1. Introduction

I built a Python-based recommendation engine whose goal is to surface high-quality, low-visibility books that typically fall outside mainstream algorithms. By filtering for titles with an average rating above 4.0 but fewer than 200 reviews, the system focuses on a curated “niche” subset that often goes overlooked on Goodreads.

This report walks through each component of the program, explaining the logic and why the recommendations align with user preferences.

---

2. Data Sources & Preprocessing

Full Catalog (`books.xlsx`)

- Contains all scraped metadata: title, authors, average_rating, ratings_count, text_reviews_count, num_pages, publication_date, publisher, ISBN, and genre.

Niche Subset (`over 4 and under 200.xlsx`)

- A filtered view of the full catalog: `average_rating > 4.0` and `ratings_count < 200`.

Key Steps in Preprocessing:

1. **Genre Parsing:** Split the comma-separated `genre` field into lists (`genre_list`).
2. **Page-Count Conversion:** Coerce the `num_pages` column to numeric values, handling any parsing errors.
3. **Size Buckets:** Assign each book to one of four fixed buckets based on page count:
    - Short: 0–249 pages
    - Medium: 250–499 pages
    - Long: 500–699 pages
    - Extra-Long: 700+ pages

These steps standardize raw data, preparing it for vectorization and filtering.

---

3. User Interaction & Preference Collection

Upon startup, the script prints a concise introduction describing the recommendation process. It then prompts the user through three stages:

1. **Genre Selection:** The user can choose multiple genres from the list of available values.
2. **Length Preference:** The user selects one size bucket, with page ranges displayed alongside each option.
3. **Reference-Book Rounds:** Three rounds of pairwise comparisons. In each round, the system:
    - Filters the **full** catalog by the user's genres and chosen size bucket.
    - Sorts candidates by `ratings_count` to prioritize better-known titles within the niche.
    - Presents two books to the user (or "n" if they haven't read either).
    - Carries forward the chosen book as a taste reference for subsequent rounds.

These inputs form a composite profile capturing explicit (genre, length) and implicit (taste via reference books) preferences.

---

4. Profile Construction

The program translates user choices into numeric vectors for scoring:

- **Genre Vector:** A binary vector over all possible genres, with 1's marking the user's picks.
- **Length Target:** The midpoint of the selected bucket (e.g. Medium → 375 pages), normalized to [0,1] via MinMax scaling.
- **Reference Centroid:** The average feature vector of the three chosen reference books. Each book's vector concatenates its one-hot genre encoding with its normalized page count.

This unified profile allows direct similarity calculations against candidates in the niche subset.

---

5. Scoring Logic

For each candidate in `over 4 and under 200`, the system computes:

1. **Genre Score (weight 0.50):** Fraction of the user's chosen genres that the book shares (equal weight per genre).
2. **Length Score (weight 0.25):** Full credit if the book falls in the selected bucket; otherwise a linear penalty proportional to distance from the bucket's boundaries.
3. **Reference Similarity (weight 0.25):** Cosine similarity between the candidate's feature vector (genre one-hot + normalized pages) and the reference centroid.

The final **composite score** is the weighted sum:

```
Unset
composite = 0.50 * genre_score
          + 0.25 * length_score
          + 0.25 * reference_score
```

A higher score indicates a stronger alignment with explicit and implicit user preferences.

---

6. Recommendation Selection & Tie-Breaking

- After scoring, the system sorts candidates by descending composite score.
- Exact ties are broken by preferring the book with a lower `ratings_count`, reinforcing the niche focus.
- The top ten entries form the final recommendation list.

---

7. Transparency & Explanation

Immediately before displaying recommendations, the script prints:

- The three weights (genre, length, reference) used in scoring.
- The user's explicit selections (genres and size bucket).
- The reference books they chose in the three rounds.

This transparency ensures the user can see how their inputs influence outcomes and why each book surfaces in the list.

---

8. Why These Recommendations Make Sense

1. **Genre Alignment:** By weighting genre the most, the system guarantees that every recommendation matches the user's favorite categories.
2. **Length Fit:** The size-bucket logic balances user desires for book length, penalizing but not outright excluding outliers.
3. **Taste Calibration:** The reference-book similarity captures subtler preferences—writing style, thematic complexity, pacing—via the centroid of known favorites.
4. **Niche Emphasis:** Filtering for under-200-reviewed titles and breaking ties by lower review counts ensures fresh discoveries instead of over-exposed bestsellers.

Together, these components create a recommendation pipeline that is both data-driven and user-centered, surfacing hidden gems tailored to each reader.
