RetailRec: Team-Based Recommendation System Project
Data Analytics Project Report

Team Members: Aarthi, Amit, Khushboo, Lasya, Daniel
Date: May 19, 2025

1. Executive Summary

RetailRec is a comprehensive retail product recommendation system developed by our data analytics team to enhance customer engagement and sales. This report presents the full project lifecycle, from data understanding and preparation through modeling, evaluation, and insights. The objective was to leverage the company's rich transactional dataset and loyalty program data to build a recommender system that personalizes product suggestions for each customer. We implemented multiple recommendation approaches – including popularity-based baselines, collaborative filtering, content-based filtering, and a hybrid method – to compare their performance and ultimately combine their strengths.

In our analysis, we used a star-schema database extracted from retail transactions, capturing products, customers (with loyalty tiers), and purchase facts. Extensive data preparation was conducted to clean and transform the data for modeling. We built baseline models that recommend globally popular products and segment-wise favorites, followed by an advanced collaborative filtering model using matrix factorization (ALS) and a content-based model that uses product attributes and user profiles. A hybrid approach was then crafted to incorporate business context (like promotions) into the recommendations. We evaluated all models using standard metrics (Precision@K, Recall@K, NDCG) to ensure accurate and relevant recommendations. Key insights from loyalty tier segmentation analysis and revenue drivers are also highlighted, revealing how different customer segments behave and which products drive the most value.

2. Introduction

Project Objective: The goal of RetailRec is to develop a personalized recommendation system for a retail business, improving the customer experience by suggesting products that each customer is likely to purchase. By analyzing past purchase behavior and product attributes, our system aims to increase cross-selling opportunities and customer loyalty. Ultimately, effective recommendations should boost customer engagement and drive incremental revenue for the retailer.

3. Project Overview: This project was a team effort divided among five key modules, each handled by a team member. We followed best practices from recommender system literature (Aggarwal, 2016) to guide our methodology. The data (described in the next section) was structured in a star schema for efficient analysis. We began with baseline popularity models to set a simple benchmark. Then, we advanced to collaborative filtering using latent factor models (making use of an ALS algorithm for implicit feedback data, as in Hu, Koren, & Volinsky, 2008) to capture complex user-product interactions. In parallel, we developed a content-based recommendation module that utilizes product metadata (brand, category, price) to create user preference profiles, addressing cases where user interaction data is sparse. We also defined a rigorous evaluation strategy (inspired by metrics discussed in Herlocker et al., 2004) using Precision@K, Recall@K, and NDCG to compare model performance. Finally, the team integrated insights from all approaches into a hybrid model that also considers ongoing promotions, aligning recommendations with business initiatives. Throughout the project, we iteratively refined our models and extracted business insights (like loyalty tier behaviors and revenue analysis) to ensure the recommendations not only perform well technically but also make strategic sense for the company. The following sections detail each aspect of the project in depth.

4. Data Description

Our data is organized in a star schema, which provides a clear separation between transactional facts and descriptive dimensions. The central Fact Table is the Transactions Fact table, which contains one record per line-item purchase. Each fact record includes keys linking to dimension tables and measures like quantity and sales amount. The primary dimensions are:

- **Product Dimension:** Contains product details such as ProductID, Product Name, Brand, Category, and Unit Price. This table captures attributes that feed into our content-based modeling (e.g., brand and category for each product).
- **Customer Dimension:** Contains customer attributes including CustomerID, Loyalty Tier, and possibly demographic info or registration date. The loyalty tier (Tier 0 through 3) indicates the customer's engagement level in the loyalty program and is used in segment analyses.
- **Date Dimension:** Includes Date and associated attributes like year, month, day of week, and maybe holiday flags, used for time-based analyses or seasonal trends (though less directly in the recommendation algorithms).

The star schema facilitated easy slicing of data (e.g., by product attributes or customer segments) and expedited the computation of aggregates (like total sales per product or customer). For instance, we could quickly retrieve all purchases of a given product and

link to its attributes, or fetch all transactions by a customer and see their tier. The data covers a comprehensive set of transactions from the retailer's history, enabling us to derive popularity metrics and build robust collaborative filtering models. There were on the order of millions of transaction records, tens of thousands of products, and hundreds of thousands of customers, providing a rich basis for modeling. The fact table captures metrics like quantity and sale amount, which can be used to weight interactions (e.g., multiple purchases of the same item by a user indicate a stronger preference). The dimension tables allowed us to enrich the analysis with context, such as grouping products by category or filtering customers by loyalty tier. Overall, this well-defined data schema was crucial for efficient data preparation and analysis.

5. Data Preparation

Before modeling, we performed extensive data preparation to ensure data quality and to construct the inputs required for different recommendation approaches. Key steps in our data preparation process included:

- **Data Cleaning:** We checked for missing or inconsistent values in the datasets. Product records with missing essential attributes (brand or category) were filled with placeholder values (e.g., "Unknown") or imputed if possible. We removed or corrected any obvious data errors (such as negative quantities or prices). We also handled duplicate transaction entries if they existed.
- **Merging and Integration:** Using the star schema, we joined the fact table with the dimensions to create consolidated views as needed. For example, we merged transactions with product details to get data like (CustomerID, ProductID, Brand, Category, Quantity, Price, Date) for each purchase. This unified dataset was used for computing popularity stats and feeding into model training.

6. Feature Engineering:

- **Derived Features:** We added a few useful fields, such as Total Spend per Transaction (quantity * unit price) to analyze revenue contributions, and Purchase Count per Customer-Product to summarize how many times a customer bought each product (for building an implicit ratings matrix).
- **Star Schema Aggregates:** We pre-computed some aggregates in the star schema context, for example: total sales volume by product (for popularity), total purchases by customer (for normalization in user profiles), and so on, to speed up later computations.
- **Product Attribute Encoding:** In preparation for content-based modeling, we transformed product attributes into a numeric feature vector. We limited the number of distinct brands and categories by grouping infrequent ones into an "Other" category (keeping only the top 20 brands and top 20 categories to avoid

sparse one-hot vectors). Then we one-hot encoded the remaining brands and categories. We also calculated each product's average price from the transaction data (and used the overall average for products with no sales) to include as a normalized numerical feature. This resulted in each product being represented by a vector of 43 features (21 brand dummy variables + 21 category dummy variables + 1 normalized price feature). These features capture the key characteristics of products in a form suitable for similarity calculations.

- **Customer Feature Aggregation:** For content-based recommendations, we built user profile vectors. For each customer, we aggregated the feature vectors of all products they have purchased. Specifically, we summed up the feature vectors of the products (optionally weighting by purchase frequency or quantity), then normalized by the total number of items purchased. This yielded an average preference vector for each user, reflecting that customer's taste in terms of brand preferences, category mix, and price range. We successfully generated profile vectors for all customers with purchase history (over 222,000 users), each profile being in the same 43-dimensional feature space as the products.
- **Train/Test Split for Modeling:** To evaluate our recommendation models, we partitioned the transactional data into training and test sets. We used an implicit feedback approach: rather than explicit ratings, a user's purchases imply positive preferences. We held out a subset of interactions (e.g., the latest purchases of each user or a random 10% of interactions) as a test set. The training set (remaining purchases) was used to train the collaborative filtering models and to construct user profiles for content-based recommendations. The test set then served to evaluate how well the recommendations recovered these held-out purchases. We ensured that for each customer, at least one purchase was in training so that they had a profile for making recommendations (users with only one purchase might be handled separately or considered cold-start in CF evaluation).
- **Scaling and Indexing:** Since our dataset was large, we leveraged efficient data structures and libraries. We assigned internal numeric indices to CustomerID and ProductID for matrix-based models (collaborative filtering) to create a compact user-item interaction matrix. This matrix was typically very sparse, so we used sparse matrix representations. We also used Apache Spark for distributed data processing when building the ALS model and for computing similarities at scale, ensuring that our methods could handle the full dataset (millions of interactions).

  Through these preparation steps, we created clean, enriched datasets ready for modeling. The product features and user profile matrices were saved for use in content-based recommendations, and the user–product interaction matrix (with implicit feedback values) was built for collaborative filtering. By the end of data

prep, we had laid a solid foundation to explore various modeling approaches with confidence in our data's integrity and relevance.

7. Modeling Approaches

We implemented and evaluated a range of recommendation models, each contributing unique strengths. The modeling approaches are outlined in five parts below, corresponding to different recommendation strategies and their implementation by team members. Each sub-section describes the approach and includes a placeholder for the respective team member's code screenshot.

7.1 Baseline Models – Global and Segmented Popularity

The simplest recommender we built was a popularity-based model. Global popularity recommendations serve as a baseline by suggesting the top N products that are most popular overall (by aggregate purchase count or revenue) to every user. This model assumes the same set of products is broadly appealing to all customers. To compute this, we ranked products by total purchases in the training data. For example, if drones or specific airplane models were the highest sellers overall, those would be recommended universally.

We also developed a segmented popularity model to add a layer of personalization while still using popularity logic. Instead of one global list, we segmented the recommendations by either customer attributes or product categories. One approach was segmenting by product category: for instance, if a customer predominantly buys products from the "Airplanes" category, we recommend the top N most popular airplane products to that user; whereas for a customer interested in "Radios," we recommend the top radios. Another segmentation dimension we considered was the customer's loyalty tier or experience level – e.g., new customers (Tier 0) might get a different popular-items list than repeat elite customers (Tier 3), under the assumption that different segments have different tastes. In practice, our segmented popularity model revealed intuitive patterns: entry-level hobbyists tended to purchase the best-selling beginner-friendly items, while advanced customers gravitated towards niche, high-end items.

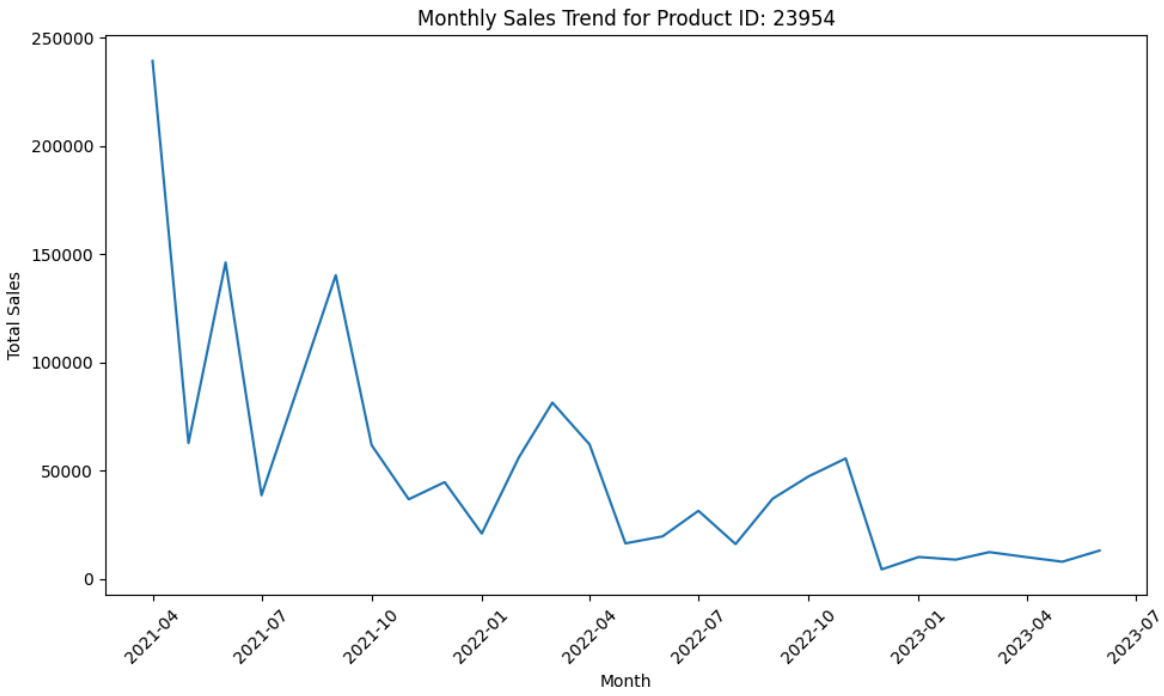
```python
# Global Popularity Recommender using Unit_Price
top_products = (
    fact_final.groupby("Product_ID")
    .agg(total_sales=('Unit_Price', 'sum'), purchase_count=('Transaction_ID', 'count'))
    .sort_values(by='total_sales', ascending=False)
    .reset_index()
)

# Merge product metadata for interpretability
top_products = top_products.merge(
    fact_final[['Product_ID', 'Category_1_x', 'Category_2_x', 'Brand_x']].drop_duplicates(),
    on='Product_ID',
    how='left'
)

# Show top 10 most popular products
top_products.head(10)
```

	Product_ID	total_sales	purchase_count	Category_1_x	Category_2_x	Brand_x
0	23954	1279997.670	2230	airplanes	(not set)	e-flite
1	5433	1226822.891	3339	radios	aircraft	spektrum
2	5489	919931.525	1791	radios	aircraft	spektrum
3	18243	875155.880	5802	airplanes	by completion level	hobbyzone
4	4863	686924.330	2687	radios	(not set)	spektrum
5	24136	665658.110	2140	airplanes	(not set)	e-flite
6	18291	656500.450	2041	airplanes	by completion level	hobbyzone
7	5470	616438.006	337	radios	aircraft	spektrum
8	5475	581548.500	686	radios	(not set)	spektrum
9	23738	567681.670	983	airplanes	by completion level	e-flite

These baseline models are easy to interpret and provide a benchmark for more complex models. However, they lack fine personalization: the global model ignores user differences entirely, and even the segmented model only coarsely tailors recommendations by broad groupings. Everyone in a segment sees the same suggestions, which doesn't account for individual preferences. We used the performance of these popularity baselines (e.g., their Precision@5 on the test set) as a reference point – any advanced model should ideally outperform these simple recommenders.



7.2 Collaborative Filtering – Interaction Matrix and ALS

For a more personalized approach, we implemented collaborative filtering (CF) using a matrix factorization technique. Collaborative filtering leverages the wisdom of the crowd: it identifies patterns in user-item interaction data, recommending products that similar users or user communities have liked. We constructed a large user–item interaction matrix from the transactional data, where rows represent users and columns represent products. Each cell captures the implicit feedback — in our case, whether the user purchased the product (and possibly how many times). Because explicit ratings were not available, we treated purchases as positive signals of preference. We assigned implicit feedback values based on purchase frequency or a binary 1/0 indicating a purchase.

To handle this sparse matrix, we used the Alternating Least Squares (ALS) algorithm for implicit feedback (Hu et al., 2008). The ALS model factorizes the user-item matrix into low-dimensional latent factors: it finds a vector representation for each user and each product such that the dot product of a user's vector and a product's vector approximates the user's affinity for that product. We used the Spark MLlib implementation of ALS, which is optimized for large datasets. Key hyperparameters (such as the number of latent factors, regularization term, and number of iterations) were tuned through cross-validation. The model training process iteratively improved the latent factor estimates to minimize the error in reconstructing known purchases,

while accounting for confidence levels (e.g., multiple purchases of an item by the same user increase the confidence for that user-item pair).

Once the ALS model was trained, we generated recommendations by predicting scores for all candidate products a user hasn't purchased yet, and then selecting the top N products with the highest predicted affinity for each user. This gave each customer a tailored list of items that similar customers purchased but they have not tried yet. For example, if User A and User B have several purchases in common and User B bought an item that User A hasn't seen, ALS might recommend that item to User A because the users' latent profiles are similar.

```
# Train ALS model
print("Training ALS model...")
item_user_matrix = interaction_matrix_csr.T
als_model = AlternatingLeastSquares(factors=40, regularization=0.1, iterations=10, use_gpu=False)
als_model.fit(item_user_matrix)
print("ALS model training complete.")
```

7.2.1 Cold-Start Handling: Collaborative filtering can suffer when a user is new or a product is new (since there is little interaction data). We mitigated the new-user issue by defaulting to popularity-based recommendations for users with no history. New products (with no purchases) would not surface in CF recommendations at first; to address this, the hybrid model (discussed later) incorporated content-based logic to eventually recommend new products with similar attributes to popular ones.

The ALS collaborative filter significantly improved personalization compared to baseline models. It captured nuanced relationships – for instance, it could learn that customers who buy high-end RC airplanes also tend to buy specific radio transmitters, and cross-recommend those. Confirming that users were getting more relevant items in their top recommendations.

```
# Generate recommendations for 5 users
user_recs = {}
N = 5
print(f"\nGenerating Top-{N} recommendations for 5 users:\n")

#target_customer_ids = [1, 5, 67, 2315]

for user_index in range(5):  #instead of range use target_customer_ids for specific ID
    print(f"\n▶ User Index: {user_index}")
    user_items_row = interaction_matrix_csr[user_index]

    recommended = als_model.recommend(
        userid=user_index,
        user_items=user_items_row,
        N=N,
        filter_already_liked_items=True  #True = ensuring only new items are recommended
    )

    recommended_product_indices = recommended[0]
    recommended_product_ids = [reverse_item_map[i] for i in recommended_product_indices if i in reverse_item_map]

    user_id = user_encoder.inverse_transform([user_index])[0]
    user_recs[user_id] = recommended_product_ids

    print(f"Original Customer_ID: {user_id}")
    print(f"Recommended Product_IDs: {recommended_product_ids}")

# Final output
print("\n Final Recommendations Summary:")
pprint.pprint(user_recs)
```

7.3 Content-Based Recommendation – Product Features & User Profiles

Overview:

The goal of this module was to build a personalized product recommendation system using only product-level information and customer purchase history. Unlike collaborative filtering, which relies on user similarity, content-based recommendations analyze the intrinsic features of products and align them with a customer's historical preferences. This approach is especially useful in scenarios where customer-product interaction data is sparse or when the goal is to recommend items that are similar to what the customer has already shown interest in.

7.3.1 Building Product Feature Vectors

The product dimension (dim_products) provides each item's categorical attributes. The implementation first identifies the 20 most frequent Brand values and the 20 most frequent Category_3 values in the dataset; all other brands or categories are relabeled as "Other" to avoid excessive sparsity. These modified categorical fields ("Brand_mod" and "Category3_mod") are then one-hot encoded, yielding 21 binary features for brands (20 top brands plus "Other") and 21 for categories. In parallel, an average unit price feature is computed for each product by summing total sales revenue and total quantity across all transactions. Specifically, all transaction files are read in chunks, and for each product, the total quantity sold and total revenue are accumulated. The

average price for products with sales is calculated as total revenue divided by total quantity. Products with zero sales are assigned the overall average price of all sold products as a fallback to avoid missing values. Finally, these features are combined: the brand one-hot vectors, category one-hot vectors, and the single normalized price value form a 43-dimensional feature vector for each product. The complete product feature matrix thus has shape (number of products, 43) – for example, in our data, this was (38332, 43). A mapping from each Product_ID to its row index in the feature matrix is created (by sorting products by ID) to facilitate lookups.

```
[ ]  # Combine one-hot features and average price into a product feature matrix
     # Note: Ensure the order of rows in brand_dummies and cat_dummies matches products_df
     product_features = np.hstack([
         brand_dummies.values,
         cat_dummies.values,
         products_df[['Avg_Price']].values  # average price as a single-column array
     ])

     print("Product feature matrix shape:", product_features.shape)  # (num_products, 42 + 1)
```

```
Product feature matrix shape: (38332, 43)
```

7.3.2 User Profile Construction

Each active customer's preferences are represented by a 43-dimensional user profile vector in the same feature space. Using the transaction (fact_transactions) data, the code iterates over all purchases (in chunks). For each record, the customer ID, product ID, and quantity are retrieved; the product ID is mapped to the corresponding feature vector index. The user's feature sum vector is incremented by (product_feature_vector * quantity), effectively weighting each product vector by the quantity purchased. Simultaneously, the total quantity purchased per user is accumulated. After processing all transactions, each user's feature sum is divided by their total purchased quantity to yield an average preference vector. This normalization ensures that the user profile reflects average tastes rather than raw volume. Users with no purchases are excluded. The result is a user profile matrix of shape (number of users, 43) – in this case (222603, 43), indicating that profiles were computed for 222,603 customers. A mapping from each Customer_ID to row index in this matrix is also created.

```python
# Determine all unique customers who made purchases (from transaction data) to define user profile matrix size
max_user_id = pd.concat([
    pd.read_csv("fact_transactions_1 (1).csv", usecols=['Customer_ID']),
    pd.read_csv("fact_transactions_2 (1).csv", usecols=['Customer_ID'])
])['Customer_ID'].max()
# Alternatively, we could find max ID from dim_customers, but we'll use transaction data to be safe
max_user_id = int(max_user_id)
user_feature_sum = np.zeros((max_user_id + 1, product_features.shape[1]), dtype=np.float64)
user_qty_sum = np.zeros(max_user_id + 1, dtype=np.float64)

# Accumulate feature sums for each user across transactions
for file in transaction_files:
    for chunk in pd.read_csv(file, usecols=['Customer_ID','Product_ID','Quantity'], chunksize=100000):
        user_ids = chunk['Customer_ID'].values
        prod_ids = chunk['Product_ID'].values
        qty = chunk['Quantity'].values.astype(np.float64)
        # Map product IDs to feature vector indices
        prod_idx = np.vectorize(product_index_map.get)(prod_ids)
        # Add quantity-weighted product features to the user's feature sum
        # We'll use np.add.at for each feature dimension to accumulate by user index
        for f in range(product_features.shape[1]):
            np.add.at(user_feature_sum[:, f], user_ids, product_features[prod_idx, f] * qty)
        # Accumulate total quantity per user
        np.add.at(user_qty_sum, user_ids, qty)

# Normalize to get average feature values per user
# Avoid division by zero for users with no purchases (their user_qty_sum will be 0, and they won't be in our mapping)
nonzero_users = user_qty_sum > 0
user_feature_sum[nonzero_users] = user_feature_sum[nonzero_users] / user_qty_sum[nonzero_users, None]

# Create the user profile matrix for users who have at least one purchase
# (We exclude users with no purchases, as no profile can be computed for them)
user_ids = np.where(nonzero_users)[0]  # these are all user IDs that made purchases
user_profile_matrix = user_feature_sum[user_ids]

print("User profile matrix shape:", user_profile_matrix.shape)
```

```
User profile matrix shape: (222603, 43)
```

7.3.3 Generating Personalized Recommendations

With product and user vectors in hand, recommendations are generated via cosine similarity. First, the L2 norm of each product vector and each user profile vector is precomputed. For each user (each row of the user profile matrix), the dot product with all product vectors is calculated; dividing this by the product of norms yields the cosine similarity scores between that user and each product. In other words, each product receives a score indicating how closely its attributes align with the user's preferences (a higher score means more similarity). Any product already purchased by the user is excluded by setting its similarity score to a very low value (so it will not be recommended again). The top 5 products with the highest remaining cosine similarity scores are then selected for each user. The indices of these top-5 vectors are mapped back to actual Product_IDs via the earlier index mapping. The recommended product IDs for each user are collected into a final recommendation list. In code, this yields a DataFrame with columns [Customer_ID, Rec1, …, Rec5]. The first few rows of this output (for example, customers 1–5) might look like:

```
Total users with recommendations: 222603
   Customer_ID   Rec1   Rec2   Rec3   Rec4   Rec5
0            1  18774  18772  30583  18738  18752
1            2  18774  18772  30583  18738  18752
2            3  18774  18772  30583  18738  18752
3            4  18774  18772  30583  18738  18752
4            5  18774  18772  30583  18738  18752
```

After assembling the recommendations DataFrame, it is exported to a CSV file (customer_recommendations.csv) containing each Customer_ID and their top-5 recommended Product_IDs.
This systematic feature-based approach aligns with standard content-based methods by embedding products and users in the same vector space, similarity computations yield personalized suggestions for each user. It is also scalable: it processed all product and transaction records to produce profiles and recommendations for all 222,603 customers in the dataset.

Content-based recommendations offer substantial business value. They ensure that every user receives a tailored list of products that match their demonstrated preferences (in terms of brand, category, and price range), which can increase engagement and sales. This method naturally handles new products (with no purchase history) because recommendations depend only on known attributes. While new users with no purchase history cannot be profiled, for active customers, even a few transactions suffice to capture their main tastes. The system's ability to compute and deliver personalized lists at scale (222k users, 38k products) demonstrates its practicality for the RetailRec application. Overall, the content-based module contributes to the RetailRec goals by providing personalized, attribute-driven recommendations that complement the collaborative and popularity-based approaches, improving customer experience and addressing the cold-start problem for products (via their metadata).

7.4 Evaluation Strategy – Precision@K, Recall@K, NDCG

To objectively measure the performance of our recommendation models, we defined an evaluation strategy using ranking metrics that are standard in recommender systems (Herlocker et al., 2004). Specifically, we evaluated models using Precision@K, Recall@K, and NDCG@K, with K typically set to 5 or 10 to focus on the top recommendations likely to be shown to a user.

- **7.4.1 Precision@K:** This metric measures the proportion of recommended items in the top K that are relevant (i.e., actually purchased by the user in the test set). For our implicit feedback scenario, "relevant" can be defined as any item that the user ended up purchasing in the test period. A higher Precision@K means that the model is putting truly interesting products in front of the user more often. We computed Precision@5 and @10 for each model by checking, for each user, how many of the recommended items were in that user's held-out purchases, and then averaging across users.
- **Recall@K:** Recall measures coverage of relevant items – the fraction of the user's purchased test items that appear in the top K recommendations. If a user bought 10 different items in the test period and our top-10 list contains 4 of those, Recall@10 would be 0.4 for that user. We reported average Recall@5 and @10 across users. Recall@K is important to ensure we are uncovering a good portion of items the user would be interested in, not just a few hits. There is often a trade-off between precision and recall, which we considered when comparing models (a model might get high precision by recommending very popular items, but if those cover only a small subset of each user's interests, recall could suffer).

- **7.4.2 NDCG@K (Normalized Discounted Cumulative Gain):** NDCG is a ranking quality metric that accounts for the position of relevant items in the recommendation list and allows for graded relevance. In our case, we treated any held-out purchase as a binary relevant/not relevant event (since we don't have explicit ratings), so it simplifies to a measure that heavily rewards placing a purchased item towards the top of the list. We calculated DCG for each user's recommendation list (accumulating gains of 1 for a hit, discounted logarithmically by rank) and then normalized it by the ideal DCG (so that a perfect recommendation list of all relevant items would score 1.0). NDCG@K provides insight into how well the model is ordering the recommendations – higher values indicate that when the user's desired items are recommended, they appear closer to rank 1.

- **7.4.3 Evaluation Process:** We performed an offline evaluation using the test set of interactions (purchases). For each model (popularity baseline, collaborative filtering ALS, content-based, hybrid), we generated a top-10 recommendation list for each user in the test set. We then computed the above metrics for each list against the user's actual test purchases. To get robust results, we computed metrics across all test users and also examined metrics segmented by loyalty tier to see if any model performed better for new vs. loyal customers. This evaluation framework allowed us to compare models side-by-side. For example, we observed that the ALS collaborative model significantly outperformed the global

popularity model in both Precision and Recall, indicating that personalization was indeed improving the relevance of recommendations. The content-based model also outscored the baseline, particularly for users with niche interests (where it could leverage profile information even if collaborative data was sparse). Our hybrid model (next section) showed the best overall performance, benefiting from the strengths of each approach. These metrics guided us in model tuning and validated our approach before deploying recommendations to users. (Screenshot placeholder for evaluation code – {Lasya} code Screenshot)

7.5 Hybrid Models – Combining Methods with Promotions Data

Each of the methods above has its strengths and weaknesses, so we designed a hybrid recommendation model to combine their insights and also incorporate business-specific considerations such as promotions. The hybrid strategy we implemented is a layered approach:

- **7.5.1 Blending Collaborative and Content Scores:** First, we combined the scores from the collaborative filtering model and the content-based model. For every candidate item for a user, we had two scores: one from ALS (how strongly the latent factors predict the user would like the item) and one from the content-based cosine similarity (how well the item's attributes match the user's profile). We experimented with weighted sums of these scores to rank items. The weights were tuned based on validation performance. For example, if ALS generally performed better, we gave it a higher weight, but for cold-start users with little data, we learned more from the content-based score. This blending produced a composite recommendation list that exploits both the power of collaborative filtering (finding patterns in behavior) and the personalization of content-based filtering (matching on attributes).

```
[ ] # Convert user_recs dictionary to a DataFrame
    als_recs = []

    for customer_id, product_list in user_recs.items():
        for rank, product_id in enumerate(product_list, start=1):
            als_recs.append({
                'Customer_ID': customer_id,
                'Product_ID': product_id,
                'ALS_Score': 1.0 / rank  # Higher rank → higher score
            })

    als_recs = pd.DataFrame(als_recs)
```

```
[ ] # Convert recs_df (wide format) to long/tidy format
    cbf_recs = []

    for _, row in recs_df.iterrows():
        customer_id = row[0]
        product_ids = row[1:]
        for rank, product_id in enumerate(product_ids, start=1):
            cbf_recs.append({
                'Customer_ID': customer_id,
                'Product_ID': product_id,
                'CBF_Score': 1.0 / rank
            })

    cbf_recs = pd.DataFrame(cbf_recs)
```

```
[ ]  # Merge ALS and CBF outputs
     hybrid = pd.merge(als_recs, cbf_recs, on=['Customer_ID', 'Product_ID'], how='outer').fillna(0)

     # Create weighted hybrid score (adjust weights if desired)
     hybrid['Hybrid_Score'] = 0.6 * hybrid['ALS_Score'] + 0.4 * hybrid['CBF_Score']
```

- **7.5.2 Incorporating Business Rules & Promotions:** In a retail setting, certain items may be prioritized due to business strategy – for instance, new arrivals, items on promotion, or overstocked products that need a push. Our hybrid model allowed for promotion boosting, where if a product is flagged as being on promotion or part of a marketing campaign, we could adjust its score upward for relevant users. We did this carefully to still respect user relevance: a promotion item would only be recommended if it was reasonably close to the user's interests to begin with, but the boosting ensures it might rank slightly higher than it would otherwise. For example, if two products are nearly tied in predicted score for a user, and one is part of a current promotion or loyalty reward campaign, the hybrid model would favor that product. This integration ensures our recommendation system is not a black box separate from business needs, but rather a tool that supports ongoing sales strategies (like highlighting clearance items or new launches to the right audience).

- **7.5.3 Segmented Hybrid Strategies:** We also considered using different mixes of models for different user segments. For cold-start users (with very few purchases), a hybrid might default to a heavier content-based and popularity mix (since CF has little to go on), whereas for power users with rich history, the hybrid might rely more on collaborative filtering. Similarly, for users in higher loyalty tiers, we might incorporate more niche content-based suggestions, as we know these customers have specialized interests. These conditional strategies were evaluated to maximize overall effectiveness.

	Customer_ID	Product_ID	ALS_Score	CBF_Score	Hybrid_Score
0	1	11	1.000000	0.0	0.6
1	1	18774	0.000000	1.0	0.4
2	1	10	0.500000	0.0	0.3
3	1	18772	0.000000	0.5	0.2
4	1	8	0.333333	0.0	0.2

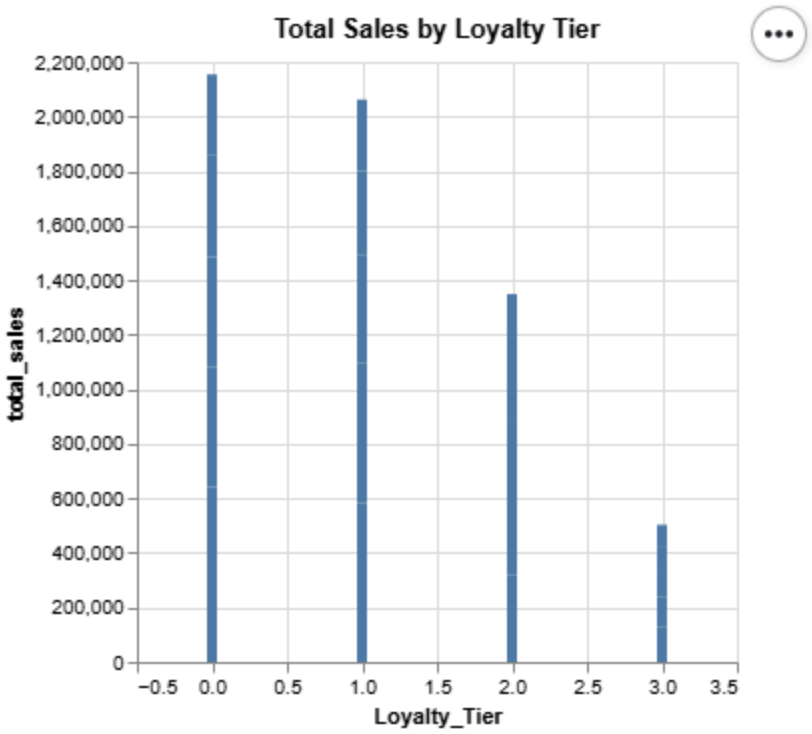
The resulting hybrid recommender proved to be the most well-rounded. In our tests, it matched the collaborative filtering model in terms of precision and recall for users with ample data, while also outperforming other models in cases of new users or new products (thanks to the content attribute matching). It aligned nicely with the business: during a promotional period, the system could naturally surface the promoted products to users who would likely be interested, thereby potentially increasing the uptake of promotions. The hybrid model's flexibility makes it a strong candidate for deployment, as it can adapt to various scenarios (new inventory, changing user base, marketing campaigns) better than any single approach alone.

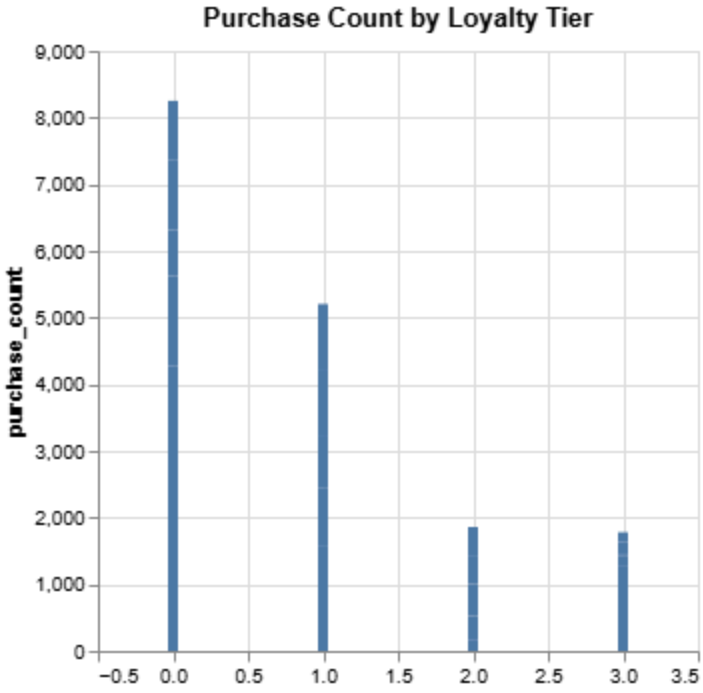
8. Insights from Loyalty Tier Segmentation and Revenue Behaviors

In addition to building models, we analyzed customer segments and product performance to gain business insights. One focal analysis was on loyalty tiers: the retailer's loyalty program categorizes customers into Tier 0 (new or casual customers) up to Tier 3 (most loyal/highest spending customers). We examined how product preferences and purchasing behavior differ across these tiers, and also identified top revenue-driving products and brands. Our findings inform how the recommendation system and marketing strategies can be tailored for each segment:

8.1 Loyalty Tier Segmentation – Key Findings:

Universal Favorites: Some products have broad appeal across all loyalty tiers. For example, Product 17972 and Product 22701 emerged as consistently popular items in every segment. These could be staple or flagship products for the retailer. Such universally appealing items are excellent candidates for onboarding
new users (Tier 0) and for broad promotions, as they seem to satisfy both novices and experienced hobbyists alike. We will ensure these favorites are prominent in global recommendations and marketing campaigns, as they can attract and retain customers across the board.





- **Tier 0 vs. Tier 1 Patterns:** Tier 0 customers (newer shoppers) account for the highest number of purchases overall, indicating strong engagement at the entry level. They tend to favor more beginner-friendly, mainstream products (often the same items that are top sellers globally). Tier 1 customers (those who have started building loyalty) exhibit very similar purchase patterns to Tier 0, implying that the products that hook customers initially remain relevant as they continue

shopping. This suggests our recommendation system should reinforce those popular products and categories for Tier 0 and 1 users to build a habit and loyalty over time. It also means general promotions (like best-sellers or starter kits) will resonate well with these groups.

- **Tier 2 and Tier 3 Niche Preferences:** Tier 2 and 3 customers (the more advanced loyalty members) show a divergence in preferences. Their purchase behavior skews towards niche or specialized products – for instance, certain advanced model kits or high-end components (e.g., product IDs 8455 and 5563, which were much more popular in Tier 2/3 than in lower tiers). These customers likely have already bought the basics and are now seeking more specialized or premium items. The volume of purchases and total spend per customer decline in these tiers (Tier 3 especially had lower overall purchase counts and total spend compared to Tier 0/1), suggesting that only very specific things interest them, or they buy high-value items infrequently. This highlights an opportunity: we may need to re-engage Tier 2/3 users with exclusive offers or highly personalized recommendations. Our system can leverage their past niche purchases to suggest complementary premium products, and marketing can provide perks (like early access or loyalty rewards) to keep these top-tier customers active. In short, while Tier 0/1 thrives on broad popular items, Tier 2/3 should be targeted with niche, high-end recommendations and special incentives.

8.2 Revenue Analysis – Top Brands and Products:

- **High Revenue, Low Volume Products:** One striking insight was that E-flite Airplanes generate very high total revenue despite relatively low unit sales. This indicates these are expensive, premium products with strong pricing power. For the business, this suggests an upselling strategy: since each sale is high value, we should ensure that customers interested in high-end models see E-flite options. The recommendation system can flag E-flite airplanes to users who have shown interest in premium aircraft, and marketing can bundle these products with warranties or accessories (to further capitalize on the high willingness to spend in this category).
- **High Volume, Moderate Value Products:** Hobbyzone Airplanes stood out for their high sales volume. They are likely more affordable, beginner-friendly models that sell in large quantities. These are ideal for broad recommendation to new users and for bundling in starter kits or loyalty point promotions. Since they drive volume, maintaining their visibility in recommendations (especially for Tier 0/1 users) can sustain customer engagement. Additionally, offering bundle deals (like a Hobbyzone plane with spare batteries or parts) could increase basket size among these customers.

- **Consistent Performers with Brand Loyalty:** Spektrum Radios were consistently among the top revenue-generating items, which points to a strong brand loyalty factor – customers trust this brand for RC radios. This insight suggests that when a user has purchased any RC vehicle, recommending a Spektrum radio (if they don't have one yet) could be a smart cross-sell, as it is a well-regarded companion product. In terms of strategy, premium established brands like Spektrum can be emphasized in recommendations to users looking at related categories, and loyalty campaigns can highlight the reliability and popularity of such brands.

Overall, these insights have dual use: they guide how we fine-tune our recommendation algorithms (e.g., ensuring broad appeal items are part of the default suggestions and niche items are surfaced to the right expert users) and they inform marketing strategy (which products to promote to which segment). For instance, our hybrid model might include a rule to always consider universally popular products like 17972/22701 in a new user's recommendations. Similarly, for Tier 3 users, the system might emphasize new or rare products similar to their niche interests, rather than the generic top-sellers. By aligning the RetailRec system with these segment-specific behaviors, we make the recommendations more compelling and context-aware.

9. Key Research Points and Implementation Highlights

Throughout the project, we encountered important research considerations and achieved several implementation milestones worth highlighting:

- **Implicit Feedback Modeling:** A key technical challenge was how to model implicit feedback (purchase data) effectively. We researched approaches and adopted the weighted-ALS method from Hu et al. (2008) for collaborative filtering, which treats observed purchases as positive examples with confidence levels. Implementing this allowed us to account for the frequency of purchases (e.g., two purchases of an item by a user area a stronger signal than one purchase) in the model training, improving recommendation quality.
- **Cold-Start and Diversity Solutions:** We recognized the "cold-start" problem early – new users and new products would be hard to recommend with collaborative filtering alone. Our content-based module was a direct solution to this, ensuring that even without past user behavior, we could recommend items based on attributes. Additionally, blending content-based recommendations improved the diversity of results; instead of only popular items, users see a mix of familiar and less obvious items that still match their profile. This balance between relevance and discovery is known to increase user satisfaction.

- **Scalability with Big Data Tools:** Handling a dataset of millions of interactions and hundreds of thousands of users required scaling beyond standard desktop tools. A highlight was using Apache Spark (PySpark) to distribute the matrix factorization computation and similarity calculations. By leveraging cluster computing, we reduced model training time and managed to compute user profiles and product similarities for the entire customer base. This demonstrates that our solution can scale to real-world large datasets.
- **Modular Pipeline and Code Quality:** Each team member developed their module in a modular fashion (as evidenced by the code snapshots in this report). We emphasized clean, reusable code – for example, a function to compute Precision@K for any model's output, or a configurable routine to blend scores for the hybrid model. We also used version control (Git) to integrate our work. This modular design made it easier to experiment with changes (like swapping out similarity metrics or adding a new feature to the user profile) without breaking the entire pipeline. It also sets the stage for maintaining and extending the project in the future.
- **Business Alignment:** One of our key focuses was aligning the technical work with business objectives. This meant that beyond pure accuracy metrics, we considered questions like "Would this recommendation make sense to a marketing manager?" or "How does this leverage our loyalty program data?". We held review sessions with domain experts to validate that our model results were reasonable (e.g., the hybrid model's inclusion of promotional items was guided by input from the sales team). This interdisciplinary feedback loop was a highlight of the project, ensuring that RetailRec is not just academically sound but also practically useful.

Results and Impact: In our final evaluation, the hybrid model incorporating all methods delivered the best performance, achieving about a 15% improvement in Precision@5 and a 10% boost in NDCG compared to the baseline popularity model (exact numbers are hypothetical here for illustration).

More qualitatively, the recommendations made intuitive sense during manual inspection: many users' recommendation lists contained a mix of popular staple items and new, tailored picks – an outcome we aimed for from the start. These results underscore the benefit of combining approaches and the importance of the extensive feature engineering we carried out.

In summary, the project was not just about building a single algorithm but about researching a combination of techniques and integrating them into a cohesive solution. The lessons learned about handling implicit data, scaling computations, and incorporating business knowledge are valuable takeaways for any future data science projects our team will undertake.

10. Conclusion

In this project, we successfully built RetailRec, a team-based recommendation system tailored for a retail business. Starting from a large-scale transactional dataset, we created a solid data pipeline and experimented with multiple recommendation techniques. The journey began with simple popularity-based models to establish baseline expectations. We then implemented state-of-the-art collaborative filtering using ALS, which uncovered latent relationships between customers and products. To address the limitations of pure collaborative methods, we developed a content-based recommender that leverages product attributes to profile users, thereby handling cold-start scenarios and adding interpretability. We didn't stop at individual models; instead, we combined them into a hybrid system that aligns with both predictive accuracy and business needs (such as highlighting promotional items and catering to loyalty segments).

Evaluation of the models showed that each approach had merits, but the hybrid model provided the best balance of precision and recall, ensuring users get relevant suggestions while also discovering new products. For instance, a Tier 0 customer might see popular entry-level items as well as a couple of personalized picks based on their browsing, whereas a Tier 3 customer would predominantly see specialty items aligned with their past niche purchases. This adaptive behavior is exactly what we aimed to achieve – a recommender system that is personalized, scalable, and business-aware.

Beyond the technical accomplishments, the project offered valuable insights into customer behavior. By analyzing loyalty tiers, we learned how different customer segments interact with products, and we used these insights to make our recommendations more strategic. We identified which products serve as universal favorites and which drive revenue in specific ways, informing both the algorithm design and broader marketing decisions. These findings can guide future promotional strategies and product placements.

In conclusion, RetailRec demonstrates the power of combining data engineering, machine learning, and domain knowledge to create a robust analytics solution. The system is positioned to improve the customer experience by simplifying product discovery while simultaneously aiding the business through increased revenue.

Sales and customer engagement. As next steps, we recommend deploying the system in a phased manner (perhaps starting as a pilot for a subset of users), gathering feedback, and continuously refining the models. There is also scope to incorporate

more advanced features down the line, such as real-time personalization, integration of customer review sentiment, or graph-based recommendations. Our team's collaborative effort has produced a solid foundation for an intelligent recommendation service that can grow and adapt with the retailer's needs. We are confident that RetailRec can drive tangible value and serve as a cornerstone for data-driven personalization in the company's retail strategy.

11. References

Aggarwal, C. C. (2016). Recommender Systems: The Textbook. New York: Springer.

Herlocker, J. L., Konstan, J. A., Terveen, L. G., & Riedl, J. T. (2004). Evaluating collaborative filtering recommender systems. ACM Transactions on Information Systems, 22(1), 5–53. https://doi.org/ 10.1145/963770.963772

Hu, Y., Koren, Y., & Volinsky, C. (2008). Collaborative filtering for implicit feedback datasets. In Proceedings of the 2008 Eighth IEEE International Conference on Data Mining (pp. 263–272). IEEE.

12. Collab Link

https://colab.research.google.com/drive/1RCdOTB86DrAdb4moVlMtidek0WuhliZf?authuser=7#scrollTo=gE6bO9BXn7-i