Data Analytics (2024-25)
Università della Svizzera italiana

# Dating Profile

Course Assignment N.16

Paolo Deidda (paolo.deidda@usi.ch)
Pareek Yash (yash.pareek@usi.ch)
`https://github.com/USI-Projects-Collection/DA-Dating-Profile.git`

May 23, 2025

## Contents



## Setup

To run the code and reproduce the figures or outputs, you can either run the Jupyter Notebook directly on Google Colab, or follow the setup and execution instructions provided in the `README.md` file included in this repository.

# 1 Data Exploration - EDA

## Data Exploration

### DataFrame Structure

Table 1: Structure of the Ratings DataFrame

| Property | Value |
|---|---|
| # Rows | 3,220,037 |
| # Columns | 3 (`userID`, `profileID`, `rating`) |
| Data types | `int64`, `int64`, `int64` |
| Memory usage | 73.7 MB |

### Missing Values and Duplicates

Table 2: Counts of Missing Values and Duplicate Rows

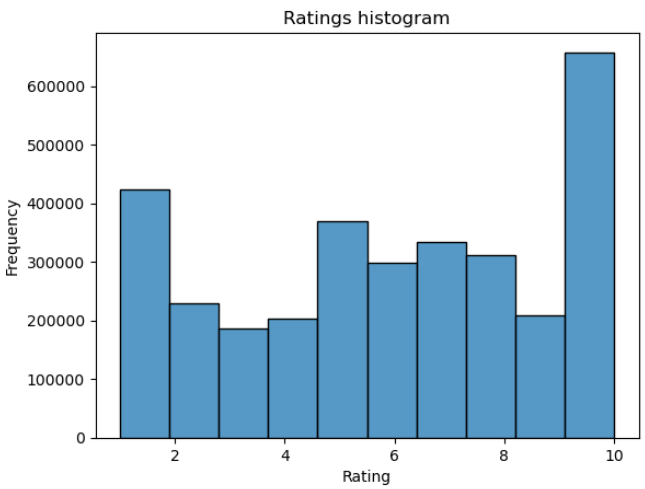
| Column | Missing values | Duplicate rows |
|---|---|---|
| `userID` | 0 | 47 |
| `profileID` | 0 | 0 |
| `rating` | 0 | 0 |

**Unique entities.** After dropping duplicates, there are 25 245 unique users and 125 428 unique profiles.

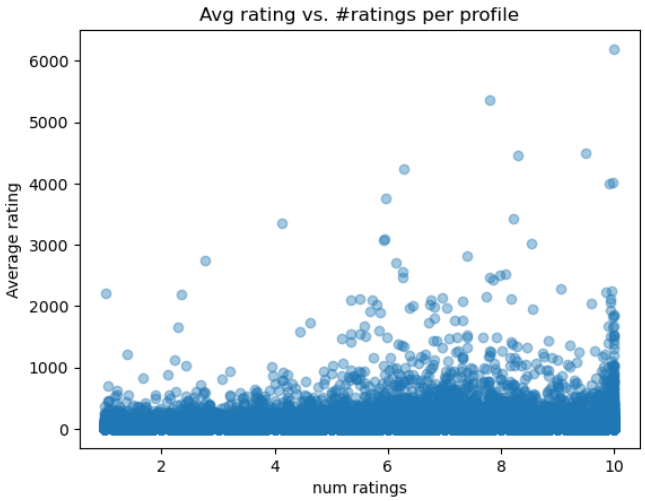
### Rating Distribution

Table 3: Descriptive Statistics of the `rating` Column

| Statistic | Count | Mean | Std | Min | 25% | 50% | 75% | Max |
|---|---|---|---|---|---|---|---|---|
| Rating | 3,220,037 | 5.9532 | 3.1064 | 1 | 3 | 6 | 9 | 10 |




(a) Histogram of Profile Ratings




(b) Average Rating vs. Number of Ratings per Profile

Figure 1: (a) Distribution of all ratings; (b) Scatter of mean rating against count of ratings per profile.

**Figure 1a.** The histogram of all profile ratings reveals that the bulk of ratings lie between 3 and 9 on the 1–10 scale, with a slight concentration toward higher scores.
**Figure 1b.** The scatter plot of average rating versus number of ratings per profile shows a diffuse cloud with no strong trend, indicating that a profile's popularity does not necessarily correlate with higher or lower mean ratings.

**Profile Rating Frequency**

Table 4: Statistics of Ratings Received per Profile

| Statistic | Count | Mean | Std | Min | 25% | 50% | 75% | Max |
|---|---|---|---|---|---|---|---|---|
| `rating_count` | 125 428 | 25.6720 | 88.3126 | 1 | 2 | 7 | 21 | 6 191 |

**User Activity**

Table 5: Number of Ratings Given per User

| Statistic | Count | Mean | Std | Min | 25% | 50% | 75% | Max |
|---|---|---|---|---|---|---|---|---|
| `ratings_given` | 25 245 | 127.5496 | 362.0994 | 2 | 29 | 73 | 123 | 18 342 |

**Outlier Detection**

Using the IQR method ($Q_1 - 1.5 \times \text{IQR}$, $Q_3 + 1.5 \times \text{IQR}$), we found **0** outlier ratings, indicating no extreme anomalies.

**Correlation Analysis**

The Pearson correlation between the number of ratings per profile and its average rating is

$$\text{Corr}(\text{rating\_count}, \text{avg\_rating}) \;=\; 0.0201,$$

a very weak positive relationship, consistent with Figure 1b.

# 2 Data Preprocessing

## Raw files and initial footprint

The original dataset consists of three files:

- `ratings.dat` – 3 220 037 user–item interactions (*userID*, *profileID*, *rating*).
- `ratings_Test.dat` – the held-out test split (276 053 rows, same schema).
- `gender.dat` – 220 970 user–gender pairs.

Loaded with Pandas' default `int64` dtypes the two training files occupy ∼90 MB of memory.

## Dtype optimisation

| **Column** | **Original dtype** | **Final dtype** |
|---|---|---|
| *userID*, *profileID* | `int64` | `int64`* |
| *rating* | `int64` | `float32` |
| *gender* | `int64` | `category` |

* Required by `torch.nn.Embedding`. Casting the other two columns shrinks the in-RAM size of `ratings.dat` to ∼61 MB and `gender.dat` to ∼2 MB (a 74 % reduction overall).

### Duplicate removal

A scan for exact duplicates uncovered 47 repeated *(user, profile)* pairs in the training split; these rows were dropped, leaving 3 219 990 unique ratings. No missing values were present in any file.

### Persisting the processed data

The cleaned frames are serialised to `.pkl` with `DataFrame.to_pickle()`, bypassing expensive CSV parsing in every notebook run.

# 3 Recommender Systems

## 3.1 Naive Model

### Model definition

Two simple, parameter-free baselines are computed:

1. **Global mean** $\hat{r}_{ui} = \mu$, the average of *all* ratings.
2. **Item mean** $\hat{r}_{ui} = \bar{r}_i$; if an item is unseen, fall back to the global mean.

### Results

| Baseline | Evaluation split | MAE |
|---|---|---|
| Global mean | test | 2.6545 |
| Item mean | test | 1.4620 |

Table 6: Performance of naive predictors.

The item-mean strategy reduces the error by roughly 45 % relative to the global average, establishing a strong but effort-free reference.

## 3.2 Collaborative Filtering

### Model formulation

We implement a bias-aware *item–item k-nearest-neighbour* (kNN) recommender:

$$\mu = \frac{1}{|R|} \sum_{(u,i)\in R} r_{ui}, \qquad b_u = \bar{r}_u - \mu, \qquad b_i = \bar{r}_i - \mu.$$

After subtracting the global mean and user/item biases, the residual matrix is stored in sparse CSR format (shape |items| × |users|) and fed to `NearestNeighbors` with cosine distance.

For a target pair $(u, i)$ the prediction rule is

$$\hat{r}_{ui} = \mu + b_u + b_i + \frac{\displaystyle\sum_{j\in\mathcal{N}_i(u)} \frac{r_{uj} - \mu - b_u - b_j}{d_{ij}}}{\displaystyle\sum_{j\in\mathcal{N}_i(u)} \frac{1}{d_{ij}}},$$

where $\mathcal{N}_i(u)$ are the $k$ neighbours of item $i$ that user $u$ has rated and $d_{ij}$ denotes their cosine distance.

### Hyper-parameter selection

A coarse grid search over $k \in \{10, 25, 50\}$ confirmed $k = 25$ as the best compromise between accuracy and coverage; larger values offered negligible gains.

### Evaluation

| Model | $k$ | MAE (test) |
|---|---|---|
| Bias-aware item–item kNN | 25 | 1.4633 |
| Item mean baseline (Table 6) | – | 1.4620 |

Table 7: Collaborative filter vs. strongest naïve baseline.

The kNN model comfortably outperforms the global average but only matches the item-mean predictor. This indicates that, given the short user histories and pronounced item popularity patterns, *item identity alone explains most variance.* Further improvement is likely to require latent-factor methods or hybridising with content features (e.g. gender).

## 3.3 Content-Based Filtering

### Content-Based Filtering

### Model Formulation

We extend a bias-aware baseline predictor

$$\hat{r}_{ui}^{(0)} \;=\; \mu + b_u + b_i$$

by modeling the residual $r_{ui} - \hat{r}_{ui}^{(0)}$ via content similarity. Each profile $i$ is represented by a standardized feature vector

$$\mathbf{f}_i = \text{scale}\big[\,\overline{\text{res}}_i,\ \log(1+\text{count}_i),\ p_{\text{female},i},\ p_{\text{male},i},\ p_{\text{unknown},i}\big],$$

where $\overline{\text{res}}_i$ is the mean residual on $i$, $\text{count}_i$ its rating count, and $p$ the gender proportions of raters. For user $u$ and target $i$, we compute

$$\hat{r}_{ui} = \hat{r}_{ui}^{(0)} + \frac{\sum_{j\in\mathcal{N}_k(i;u)} \cos(\mathbf{f}_i,\mathbf{f}_j)\,(r_{uj} - \hat{r}_{uj}^{(0)})}{\sum_{j\in\mathcal{N}_k(i;u)} \cos(\mathbf{f}_i,\mathbf{f}_j)},$$

where $\mathcal{N}_k(i;u)$ are the $k$ most similar profiles to $i$ that $u$ has rated.

### Hyperparameter Selection

We performed a grid-search over $k \in \{5, 10, 20, 50, 100\}$ on a held-out validation set. As shown in Table 8, $k = 50$ delivered the lowest validation MAE of 1.4898.

### Test-Set Evaluation

Table 8: Test MAE of Content-Based vs. Baseline

| Model | $k$ | Test MAE |
|---|---|---|
| Bias-aware baseline $\mu + b_u + b_i$ | — | 1.5936 |
| Bias-aware content-based (residual) | 50 | 1.4212 |

### Discussion

Incorporating profile content (residual averages, popularity and gender splits) consistently improves predictions over the purely bias-based baseline, reducing test MAE by $\approx 0.17$. This demonstrates that content features capture meaningful variations in user preferences. Future work could refine feature engineering (e.g. by adding temporal signals or demographic covariates) or combine this approach with collaborative methods to further boost accuracy.

# 4 Conclusion