for building connectivity graph (Ward
requires euclidean)
K_PROPAGATE          = 30      # k for kNN label propagation to full set

# DBSCAN (classic)
USE_DBSCAN               = True
```

```
DBSCAN_EPS            = None       # None to evaluate from k-distance elbow
(or evaluated from previous calibrations)
DBSCAN_MIN_SAMPLES        = 10
DBSCAN_METRIC           = "euclidean"
# DBSCAN sampling for eps estimation and fit
DBSCAN_EPS_SAMPLE_N    = 25_000  # sample size for k-distance eps
estimation
DBSCAN_FIT_SAMPLE_N    = 30_000  # sample size for DBSCAN fit when
sampling + propagation

# OPTICS (alternative density method)
USE_OPTICS            = False
OPTICS_MIN_SAMPLES        = 20
OPTICS_XI             = 0.05       # smaller -> more clusters; larger -> fewer
clusters
OPTICS_MIN_CLUSTER_SIZE    = 50
OPTICS_MAX_EPS          = np.inf      # set to a reasonable cap to speed up
(e.g., 2.0) if known
OPTICS_METRIC           = "cosine"

# Evaluation (internal metrics + stability)
EVAL_SAMPLE_SIZE         = 70_000      # for fast metric computation on large
sets
COMPUTE_SILHOUETTE        = True        # only on non-noise labels
COMPUTE_CH_DB           = True        # Calinski–Harabasz & Davies–Bouldin
COMPUTE_DBCV            = False        # density-based validity (better for
HDBSCAN)
NOISE_LABEL            = -1

# Temporal stability (early vs late split)
CHECK_TEMPORAL_DRIFT       = True
EARLY_LATE_SPLIT_PCT       = 0.5         # 0.5 = first half vs second half by
time

# Cross-model agreement (optional)
COMPUTE_ARI_AMI          = True
```

```
# Output paths for clustering evaluations
def run_stem(name: str) -> Path:
    return OUT_DIR / f"{RUN_ID}_{name}"
SCORECARD_OUT          = run_stem("cluster_scorecard.csv")

# Profiling & naming clusters
# These are simple session-level features for interpretation/profiling tables
PROFILE_FEATURES = [
    "events", "view", "addtocart", "transaction",
    "duration_sec", "events_per_min", "item_diversity",
    "last_evt_view", "last_evt_addtocart", "last_evt_transaction",
    "hour", "dow", "is_evening"
]

# RFM-style baseline (for a simple, transparent benchmark)
BASELINE_LEVEL              = "session"    # or "visitor"
MONETARY_MODE               = "transactions"  # "transactions" or "none"
K_GRID_BASE                 = [4, 6, 8, 10, 12]
SIL_SAMPLE_N = 5_000
ATC_WEIGHT   = 0.5   # add-to-cart counts as “half” a purchase in M proxy

#  SHAP (surrogate explainability for cluster membership)
USE_SHAP_SURROGATE          = True
SHAP_SAMPLE_SIZE            = 30_000      # sample for surrogate modelling
SHAP_TOP_N_FEATURES         = 15          # report top-N features globally +
per-cluster
RF_N_ESTIMATORS             = 500
RF_MAX_DEPTH                = 12
RF_RANDOM_STATE             = RANDOM_SEED
SHAP_BACKGROUND_SIZE        = 5_000       # interventional background size
for TreeSHAP
SAVE_BEESWARM_PNG           = True

# Saving and file naming
SAVE_CSV_SUMMARIES          = True
SAVE_VIZ_PNG                = True
FIG_DPI                     = 150
```

```
#  Utility: consistent file stem for this run
def run_stem(name: str) -> Path:
    """Create a consistent file stem under outputs/ with run id and name."""
    return OUT_DIR / f"{RUN_ID}_{name}"

# printing configuration summary for current run
def print_config_summary():
    print("CONFIGURATION SUMMARY")
    print(f"PROJECT_NAME: {PROJECT_NAME}")
    print(f"RUN_ID:       {RUN_ID}")
    print(f"Random seed:  {RANDOM_SEED}")
    print(f"Events CSV:   {EVENTS_CSV}")
    print(f"PCA dims:     {PCA_N_COMPONENTS} | PCA for model:
{USE_PCA_FOR_MODEL}")
    print(f"UMAP for viz: {USE_UMAP_FOR_VIZ} (n={UMAP_N_COMPONENTS},
metric={UMAP_METRIC})")
    print(f"HDBSCAN: min_cluster_size={HDBSCAN_MIN_CLUSTER_SIZE},
min_samples={HDBSCAN_MIN_SAMPLES}, "
          f"metric={HDBSCAN_METRIC},
selection={HDBSCAN_CLUSTER_SELECTION}, "
          f"epsilon={HDBSCAN_SELECTION_EPSILON},
sample={HDBSCAN_SAMPLE_SIZE}")
    print(f"OPTICS: use={USE_OPTICS}, xi={OPTICS_XI},
min_samples={OPTICS_MIN_SAMPLES}, max_eps={OPTICS_MAX_EPS}")
    print(f"DBSCAN: use={USE_DBSCAN}, eps={DBSCAN_EPS},
min_samples={DBSCAN_MIN_SAMPLES}, metric={DBSCAN_METRIC}")
    print(f"Eval sample:  {EVAL_SAMPLE_SIZE}, DBCV={COMPUTE_DBCV},
silhouette={COMPUTE_SILHOUETTE}")
    print(f"SHAP: use={USE_SHAP_SURROGATE},
sample={SHAP_SAMPLE_SIZE}, topN={SHAP_TOP_N_FEATURES}")
    print(f"Outputs -> {OUT_DIR}")



print_config_summary()
```

```
# Loading data

print("Loading events and building sessions")

# 1) Load events (minimal schema)

usecols = ["visitorid", "timestamp", "event", "itemid"]
events = pd.read_csv(EVENTS_CSV, usecols=usecols, nrows=None)

# Basic cleaning:
# - drop rows missing core fields
# - keep timestamps as int64 (UNIX seconds)
events = events.dropna(subset=["visitorid", "timestamp", "event"]).copy()

# If timestamps are in milliseconds, downscale to seconds (TS_DIV from
CONFIG)
ts = events["timestamp"].astype("int64")
if ts.max() > 10**11 or TS_DIV == 1000:  # rough check + config guardrail
    events["timestamp"] = (ts // 1000).astype("int64")
else:
    events["timestamp"] = ts

# Normalise event labels (keeps downstream logic simple)
events["event"] = (
    events["event"]
    .astype("string")
    .str.strip()
    .str.lower()
    .replace({
        "add-to-cart": "addtocart",
        "add_to_cart": "addtocart",
        "cart": "addtocart",
        "purchase": "transaction",
        "buy": "transaction",
    })
)
```

```
# Sorting is important for session building
events = events.sort_values(["visitorid", "timestamp"]).reset_index(drop=True)

# 2) Build sessions (30 min inactivity gap)

gap = SESSION_GAP_SEC
# New session when gap between consecutive events for the same visitor >
threshold
dt = events.groupby("visitorid")["timestamp"].diff().fillna(gap + 1)
is_new_session = (dt > gap) | events["visitorid"].ne(events["visitorid"].shift())
events["session_id"] = np.cumsum(is_new_session)

# Dropping super-short sessions (e.g., single-click noise)
if MIN_EVENTS_PER_SESSION and MIN_EVENTS_PER_SESSION > 1:
    sizes = events.groupby("session_id").size()
    keep = sizes[sizes >= MIN_EVENTS_PER_SESSION].index
    events = events[events["session_id"].isin(keep)].reset_index(drop=True)

print(f"Rows after load/sessionise: {len(events):,}")
print(f"Sessions (pre-filter):      {events['session_id'].nunique():,}")
print(f"Visitors (pre-filter):      {events['visitorid'].nunique():,}")


# 3) Helper function to compute per-session features for profiling

def compute_session_features(df: pd.DataFrame) -> pd.DataFrame:
    """
    Returns one row per session_id with:
    - events, duration_sec, events_per_min
    - item_diversity (unique items)
    - last event flags (view/addtocart/transaction)
    - basic time features (hour, dow, is_evening)
    """
    agg = {
        "timestamp": ["min", "max", "count"],
        "itemid": pd.Series.nunique,
    }
    g = df.groupby("session_id").agg(agg)
    g.columns = ["ts_min", "ts_max", "events", "item_diversity"]
    g["duration_sec"] = (g["ts_max"] - g["ts_min"]).clip(lower=0)
    g["events_per_min"] = g["events"] / np.maximum(1, g["duration_sec"] / 60.0)

    # Last event per session
    last_evt = (
        df.sort_values(["session_id", "timestamp"])
          .groupby("session_id")["event"].last()
    )
    for ev in ["view", "addtocart", "transaction"]:
        g[f"last_evt_{ev}"] = (last_evt == ev).astype(int)

    # Lightweight time-of-day features based on last timestamp
    last_ts = (
        df.sort_values(["session_id", "timestamp"])
          .groupby("session_id")["timestamp"].last()
    )
    last_dt = pd.to_datetime(last_ts, unit="s", utc=True).dt.tz_convert(None)
    g["hour"] = last_dt.dt.hour
    g["dow"] = last_dt.dt.dayofweek
    g["is_evening"] = g["hour"].between(17, 22).astype(int)

    return g.reset_index()

session_feats = compute_session_features(events)



# 4) dropping abnormal users by volume
DROP_ABNORMAL_USERS = False
if DROP_ABNORMAL_USERS and "visitorid" in events.columns:
    before_users = events["visitorid"].nunique()
    user_events = events.groupby("visitorid").size()
    cutoff = np.percentile(user_events, ABNORMAL_USER_PCTL)
    heavy_users = user_events[user_events > cutoff].index

    if len(heavy_users) > 0:
        events =
events.loc[~events["visitorid"].isin(heavy_users)].reset_index(drop=True)
        print(
            f"[users] Dropped {len(heavy_users):,} heavy users "
            f"(>{ABNORMAL_USER_PCTL}th pct). Users: {before_users:,} ->
{events['visitorid'].nunique():,}"
        )
        # Recomputing session features (as some sessions might have vanished)
        session_feats = compute_session_features(events)


# 5) Defining outlier thresholds from data (p99 and safe floors threshold values)

def p99(series: pd.Series) -> float:
    return float(np.nanpercentile(series.values, 99))

p99_events       = p99(session_feats["events"])
p99_epm          = p99(session_feats["events_per_min"])
p99_duration_sec = p99(session_feats["duration_sec"])
p99_itemdiv      = p99(session_feats["item_diversity"])

# Defining floors helps in keeping thresholds sensible
THRESH_EVENTS_MAX   = max(100, int(p99_events))
THRESH_EPM_MAX      = max(30.0, p99_epm)
THRESH_DURATION_MAX = max(3 * 3600, int(p99_duration_sec))  # at least 3
hours
THRESH_ITEMDIV_MAX  = max(200, int(p99_itemdiv))

print("Outlier thresholds (p99 with floors):")
print(f"  events <= {THRESH_EVENTS_MAX:,}")
print(f"  events_per_min <= {THRESH_EPM_MAX:,.2f}")
print(f"  duration_sec <= {THRESH_DURATION_MAX:,}
(~{THRESH_DURATION_MAX/3600:.1f}h)")
print(f"  item_diversity <= {THRESH_ITEMDIV_MAX:,}")


# 6) Flagging and removing potential outliers
```

```
outlier_mask = (
    (session_feats["events"] > THRESH_EVENTS_MAX) |
    (session_feats["events_per_min"] > THRESH_EPM_MAX) |
    (session_feats["duration_sec"] > THRESH_DURATION_MAX) |
    (session_feats["item_diversity"] > THRESH_ITEMDIV_MAX)
)
potential_outliers = session_feats.loc[outlier_mask].copy()

before_rows = len(events)
before_sessions = events["session_id"].nunique()

kept_session_ids = set(session_feats.loc[~outlier_mask, "session_id"].values)
events =
events[events["session_id"].isin(kept_session_ids)].reset_index(drop=True)

after_rows = len(events)
after_sessions = events["session_id"].nunique()

print(
    f"[sessions] Outlier removal: {before_sessions:,} -> {after_sessions:,} sessions
"
    f"({before_sessions - after_sessions:,} dropped); rows {before_rows:,} ->
{after_rows:,}"
)

# Keeping features aligned with filtered events
session_feats = compute_session_features(events)



# 7) Saving for audit and dashboard
# Percentiles
pct = (session_feats[["events", "events_per_min", "duration_sec",
"item_diversity"]].describe(percentiles=[0.5, 0.9, 0.95, 0.99]).round(3))
pct_path = TABLES_DIR / f"{RUN_ID}_session_feature_percentiles.csv"
pct.to_csv(pct_path)
```

```
# Outliers we removed (transparency)
out_path = TABLES_DIR / f"{RUN_ID}_potential_automation_or_outliers.csv"
potential_outliers.to_csv(out_path, index=False)

# Dataset overview (post-cleaning)
overview = pd.DataFrame([{
    "rows": after_rows,
    "sessions": after_sessions,
    "visitors": events["visitorid"].nunique() if "visitorid" in events.columns else
np.nan,
    "items": events["itemid"].nunique(),
    "date_start": pd.to_datetime(events["timestamp"], unit="s").min(),
    "date_end": pd.to_datetime(events["timestamp"], unit="s").max()
}])
ov_path = TABLES_DIR / f"{RUN_ID}_dataset_overview.csv"
overview.to_csv(ov_path, index=False)

# Building a simple session-level funnel (post-cleaning)
evt_flags = (
    events.assign(flag=1)
        .pivot_table(index="session_id", columns="event",
        values="flag", aggfunc="sum", fill_value=0)
)
# Ensuring all columns exist
for c in ["view", "addtocart", "transaction"]:
    if c not in evt_flags.columns:
        evt_flags[c] = 0
# Saving funnel data
funnel = pd.DataFrame({
    "stage": ["view", "addtocart", "transaction"],
    "sessions_reached": [
        int((evt_flags["view"] > 0).sum()),
        int(((evt_flags["view"] > 0) & (evt_flags["addtocart"] > 0)).sum()),
        int(((evt_flags["addtocart"] > 0) & (evt_flags["transaction"] > 0)).sum())
    ]
})
funnel["prop_of_total"] = funnel["sessions_reached"] / after_sessions
```

```
fun_path = TABLES_DIR / f"{RUN_ID}_funnel_session_level.csv"
funnel.to_csv(fun_path, index=False)

print("Session features ready and aligned with filtered data.")



# ## 2. Deriving features via neural embeddings (session-level)

# 2) Session-level neural embeddings


# 1) Imports
from gensim.models import Word2Vec, Doc2Vec
from gensim.models.doc2vec import TaggedDocument
from sklearn.preprocessing import StandardScaler
import numpy as np
from collections import Counter
import math, os, time

# helpers for timing
def tic(section):
    print(f"\n>>> Starting: {section}")
    return time.perf_counter()

def toc(t0, section):
    dt = time.perf_counter() - t0
    print(f"completed: {section} in {dt:.2f}s")

# worker threads logic
if isinstance(N_JOBS, int) and N_JOBS == -1:
    THREADS = max(1, (os.cpu_count() or 2) - 1)
elif isinstance(N_JOBS, int) and N_JOBS > 0:
    THREADS = N_JOBS
else:
    THREADS = 1
```

```
# 2) Token sequences per session (items as strings)
t0 = tic("Preparing token sequences per session")
events_seq = events.sort_values(["session_id", "timestamp"], kind="mergesort")
session_to_items = (
    events_seq.groupby("session_id", sort=False)["itemid"]
            .apply(lambda s: [str(x) for x in s.values])
)
sentences = session_to_items.tolist()
print("token sequences per session prepared.")
toc(t0, "Preparing token sequences per session")

# 3) Frequency dictionaries (for SIF/TF-IDF pooling)
t0 = tic("Building frequency dictionaries")
global_counts = Counter(t for seq in sentences for t in seq)
N_tokens      = sum(global_counts.values())
N_docs        = len(sentences)
df_counts     = Counter(t for seq in map(set, sentences) for t in seq) if
USE_TFIDF_WEIGHTING else None
toc(t0, "Building frequency dictionaries")

# SIF weight function
a_sif = 5e-4
def sif_weight(token: str) -> float:
    pw = global_counts[token] / max(1, N_tokens)
    return a_sif / (a_sif + pw)

# TF-IDF helper
def tfidf_pool(tokens, get_vec):
    if not tokens:
        return None, 0.0
    cnt = Counter(tokens)
    L = float(len(tokens))
    acc = None; wsum = 0.0
    for t, tf in cnt.items():
        vec = get_vec(t)
        if vec is None:
            continue
        idf = math.log((N_docs + 1) / (df_counts[t] + 1)) + 1.0
        w = (tf / L) * idf
        acc = (vec * w) if acc is None else (acc + vec * w)
        wsum += w
    return acc, wsum

# 4) Word2Vec + pooling
t0 = tic("Training Word2Vec + pooling")
w2v_features = None
if USE_WORD2VEC:
    w2v = Word2Vec(
        sentences=sentences,
        vector_size=W2V_VECTOR_SIZE,
        window=W2V_WINDOW,
        min_count=W2V_MIN_COUNT,
        workers=THREADS,
        sg=W2V_SG,
        negative=W2V_NEGATIVE,
        sample=W2V_SUBSAMPLE,
        epochs=W2V_EPOCHS,
        seed=RANDOM_SEED,
        hs=0
    )

    # pooling rules
    def mean_pool(tokens):
        vs = [w2v.wv[t] for t in tokens if t in w2v.wv]
        return (np.mean(vs, axis=0) if vs else None), 1.0

    def sif_pool(tokens):
        acc = None; wsum = 0.0
        for t in tokens:
            if t in w2v.wv:
                v = w2v.wv[t]
                w = sif_weight(t)
                acc = (v * w) if acc is None else (acc + v * w)
                wsum += w
        return acc, wsum

    def get_vec(token):
        return w2v.wv[token] if token in w2v.wv else None

    def pooled_vector(tokens):
        if USE_TFIDF_WEIGHTING:
            acc, wsum = tfidf_pool(tokens, get_vec)
        elif USE_SIF_WEIGHTING:
            acc, wsum = sif_pool(tokens)
        else:
            acc, wsum = mean_pool(tokens)
        if (acc is None) or (wsum <= 0):
            return np.zeros(W2V_VECTOR_SIZE, dtype="float32")
        return (acc / wsum).astype("float32")

    w2v_features = np.vstack([pooled_vector(tokens) for tokens in sentences])

print("Word2Vec training completed.")
toc(t0, "Training Word2Vec + pooling")

# 5) Doc2Vec (DM + optional DBOW)
t0 = tic("Training Doc2Vec (DM) [and optional DBOW]")
d2v_features_list = []
if USE_DOC2VEC:
    docs = [TaggedDocument(words=toks, tags=[str(sid)])
            for sid, toks in zip(session_to_items.index.values, sentences)]

    # Distributed Memory
    d2v_dm = Doc2Vec(
        documents=docs,
        vector_size=D2V_VECTOR_SIZE,
        window=D2V_WINDOW,
        min_count=D2V_MIN_COUNT,
        workers=THREADS,
        dm=D2V_DM, dm_mean=1,
        negative=W2V_NEGATIVE,
        epochs=D2V_EPOCHS,
        seed=RANDOM_SEED
    )
    d2v_dm_feats = np.vstack([d2v_dm.dv[str(sid)] for sid in
session_to_items.index.values]).astype("float32")
    d2v_features_list.append(d2v_dm_feats)

    # DBOW for robustness on short sessions
    if USE_DBOW_EXTRA:
        d2v_dbow = Doc2Vec(
            documents=docs,
            vector_size=max(32, D2V_VECTOR_SIZE // 2),
            window=D2V_WINDOW,
            min_count=D2V_MIN_COUNT,
            workers=THREADS,
            dm=0, dbow_words=0,
            negative=W2V_NEGATIVE,
            epochs=max(10, D2V_EPOCHS // 2),
            seed=RANDOM_SEED
        )
        d2v_dbow_feats = np.vstack([d2v_dbow.dv[str(sid)] for sid in
session_to_items.index.values]).astype("float32")
        d2v_features_list.append(d2v_dbow_feats)

d2v_features = np.hstack(d2v_features_list) if d2v_features_list else None
print("Doc2Vec training completed.")
toc(t0, "Training Doc2Vec (DM) [and optional DBOW]")

# 6) Concatenate embeddings
t0 = tic("Concatenating feature blocks")
feat_blocks = [f for f in (w2v_features, d2v_features) if f is not None]
assert len(feat_blocks) >= 1, "No embeddings produced, check
USE_WORD2VEC / USE_DOC2VEC."
X = feat_blocks[0] if len(feat_blocks) == 1 else np.hstack(feat_blocks)
print("Concatenation completed. Raw feature shape:", X.shape)
toc(t0, "Concatenating feature blocks")
```

```
# 7) Standardise + L2-normalise
t0 = tic("Standardising & L2-normalising")
scaler = StandardScaler(with_mean=True, with_std=True)
X_scaled = scaler.fit_transform(X).astype("float32", copy=False)
row_norms = np.linalg.norm(X, axis=1, keepdims=True)
X_l2 = X / np.clip(row_norms, 1e-12, None)
print("Standardisation and L2-normalisation completed.")
print("Feature matrix (scaled) shape:", X_scaled.shape)
print("Feature matrix (L2) shape:", X_l2.shape)
toc(t0, "Standardising & L2-normalising")

# 8) Health checks
t0 = tic("Health checks")
n_zero_w2v = 0 if w2v_features is None else int((w2v_features ==
0).all(axis=1).sum())
vocab_size = 0 if not USE_WORD2VEC else len(w2v.wv)
print("W2V vocab size:", vocab_size)
print("Zero-vector sessions from W2V pooling:", n_zero_w2v)
toc(t0, "Health checks")


# ## 3. Dimensionality reduction of derived neural embeddings (PCA AND
UMAP)

# 3) Dimensionality reduction (PCA + UMAP)

# 1) Imports
from sklearn.decomposition import PCA
import numpy as np
import umap
import matplotlib.pyplot as plt
import warnings

warnings.filterwarnings("ignore", category=FutureWarning)

# 2) PCA on scaled features (denoise + numerical stability)
X_model = X_scaled
```

```
pca_dim_eff = int(min(PCA_N_COMPONENTS, X_scaled.shape[1]))
assert pca_dim_eff >= 2, "PCA_N_COMPONENTS must be >= 2."

pca = PCA(
    n_components=pca_dim_eff,
    whiten=PCA_WHITEN,
    random_state=PCA_RANDOM_STATE
)
X_pca = pca.fit_transform(X_scaled)

# Cosine-friendly copy (L2 after PCA): Euclidean is approximately same as
cosine
row_norms_pca = np.linalg.norm(X_pca, axis=1, keepdims=True)
X_pca_l2 = X_pca / np.clip(row_norms_pca, 1e-12, None)

# Downstream model inputs (choose L2 if metric is cosine)
X_model_kmeans  = X_pca_l2                  # k-means prefers L2
X_model_gmm     = X_pca                     # GMM prefers PCA/standardised
space
X_model_hdbscan = X_pca_l2 if HDBSCAN_METRIC == "cosine" else X_pca
X_model_dbscan  = X_pca_l2 if DBSCAN_METRIC  == "cosine" else X_pca

# Sample sizes for any later calibration/fit subsampling
DBSCAN_SAMPLE_N  = min(90_000, X_model_dbscan.shape[0])
HDBSCAN_SAMPLE_N = min(HDBSCAN_SAMPLE_SIZE,
X_model_hdbscan.shape[0])  # tied to configuration

# Reporting
expl_var = float(np.sum(pca.explained_variance_ratio_))
print(f"PCA | components: {pca_dim_eff} | total explained variance:
{expl_var:.3f}")
print("k-means/HDBSCAN input shape:", X_model_kmeans.shape)
print("GMM input shape:", X_model_gmm.shape)
print("DBSCAN sample size:", DBSCAN_SAMPLE_N)
print("HDBSCAN sample size:", HDBSCAN_SAMPLE_N)

# 3) UMAP for 2D visualisation (for plots only; not used for modelling)
```

```
umap_2d = None
if USE_UMAP_FOR_VIZ:
    umap_2d = umap.UMAP(
        n_neighbors=UMAP_N_NEIGHBORS,
        min_dist=UMAP_MIN_DIST,
        n_components=2,
        metric=UMAP_METRIC,            # from config (e.g., "cosine")
        random_state=UMAP_RANDOM_STATE
    ).fit_transform(X_pca)             # run on denoised PCA space

    print("UMAP ->", umap_2d.shape)
    print("Ethical Note: UMAP is for visual inspection only; global
distances/densities are not faithful.")

# 4) Save UMAP scatter to FIGS_DIR (with optional downsampling)
if USE_UMAP_FOR_VIZ and umap_2d is not None:
    rng = np.random.default_rng(RANDOM_SEED)
    UMAP_PLOT_MAX = 250_000
    n_points = umap_2d.shape[0]
    if n_points > UMAP_PLOT_MAX:
        idx = rng.choice(n_points, size=UMAP_PLOT_MAX, replace=False)
        umap_plot = umap_2d[idx]
    else:
        umap_plot = umap_2d

    FIGS_DIR.mkdir(parents=True, exist_ok=True)

    plt.figure(figsize=(8, 8), dpi=200)
    plt.scatter(umap_plot[:, 0], umap_plot[:, 1], s=1, alpha=0.5, linewidths=0,
rasterized=True)
    plt.title("UMAP of Sessions (2D projection)")
    plt.xlabel("UMAP-1"); plt.ylabel("UMAP-2")
    plt.tight_layout()
    out_path = FIGS_DIR / f"{RUN_ID}_umap_sessions_2d.png"
    plt.savefig(out_path, dpi=200, bbox_inches="tight")
    plt.close()
    print("UMAP plot saved to:", out_path)


# ## 4. Clustering (multiple unsupervised clustering models compared)

# 4) Clustering (KMeans, GMM[BIC], HDBSCAN + propagation, Agglomerative +
propagation, DBSCAN + eps tuning)

from sklearn.cluster import MiniBatchKMeans, AgglomerativeClustering,
DBSCAN as SK_DBSCAN
from sklearn.mixture import GaussianMixture
from sklearn.neighbors import NearestNeighbors, KNeighborsClassifier,
kneighbors_graph
from sklearn.metrics import silhouette_score, calinski_harabasz_score,
davies_bouldin_score
import hdbscan
import numpy as np
import pandas as pd
import time, os, warnings

warnings.filterwarnings("ignore", category=FutureWarning)

def tic(section):
    print(f"\n>>> Starting: {section}")
    return time.perf_counter()

def toc(t0, section):
    dt = time.perf_counter() - t0
    print(f"completed: {section} in {dt:.2f}s")

#  Shared evaluation sample (deterministic)
rng = np.random.default_rng(RANDOM_SEED)
if EVAL_ON_SAMPLE:
    EVAL_IDX = rng.choice(X_model_kmeans.shape[0],
size=min(EVAL_SAMPLE_SIZE, X_model_kmeans.shape[0]), replace=False)
else:
    EVAL_IDX = np.arange(X_model_kmeans.shape[0])
```

```
def _eval_scores(X, y, metric="euclidean"):
    """Internal scores on the shared evaluation sample; ignores noise (-1)."""
    y_eval = y[EVAL_IDX]
    mask = (y_eval != NOISE_LABEL) if (NOISE_LABEL in y_eval) else
np.ones_like(y_eval, bool)
    if mask.sum() < 50 or len(np.unique(y_eval[mask])) < 2:
        return {"sil": np.nan, "ch": np.nan, "db": np.nan}
    Xs, ys = X[EVAL_IDX][mask], y_eval[mask]
    out = {"sil": np.nan, "ch": np.nan, "db": np.nan}
    if COMPUTE_SILHOUETTE:
        out["sil"] = float(silhouette_score(Xs, ys, metric=metric))
    if COMPUTE_CH_DB:
        out["ch"]  = float(calinski_harabasz_score(Xs, ys))
        out["db"]  = float(davies_bouldin_score(Xs, ys))
    return out

labels_dict = {}
score_rows  = []

#  1) MiniBatch K-Means (on cosine-like L2 space)
t0 = tic("MiniBatch K-Means")
best_km, best_k, best_sil = None, None, -np.inf
for k in KMEANS_K_GRID:
    km = MiniBatchKMeans(
        n_clusters=k,
        batch_size=KMEANS_BATCH_SIZE,
        max_iter=KMEANS_MAX_ITER,
        n_init=KMEANS_N_INIT,
        random_state=RANDOM_SEED,
        init="k-means++"
    )
    km.fit(X_model_kmeans)
    labs = km.predict(X_model_kmeans)
    labels_dict[f"kmeans_{k}"] = labs
    s = _eval_scores(X_model_kmeans, labs)["sil"]
    if np.nan_to_num(s, nan=-1) > best_sil:
        best_sil, best_km, best_k = s, km, k
print("K-Means tried:", KMEANS_K_GRID, "| best k:", best_k, "| silhouette(eval):",
None if np.isnan(best_sil) else round(best_sil, 4))
toc(t0, "MiniBatch K-Means")

#  2) GMM (choose K by BIC on PCA space)
t0 = tic("Gaussian Mixture Models (BIC)")
best_gmm, best_bic, best_g = None, np.inf, None
for g in GMM_K_GRID:
    gmm = GaussianMixture(
        n_components=g,
        covariance_type=GMM_COV_TYPE,
        random_state=GMM_RANDOM_STATE,
        max_iter=GMM_MAX_ITER,
        n_init=GMM_N_INIT,
        reg_covar=1e-6,
        init_params="kmeans"
    )
    gmm.fit(X_model_gmm)
    bic = gmm.bic(X_model_gmm)
    labels_dict[f"gmm_{g}"] = gmm.predict(X_model_gmm)
    if bic < best_bic:
        best_bic, best_g, best_gmm = bic, g, gmm
print("GMM best k by BIC:", best_g, "| BIC:", round(best_bic, 2))
toc(t0, "Gaussian Mixture Models (BIC)")

# 3) HDBSCAN (fit on sample if configured) + approximate_predict
t0 = tic("HDBSCAN + propagation")
n = X_model_hdbscan.shape[0]
fit_n = min(HDBSCAN_SAMPLE_SIZE, n) if USE_SAMPLED_CLUSTERING
else n
fit_idx = (rng.choice(n, size=fit_n, replace=False) if fit_n < n else np.arange(n))

print(f"[HDBSCAN] fit_size={fit_idx.size:,}/{n:,} |
min_cluster_size={HDBSCAN_MIN_CLUSTER_SIZE} |
min_samples={HDBSCAN_MIN_SAMPLES} | metric={HDBSCAN_METRIC} |
selection={HDBSCAN_CLUSTER_SELECTION}")
```

```
hdb = hdbscan.HDBSCAN(
    min_cluster_size=HDBSCAN_MIN_CLUSTER_SIZE,
    min_samples=HDBSCAN_MIN_SAMPLES,
    metric=HDBSCAN_METRIC,
    cluster_selection_method=HDBSCAN_CLUSTER_SELECTION,
    cluster_selection_epsilon=HDBSCAN_SELECTION_EPSILON,
    allow_single_cluster=HDBSCAN_ALLOW_SINGLE_CLUSTER,
    core_dist_n_jobs=(os.cpu_count() or 1),
    prediction_data=True
).fit(X_model_hdbscan[fit_idx])

from hdbscan import prediction as hdb_pred
if fit_idx.size < n and HDBSCAN_APPROX_PREDICT:
    labs_full, strengths = hdb_pred.approximate_predict(hdb, X_model_hdbscan)
    labs_full = labs_full.astype(int)
else:
    labs_full = hdb.labels_.astype(int)
    strengths = getattr(hdb, "probabilities_", np.ones_like(labs_full, dtype=float))

# optional confidence thresholding
if HDBSCAN_CLUSTER_PROB_THRESH > 0.0:
    low_conf = strengths < HDBSCAN_CLUSTER_PROB_THRESH
    labs_full = labs_full.copy()
    labs_full[low_conf] = NOISE_LABEL

labels_dict["hdbscan"] = labs_full
labels_dict["hdbscan_conf"] = strengths
hdb_scores = _eval_scores(X_model_hdbscan, labs_full, metric=("cosine" if
HDBSCAN_METRIC == "cosine" else "euclidean"))
print("HDBSCAN clusters (excl. noise):", len(np.unique(labs_full[labs_full !=
NOISE_LABEL])))
print("HDBSCAN eval:", hdb_scores)
toc(t0, "HDBSCAN + propagation")

# 4) Agglomerative (Ward) + kNN propagation (kept as in prior runs)
if USE_AGGLO:
    t0 = tic("Agglomerative (Ward) + kNN propagation")
    # Fit on sample if sampling enabled
    agg_fit_n = min( max(AGGLOMERATIVE_NCL * 4_000, 60_000),
X_model_kmeans.shape[0]) if USE_SAMPLED_CLUSTERING else
X_model_kmeans.shape[0]
    agg_idx = (rng.choice(X_model_kmeans.shape[0], size=agg_fit_n,
replace=False)
               if agg_fit_n < X_model_kmeans.shape[0] else
np.arange(X_model_kmeans.shape[0]))
    X_agg_fit = X_model_kmeans[agg_idx]

    print(f"[Agglo] building connectivity graph (n={X_agg_fit.shape[0]:,},
k={K_CONNECTIVITY})")
    graph_conn = kneighbors_graph(X_agg_fit, n_neighbors=K_CONNECTIVITY,
mode="connectivity", metric="euclidean")

    print(f"[Agglo] fitting Ward linkage with n_clusters={AGGLOMERATIVE_NCL}")
    agg_ward = AgglomerativeClustering(n_clusters=AGGLOMERATIVE_NCL,
linkage="ward", connectivity=graph_conn)
    labels_agg_fit = agg_ward.fit_predict(X_agg_fit)

    print("Agglomerative: propagating labels to full dataset via kNN")
    knn_ag = KNeighborsClassifier(n_neighbors=K_PROPAGATE,
weights="distance")
    knn_ag.fit(X_agg_fit, labels_agg_fit)
    labels_agg_full = knn_ag.predict(X_model_kmeans).astype(int)
    labels_dict[f"agg_ward_{AGGLOMERATIVE_NCL}"] = labels_agg_full

    agg_scores = _eval_scores(X_model_kmeans, labels_agg_full,
metric="euclidean")
    print("Agglomerative clusters:", len(np.unique(labels_agg_full)))
    print("Agglomerative eval:", agg_scores)
    toc(t0, "Agglomerative (Ward) + kNN propagation")

# 5) DBSCAN (k-distance eps estimation on sample) + optional propagation
if USE_DBSCAN:
    t0 = tic("DBSCAN (eps via k-distance) + optional propagation")
    # choose sample for eps estimation (and for fit if sampling)
    eps_idx = (rng.choice(X_model_dbscan.shape[0],
size=min(DBSCAN_EPS_SAMPLE_N, X_model_dbscan.shape[0]),
replace=False)
               if USE_SAMPLED_CLUSTERING else
np.arange(X_model_dbscan.shape[0]))
    X_db_eps = X_model_dbscan[eps_idx]

    print(f"[DBSCAN] estimating eps via k-distance on {X_db_eps.shape[0]:,}
points (k={DBSCAN_MIN_SAMPLES}, metric={DBSCAN_METRIC})")
    nn = NearestNeighbors(n_neighbors=DBSCAN_MIN_SAMPLES,
metric=DBSCAN_METRIC).fit(X_db_eps)
    dists, _ = nn.kneighbors(X_db_eps)
    kdist = np.sort(dists[:, -1])
    eps_grid = [float(np.percentile(kdist, p)) for p in [95, 96, 97, 98, 99]]

    best_eps, best_s = None, -np.inf
    for eps in eps_grid:
        tmp_labels = SK_DBSCAN(eps=eps,
min_samples=DBSCAN_MIN_SAMPLES,
metric=DBSCAN_METRIC).fit_predict(X_db_eps)
        # evaluate fairly on same eval index — propagate if sample != full
        if X_db_eps.shape[0] == X_model_dbscan.shape[0]:
            s = _eval_scores(X_model_dbscan, tmp_labels, metric=("cosine" if
DBSCAN_METRIC == "cosine" else "euclidean"))["sil"]
        else:
            mask_fit = (tmp_labels != NOISE_LABEL)
            if mask_fit.sum() > 0:
                knn_db = KNeighborsClassifier(n_neighbors=K_PROPAGATE,
weights="distance").fit(X_db_eps[mask_fit], tmp_labels[mask_fit])
                lab_full_tmp = np.full(X_model_dbscan.shape[0], NOISE_LABEL,
dtype=int)
                lab_full_tmp[eps_idx] = tmp_labels
                rest_idx = np.setdiff1d(np.arange(X_model_dbscan.shape[0]), eps_idx,
assume_unique=True)
                lab_full_tmp[rest_idx] = knn_db.predict(X_model_dbscan[rest_idx])
            s = _eval_scores(X_model_dbscan, lab_full_tmp, metric=("cosine" if
DBSCAN_METRIC == "cosine" else "euclidean"))["sil"]
        else:
            s = -np.inf
        if np.nan_to_num(s, nan=-1) > best_s:
            best_s, best_eps = s, eps

    print("DBSCAN chosen eps:", round(best_eps, 5), "| silhouette(eval):", None if
np.isnan(best_s) else round(best_s, 4))

    # final fit (sample + propagate, or full)
    if USE_SAMPLED_CLUSTERING:
        fit_idx = (rng.choice(X_model_dbscan.shape[0],
size=min(DBSCAN_FIT_SAMPLE_N, X_model_dbscan.shape[0]),
replace=False))
        X_db_fit = X_model_dbscan[fit_idx]
        db = SK_DBSCAN(eps=best_eps,
min_samples=DBSCAN_MIN_SAMPLES,
metric=DBSCAN_METRIC).fit(X_db_fit)
        lab_fit = db.labels_.astype(int)

        print("DBSCAN: propagating labels to full dataset via kNN...")
        labels_db_full = np.full(X_model_dbscan.shape[0], NOISE_LABEL,
dtype=int)
        labels_db_full[fit_idx] = lab_fit
        mask_fit = (lab_fit != NOISE_LABEL)
        if mask_fit.sum() > 0:
            knn = KNeighborsClassifier(n_neighbors=K_PROPAGATE,
weights="distance").fit(X_db_fit[mask_fit], lab_fit[mask_fit])
            rest_idx = np.setdiff1d(np.arange(X_model_dbscan.shape[0]), fit_idx,
assume_unique=True)
            labels_db_full[rest_idx] = knn.predict(X_model_dbscan[rest_idx])
    else:
        db = SK_DBSCAN(eps=best_eps,
min_samples=DBSCAN_MIN_SAMPLES,
metric=DBSCAN_METRIC).fit(X_model_dbscan)
        labels_db_full = db.labels_.astype(int)
    db_key = f"dbscan_eps{round(best_eps,5)}_ms{DBSCAN_MIN_SAMPLES}"
    labels_dict[db_key] = labels_db_full
    db_scores = _eval_scores(X_model_dbscan, labels_db_full, metric=("cosine" if
DBSCAN_METRIC == "cosine" else "euclidean"))
    print("DBSCAN clusters (excl. noise):",
len(np.unique(labels_db_full[labels_db_full != NOISE_LABEL])))
    print("DBSCAN eval:", db_scores)
    toc(t0, "DBSCAN (eps via k-distance) + optional propagation")

# 6) Internal score summary + save
t0 = tic("Score summary + save")

def _n_clusters(y):
    u = np.unique(y)
    return int(len(u) - (1 if NOISE_LABEL in u else 0))

rows = []
rows.append(("kmeans_" + str(best_k),) + tuple(_eval_scores(X_model_kmeans,
labels_dict[f"kmeans_{best_k}"]).values()))
rows.append(("gmm_" + str(best_g),) + tuple(_eval_scores(X_model_gmm,
labels_dict[f"gmm_{best_g}"]).values()))
rows.append(("hdbscan",)           + tuple(_eval_scores(X_model_hdbscan,
labels_dict["hdbscan"], metric=("cosine" if HDBSCAN_METRIC=="cosine" else
"euclidean")).values()))
if USE_AGGLO:
    rows.append((f"agg_ward_{AGGLOMERATIVE_NCL}",) +
tuple(_eval_scores(X_model_kmeans,
labels_dict[f"agg_ward_{AGGLOMERATIVE_NCL}"]).values()))
if USE_DBSCAN:
    rows.append((db_key,) + tuple(_eval_scores(X_model_dbscan,
labels_dict[db_key], metric=("cosine" if DBSCAN_METRIC=="cosine" else
"euclidean")).values()))

score_df = pd.DataFrame(rows, columns=["model", "silhouette", "ch", "db"])
# add #clusters and noise%
def _noise_pct(name):
    if name == "hdbscan":
        return float((labels_dict["hdbscan"] == NOISE_LABEL).mean())
    if name.startswith("dbscan_"):
        return float((labels_dict[name] == NOISE_LABEL).mean())
    return 0.0

score_df["n_clusters"] = score_df["model"].map(lambda m:
_n_clusters(labels_dict["hdbscan"]) if m=="hdbscan"
                                              else (_n_clusters(labels_dict[db_key]) if
m.startswith("dbscan_")
                                                    else _n_clusters(labels_dict[m])))
score_df["noise_pct"]  = score_df["model"].map(_noise_pct)

score_df.to_csv(SCORECARD_OUT, index=False)
print("Saved scorecard ->", SCORECARD_OUT)
print(score_df)
toc(t0, "Score summary + save")

# 7) Save full labels (all models)
t0 = tic("Saving labels (full)")
labels_df = pd.DataFrame({"session_id": session_to_items.index.to_numpy()})
for name, lab in labels_dict.items():
    if name.endswith("_conf"):
        continue
    labels_df[name] = lab
if "hdbscan_conf" in labels_dict:
    labels_df["hdbscan_conf"] = labels_dict["hdbscan_conf"]

LABELS_OUT = run_stem("cluster_labels_all_models.csv")
labels_df.to_csv(LABELS_OUT, index=False)
print("Saved labels ->", LABELS_OUT)
print("Clusterers saved:", list(labels_dict.keys()))
toc(t0, "Saving labels (full)")


# ## 5. Baseline Static Model Implementation RFM Inspired
# ### 5.1. Session level RFM-style feature extraction

# 5.1) Feature extraction for session-level RFM proxies

# 1) Imports
import time, numpy as np, pandas as pd, math

def tic(msg):
   print(f">>> {msg}"); return time.perf_counter()
def toc(t0, msg):
   print(f"---- {msg} in {time.perf_counter()-t0:.2f}s")

assert {"session_id","timestamp","event","itemid"}.issubset(events.columns)

t0_all = tic("Session feature extraction (fast path)")

# 2) dtypes and normalisation (optimises groupby performance)
t0 = tic("cast dtypes")
ev = events[["session_id","timestamp","event","itemid"]].copy()
ev["event"] = ev["event"].astype(str).str.lower().astype("category")
if ev["itemid"].dtype != "category":
   ev["itemid"] = ev["itemid"].astype("category")
# respect global timestamp unit (supports ms->s if TS_DIV=1000)
ev["timestamp"] = (ev["timestamp"].astype("int64") // TS_DIV).astype("int64")
toc(t0, "cast dtypes")

# 3) Vectorised counts per (session,event)
t0 = tic("event counts (crosstab)")
counts = (ev.groupby(["session_id","event"], observed=True)
          .size()
          .unstack(fill_value=0))
# ensure expected columns exist (singular names to match
PROFILE_FEATURES)
for col in ["view","addtocart","transaction"]:
   if col not in counts.columns:
      counts[col] = 0
# keep singular column names
```

```
counts = counts[["view","addtocart","transaction"]].astype("int32")
counts["events"] = counts.sum(axis=1).astype("int32")
toc(t0, "event counts done")

# 4) Timing statistics (min/max/duration) without sorting
t0 = tic("timings (min/max/duration)")
g = ev.groupby("session_id", observed=True, sort=False)
t_min = g["timestamp"].min()
t_max = g["timestamp"].max()
duration_sec = (t_max - t_min).astype("float64").clip(lower=0.0)
toc(t0, "timings done")

# 5) Unique items per session (item_diversity)
t0 = tic("unique items (item_diversity)")
item_diversity = g["itemid"].nunique().astype("int32")
toc(t0, "unique items done")

# 6) Last event per session -> one-hot
t0 = tic("last event via idxmax")
idx_last = g["timestamp"].idxmax()
last_event = ev.loc[idx_last,
["session_id","event"]].set_index("session_id")["event"]
last_dummies = pd.get_dummies(last_event, prefix="last_evt", dtype="int8")
for col in ["last_evt_view","last_evt_addtocart","last_evt_transaction"]:
    if col not in last_dummies.columns:
        last_dummies[col] = 0
toc(t0, "last event done")

# 7) Assemble features (rates, time-of-day, recency, bounce)
t0 = tic("assembling features")
sess = pd.DataFrame({
    "duration_sec": duration_sec,
    "item_diversity": item_diversity,
}).join(counts, how="left").join(last_dummies, how="left")

# rates/ratios
```

```
sess["events_per_min"] = (sess["events"] /
np.maximum(sess["duration_sec"]/60.0, 1e-6)).astype("float64")

# time-of-day + day-of-week from session start
hour = pd.to_datetime(t_min, unit="s",
errors="coerce").dt.hour.fillna(0).astype("int16")
dow  = pd.to_datetime(t_min, unit="s",
errors="coerce").dt.dayofweek.fillna(0).astype("int8")
sess["hour"] = hour
sess["dow"]  = dow
sess["is_evening"] = ((hour >= 18) & (hour <= 23)).astype("int8")

# recency (days from dataset end to session end)
dataset_end = ev["timestamp"].max()
sess["recency_days"] = ((dataset_end - t_max) / (60*60*24)).astype("float64")

# bounce flag
sess["is_bounce"] = (sess["events"] <= 2).astype("int8")

# clean index name
sess.index.name = "session_id"
toc(t0, "features assembled")
toc(t0_all, "Session feature extraction")

# persist features for downstream SHAP/personas/reporting
out_feat = run_stem("session_features_rfm_proxies.csv")
sess.to_csv(out_feat)
print("Session-level feature table saved to:", out_feat)



# ### 5.2. Session-level RFM-proxy model (K-Means & profiling)

# 5.2) Session-level RFM-proxy model (K-Means & profiling)



# 1) Import
import numpy as np
import pandas as pd
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
from sklearn.preprocessing import StandardScaler, RobustScaler
import time

# pulling from config
RANDOM_SEED  = globals().get("RANDOM_SEED", 42)
K_GRID_BASE  = globals().get("K_GRID_BASE", [2, 4, 6, 8, 10, 12])

# tiny timers
def tic(msg): print(f"\n>>> {msg}"); return time.perf_counter()
def toc(t0, msg): print(f"---- {msg} in {time.perf_counter()-t0:.2f}s")

# Preconditions
t0 = tic("RFM-proxy model — data preprocessing checks")
assert "sess" in globals(), "Expected 'sess' from the fast extractor."
need = {"recency_days","events","addtocart","transaction"}   # singular names per
extractor
missing = need - set(sess.columns)
assert not missing, f"'sess' is missing columns: {missing}"
toc(t0, "pre-checks complete")

# 2) Construct R/F/M proxies
t0 = tic("Constructing R/F/M proxies")

R_proxy = sess["recency_days"].astype("float64")
F_proxy = np.log1p(sess["events"].astype("float64"))

tx   = sess["transaction"].astype("float64")
atc  = sess["addtocart"].astype("float64")
views = (sess["events"].astype("float64") - atc - tx).clip(lower=0)  # crude proxy

print(f"Non-zero rates -> tx: {(tx>0).mean():.4%}, atc: {(atc>0).mean():.4%}")

M_candidates = {
    "M_tx_plus_atc_log1p": np.log1p(tx + ATC_WEIGHT * atc),         #
recommended
    "M_tx_log1p":          np.log1p(tx),                          # strict monetary
    "M_intent_per_view":   np.log1p((tx + ATC_WEIGHT * atc) / (1.0 + views))
}

chosen_M_name, M_proxy = None, None
for name, arr in M_candidates.items():
    if np.nanstd(arr) > 0:
        chosen_M_name, M_proxy = name, arr
        break

if M_proxy is None:
    print("All M candidates are constant -> falling back to RF baseline.")
    rfm_proxy = pd.DataFrame({
        "R_recency_days": R_proxy,
        "F_events_log1p": F_proxy,
    }, index=sess.index)
else:
    print(f"Chosen M proxy: {chosen_M_name}")
    rfm_proxy = pd.DataFrame({
        "R_recency_days": R_proxy,
        "F_events_log1p": F_proxy,
        "M_intent_log1p": M_proxy,
    }, index=sess.index)

# Scaling
USE_ROBUST = True  # set False to use StandardScaler
scaler = RobustScaler(quantile_range=(5, 95)) if USE_ROBUST else
StandardScaler()
X_rfm = scaler.fit_transform(rfm_proxy.values).astype("float32", copy=False)

print("RFM-proxy preview:")
print(rfm_proxy.head().to_string())
toc(t0, "R/F/M proxies ready")

# 3) K-Means with sampled silhouette over the K grid
```

```
t0 = tic("Clustering RFM-proxy with K-Means + silhouette(K grid)")
n = X_rfm.shape[0]
rng = np.random.default_rng(RANDOM_SEED)
idx_sil = rng.choice(n, size=min(SIL_SAMPLE_N, n), replace=False)

best_k, best_sil, best_labels = None, -1.0, None
labels_by_k = {}

for k in K_GRID_BASE:
    if k < 2 or k >= n:
        continue
    km = KMeans(n_clusters=k, n_init=10, random_state=RANDOM_SEED)
    y  = km.fit_predict(X_rfm)
    labels_by_k[k] = y
    sil = silhouette_score(X_rfm[idx_sil], y[idx_sil], metric="euclidean")
    print(f"k={k:<2d} -> silhouette(sample)={sil:.4f}")
    if np.isfinite(sil) and sil > best_sil:
        best_k, best_sil, best_labels = k, sil, y

if best_k is None:
    raise RuntimeError("No valid K found. Please adjust K_GRID_BASE.")

print(f"Chosen K = {best_k} (silhouette={best_sil:.4f})")
toc(t0, "K-Means clustering completed for RFM Baseline model")

# 4) Profiling
t0 = tic("Profiling clusters (size, share, quantiles, intent rate)")
full_sil = float(silhouette_score(X_rfm, best_labels, metric="euclidean"))
print(f"[RFM-proxy] Full-data silhouette at K={best_k}: {full_sil:.4f}")

prof = rfm_proxy.copy()
prof["label"] = best_labels
prof["tx"]    = (sess["transaction"] > 0).astype(int)
prof["atc"]   = (sess["addtocart"] > 0).astype(int)

g = prof.groupby("label", observed=True)
cluster_size = g.size().rename("n")
```

```
share = (cluster_size / len(prof)).rename("share")
tx_rate  = g["tx"].mean().rename("tx_rate")
atc_rate = g["atc"].mean().rename("atc_rate")

def q(x):
   return pd.Series({"p25": np.percentile(x,25), "median": np.median(x), "p75":
np.percentile(x,75)})

desc_parts = [g["R_recency_days"].apply(q).unstack().add_prefix("R_"),
              g["F_events_log1p"].apply(q).unstack().add_prefix("F_")]
if "M_intent_log1p" in prof.columns:
   desc_parts.append(g["M_intent_log1p"].apply(q).unstack().add_prefix("M_"))

desc = pd.concat(desc_parts, axis=1)
summary = pd.concat([cluster_size, share, tx_rate, atc_rate], axis=1).join(desc)
print(summary.round(3).to_string())
toc(t0, "profiling completed")

# 5) Saving results
t0 = tic("Saving results")

out = sess.join(rfm_proxy)
out["rfm_proxy_k"] = best_k
out["rfm_proxy_label"] = best_labels
out["rfm_full_silhouette"] = full_sil
out["rfm_chosen_M"] = chosen_M_name if chosen_M_name is not None else
"none"
out["rfm_scaler"] = "Robust(5-95)" if isinstance(scaler, RobustScaler) else
"Standard"

feat_path = run_stem("session_rfm_proxy_features.parquet")
lab_path  = run_stem(f"session_rfm_proxy_labels_k{best_k}.csv")
prof_path = run_stem(f"session_rfm_proxy_profile_k{best_k}.csv")

out.to_parquet(feat_path, index=True)
summary.reset_index().to_csv(prof_path, index=False)
```

```
pd.DataFrame({"session_id": out.index, "label": best_labels}).to_csv(lab_path,
index=False)

print(f"Saved features -> {feat_path}")
print(f"Saved labels  -> {lab_path}")
print(f"Saved profile -> {prof_path}")

# expose labels for downstream agreement/SHAP steps
if "labels_dict" not in globals():
    labels_dict = {}
labels_dict[f"rfm_proxy_kmeans_session_{best_k}"] = best_labels

toc(t0, "results saved")



# ## 6. Visualising clusters across all algorithms (using UMAP 2D overlays)

# 6) Visualising clusters across all algorithms (UMAP overlays)



# 1) Imports
import numpy as np
import pandas as pd
import umap
import matplotlib.pyplot as plt
from pathlib import Path
import warnings, time, os

# 2) Progress helpers
def tic(msg): print(f"\n>>> {msg}"); return time.perf_counter()
def toc(t0, msg): print(f"---- {msg} in {time.perf_counter()-t0:.2f}s")

warnings.filterwarnings("ignore")

# 3) Preconditions & defaults
t0 = tic("UMAP overlays — prechecks")
```

```
assert "labels_dict" in globals() and isinstance(labels_dict, dict) and labels_dict,
"labels_dict is missing/empty."

# pull-through from config
RANDOM_SEED = globals().get("RANDOM_SEED", 42)
RUN_ID      = globals().get("RUN_ID", "run")
UMAP_N_NEIGHBORS = int(globals().get("UMAP_N_NEIGHBORS", 30))
UMAP_MIN_DIST    = float(globals().get("UMAP_MIN_DIST", 0.05))
UMAP_METRIC      = globals().get("UMAP_METRIC", "cosine")

# figure/output dirs
FIGS_DIR = globals().get("FIGS_DIR", Path("outputs/figs"))
OUT_DIR  = globals().get("OUT_DIR", Path("outputs"))
FIGS_DIR.mkdir(parents=True, exist_ok=True)
OUT_DIR.mkdir(parents=True, exist_ok=True)

# reproducible RNG
RNG = np.random.default_rng(RANDOM_SEED)

# Base neural-embedding matrix (prefer L2-PCA for cosine-like geometry)
if "X_model_kmeans" in globals():
    BASE_X = X_model_kmeans
elif "X_pca_l2" in globals():
    BASE_X = X_pca_l2
elif "X_pca" in globals():
    BASE_X = X_pca
else:
    raise AssertionError("Could not find BASE_X
(X_model_kmeans/X_pca_l2/X_pca). Run the PCA block again (section 3).")
N = BASE_X.shape[0]

# Session index for alignment
if "SESSION_INDEX" in globals():
    SESSION_INDEX = SESSION_INDEX
elif "session_to_items" in globals():
    SESSION_INDEX = session_to_items.index.to_numpy()
else:
    SESSION_INDEX = (events[["session_id"]].drop_duplicates()
                     .astype({"session_id":"int64"}).values.ravel())

# Local plotting/runtime knobs
PLOT_MAX      = 100_000
POINT_SIZE    = 1.8
PLOT_DPI      = 140
REUSE_COORDS  = True
FIT_SAMPLE_N  = 120_000
BATCH_SIZE    = 100_000
LOW_MEMORY    = True
toc(t0, "prechecks completed")

# 4) Build or reuse UMAP for neural-embedding space
t0 = tic("UMAP base (neural-embedding space): sample-fit + transform")

coords_all = None
if REUSE_COORDS and ("umap_2d" in globals()) and (umap_2d is not None)
and (getattr(umap_2d, "shape", (0,0))[0] == N):
    coords_all = umap_2d.astype(np.float32, copy=False)
    print("Reused precomputed umap_2d.")
else:
    fit_idx = RNG.choice(N, size=min(FIT_SAMPLE_N, N), replace=False)
    umap_model_all = umap.UMAP(
        n_neighbors=UMAP_N_NEIGHBORS,
        min_dist=UMAP_MIN_DIST,
        n_components=2,
        random_state=RANDOM_SEED,
        metric=("euclidean" if BASE_X is X_model_kmeans else UMAP_METRIC),
# L2-PCA -> euclidean ok
        low_memory=LOW_MEMORY,
    ).fit(BASE_X[fit_idx].astype(np.float32, copy=False))

    coords_all = np.empty((N, 2), dtype=np.float32)
    coords_all[fit_idx] = umap_model_all.embedding_.astype(np.float32,
copy=False)
```

```
    rest = np.setdiff1d(np.arange(N), fit_idx, assume_unique=True)
    for s in range(0, len(rest), BATCH_SIZE):
        b = rest[s:s+BATCH_SIZE]
        coords_all[b] = umap_model_all.transform(BASE_X[b].astype(np.float32,
copy=False)).astype(np.float32, copy=False)

    # expose for later reuse
    umap_2d = coords_all

# Save coordinates (run-stamped under outputs/)
coords_all_df = pd.DataFrame({
    "session_id": SESSION_INDEX,
    "umap_x": coords_all[:,0],
    "umap_y": coords_all[:,1],
})
coords_all_df.to_parquet(run_stem("umap_all_models_coords.parquet"),
index=False)
coords_all_df.to_csv(run_stem("umap_all_models_coords.csv"), index=False)
print("Saved base UMAP coords to:",
run_stem("umap_all_models_coords.parquet"), "and",
run_stem("umap_all_models_coords.csv"))
print("Ethical Note: UMAP is for visual inspection only and not a clustering tool.")
toc(t0, "UMAP base ready")

# 5) Plot helper (centroids computed from plotted subset only)
def _plot_clusters(coords2d, labels, title, out_png, max_points=PLOT_MAX):
    n = coords2d.shape[0]
    if n > max_points:
        idx = RNG.choice(n, size=max_points, replace=False)
        pts, labs = coords2d[idx], labels[idx]
    else:
        pts, labs = coords2d, labels

    # color mapping: noise in light gray
    base_colors = plt.rcParams['axes.prop_cycle'].by_key().get('color',
['C0','C1','C2','C3','C4','C5','C6','C7','C8','C9'])
    uniq = np.unique(labs)
    uniq_sorted = [u for u in uniq if u != -1] + ([-1] if -1 in uniq else [])
    color_map, c_idx = {}, 0
    for u in uniq_sorted:
        if u == -1:
            color_map[u] = "#D3D3D3"
        else:
            color_map[u] = base_colors[c_idx % len(base_colors)]
            c_idx += 1

    plt.figure(figsize=(8,8), dpi=PLOT_DPI)
    for u in uniq_sorted:
        m = (labs == u)
        if not np.any(m):
            continue
        plt.scatter(pts[m,0], pts[m,1], s=POINT_SIZE, alpha=0.55, linewidths=0,
                    color=color_map[u], label=("Noise" if u == -1 else f"Cluster {u}"),
rasterized=True)

    # centroids from the plotted subset
    cents = []
    for u in [u for u in uniq_sorted if u != -1]:
        m = (labs == u)
        if np.any(m):
            cents.append(pts[m].mean(axis=0))
    if cents:
        cents = np.vstack(cents)
        plt.scatter(cents[:,0], cents[:,1], s=120, marker="X", edgecolor="black",
                    linewidths=0.5, color=[color_map[u] for u in uniq_sorted if u != -1],
label="Centroids")

    plt.title(title)
    plt.xlabel("UMAP-1"); plt.ylabel("UMAP-2")
    plt.legend(markerscale=4, fontsize=8, frameon=True)
    plt.tight_layout()
    plt.savefig(out_png, dpi=PLOT_DPI, bbox_inches="tight")
    plt.close()
    print("Saved:", out_png)
```

```
# 6) Label alignment helper
def _align_labels(name, arr):
    """Ensures that labels align with BASE_X row order (SESSION_INDEX).
Returns None if not alignable."""
    if len(arr) == len(SESSION_INDEX):
        return np.asarray(arr)
    if ("rfm_proxy" in globals()) and isinstance(rfm_proxy, pd.DataFrame):
        if len(arr) == rfm_proxy.shape[0]:
            lab_series = pd.Series(arr, index=rfm_proxy.index)
            aligned = lab_series.reindex(SESSION_INDEX).to_numpy()
            return aligned if aligned.shape[0] == len(SESSION_INDEX) else None
        if name == "rfm_proxy" and "rfm_proxy_label" in rfm_proxy.columns:
            aligned =
rfm_proxy["rfm_proxy_label"].reindex(SESSION_INDEX).to_numpy()
            return aligned if aligned.shape[0] == len(SESSION_INDEX) else None
    return None

# 7) Plot overlays for ALL models on the neural embedding UMAP
t0 = tic("Plotting neural-embedding overlays for all models")
all_plots = []  # (title, filename)

for key, lab in labels_dict.items():
    if key.endswith("_conf"):  # skip HDBSCAN strengths
        continue
    labels_aligned = _align_labels(key, lab)
    if labels_aligned is None:
        print(f"[skip] {key} — label array length mismatch (expected
{len(SESSION_INDEX)})")
        continue

    pretty = (key
              .replace("_knnprop_conf","")
              .replace("_knnprop","")
              .replace("agg_","Agglomerative ")
              .replace("kmeans_","KMeans k=")
              .replace("gmm_","GMM k=")
            .replace("dbscan_","DBSCAN ")
            .replace("hdbscan","HDBSCAN"))
    fname = f"{RUN_ID}_umap_{key}.png".replace("__","_")
    out_png = FIGS_DIR / fname
    _plot_clusters(coords_all, labels_aligned, f"Neural-embedding UMAP —
{pretty}", out_png)
    all_plots.append((pretty, out_png))

toc(t0, "neural overlays completed")

# 8) Separate UMAP for RFM baseline (3 features)
if ("X_rfm" in globals()) and ("rfm_proxy" in globals()):
    t0 = tic("UMAP for session RFM-proxy (3 features): sample-fit + transform")
    X_rfm = X_rfm.astype(np.float32, copy=False)
    n_rfm  = X_rfm.shape[0]
    fit_n  = min(60_000, n_rfm)
    fit_idx = RNG.choice(n_rfm, size=fit_n, replace=False)

    umap_rfm = umap.UMAP(
        n_neighbors=UMAP_N_NEIGHBORS,
        min_dist=UMAP_MIN_DIST,
        n_components=2,
        random_state=RANDOM_SEED,
        metric="euclidean",
        low_memory=LOW_MEMORY,
    ).fit(X_rfm[fit_idx])

    coords_rfm = np.empty((n_rfm, 2), dtype=np.float32)
    coords_rfm[fit_idx] = umap_rfm.embedding_.astype(np.float32, copy=False)
    rest = np.setdiff1d(np.arange(n_rfm), fit_idx, assume_unique=True)
    for s in range(0, len(rest), BATCH_SIZE):
        b = rest[s:s+BATCH_SIZE]
        coords_rfm[b] = umap_rfm.transform(X_rfm[b]).astype(np.float32,
copy=False)

    # plot any RFM labels present
    for key, lab in labels_dict.items():
        if key.startswith("rfm_proxy_") and (not key.endswith("_conf")):
            labels_rfm = _align_labels("rfm_proxy", lab)
            if labels_rfm is None:
                print(f"[skip RFM plot] {key} — label array length mismatch (expected
{len(SESSION_INDEX)})")
                continue
            out_png = FIGS_DIR / f"{RUN_ID}_umap_{key}.png"
            _plot_clusters(coords_rfm, labels_rfm, f"RFM-proxy UMAP — {key}",
out_png)
            all_plots.append((key, out_png))

    # saving RFM coordinates (run-stamped)
    coords_rfm_df = pd.DataFrame({
        "session_id": rfm_proxy.index.values,
        "umap_x": coords_rfm[:,0],
        "umap_y": coords_rfm[:,1],
    })
    coords_rfm_df.to_parquet(run_stem("umap_rfm_proxy_coords.parquet"),
index=False)
    coords_rfm_df.to_csv(run_stem("umap_rfm_proxy_coords.csv"), index=False)
    toc(t0, "RFM-proxy UMAP completed")
else:
    print("Skipped RFM-proxy UMAP: X_rfm/rfm_proxy not found (run 5.1 & 5.2
first).")

print("\nGenerated plots:")
for ttitle, p in all_plots:
    print("-", ttitle, "->", p)


# ## 7. Model evaluation (internal metrics, cross-model agreement and temporal
drift)

# 7) Model evaluation (internal metrics, cross-model agreement, temporal drift)


# 1) Imports
```

```
import numpy as np
import pandas as pd
import time, os
from sklearn.metrics import (
    silhouette_score, calinski_harabasz_score, davies_bouldin_score,
    adjusted_rand_score, adjusted_mutual_info_score
)

# 2) small timers
def tic(section):
    print(f"\n>>> Starting: {section}")
    return time.perf_counter()

def toc(t0, section):
    dt = time.perf_counter() - t0
    print(f"---- completed: {section} in {dt:.2f}s")

# 3) Pre-checks & pulling through config
t0 = tic("Evaluation — prechecks")
assert "labels_dict" in globals() and len(labels_dict) > 0, "labels_dict not found;
run clustering first."

RANDOM_SEED        = globals().get("RANDOM_SEED", 42)
RNG                = globals().get("RNG", np.random.default_rng(RANDOM_SEED))
EVAL_SAMPLE_SIZE   = int(globals().get("EVAL_SAMPLE_SIZE", 10_000))
COMPUTE_SILHOUETTE = bool(globals().get("COMPUTE_SILHOUETTE",
True))
COMPUTE_CH_DB      = bool(globals().get("COMPUTE_CH_DB", True))
COMPUTE_DBCV       = bool(globals().get("COMPUTE_DBCV", False))
NOISE_LABEL        = int(globals().get("NOISE_LABEL", -1))

OUT_DIR.mkdir(parents=True, exist_ok=True)

# Reuse existing S_EVAL_IDX if present; else deterministic sample
if "S_EVAL_IDX" not in globals():
    N_ref = next(v.shape[0] for k, v in labels_dict.items() if not k.endswith("_conf"))
    S_EVAL_IDX = RNG.choice(N_ref, size=min(EVAL_SAMPLE_SIZE, N_ref),
replace=False)
print(f"Eval sample size: {len(S_EVAL_IDX)}")
toc(t0, "Evaluation — prechecks completed")

# 4) Choose the feature space matching each model’s labels
def _X_for_label(key: str):
    """
    Return the feature matrix that matches how 'key' labels were produced.
    Falls back to X_model_kmeans (L2-PCA) if ambiguous.
    """
    if key.startswith("kmeans_") or key.startswith("agg_"):
        return X_model_kmeans
    if key.startswith("gmm_"):
        return X_model_gmm
    if key.startswith("hdbscan_"):
        return X_model_hdbscan
    if key.startswith("dbscan_"):
        return X_model_dbscan
    if key.startswith("rfm_simple") or key.startswith("rfm_proxy"):
        return globals().get("X_rfm", X_model_kmeans)
    return X_model_kmeans

# 5) Internal metrics per model (silhouette / CH / DB / optional DBCV)
t0 = tic("Internal metrics per model")

rows = []
for name, y_full in labels_dict.items():
    if name.endswith("_conf"):
        continue

    X = _X_for_label(name)
    idx = S_EVAL_IDX if len(S_EVAL_IDX) < X.shape[0] else
np.arange(X.shape[0])

    y = y_full[idx]
    # ignore noise for geometry metrics
    mask = (y != NOISE_LABEL) if (NOISE_LABEL in y) else np.ones_like(y,
dtype=bool)
    Xs, ys = X[idx][mask], y[mask]

    sil = ch = db = np.nan
    dbcv = np.nan

    # need at least 2 clusters & enough points
    if (np.unique(ys).size >= 2) and (Xs.shape[0] >= 100):
        # stratified subsample for silhouette (keep ≥2 labels)
        Xsub, ysub = Xs, ys
        sub_n = min(5_000, Xs.shape[0])
        if Xs.shape[0] > sub_n:
            labels_u = np.unique(ys)
            per = max(1, int(np.ceil(sub_n / labels_u.size)))
            take_idx = []
            for c in labels_u:
                idx_c = np.where(ys == c)[0]
                if idx_c.size > 0:
                    k = min(per, idx_c.size)
                    take_idx.append(RNG.choice(idx_c, size=k, replace=False))
            take_idx = np.concatenate(take_idx)
            if np.unique(ys[take_idx]).size >= 2:
                Xsub, ysub = Xs[take_idx], ys[take_idx]

        if COMPUTE_SILHOUETTE:
            sil = float(silhouette_score(Xsub, ysub, metric="euclidean"))
        if COMPUTE_CH_DB:
            ch  = float(calinski_harabasz_score(Xs, ys))
            db  = float(davies_bouldin_score(Xs, ys))

        # Optional: DBCV (density-based validity), best for density models
        if COMPUTE_DBCV:
            try:
                from hdbscan.validity import validity_index
                # Use cosine-like geometry when X is L2-normalised
(kmeans/dbscan/hdbscan on L2-PCA)
```

```
            # validity_index expects condensed distances or a metric name
supported by pairwise distances.
            # We'll use 'euclidean' here because BASE_X is L2-normalised for
cosine-like setups.
            dbcv = float(validity_index(Xs, y=ys, metric='euclidean'))
        except Exception as e:
            # keep robust; just log once per model
            print(f"[DBCV skipped for {name}] {e}")

    # count clusters excl. noise
    n_clusters = int(np.unique(y_full[y_full != NOISE_LABEL]).size) if
(NOISE_LABEL in y_full) else int(np.unique(y_full).size)
    rows.append({
        "model": name, "n_clusters": n_clusters,
        "silhouette": sil, "calinski_harabasz": ch, "davies_bouldin": db,
        "dbcv": dbcv
    })

internal_df = pd.DataFrame(rows).sort_values(["silhouette"], ascending=False)
internal_path = run_stem("eval_internal_metrics.csv")
internal_df.to_csv(internal_path, index=False)
print("Saved internal metrics to:", internal_path)
print(internal_df.head(10))
toc(t0, "Internal metrics per model")

# 6) Cross-model agreement (ARI / AMI)
t0 = tic("Cross-model agreement (ARI / AMI)")

keys = [k for k in labels_dict.keys() if not k.endswith("_conf")]
Y = {k: labels_dict[k] for k in keys}

ari = pd.DataFrame(index=keys, columns=keys, dtype="float32")
ami = pd.DataFrame(index=keys, columns=keys, dtype="float32")

for ki in keys:
    yi = Y[ki]
    for kj in keys:
        yj = Y[kj]
        if len(yi) == len(yj):
            ari.loc[ki, kj] = adjusted_rand_score(yi, yj)
            ami.loc[ki, kj] = adjusted_mutual_info_score(yi, yj)
        else:
            ari.loc[ki, kj] = np.nan
            ami.loc[ki, kj] = np.nan

ari_path = run_stem("eval_cross_model_ARI.csv")
ami_path = run_stem("eval_cross_model_AMI.csv")
ari.to_csv(ari_path); ami.to_csv(ami_path)
print("Saved ARI ->", ari_path)
print("Saved AMI ->", ami_path)

def _top_pairs(M, k=5):
    vals = []
    for i in M.index:
        for j in M.columns:
            if i >= j:  # avoid dup/self (matrix is symmetric)
                continue
            v = M.loc[i, j]
            if pd.notnull(v):
                vals.append((i, j, float(v)))
    vals.sort(key=lambda x: x[2], reverse=True)
    return vals[:k], vals[-k:]

top5, bot5 = _top_pairs(ari)
print("\nTop 5 ARI pairs:")
for a,b,v in top5:  print(f"{a:>30s}  vs  {b:<30s}  ->  ARI={v:.3f}")
print("\nBottom-5 ARI pairs:")
for a,b,v in bot5:  print(f"{a:>30s}  vs  {b:<30s}  ->  ARI={v:.3f}")

toc(t0, "Cross-model agreement (ARI / AMI)")

# 7) Temporal drift (early vs late halves by session start time)
t0 = tic("Temporal drift check (early vs late halves)")
```

```
# Build/confirm SESSION_INDEX
if "SESSION_INDEX" not in globals():
    if "session_to_items" in globals():
        SESSION_INDEX = session_to_items.index.to_numpy()
    else:
        SESSION_INDEX = (events[["session_id"]].drop_duplicates()
                         .astype({"session_id":"int64"}).values.ravel())

# Respect TS_DIV from config
TS_DIV = int(globals().get("TS_DIV", 1))
sess_first_ts = (events.groupby("session_id")["timestamp"].min() // TS_DIV)
sess_first_ts = sess_first_ts.reindex(SESSION_INDEX).to_numpy()
median_ts = np.nanmedian(sess_first_ts)
early = (sess_first_ts <= median_ts)
late  = ~early

drift_rows = []
for name, y in Y.items():
    labs = np.asarray(y)
    uniq = np.unique(labs)
    for c in uniq:
        if c == NOISE_LABEL:  # skip noise here
            continue
        p_early = float((labs[early] == c).mean())
        p_late  = float((labs[late]  == c).mean())
        drift_rows.append({
            "model": name, "cluster": int(c),
            "p_early": p_early, "p_late": p_late,
            "abs_diff": abs(p_early - p_late)
        })

drift_df = pd.DataFrame(drift_rows).sort_values(["abs_diff"], ascending=False)
TEMPORAL_OUT = run_stem("eval_temporal_drift.csv")
drift_df.to_csv(TEMPORAL_OUT, index=False)
print("Saved temporal drift summary to:", TEMPORAL_OUT)
print(drift_df.head(10))
toc(t0, "Temporal drift check")
```

```
# ## 8. Explainability via a surrogate RandomForest + SHAP

# 8) Surrogate explainability with SHAP

# 1) Imports
import os, gc, joblib, numpy as np, pandas as pd, shap
from pathlib import Path
from sklearn.ensemble import RandomForestClassifier

# 2) pulling through from config (defaults safe if missing)
USE_SHAP_SURROGATE   = bool(globals().get("USE_SHAP_SURROGATE",
True))
if not USE_SHAP_SURROGATE:
   print("[SHAP] USE_SHAP_SURROGATE=False, skipping this section.")
else:
   RANDOM_SEED        = int(globals().get("RANDOM_SEED", 42))
   RNG                = globals().get("RNG",
np.random.default_rng(RANDOM_SEED))
   N_JOBS             = int(globals().get("N_JOBS", -1))
   NOISE_LABEL        = int(globals().get("NOISE_LABEL", -1))

   OUT_DIR            = globals().get("OUT_DIR", Path("outputs"));
OUT_DIR.mkdir(parents=True, exist_ok=True)
   SHAP_DIR           = globals().get("SHAP_DIR", OUT_DIR / "shap_artifacts");
SHAP_DIR.mkdir(parents=True, exist_ok=True)

   RF_N_ESTIMATORS     = int(globals().get("RF_N_ESTIMATORS", 400))
   RF_MAX_DEPTH        = globals().get("RF_MAX_DEPTH", None)
   RF_RANDOM_STATE     = RANDOM_SEED
   SHAP_SAMPLE_SIZE    = int(globals().get("SHAP_SAMPLE_SIZE",
100_000))     # total cap across clusters
   SHAP_TOP_N_FEATURES = int(globals().get("SHAP_TOP_N_FEATURES",
15))
   SHAP_BACKGROUND_SIZE=
int(globals().get("SHAP_BACKGROUND_SIZE", 2_000))
```

```
    SAVE_BEESWARM_PNG   = bool(globals().get("SAVE_BEESWARM_PNG",
True))

    assert "labels_dict" in globals() and labels_dict, "labels_dict is missing. Run
clustering first."

    # 3) Pick the lead clustering model (prefer neural over RFM unless LEAD_KEY
provided)
    lead_key = globals().get("LEAD_KEY", None)

    def _pick_lead_key():
        if "internal_df" in globals() and isinstance(internal_df, pd.DataFrame) and not
internal_df.empty:
            order = internal_df.sort_values("silhouette",
ascending=False)["model"].tolist()
            for m in order:
                if not m.startswith("rfm_proxy_"):
                    return m
            return order[0]
        path = OUT_DIR / "eval_internal_metrics.csv"
        if path.exists():
            df = pd.read_csv(path)
            order = df.sort_values("silhouette", ascending=False)["model"].tolist()
            for m in order:
                if not m.startswith("rfm_proxy_"):
                    return m
            return order[0]
        # fallback: any kmeans_* else first non-conf entry
        return next((k for k in labels_dict if k.startswith("kmeans_") and not
k.endswith("_conf")),
                    next(k for k in labels_dict if not k.endswith("_conf")))

    if lead_key is None:
        lead_key = _pick_lead_key()

    print(f"[SHAP] Lead model: {lead_key}")
    y_all = np.asarray(labels_dict[lead_key])
    # 4) Feature table for surrogate (use 'out' if present, else 'sess')
    feat_df = out.copy() if "out" in globals() else sess.copy()

    # Align row order to SESSION_INDEX (same as clustering inputs)
    if "SESSION_INDEX" in globals():
        if "session_id" in feat_df.columns:
            feat_df = feat_df.set_index("session_id")
        feat_df = feat_df.reindex(pd.Index(SESSION_INDEX))

    # Drop non-numeric + housekeeping columns, clean NaN/inf, remove constant
cols
    drop_cols = [c for c in feat_df.columns if c.startswith("rfm_proxy_")] +
["session_id"]
    feat_df = (feat_df.drop(columns=drop_cols, errors="ignore")
                  .select_dtypes(include=[np.number])
                  .replace([np.inf, -np.inf], np.nan)
                  .fillna(0.0))
    feat_df = feat_df.loc[:, feat_df.nunique(dropna=False) > 1]

    X_full = feat_df.to_numpy().astype("float32", copy=False)
    y_full = y_all

    # Remove noise labels from training
    mask_train = (y_full != NOISE_LABEL)
    X_train, y_train = X_full[mask_train], y_full[mask_train]
    assert X_train.shape[0] > 0, "No non-noise labels available for SHAP
surrogate."

    # 5) Train/load RandomForest surrogate (fully configurable)
    rf_path = SHAP_DIR / f"rf_surrogate_{lead_key}.joblib"
    if rf_path.exists():
        rf = joblib.load(rf_path)
        print("[SHAP] Loaded surrogate RF from disk.")
    else:
        rf = RandomForestClassifier(
            n_estimators=RF_N_ESTIMATORS,
            max_depth=RF_MAX_DEPTH,
            max_features="sqrt",
            min_samples_leaf=20,
            n_jobs=N_JOBS if isinstance(N_JOBS, int) else -1,
            random_state=RF_RANDOM_STATE
        ).fit(X_train, y_train)
        joblib.dump(rf, rf_path)
        print("[SHAP] Trained & saved surrogate RF.")

    # 6) Balanced, capped SHAP sample across clusters (total cap =
SHAP_SAMPLE_SIZE)
    classes = np.sort(np.unique(y_train))
    K = len(classes)
    per_cls = max(500, int(np.ceil(SHAP_SAMPLE_SIZE / max(1, K))))
    take_idx = []
    for c in classes:
        idx_c = np.where(y_train == c)[0]
        if idx_c.size == 0:
            continue
        take = min(per_cls, idx_c.size)
        take_idx.append(RNG.choice(idx_c, size=take, replace=False))
    take_idx = np.concatenate(take_idx) if take_idx else np.arange(min(10_000,
X_train.shape[0]))
    X_shap, y_shap = X_train[take_idx], y_train[take_idx]
    print(f"[SHAP] Sampled for SHAP: {X_shap.shape[0]} rows across {K}
clusters.")
    pd.Series(y_shap).value_counts().sort_index().to_csv(SHAP_DIR /
f"shap_sample_class_sizes_{lead_key}.csv")

    # 7) Background for TreeExplainer (interventional): small, stratified
    bg_per_cls = max(50, int(np.ceil(SHAP_BACKGROUND_SIZE / max(1, K))))
    bg_idx = []
    for c in classes:
        idx_c = np.where(y_train == c)[0]
        if idx_c.size == 0:
            continue
        take = min(bg_per_cls, idx_c.size)
        bg_idx.append(RNG.choice(idx_c, size=take, replace=False))
    bg_idx = np.concatenate(bg_idx) if bg_idx else RNG.choice(X_train.shape[0],
size=min(SHAP_BACKGROUND_SIZE, X_train.shape[0]), replace=False)
    X_bg = X_train[bg_idx]

    # Helper to normalise SHAP outputs across versions
    def _as_class_list(sv):
        if isinstance(sv, list):
            return sv
        if isinstance(sv, np.ndarray):
            if sv.ndim == 3:              # (n,F,C)
                return [sv[:, :, i] for i in range(sv.shape[2])]
            if sv.ndim == 2:              # (n,F)
                return [sv]
        raise ValueError(f"Unexpected SHAP output shape/type: {type(sv)},
ndim={getattr(sv,'ndim',None)}")

    # 8) Streaming SHAP aggregation (memory friendly)
    explainer = shap.TreeExplainer(rf, data=X_bg,
feature_perturbation="interventional", model_output="raw")
    BATCH = 2000
    F     = X_shap.shape[1]
    C     = len(rf.classes_)
    sum_abs_global     = np.zeros(F, dtype=np.float64)
    count_global       = 0
    sum_abs_by_cluster = np.zeros((C, F), dtype=np.float64)
    count_by_cluster   = np.zeros(C, dtype=np.int64)

    for s in range(0, X_shap.shape[0], BATCH):
        xb = X_shap[s:s+BATCH]; yb = y_shap[s:s+BATCH]
        raw = explainer.shap_values(xb, check_additivity=False)
        clist = _as_class_list(raw)  # list of (n_batch,F)

        # multiclass alignment; SHAP may return 1 array in binary case
        if len(clist) == 1 and C == 2:
            clist = [clist[0], clist[0]]
        for ci, c in enumerate(rf.classes_):
            sv_ci = clist[min(ci, len(clist)-1)]
            m = (yb == c)
            if not np.any(m):
                continue
            v = np.abs(sv_ci[m]).sum(axis=0)    # (F,)
            sum_abs_by_cluster[ci] += v
            count_by_cluster[ci]   += int(m.sum())
            sum_abs_global         += v
            count_global           += int(m.sum())

        del raw, clist, xb, yb
        if (s // BATCH) % 5 == 0:
            np.savez_compressed(
                SHAP_DIR / f"shap_ckpt_{lead_key}.npz",
                sum_abs_global=sum_abs_global,
count_global=np.array([count_global]),
                sum_abs_by_cluster=sum_abs_by_cluster,
count_by_cluster=count_by_cluster
            )
            gc.collect()

    # 9) Save summaries (timestamped via run_stem if available)
    feat_names = feat_df.columns.to_list()

    def _out(path_like):
        # use run_stem if available for consistent RUN_ID prefix
        return run_stem(path_like) if "run_stem" in globals() else (OUT_DIR /
path_like)

    global_imp = pd.DataFrame({
        "feature": feat_names,
        "mean_abs_shap": sum_abs_global / max(1, count_global)
    }).sort_values("mean_abs_shap", ascending=False)
    global_path = _out(f"shap_global_{lead_key}.csv")
    global_imp.to_csv(global_path, index=False)
    per_cluster_rows = []
    for ci, c in enumerate(rf.classes_):
        if count_by_cluster[ci] == 0:
            continue
        tmp = pd.DataFrame({
            "cluster": int(c),
            "feature": feat_names,
            "mean_abs_shap": (sum_abs_by_cluster[ci] / max(1,
count_by_cluster[ci]))
        }).sort_values("mean_abs_shap",
ascending=False).head(SHAP_TOP_N_FEATURES)
        tmp["rank"] = np.arange(1, len(tmp) + 1)
        per_cluster_rows.append(tmp)

    top_path = _out(f"shap_top_features_per_cluster_{lead_key}.csv")
    if per_cluster_rows:
        pd.concat(per_cluster_rows, ignore_index=True).to_csv(top_path,
index=False)

    # Optional extra export if SHAP_TABLE_OUT set in config
    SHAP_TABLE_OUT = globals().get("SHAP_TABLE_OUT", None)
    if SHAP_TABLE_OUT is not None and per_cluster_rows:
        pd.concat(per_cluster_rows,
ignore_index=True).to_csv(SHAP_TABLE_OUT, index=False)

    print("[SHAP] Saved:", global_path, "and", top_path)

    # 10) Tiny beeswarm (subsampled) if enabled
    if SAVE_BEESWARM_PNG and X_shap.shape[0] > 0:
        import matplotlib.pyplot as plt
        idx = RNG.choice(X_shap.shape[0], size=min(3000, X_shap.shape[0]),
replace=False)
        raw   = explainer.shap_values(X_shap[idx], check_additivity=False)
        clist = _as_class_list(raw)

        plt.figure(figsize=(8,6), dpi=140)
        if len(clist) > 1:  # multiclass or mirrored binary
```

```
            counts = [np.sum(y_shap[idx] == c) for c in rf.classes_]
            top_ci = int(np.argmax(counts))
            shap.summary_plot(clist[min(top_ci, len(clist)-1)], X_shap[idx],
                              feature_names=feat_names, show=False)
        else:
            shap.summary_plot(clist[0], X_shap[idx], feature_names=feat_names,
show=False)
        plt.tight_layout()
        fig_path = _out(f"shap_beeswarm_{lead_key}.png")
        plt.savefig(fig_path, dpi=140, bbox_inches="tight")
        plt.close()
        print("[SHAP] Saved beeswarm:", fig_path)

    print("Ethical Note: SHAP indicates feature association with cluster
assignments, not causality. Correlated features may share credit.")


# ## 9. Model Evaluation & Identifying Personas

# 9). Model Evaluation & Identifying Personas

# 1) Imports
import numpy as np
import pandas as pd
from pathlib import Path
from IPython.display import display

# 2) Config & paths (pulled from global CONFIG block where possible)
BASELINE_LEVEL = globals().get("BASELINE_LEVEL", "session")        #
"session" | "visitor"
MONETARY_MODE  = globals().get("MONETARY_MODE", "transactions")    #
"transactions" | "none"
OUT_DIR        = globals().get("OUT_DIR", Path("outputs"));
OUT_DIR.mkdir(parents=True, exist_ok=True)
PERSONA_OUT    = globals().get("PERSONA_OUT", OUT_DIR /
"personas.csv")
```

```
TS_DIV        = float(globals().get("TS_DIV", 1))              # respect timestamp
units
RANDOM_SEED    = globals().get("RANDOM_SEED", 42)
RNG           = globals().get("RNG", np.random.default_rng(RANDOM_SEED))

# 3) Preconditions
assert "events" in globals() and isinstance(events, pd.DataFrame), "Missing
'events' dataframe from earlier blocks."
need_cols = {"session_id", "visitorid", "timestamp", "event"}
missing   = need_cols - set(events.columns)
assert not missing, f"'events' is missing columns: {missing}"

# 4) Build unit table (R/F/M inputs)
if BASELINE_LEVEL == "session":
    unit_ids  = events["session_id"].drop_duplicates().astype("int64")
    unit_tbl  = pd.DataFrame({"unit_id": unit_ids.values})
    sess_last = (events.groupby("session_id")["timestamp"].max() // TS_DIV)
    sess_freq = events.groupby("session_id").size()
    tx_counts = (events.assign(event=events["event"].astype(str).str.lower())
                 .loc[lambda d: d["event"] == "transaction"]
                 .groupby("session_id").size())
    unit_tbl = (unit_tbl
        .merge(sess_last.rename("last_ts"), left_on="unit_id", right_index=True,
how="left")
        .merge(sess_freq.rename("frequency"), left_on="unit_id", right_index=True,
how="left")
        .merge(tx_counts.rename("tx_count"), left_on="unit_id", right_index=True,
how="left"))
else:  # visitor-level
    unit_ids = events["visitorid"].drop_duplicates()
    unit_tbl = pd.DataFrame({"unit_id": unit_ids.values})
    vis_last = (events.groupby("visitorid")["timestamp"].max() // TS_DIV)
    vis_freq = events.groupby("visitorid").size()
    tx_counts = (events.assign(event=events["event"].astype(str).str.lower())
                 .loc[lambda d: d["event"] == "transaction"]
                 .groupby("visitorid").size())
    unit_tbl = (unit_tbl
        .merge(vis_last.rename("last_ts"), left_on="unit_id", right_index=True,
how="left")
        .merge(vis_freq.rename("frequency"), left_on="unit_id", right_index=True,
how="left")
        .merge(tx_counts.rename("tx_count"), left_on="unit_id", right_index=True,
how="left"))

# 5) Recency (days) & monetary
max_ts = int((events["timestamp"].max() // TS_DIV))
unit_tbl["recency_days"] = ((max_ts - unit_tbl["last_ts"]) / (60 * 60 *
24)).astype("float64")
unit_tbl["tx_count"]     = unit_tbl["tx_count"].fillna(0).astype("float64")
unit_tbl["monetary"]     = (1.0 if MONETARY_MODE == "none" else
unit_tbl["tx_count"])

# 6) Attach RFM cluster labels (prefer labels from 'out', else fall back to
labels_dict)
label_series = None

if BASELINE_LEVEL == "session":
    # try from 'out' (RFM KMeans block)
    if "out" in globals() and isinstance(out, pd.DataFrame) and ("rfm_proxy_label"
in out.columns):
        lab_src = out["rfm_proxy_label"]
        if lab_src.index.name != "session_id":
            if "session_id" in out.columns:
                lab_src = pd.Series(lab_src.values, index=out["session_id"].values)
            elif "SESSION_INDEX" in globals():
                lab_src = pd.Series(lab_src.values, index=pd.Index(SESSION_INDEX,
name="session_id"))
        label_series = lab_src.rename("label").astype(int, copy=False)

    # fallback to labels_dict
    if (label_series is None) and ("labels_dict" in globals()):
        rfm_keys = [k for k in labels_dict if
k.startswith("rfm_proxy_kmeans_session_")]
        if rfm_keys:
            k0 = sorted(rfm_keys)[0]
            if "sess" in globals():
               idx = sess.index
            elif "SESSION_INDEX" in globals():
               idx = pd.Index(SESSION_INDEX, name="session_id")
            else:
               idx = events["session_id"].drop_duplicates().astype("int64")
            label_series = pd.Series(labels_dict[k0], index=idx,
name="label").astype(int, copy=False)

   assert label_series is not None, "RFM session labels not found. Run Section
5.2 first."
   unit_lbl = unit_tbl.merge(label_series.rename_axis("session_id"),
                        left_on="unit_id", right_index=True, how="left")
else:
   # visitor-level: derive majority label across that visitor's sessions
   if "out" in globals() and isinstance(out, pd.DataFrame) and ("rfm_proxy_label"
in out.columns):
      sess_lab = out["rfm_proxy_label"]
      if sess_lab.index.name != "session_id":
         if "session_id" in out.columns:
            sess_lab = pd.Series(sess_lab.values, index=out["session_id"].values)
         elif "SESSION_INDEX" in globals():
            sess_lab = pd.Series(sess_lab.values,
index=pd.Index(SESSION_INDEX, name="session_id"))
   else:
      assert "labels_dict" in globals(), "labels_dict not found; run the RFM KMeans
block (5.2)."
      rfm_keys = [k for k in labels_dict if
k.startswith("rfm_proxy_kmeans_session_")]
      assert rfm_keys, "No RFM session labels found in labels_dict."
      k0 = sorted(rfm_keys)[0]
      if "sess" in globals():
         idx = sess.index
      elif "SESSION_INDEX" in globals():
         idx = pd.Index(SESSION_INDEX, name="session_id")
      else:
        idx = events["session_id"].drop_duplicates().astype("int64")
      sess_lab = pd.Series(labels_dict[k0], index=idx, name="rfm_proxy_label")

    sess_to_vis = (events[["session_id", "visitorid"]]
                  .drop_duplicates()
                  .set_index("session_id")["visitorid"])
    tmp = pd.DataFrame({
      "visitorid": sess_to_vis.reindex(sess_lab.index).values,
      "label": sess_lab.values
    }).dropna()
    vis_lab = (tmp.groupby("visitorid")["label"]
                 .agg(lambda s: s.value_counts().idxmax())
                 .astype(int))
    unit_lbl = unit_tbl.merge(vis_lab.rename_axis("unit_id").rename("label"),
                              left_on="unit_id", right_index=True, how="left")

# 7) Event shares (views / add-to-cart / transactions)
evt = events[["session_id", "visitorid", "event"]].copy()
evt["event"] = evt["event"].astype(str).str.lower()
evt["is_view"]        = (evt["event"] == "view").astype(int)
evt["is_addtocart"]   = (evt["event"] == "addtocart").astype(int)
evt["is_transaction"] = (evt["event"] == "transaction").astype(int)

if BASELINE_LEVEL == "session":
    e_agg = (evt.groupby("session_id")[["is_view","is_addtocart","is_transaction"]]
               .sum().reset_index().rename(columns={"session_id":"unit_id"}))
else:
    e_agg = (evt.groupby("visitorid")[["is_view","is_addtocart","is_transaction"]]
               .sum().reset_index().rename(columns={"visitorid":"unit_id"}))
e_agg["events"] = e_agg[["is_view","is_addtocart","is_transaction"]].sum(axis=1)

prof = unit_lbl.merge(e_agg, on="unit_id", how="left")

# proportions (safe for zero denominators)
denom = prof["events"].replace(0, np.nan)
prof["p_view"]        = (prof["is_view"]        / denom).fillna(0.0)
prof["p_addtocart"]   = (prof["is_addtocart"]   / denom).fillna(0.0)
```

```
prof["p_transaction"] = (prof["is_transaction"] / denom).fillna(0.0)

# 8) Cluster-level summary
summary = (prof
    .groupby("label", dropna=False)
    .agg(
        n=("unit_id", "count"),
        share=("unit_id", lambda s: len(s) / max(1, len(prof))),   # added share for
readability
        recency_days_median=("recency_days", "median"),
        frequency_median=("frequency", "median"),
        monetary_median=("monetary", "median"),
        events_mean=("events", "mean"),
        p_view_mean=("p_view", "mean"),
        p_addtocart_mean=("p_addtocart", "mean"),
        p_transaction_mean=("p_transaction", "mean"),
    )
    .reset_index()
    .sort_values(["n"], ascending=False)
)

# 9) Save profile & personas
summary_path = OUT_DIR / f"rfm_cluster_profile_{BASELINE_LEVEL}.csv"
summary.to_csv(summary_path, index=False)
summary.to_csv(PERSONA_OUT, index=False)

print("Baseline RFM profile (top rows):")
try:
    display(summary.head(10))
except Exception:
    print(summary.head(10).to_string(index=False))

print(f"Saved profile table : {summary_path}")
print(f"Saved personas to   : {PERSONA_OUT}")
```

9.2. Implementation Pipeline Code Output

All code output from last run have been placed on Google Drive, and can be accessed through following URL:

10. APPENDIX C

10.1. Streamlit Dashboard Code

```
# app.py

import streamlit as st
import pandas as pd
import plotly.express as px
from pathlib import Path

# 1. Configuration
DATA_DIR     =  Path(__file__).parent.parent
FIGS_DIR     = f"{DATA_DIR}/figs"
TABLES_DIR    = f"{DATA_DIR}/data"
```

```
SHAP_DIR      = f"{DATA_DIR}/shap"
SESSION_INDEX =
pd.read_csv(f"{TABLES_DIR}/20250827_195718_dataset_overview.csv")["sessio
ns"]  # adjust

st.set_page_config(page_title="Rocket Retail Segmentation", layout="wide")

# 2. Sidebar controls
st.sidebar.title("Controls")
model_list =
pd.read_csv(f"{TABLES_DIR}/20250827_195718_eval_internal_metrics.csv")["m
odel"].tolist()
lead_model = st.sidebar.selectbox("Select clustering model:", model_list)
show_shap  = st.sidebar.checkbox("Show SHAP explanations", value=False)
time_split = st.sidebar.checkbox("Show temporal drift", value=False)

# 3. EDA Summary
st.header("Dataset Overview")
overview =
pd.read_csv(f"{TABLES_DIR}/20250827_195718_dataset_overview.csv")
st.metric("Total Events", f"{int(overview.rows):,}")
st.metric("Total Sessions", f"{int(overview.sessions):,}")
st.metric("Unique Visitors", f"{int(overview.visitors):,}")

# Event distribution
evt = pd.read_csv(f"{TABLES_DIR}/event_distribution.csv")
fig_evt = px.bar(evt, x="event", y="count", title="Event Distribution")
st.plotly_chart(fig_evt, use_container_width=True)

# 4. Temporal Patterns
st.header("Temporal Patterns")
hr = pd.read_csv(f"{TABLES_DIR}/temporal_hour_event_counts.csv")
fig_hr = px.line(hr, x="hour", y="count", color="event", title="Hourly Event
Volume")
st.plotly_chart(fig_hr, use_container_width=True)
dow = pd.read_csv(f"{TABLES_DIR}/temporal_dow_event_counts.csv")
```

```
fig_dow = px.line(dow, x="dow_label", y="count", color="event", title="Weekly
Event Volume")
st.plotly_chart(fig_dow, use_container_width=True)

# 5. Session Funnel
st.header("Session Funnel")
funnel = pd.read_csv(f"{TABLES_DIR}/funnel_session_level.csv")
fig_funnel = px.bar(funnel, x="stage", y="sessions_reached",
text="sessions_reached",
               title="Session-level Funnel")
fig_funnel.update_traces(texttemplate="%{text:,}", textposition="outside")
st.plotly_chart(fig_funnel, use_container_width=True)

# 6. UMAP Projection
st.header("UMAP Visualization")
coords =
pd.read_csv(f"{TABLES_DIR}/20250827_195718_umap_all_models_coords.csv"
)

if lead_model == "rfm_proxy_kmeans_session_12":
   labels =
pd.read_csv(f"{TABLES_DIR}/20250827_195718_session_rfm_proxy_labels_k12
.csv")
   if "label" in labels.columns:
      labels = labels.rename(columns={"label": lead_model})
else:
   labels =
pd.read_csv(f"{TABLES_DIR}/20250827_195718_cluster_labels_all_models.csv"
)


df_umap = coords.merge(labels[["session_id", lead_model]], on="session_id")
fig_umap = px.scatter(df_umap, x="umap_x", y="umap_y", color=lead_model,
                title=f"UMAP – {lead_model}", width=800, height=600, opacity=0.6)
st.plotly_chart(fig_umap, use_container_width=True)
```

```
# 7. Internal Metrics Comparison
st.header("Model Evaluation Metrics")
metrics =
pd.read_csv(f"{TABLES_DIR}/20250827_195718_eval_internal_metrics.csv")
st.dataframe(metrics.style.format({"silhouette":"{:.4f}",
                                   "calinski_harabasz":"{:.0f}",
                                   "davies_bouldin":"{:.4f}"}))

# 8. Cross-Model Agreement (ARI / AMI)
st.header("Cross-Model Agreement")
ari =
pd.read_csv(f"{TABLES_DIR}/20250827_195718_eval_cross_model_ARI.csv",
index_col=0)
ami =
pd.read_csv(f"{TABLES_DIR}/20250827_195718_eval_cross_model_AMI.csv",
index_col=0)
st.subheader("Adjusted Rand Index (ARI)")
st.dataframe(ari)
st.subheader("Adjusted Mutual Information (AMI)")
st.dataframe(ami)

# 9. Temporal Drift
if time_split:
    st.header("Cluster Temporal Drift")
    drift =
pd.read_csv(f"{TABLES_DIR}/20250827_195718_eval_temporal_drift.csv")
    st.dataframe(drift.sort_values("abs_diff", ascending=False).head(10))

# 10. Segment Personas
st.header("Segment Personas")
persona =
pd.read_csv(f"{DATA_DIR}/20250827_195718_session_rfm_proxy_profile_k12.c
sv")
st.dataframe(persona.style.format({"share":"{:.2%}",
                                   "tx_rate":"{:.2%}",
                                   "atc_rate":"{:.2%}"}))
```

```
# 11. SHAP Explanations
if show_shap:
   st.header("SHAP Feature Importance")
   shap_feats =
pd.read_csv(f"{SHAP_DIR}/shap_sample_class_sizes_{lead_model}.csv")  #
example
   st.dataframe(shap_feats)
   shap_img = f"{FIGS_DIR}/{lead_model}_shap_beeswarm.png"  # adjust
naming
   st.image(shap_img, caption=f"SHAP Beeswarm – {lead_model}",
use_column_width=True)

# 12. Footer
st.markdown("---")
st.markdown("* 2025 Rocket Retail Segmentation Dashboard")
```

10.2. **Streamlit Dashboard URL:** https://rocketretaildashboard.streamlit.app/