Advanced 🎬 Netflix Data Analysis — SQL + R End-to-End Data Engineering Project

Last Updated: 23 November 2025
Author: *Parkhe Ankit*
Tech Stack: PostgreSQL, R (tidyverse), Analytical SQL, Data Cleaning Pipelines, PL/pgSQL, Materialized Views, Clustering, Visualization, Dashboard Design

Project Overview

This project demonstrates a complete, production-style data analysis and SQL engineering workflow applied to the Netflix Titles dataset.

It combines:

- **Advanced PostgreSQL engineering**
- **R-based data cleaning**
- **Exploratory data analysis (EDA)**
- **Machine learning (k-means clustering)**
- **Visualization**
- **Data quality enforcement**
- **Recommendation logic**
- **Materialized views**
- **Dashboard design for Power BI / Tableau**

The project is intentionally designed to be portfolio-ready, suitable for showcasing to hiring managers for Data Analyst, SQL Developer, or Data Engineering roles.

Schema (PostgreSQL) - Explanation and Create Table

The PostgreSQL schema follows a hybrid **normalized + analytical** style.

Core Tables:

- users
- books (represents netflix titles)
- genres
- ratings
- book_tags
- user_preferences
- activity_log
-

Create table (PostgreSQL):

```
CREATE TABLE netflix_title(
    show_id INT PRIMARY KEY,
    title TEXT NOT NULL,
    director TEXT,
    cast_list TEXT,
    country TEXT,
    date_added DATE,
    release_year INT CHECK (
        release_year >= 1800
        AND release_year <= EXTRACT(YEAR FROM CURRENT_DATE)
    ),
    rating TEXT,
    duration TEXT,
    listed_in TEXT,
    description TEXT,
    type_ TEXT CHECK (type_ IN ('Movie', 'TV Show'))
);
```

Data Quality Strategy & Defensive SQL

The SQL portion uses defensive programming techniques to avoid:

❌ invalid casts
❌ corrupted dates
❌ irregular duration formats
❌ inconsistent categories
❌ broken transactions

Key techniques:

- NULLIF + COALESCE
- Regex extraction for duration
- Strong CHECK constraints
- Safe casting inside PL/pgSQL
- Proper transactions using SAVEPOINT
- Materialized views for clean analytics

Advanced Analytical SQL Queries

1 **Top-Rated Titles Query**

```
SELECT b.title, g.genre_name, ROUND(AVG(r.rating), 2) AS avg_rating
FROM books b
JOIN ratings r ON b.book_id = r.book_id
JOIN genres g ON b.genre_id = g.genre_id
GROUP BY b.book_id, g.genre_name
ORDER BY avg_rating DESC
LIMIT 10;
```

---

2 **Recommendation Query (Collaborative Filtering)**

```
WITH similar_users AS (
    SELECT r2.user_id
    FROM ratings r1
    JOIN ratings r2 ON r1.book_id = r2.book_id
    WHERE r1.user_id = $1
      AND ABS(r1.rating - r2.rating) <= 1
      AND r2.user_id != $1
)
SELECT b.title, ROUND(AVG(r.rating), 2) AS avg_rating
FROM ratings r
JOIN books b ON r.book_id = b.book_id
WHERE r.user_id IN (SELECT user_id FROM similar_users)
  AND b.book_id NOT IN (
      SELECT book_id FROM ratings WHERE user_id = $1
  )
GROUP BY b.book_id
ORDER BY avg_rating DESC
LIMIT 10;
```

Stored Procedures, Functions, Triggers

1 Safe Integer Extraction Function

```
CREATE OR REPLACE FUNCTION extract_first_int(txt TEXT)
RETURNS INT AS $$
DECLARE
    v INT;
BEGIN
    IF txt IS NULL THEN
        RETURN NULL;
    END IF;

    BEGIN
        v := CAST(REGEXP_REPLACE(txt, '[^0-9]', '', 'g') AS INTEGER);
        RETURN v;
    EXCEPTION WHEN INVALID_TEXT_REPRESENTATION THEN
        RETURN NULL;
    END;
END;
$$ LANGUAGE plpgsql IMMUTABLE;
```

---

2 Recompute Average Rating Procedure

```
CREATE OR REPLACE FUNCTION recompute_avg_rating(book INT)
RETURNS VOID AS $$
BEGIN
    UPDATE books
    SET avg_rating = sub.avg
    FROM (
        SELECT book_id, ROUND(AVG(rating)::NUMERIC, 2) AS avg
        FROM ratings
        WHERE book_id = recompute_avg_rating.book
        GROUP BY book_id
    ) AS sub
    WHERE books.book_id = sub.book_id;
END;
$$ LANGUAGE plpgsql;
```

Data Cleaning Pipeline (SQL + Staging Layer)

Staging → Clean → Production Pattern

1. Load raw CSV into staging (_stg)
2. Normalize strings and whitespace
3. Validate release_year, rating, type
4. Parse date formats
5. Extract duration integers
6. Route invalid rows to _err
7. Insert valid rows into netflix_title

Example Staging Load

```
CREATE TABLE netflix_title_stg AS
SELECT * FROM netflix_title WITH NO DATA;
```

Production Insert with Cleaning

```
INSERT INTO netflix_title (
    show_id, title, director, cast_list, country,
    date_added, release_year, rating, duration,
    listed_in, description, type_
)
SELECT
    CAST(NULLIF(TRIM(show_id_text), '') AS INT),
    NULLIF(TRIM(title), '')::TEXT,
    NULLIF(TRIM(director), '')::TEXT,
    NULLIF(TRIM(cast_list), '')::TEXT,
    NULLIF(TRIM(country), '')::TEXT,
    COALESCE(
```

```
        TO_DATE(NULLIF(TRIM(date_added), ''), 'YYYY-MM-DD'),
        TO_DATE(NULLIF(TRIM(date_added), ''), 'Mon DD, YYYY')
    ),
    CAST(NULLIF(TRIM(release_year_text), '') AS INT),
    NULLIF(TRIM(rating), ''),
    NULLIF(TRIM(duration), ''),
    NULLIF(TRIM(listed_in), ''),
    NULLIF(TRIM(description), ''),
    CASE
        WHEN TRIM(type_) IN ('Movie', 'TV Show') THEN TRIM(type_)
        ELSE NULL
    END
FROM netflix_title_stg
WHERE NULLIF(TRIM(title), '') IS NOT NULL;
```

Dashboard Plan (Power BI / Tableau)

KPIs:

- Total Titles
- Movies vs TV Shows
- Average Rating
- Titles by Year
- Most Common Genres

Filters:

- Genre
- Country
- Rating
- Release Year

Visuals:

- Bar chart: Top Genres
- Heatmap: Releases per Year × Genre
- Time-Series: Titles Added Over Time
- Drill-through page for each title

Repository Structure

```
netflix_project_repo/
│
├── README.md
│
├── SQL/
│   ├── 01_schema.sql
│   ├── 02_cleaning.sql
│   ├── 03_views_and_materialized.sql
│   └── 04_procedures_and_triggers.sql
│
├── scripts/
│   └── netflix_analysis.R
│
├── data/
│   ├── netflix_titles_nov_2019.csv
│   ├── clean_netflix.csv
│
├── plots/
│   ├── movies_by_year.png
│   ├── clustering_plot.png
│
└── docs/
    ├── SQL_Netflix_Project_Detailed.pdf
    └── ER_Diagram.png
```

R Language Analysis Component

Below is the full R script used for cleaning, exploring, and visualizing the Netflix dataset.

```
##########################################
# Netflix Data Analysis Project - R Script
# Author: YOUR NAME
# Description: Cleaning, exploring, and
# visualizing Netflix titles dataset
##########################################

# ----------------------------
# 1. Load Libraries
# ----------------------------
library(tidyverse)

# ----------------------------
# 2. Load Dataset
# ----------------------------
df <- read_csv("data/netflix_titles_nov_2019.csv")

# ----------------------------
# 3. Keep important columns
# ----------------------------
df <- df %>%
  select(show_id, type, title, release_year, duration)

# ----------------------------
# 4. Extract numeric duration
# ----------------------------
df <- df %>%
  mutate(duration_num = as.numeric(str_extract(duration, "\\d+")))

# ----------------------------
# 5. Save cleaned dataset
# ----------------------------
write_csv(df, "data/clean_netflix.csv")

# ----------------------------
# 6. Exploratory Data Analysis
# ----------------------------

# 6.1 Average movie duration
avg_movie_duration <- df %>%
  filter(type == "Movie") %>%
  summarize(avg_duration = mean(duration_num, na.rm = TRUE))

print(avg_movie_duration)

# 6.2 Which year released the most content?
content_by_year <- df %>%
  count(release_year, sort = TRUE)

print(content_by_year)

# 6.3 Longest Movie and Longest TV Show
longest <- df %>%
  group_by(type) %>%
  slice_max(duration_num)

print(longest)

# ----------------------------
# 7. Visualization
# ----------------------------

# 7.1 Movie releases per year
movie_plot <- df %>%
```

```
  filter(type == "Movie") %>%
  count(release_year) %>%
  ggplot(aes(release_year, n)) +
  geom_col(fill = "red") +
  labs(
    title = "Movies Released Per Year",
    x = "Year",
    y = "Number of Movies"
  ) +
  theme_minimal()

ggsave("plots/movies_by_year.png", movie_plot, width = 8, height = 5)

# ----------------------------
# 8. Clustering (Bonus)
# ----------------------------

# Remove missing values for clustering
cluster_df <- df %>%
  filter(!is.na(duration_num)) %>%
  select(duration_num, release_year)

# Perform k-means
set.seed(42)
clusters <- kmeans(cluster_df, centers = 3)

cluster_df$cluster <- as.factor(clusters$cluster)

# Plot clusters
cluster_plot <- ggplot(cluster_df, aes(duration_num, release_year, color = cluster)) +
  geom_point(alpha = 0.7) +
  labs(
    title = "K-Means Clustering of Netflix Titles",
    x = "Duration (minutes/seasons)",
    y = "Release Year"
  ) +
  theme_minimal()

ggsave("plots/clustering_plot.png", cluster_plot, width = 8, height = 5)

##########################################
# End of Script
##########################################
```

Summary

This project demonstrates:

- 🎯 Clean PostgreSQL schema design
- 🔐 Defensive SQL + safe parsing
- 📊 Advanced analytical queries
- 🤖 Recommendation system logic
- ⚙️ Stored procedures & triggers
- 🚀 Materialized view optimization
- 📈 Dashboard blueprint
- 🧹 R-based data cleaning pipeline
- 🗺️ R-based visualization & clustering

It is suitable for **GitHub portfolios**, **resume links**, and **interviews** for:

- Data Analyst
- SQL Developer
- Data Engineer
- BI Analyst