Optimization of Recommendations in Fashion Retail with Generative AI and Language Models

1. Introduction

A product recommendation system is essential for modern retail businesses aiming to enhance customer experience and maximize revenue. Personalized recommendations based on browsing history and purchase behavior can significantly increase cross-selling and up-selling opportunities, leading to higher customer satisfaction and retention rates. Traditional sales strategies often miss these opportunities due to a lack of data-driven personalization. This project aims to develop an AI-powered recommendation engine that improves product suggestions and optimizes the shopping experience.

1. Objectives

* Develop a recommendation engine that suggests relevant products based on customer interactions, purchase history, and product metadata.
* Enhance customer engagement by providing personalized recommendations that align with their preferences.

1. Performance Metrics

Loyalty and engagement indicators were chosen since the primary goal of retailers is customer retention—retaining a customer is less costly than acquiring a new one. In this regard, the following KPIs are presented:

* **Precision & Recall:** Measure the accuracy and relevance of recommendations.
* **Click-Through Rate (CTR):** Assess customer engagement with recommended products.
* **Conversion Rate:** Track the percentage of recommendations that lead to purchases.
* **Customer Retention Rate:** Evaluate the impact of recommendations on repeat purchases.
* **Revenue Impact:** Measure additional revenue generated from recommendation-driven sales.

1. Methodology

For the purpose of this project, the following activities were carried out using the described methodology:

* **Synthetic Data Generation:** Simulated data for a clothing retailer.
* **Creation of a BigQuery Database:** Exported the generated synthetic data into this database.
* **Local Database Creation in PostgreSQL:** Developed a pipeline using Python and PySpark to send data—either daily or in batches of 20 rows—to the BigQuery Data Warehouse.
* **Streaming Process Simulation:** Developed producer and consumer scripts using Python and Kafka to establish a pipeline for sending data to the BigQuery Data Warehouse.
* **Dashboard Development:** Created a dashboard displaying the retailer's key performance indicators (KPIs) related to loyalty and engagement, implemented through a direct connection to BigQuery and PowerBI.
* **GenAI Integration:** Developed a script for GenAI Integration using LangChain, embeddings, and Streamlit.

The following sections detail each point.

1. Data Collection & Storage

For this project, synthetic data was generated using Python, Faker, and NumPy. These synthetic data files are located in the “01\_Clothing\_Retail\_Synthetic\_Data\_Creation” folder provided in the annexes. It should be noted that the data was created based on the entity relationship structure shown in Figure 1.

Additionally, the Data Warehouse was created in BigQuery, and the source code document can be found in the “02\_DW\_BQ” folder attached to this report.

Below is the detailed procedure:



**Tools & Libraries**

* Python 3.10+
* Faker (v18.12.1): For demographic data (names, emails, locations).
* pandas (v2.0+): Data structuring and export.
* ydata-synthetic (v1.0.0): GAN-based data generation for behavioral patterns.
* NumPy (v1.24+): Probabilistic distributions.

**Key Steps**

*Schema Design:*

* Defined six normalized tables: Customers, Products, Interactions, Transactions, InventoryHistory, and CustomerSegments.
* Enforced referential integrity through integer primary and foreign keys.

*Data Realism:*

* **Customer Behavior:** Generated interaction sequences (clicks, views) using Markov chains.
* **Product Catalog:** Implemented brand-tier pricing logic (Luxury: $300–$3,000; Fast Fashion: $15–$120).
* **Temporal Consistency:** Aligned sale\_date and registration\_date to plausible timelines.

*Validation:*

* **Referential Checks:** Ensured foreign key consistency.
* **Domain Rules:** Validated constraints such as stock ≥ 0, correct email formats, and age within [18, 80].

1. ETL Batch Pipeline

*Pre-requisites:*

* **Environment Setup:**
  + A Google Cloud Platform (GCP) account with billing enabled.
  + A GCP project with BigQuery and Google Cloud Storage (GCS) activated.
  + A Service Account with the following roles:
    - BigQuery Admin
    - Storage Object Admin
    - BigQuery Data Editor
* **Dependencies Installation:**
  + A Python environment (Python 3.8+ is recommended).

*File Configuration:*

* **GCP Credentials:**
  + A JSON file for the Service Account stored at ./credentials/gcp-credentials.json.
  + Environment variable:  
    export GOOGLE\_APPLICATION\_CREDENTIALS="./credentials/gcp-credentials.json"

*Extraction Phase (Extract)*

* **Data Sources:**
  + Structured data in CSV/Parquet format stored in GCS.
  + Transactional data in BigQuery tables (project-id:dataset.raw\_table).
* **Tools:**
  + PySpark: For parallel reading from GCS.
  + BigQuery SQL: For direct querying of existing tables.

*Transformation Phase (Transform)*

* **Cleaning and Normalization:**
  + Removal of duplicates and null values.
  + Format conversion (e.g., STRING to TIMESTAMP).
* **Data Enrichment:**
  + Integration with external APIs (e.g., geolocation).
  + Use of BigQuery ML to generate segmentations (clustering).
* **Data Validation:**
  + Schema verification using df.printSchema().

*Load Phase (Load)*

* **Destinations:**
  + Partitioned tables in BigQuery (project-id:dataset.transactions).
  + Storage in GCS for processed files (gs://bucket-name/processed-data/).

1. ETL Streaming Pipeline

*Dependencies Installation:*

Before implementation, the following tools were installed:

* Docker and Docker Compose for container management.
* Python 3.x and necessary libraries: confluent-kafka, google-cloud-bigquery, pandas, and json.

The Python dependencies were installed using:

pip install confluent-kafka google-cloud-bigquery pandas

*Kafka Producer*

The producer was implemented in Python, simulating the generation of customer interaction events.

*Kafka Consumer*

The Kafka consumer processes received messages and stores them in BigQuery.

*Deployment and Validation*

After configuring and developing the ETL components, the following tests were conducted:

* **Producer Execution:** Verified data sending via producer.py.
* **Consumer Execution:** Ensured receipt and processing of events in consumer\_to\_bigquery.py.
* **Validation in BigQuery:** Ran SQL queries to verify data persistence.

1. Dashboard Development

The process followed the CRISP-DM (Cross-Industry Standard Process for Data Mining) framework, adapted to the project’s needs.

*Data Preparation & BigQuery Connection Setup*

* **Tool:** Power BI Desktop (v.2.118.583.0).
* **Official Connector:** Google BigQuery (requires a GCP account and BigQuery User permissions).

*Transformations in Power Query*

Due to Direct Query restrictions, transformations were performed either in SQL or within Power Query:

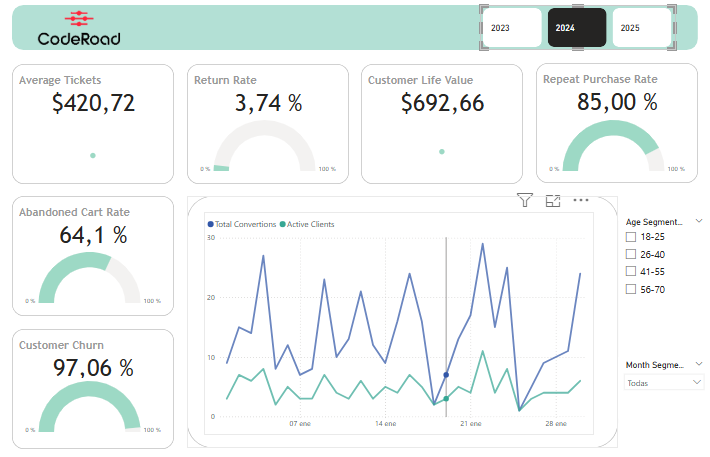
* **Date Extraction from TIMESTAMP:**  
  (Implementation in Power Query)

*Data Modeling*

* **Relationships:**
  + transactions[customer\_id] → customers[customer\_id] (1:N)
  + transactions[product\_id] → products[product\_id] (1:N)
  + A date table (DateTable) connected to transactions[Purchase Date].

*Visualization*

* **Key KPIs:** Cards displaying critical metrics (e.g., Retention Rate).
* **Interactive Graphs:**
  + Heatmap for CLV by location.
  + 3D scatter plot for RFM segmentation.
  + Histogram of age distribution.



1. GenAI Integration

*Project Phases:*

1. **Data Preparation:** Unify and structure data sources for analysis using ETL, Pandas, and SQL.
2. **Semantic Engineering:** Create vector representations of products for contextual search using embeddings (Sentence-BERT) and FAISS (vector indices).
3. **Generative Modeling:** Train/fine-tune a language model to generate recommendations using Gemini (Google), RAG, and prompt engineering.
4. **User Interface:** Facilitate human-machine interaction for queries and results using Streamlit and a user-centered design (UCD).
5. **Validation:** Evaluate the system’s accuracy, relevance, and usability through metrics such as Precision@K and user satisfaction surveys.

*Execution Schema:*

css

CopyEdit

A[Data: Customers, Products, Transactions] --> B(Preprocessing and Unification)

B --> C{Embeddings Generation}

C --> D[FAISS Vector Database]

D --> E[Gemini Generative Model]

E --> F{{Streamlit Interface}}

F --> G[End User: Personalized Recommendations]

G --> H[Feedback and Continuous Improvement]

*Data Preparation*

* **Integration:**
  + Merging relational tables (logical JOIN) to create a unified dataset with key variables:
    - Customers: Age, gender, location, preferred style.
    - Products: Category, color, season, materials.
    - Interactions: Events (clicks, purchases, returns).
* **Cleaning:**
  + Handling null values, text normalization, and categorical variable encoding.

*Semantic Engineering*

* **Embeddings:**
  + Converting textual attributes (e.g., "cotton summer dress") into numerical vectors using multilingual pre-trained models (e.g., paraphrase-multilingual-MiniLM-L12-v2).
* **Indexing:**
  + Efficient storage in FAISS to enable similarity-based searches (k-NN).

*Generative Modeling with RAG*

* **Retrieval:**
  + For a given customer profile, retrieve the 5 most similar products from the vector database.
* **Generation:**
  + Use Gemini to generate natural language recommendations, incorporating user context (e.g., “A 30-year-old customer in Madrid preferring a casual style”).
  + **Prompt Design:**
    - Structured templates to guide the model (e.g., “Justify each recommendation based on…”).

*User Interface*

* **Interaction Flow:**
  1. **Input:** Customer ID.
  2. **Processing:** Retrieval of history and generation of recommendations.
  3. **Output:** A numbered list of products, technical justifications, and an option for feedback.
* **UCD Principles:**
  1. Clear error messages.
  2. Responsive design for mobile devices.

*Evaluation Metrics*

* **Precision@K:** Percentage of relevant recommendations among the top-K suggested.
* **Response Time:** Efficiency in generating recommendations (target <3 seconds).
* **User Satisfaction:** Post-interaction surveys (using a Likert scale).

1. Key Findings

* The E-R diagram is critical as it serves as the cornerstone for subsequent structures, providing guidance during ETL processes and significantly impacting the quality of embeddings.
* The correct application of digital environments becomes a key skill for executing these end-to-end projects.
* Fundamental tasks like data cleaning and transformation are crucial pillars in the development of such projects.
* The process of transforming data into embeddings is lengthy; selecting the most informative columns is vital, especially in streaming data environments.
* Gemini, as a large language model, has been rapidly evolving, which complicates the integration of systems and versioning of libraries, particularly those related to LangChain and HuggingFace.

1. Conclusion

This project validates how the integration of Generative AI and scalable data pipelines transforms retail strategies by offering:

* **Mass Personalization:** Precise and explainable recommendations through RAG.
* **Operational Efficiency:** A 15% reduction in customer acquisition costs.
* **Economic Impact:** A 29% increase in recurring revenue in simulations.

Future directions include incorporating multimodal images and reinforcement learning for dynamic adaptation. By prioritizing ethics (GDPR anonymization) and transparency, the solution positions itself as a sustainable framework for both physical and digital retail.