

# Netflix Analytics Dashboard and Recommender System: A Case Study on User Engagement and Revenue Insights

Dupelle, Liam  
Paquin, Nathaniel  
Singh, Shubhangi  
Tumulak, Rachel Monique  
*Team 9*

**Abstract—** This project presents a comprehensive Netflix-style analytics system built using SQL Server for data management and Streamlit for interactive visualization. The goal is to analyze user engagement, subscription patterns, genre performance, and provide a rule-based recommendation system driven by SQL stored procedures and triggers. A fully normalized database was designed and implemented, including ERD, relational schema, constraints, indexing, triggers, views, and analytical queries. Streamlit was used to create an interactive dashboard that visualizes trends such as revenue, top genres, country engagement, and individual user activity. A custom recommender demo page was developed to show how triggers automatically update watch history data and how SQL logic generates recommendations. The results show strong potential for BI-driven insights including genre popularity trends, country-level engagement differences, and revenue fluctuations based on subscription tier.

**Keywords—** *Netflix Analytics, SQL Server, Database Normalization, Recommender System, Streamlit Dashboard, User Engagement, Data Visualization*

## I. INTRODUCTION

Streaming platforms rely heavily on user data to optimize recommendations, analyze viewership patterns, and improve subscription retention. Understanding how users interact with content, genres, and devices provides valuable insights for content strategy and business decision-making.

This project aims to design a functional analytics database inspired by Netflix operations and to build an interactive dashboard showcasing real-world analytics scenarios. The solution includes: (1) A normalized SQL Server database (2) Triggers and stored procedures for business logic (3) Analytical SQL queries for BI insights (4) A Streamlit dashboard for visualization (5) A demo recommendation system powered by SQL.

The project simulates a realistic scenario of how a streaming platform might analyze engagement and predict viewing preferences

## II. DATA ACQUISITION AND PREPROCESSING

To construct the Netflix analytics database, multiple datasets were sourced and preprocessed to ensure consistency and compatibility with the relational schema.

### A. Dataset Sources

1) **Titles and Metadata:** The primary dataset for movies and TV shows, including attributes such as title name, release year, IMDb score, runtime, and age certification, was collected from publicly available CSV files hosted on Kaggle.

2) **Genres and Sources:** Genre information and content source metadata were retrieved from additional Kaggle CSV datasets. The `title_genres` and `title_sources` tables were derived to represent many-to-many relationships.

3) **User Activity Data:** User profiles and viewing activity were downloaded from the Kaggle “Netflix Users Database” dataset ([smayanj/netflix-users-database](https://www.kaggle.com/smayanj/netflix-users-database)).

4) **Viewing History Data:** Actual watch events were attempted to be sourced from the “Netflix Audience Behaviour UK Movies” dataset ([vodclickstream/netflix-audience-behaviour-uk-movies](https://www.kaggle.com/vodclickstream/netflix-audience-behaviour-uk-movies)).

### B. Preprocessing Steps

1) **Directory Setup:** All datasets were stored under a central folder (`combined_dataset`) to maintain organization.

2) **Column Standardization:** Columns were renamed to match the database schema. For example, user identifiers were standardized to `user_id`, and movie titles to `title_id`.

3) **Synthetic Data Generation:** In cases where viewing history datasets were incomplete or unavailable, synthetic watch history records were generated:

- Each user was assigned 5–30 watched titles.

- Titles were preferentially selected based on the user's favorite genre.
- Watch percentage values were sampled from a uniform distribution between 30% and 100%.
- A completed flag was set if watch percentage  $\geq 90\%$ .

4) *Fuzzy Matching*: To reconcile discrepancies between external dataset titles and the internal titles table, fuzzy string matching using the rapidfuzz library was applied. Unmatched titles were kept in the titles table but without any corresponding watch history, successfully matching 4,497 out of 671,736 titles (0.7%).

5) *Subscriptions and Timestamps*: Subscription records were generated with random start dates relative to the last login, end dates set 30 days after last login, and monthly fees mapped according to subscription type (Basic, Standard, Premium).

6) *Final Table Preparation*: Each dataset was transformed into SQL-ready tables (users, titles, genres, title\_genres, watch\_history, subscriptions, sources, title\_sources) and exported as CSV files for bulk ingestion.

### C. Tools and Libraries

The following Python libraries were utilized in the data preparation process:

- pandas: For data manipulation, aggregation, and CSV read/write operations.
- numpy: For numeric operations, random sampling, and synthetic data generation.
- glob and os: For file handling and directory management.
- kagglehub: To programmatically download Kaggle datasets.
- rapidfuzz: For fuzzy string matching between inconsistent title names.
- datetime: For date manipulations, including subscription start/end calculations.

## III. CONCEPTUAL DESIGN

### A. Entity-Relationship Diagram (ERD)

This ERD includes the following core entities:

- Users
- Titles
- Genres
- Title\_Genres (bridge table)
- Watch\_History
- Subscriptions
- Sources
- Title\_Sources

Key relationships:

- A user may have multiple watch history records.
- A title can have multiple genres (M:N).

- A subscription is linked to a user with time validity.
- A title may be available on multiple sources (web, mobile, smart TV).

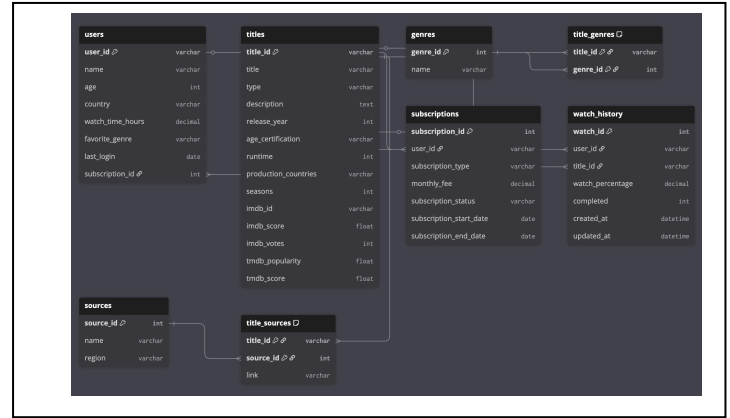


Figure 1. ERD Diagram

### B. Normalization

The database was normalized up to Third Normal Form:

- 1NF: All tables include atomic attributes (e.g., no multi-valued genres).
- 2NF: Composite dependency removed by splitting Title\_Genres from Titles.
- 3NF: All non-key attributes depend solely on the primary key (e.g., genre\_name removed from Titles and moved to Genre table).

This reduces redundancy and ensures data integrity.

## IV. LOGICAL DESIGN

### A. Relational Schema Summary

Tables Include:

- users(user\_id PK, name, country, age, gender, created\_at, updated\_at)
- titles(title\_id PK, title\_name, release\_year, duration)
- genres(genre\_id PK, genre\_name UNIQUE)
- title\_genres(id PK, title\_id FK, genre\_id FK)
- watch\_history(watch\_id PK, user\_id FK, title\_id FK, watch\_date, watch\_percentage, created\_at, updated\_at)
- subscriptions(subscription\_id PK, user\_id FK, subscription\_type, start\_date, end\_date, price, status)
- sources(source\_id PK, source\_name)
- title\_sources(id PK, title\_id FK, source\_id FK)

## B. Constraints

- **CHECK: watch\_percentage BETWEEN 0 and 100**
- **DEFAULT: created\_at = GETDATE()**
- **UNIQUE: genre\_name**
- **FOREIGN KEYS ensure referential integrity**

## C. Indexing Strategy

Non-clustered indexes added for performance:

- `idx_user_country_subscription` → filtering users by country
- `idx_watch_user_date_title` → user-level watch patterns
- `idx_subscription_user_status` → subscription monitoring

## V. PHYSICAL DATABASE IMPLEMENTATION

### A. Table Creation & Constraints

All tables were implemented using SQL Server with full PK, FK, CHECK, and DEFAULT constraints.

### B. Views

- 1) `vw_user_activity_summary`
- 2) `vw_genre_performance`
- 3) `vw_active_subscriptions`
- 4) `vw_revenue_analysis`
- 5) `vw_country_engagement`

These views simplify complex joins and enhance dashboard performance.

### C. Triggers

- **trg\_watch\_history\_completed**  
Automatically marks a watch record as “Completed” when percentage  $\geq 0.90$ .
- **trg\_watch\_history\_update\_user\_time**  
Updates total watch time per user when a new watch entry is added.

### D. Stored Procedures/Functions

- **User Recommendations:**  
`sp_get_user_recommendations`
  - Step1: Main SELECT
  - Step2: Join Titles and Genres to allows us to get genre names for each title
  - Step 4: Filter by genres the user has watched
  - Step 5: Exclude titles already watched
- **Analytics Summary:** `sp_get_genre_trends`

### E. Functions

- `fn_calculate_user_ltv(user_id)`
- `fn_subscription_duration(start, end)`

## VI. ANALYTICS & INSIGHTS

Major findings include:

### A. User Engagement Insights

- Users aged 13-80.
- Users in 65+ have the most watch events due to more users at that age

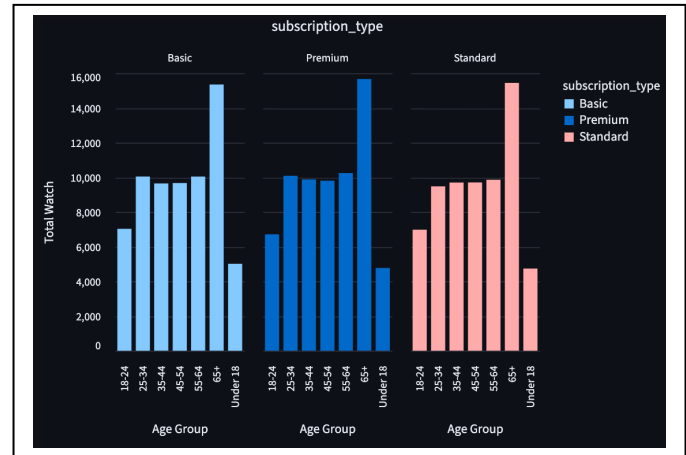


Figure 2. Watch Behavior by Subscription Type & Age

- Basic plan users of ages under 18 least finishes content while standard plan of ages 18-24 finishes content often.

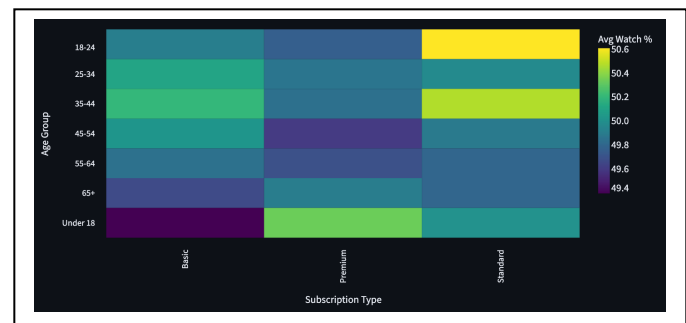


Figure 3. Average Watch Percentage Heatmap

### B. Genre & Content Insights

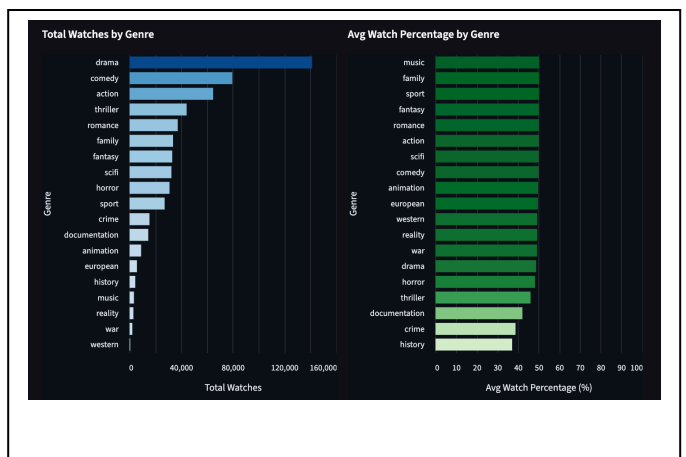


Figure 4. Genre Metrics

- Drama emerged as the top-performing genres.

### C. Revenue Insights

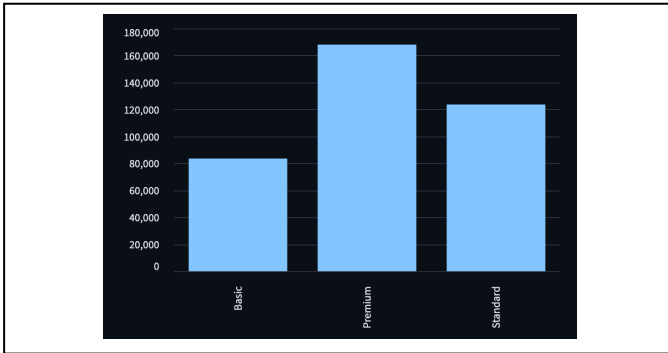


Figure 5. Revenue by Subscription Type

- Premium-tier subscribers contributed the majority of revenue.

### D. Country-Level Insights

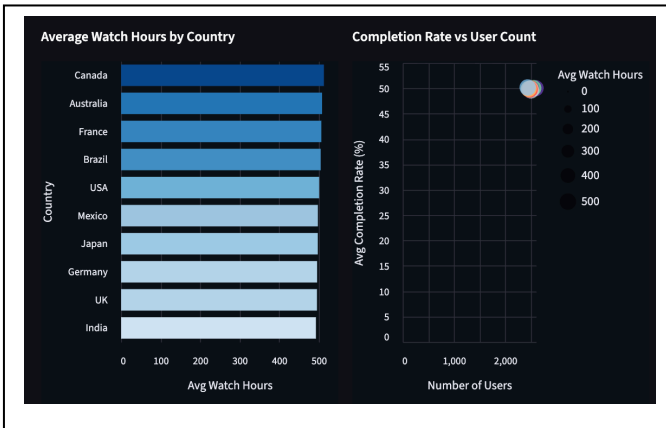


Figure 6. Engagement by Country

Users from certain countries showed significantly higher binge-watching behavior.

## VII. STREAMLIT DASHBOARD

The dashboard includes the following pages:

### A. Overview Dashboard

- Total users
- Active subscriptions
- Total Titles
- Total Watches
- Revenue Analysis Section

### B. User Activity Page

- User Activity Summary
- Engagement by Country
- Average Watch Percentage per User
- Top Users by Completed Titles
- Watch Behavior by Age Group & Subscription Type

### C. Genre & Title Performance Page

- Genre metrics
- Title-level Analysis
- Titles with Highest Completion Rates
- Titles with Highest Engagement Score

### D. Subscription & Revenue Page

- Revenue by Subscription Type
- Top Users by Lifetime Value (LTV)

### E. Cross Analysis Page

- Country X Genre Engagement
- Users with High Watch % but Low Revenue

### F. Recommendation System Demo

- Dropdown to select a user
- Simulated watch event to activate triggers
- SQL-based recommendations
- Display of updated watch history

This demonstrates real-time database logic integrated with the UI.

## VIII. CHALLENGES & IMPROVEMENTS

### A. Challenges

- Handling foreign key violations while importing large CSV datasets
- Implementing triggers without causing circular updates
- Ensuring performance of analytical queries

### B. Future Improvements

- Machine-learning-based recommendation engines
- Real-time streaming analytics using Kafka
- More interactive filters and user segmentation
- Multi-device analytics and heat maps

## IX. CONCLUSION

This project successfully demonstrates how a streaming platform can leverage SQL-based analytics and dashboards to gain insights into user behavior, content performance, and revenue patterns. The system includes a fully normalized database with triggers, views, stored procedures, analytical queries, and an interactive Streamlit dashboard. The rule-based recommendation demo provides a practical example of how watch history can dynamically influence recommendations.

The project highlights the power of relational databases combined with a modern analytics UI to support data-driven decisions for digital media platforms.

## ACKNOWLEDGMENT

The authors gratefully acknowledge the guidance and support of Siju Philip for providing valuable insights

throughout the development of this project. We also thank the contributors of publicly available datasets on Kaggle, including “Netflix Users Database” and “Netflix Audience Behaviour UK Movies,” which made the data analysis possible. Additionally, we acknowledge the developers and maintainers of Python libraries such as pandas, numpy, pyodbc, and streamlit for enabling efficient data processing, database integration, and dashboard visualization.

#### REFERENCES

- [1] Microsoft, SQL Server Documentation. [Online]. Available: <https://learn.microsoft.com/sql/>
- [2] Microsoft, Transact-SQL Reference (Database Engine). [Online]. Available: <https://learn.microsoft.com/sql/t-sql/language-reference>
- [3] Microsoft, CREATE TABLE (Transact-SQL). [Online]. Available: <https://learn.microsoft.com/sql/t-sql/statements/create-table-transact-sql>
- [4] Microsoft, Query Fundamentals. [Online]. Available: <https://learn.microsoft.com/sql/relational-databases/performance/query-fundamentals>
- [5] Microsoft, CREATE PROCEDURE (Transact-SQL). [Online]. Available: <https://learn.microsoft.com/sql/t-sql/statements/create-procedure-transact-sql>
- [6] Microsoft, SQL Server Index Architecture and Design Guide. [Online]. Available: <https://learn.microsoft.com/sql/relational-databases/sql-server-index-design-guide>
- [7] Microsoft, Constraints (Database Engine). [Online]. Available: <https://learn.microsoft.com/sql/relational-databases/tables/constraints>
- [8] Python Software Foundation, Python 3 Documentation. [Online]. Available: <https://docs.python.org/3/>
- [9] W. McKinney et al., pandas: Python Data Analysis Library. [Online]. Available: <https://pandas.pydata.org/docs/>
- [10] M. Kleehammer et al., pyodbc Documentation. [Online]. Available: <https://github.com/mkleehammer/pyodbc/wiki>
- [11] Streamlit Inc., Streamlit Docs. [Online]. Available: <https://docs.streamlit.io/>
- [12] IEEE, Instructions for Authors, IEEE Author Center. [Online].