AN ILLUSTRATED, HANDS-ON GUIDE

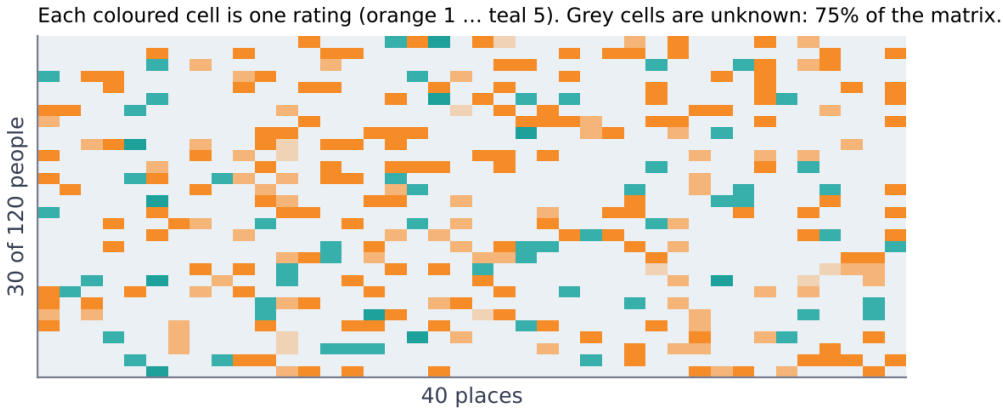
Recommendation Systems

A friendly guide to AI that learns what people like, for places in the city

Plain English, pictures first, 120 people and 40 places,
five labs in plain Python, real numbers from the code, a glossary and a cheat sheet.

Seven short chapters · Five labs · Three challenges · Cheat sheet

Prompted and directed by Mahmoud A.
Written and illustrated by Claude (Anthropic)



Prepared for the 2026 CCS: Urban AI + Agent Workshop · Harbin Institute of Technology · November 2026

How to Use This Guide

Read it in order: every chapter ends with something to try.

YouTube, TikTok and Dianping guess what you will like next. This guide uses the same ideas for a question about cities: **which places will a person find comfortable?** You will predict missing comfort ratings, recommend places to a person the system has never seen, and use the 3D city (sun, green, noise) to rate places nobody has visited.

YOU WILL BE ABLE TO

- Write people, places and ratings as one sparse matrix.
- Build and beat two simple baselines.
- Train matrix factorisation from scratch.
- Solve the cold start: a new person, three questions.
- Add place features from the city model.
- Measure everything with held-out ratings.

WHAT YOU NEED

- Python 3 with `numpy`. No internet, no GPU.
- The file `rec_lab.py` (or `rec_lab.ipynb`). Run it with `python rec_lab.py`.
- Paper and a pen for the exercises.

CONTENTS

1. What is a recommendation system?
2. The rating matrix · Lab 1
3. Baselines · Lab 2
4. Matrix factorisation · Lab 3
5. Cold start · Lab 4
6. Use the city · Lab 5
7. From comfort to the city
8. Pitfalls, glossary, cheat sheet, reading

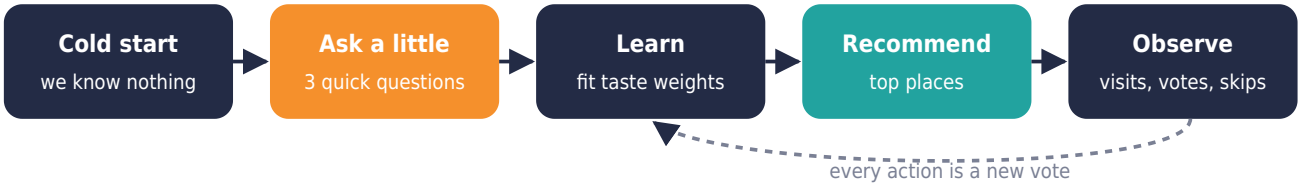
About the data. People, places and ratings are synthetic: each person has hidden tastes for sun, green and noise, and ratings follow from those tastes plus noise. The rule is in `rec_lab.py`, so you can check every result. Every number in this guide comes from running that file.

1 What Is a Recommendation System?

Ask a few, learn the rest.

A recommendation system predicts how much a person will like something they have not tried yet, and suggests the best items. It learns from two kinds of signals:

Signal	Example on a platform	Example in a city
Explicit: the person tells you	stars, likes, dislikes	a comfort vote, a review score
Implicit: the person shows you	watch time, replays, swipes away	route choice, how long people stay, which streets they avoid



Three families of methods appear in this guide:

- **Baselines:** recommend what is popular. Simple, and surprisingly hard to beat.
- **Collaborative filtering:** people who agreed in the past will agree again. Uses only the ratings.
- **Content-based:** describe each item with features (here: sun, green, noise) and learn each person's weights on those features.

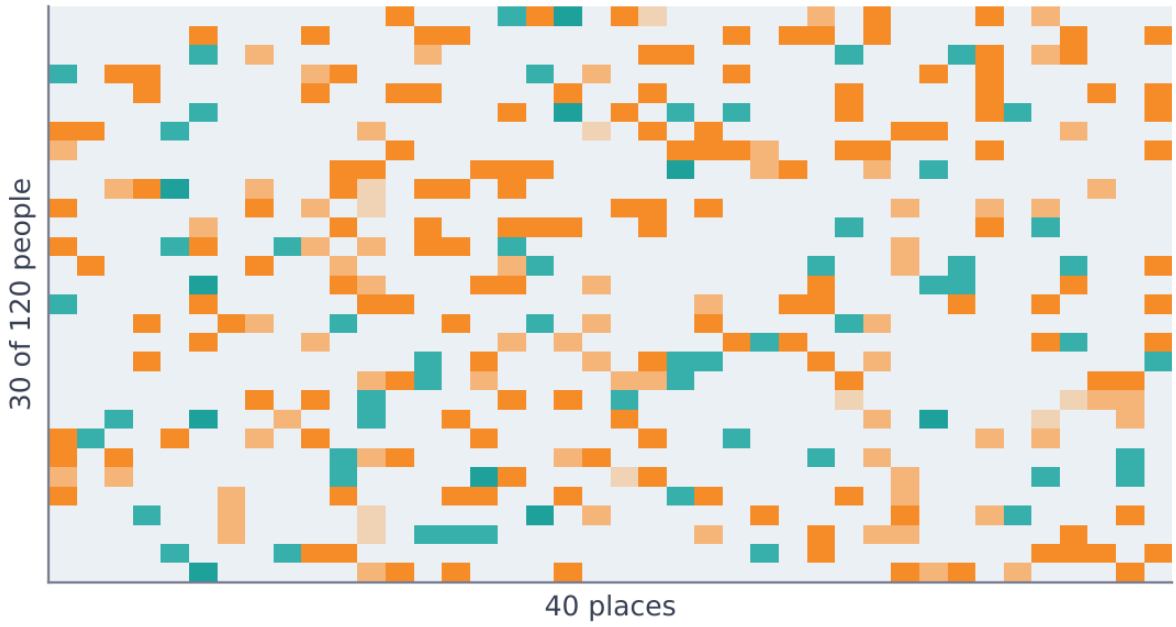
> **IN ONE LINE**
>
> A recommendation system fills in a mostly empty table of people × items, and ranks the items each person is most likely to enjoy.

2 The Rating Matrix

Lab 1 · 120 people, 40 places, mostly empty.

Put people in rows and places in columns. A cell holds a rating from 1 to 5 stars when that person rated that place, and is empty otherwise. Real rating matrices are very sparse: most people have rated almost nothing.

Each coloured cell is one rating (orange 1 … teal 5). Grey cells are unknown: 75% of the matrix.




```
F = np.c_[sun, green, noise]          # 40 places x 3 features from the 3D city model
R = np.full((120, 40), np.nan)        # ratings: 120 people x 40 places, unknown = NaN
# each person rated 10 places, 1 to 5 stars; we hold out 2 per person to test
```

```
people: 120 | places: 40 | ratings observed: 1200 (25% of the matrix)
person 0 rated places: [12, 16, 17, 18, 20, 21, 27, 29, 33, 35] -> [3.0, 4.0, 3.0, 5.0, 3.0, 1.0,
2.0, 3.0, 3.0, 2.0]
```

How we test. From each person's 10 ratings we hide 2. Every model learns from the other 8, then predicts the hidden ones. The error is the **RMSE**, the root mean squared error in stars: an RMSE of 0.7 means the predictions are off by about 0.7 stars.

TRY THIS · EXERCISE 1

1. Person 0 rated place 21 one star and place 18 five stars. What else would you want to know about those two places before recommending a third?
2. Why must we hide ratings per person, and not hide whole people?

3 Baselines

Lab 2 · the scores every model must beat.

The simplest guess for any cell is the mean of all ratings. A better guess is the mean rating of that place: some places are liked by almost everyone. A place rated by only two people should not get an extreme average, so we **shrink** the average toward the overall mean, as if every place had three extra average ratings.

```
def place_mean():
    mu = R[train].mean()                              # mean of all known ratings
    s = np.where(train, R, 0).sum(0); c = train.sum(0)
    pm = (s + 3 * mu) / (c + 3)                       # shrink rarely-rated places toward mu
    return np.tile(pm, (N, 1))
```

```
RMSE, global mean: 0.82
RMSE, place mean:  0.72
```

Knowing which places are generally pleasant already cuts the error from 0.82 to 0.72 stars. But this baseline gives every person the same list. A person who loves sun gets the same recommendations as a person who hides from sun.

WATCH OUT

Always report a baseline. A complex model that cannot beat “place mean” is not worth deploying, and a popularity list is often what a city agency already uses.

TRY THIS · EXERCISE 2

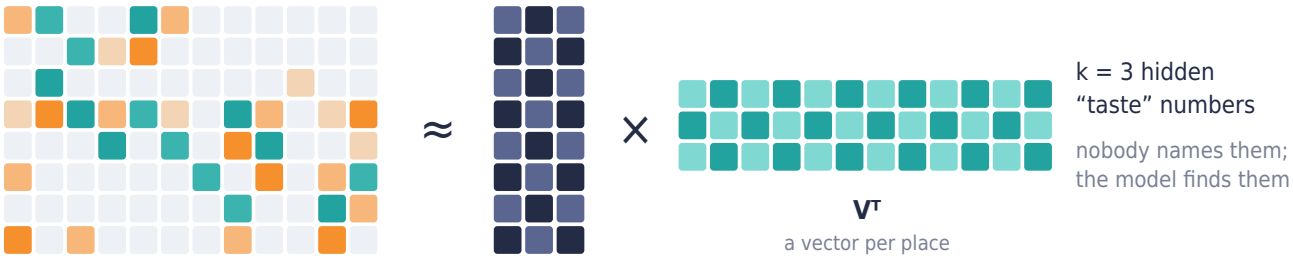
Change the shrinkage from 3 to 0 and to 20. What happens to the RMSE, and why?

4 Matrix Factorisation

Lab 3 · hidden tastes, found from the ratings alone.

Matrix factorisation guesses that each rating comes from a few hidden “taste” numbers. Every person gets a vector u_i and every place a vector v_j, both of length k. The predicted rating is their dot product plus a few biases:

$$\hat{r}_{ij} = \mu + b_i + b_j + u_i \cdot v_j$$



ratings R (people × places)

U

a taste vector per person

V^T

a vector per place

Training nudges every vector to reduce the error on the known ratings, one rating at a time (stochastic gradient descent, as in Koren, Bell & Volinsky, 2009). The term `reg` keeps the vectors small, so the model does not memorise noise.

```
for epoch in range(300):
    for i, j in shuffled(known_ratings):
        e = R[i, j] - (mu + bu[i] + bv[j] + U[i] @ V[j])   # error of the current guess
        bu[i] += lr * (e - reg * bu[i])                     # is this person generous?
        bv[j] += lr * (e - reg * bv[j])                     # is this place liked by all?
        U[i], V[j] = (U[i] + lr * (e * V[j] - reg * U[i]),  # move taste toward the place
                      V[j] + lr * (e * U[i] - reg * V[j]))  # move place toward the taste
prediction = mu + bu[:, None] + bv[None, :] + U @ V.T
```

```
RMSE, matrix factorisation (k=3): 0.69
```

Matrix factorisation (0.69) beats the place mean (0.72): the model now gives each person a personal list. Notice that the model was never told about sun, green or noise. The vectors found something similar on their own from the ratings.

5 Cold Start

Lab 4 · a new person, three questions.

Matrix factorisation needs ratings from a person before it has a vector for that person. A new resident or a new visitor has none. This is the **cold start** problem. One answer: ask a few, carefully chosen questions, and combine the answers with what everybody likes on average.

We ask the new person to rate three very different places: the sunniest, the greenest and the loudest. The new person loves sun and does not care much about green or noise, which is unusual in this population.

```
asked = [argmax(sun), argmax(green), argmax(noise)]    # 3 very different places
answers = new_person_ratings(asked)                     # e.g. [4, 3, 4]
X = Fc[asked]                                           # their features (+ a constant)
w = w0 + solve(X.T @ X + 0.3 * I, X.T @ (answers - X @ w0))   # start from the average taste
top5 = argsort(-(Fc @ w))[:5]
```

```
without asking (most popular places): [4, 5, 32, 8, 17] -> true comfort [1.8, 4.0, 2.5, 1.9, 1.7]
| mean 2.4
asked about places [5, 32, 23] (sunniest, greenest, loudest) -> answers [4.0, 3.0, 4.0]
top 5 recommended places: [5, 22, 13, 23, 12]
their true comfort for this person: [4.0, 4.0, 3.7, 4.2, 3.3] | mean 3.9
```

The popular list would give this person places they rate 2.4 on average. After only three answers, the personal list scores 3.9. Three questions cost the person ten seconds.

TRY THIS · EXERCISE 3

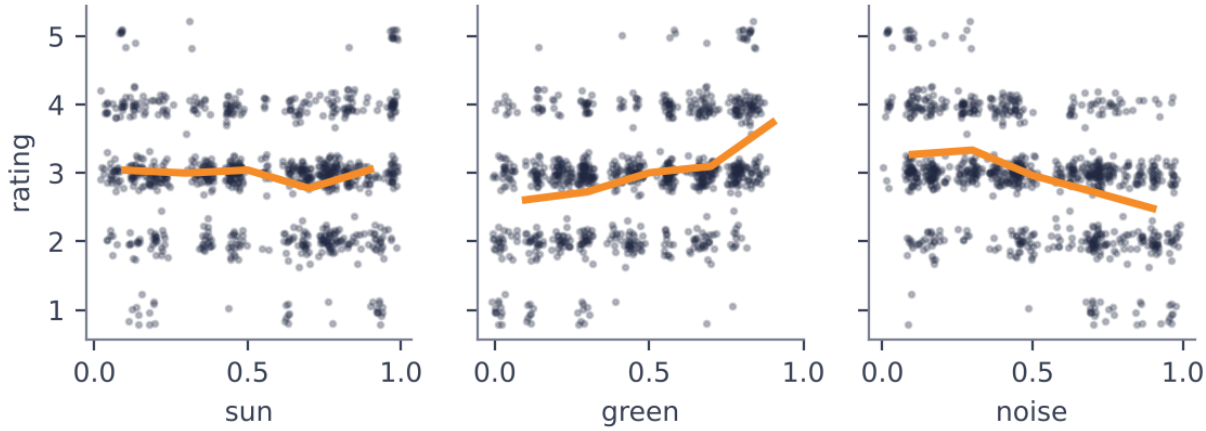
1. Ask about three random places instead. Is the list still as good? Why does the choice of questions matter?
2. In a smartwatch survey such as Cozie, which three questions would you ask a new occupant on the first day?

L5 Use the City

Place features from the 3D city model.

Each place in this lab has three numbers that a city model can compute: the share of the day in direct sun, the share of green within 50 m, and traffic noise. In a real project, sun and shade come from a 3D model such as OctreeCity, green from satellite or street-view images, and noise from a noise map.

All ratings against each place feature (orange: average). Green helps, noise hurts, sun splits people.



The content-based model learns what everybody likes on average, then lets each person differ a little, with a penalty that keeps people with few ratings close to the average.

```
w0 = lstsq(Fc[all_rated], all_ratings)          # what everybody likes, on average
for each person i:
    X = Fc[places i rated]; y = ratings of i - X @ w0
    d = solve(X.T @ X + 2.0 * I, X.T @ y)       # how person i differs from the average
    prediction[i] = Fc @ (w0 + d)               # works for every place, rated or not
```

```
RMSE, place features per person: 0.65
RMSE, hybrid (mean of the two):   0.65
a brand-new place (sun 0.2, green 0.9, noise 0.1): matrix factorisation has no vector for it;
the feature model can still score it. Predicted mean comfort over all people: 3.92 (1 to 5)
```



RMSE on 240 held-out ratings, in stars (lower is better)

Using the city gives the lowest error, 0.65 stars. A second advantage matters even more for cities: the feature model can score a **brand-new place**, such as a planned square, before anybody has visited. Matrix factorisation cannot.

6 From Comfort to the City

Actions are votes, and most are already recorded.

In this guide people rated places on purpose. In a real city most signals are implicit: what people do, not what they say. Each of these is a rating you do not have to ask for.

What people do	Where it is recorded	What a city can learn
Writes a review	Google Maps, Dianping, Weibo	how a place feels: comfort, noise, safety
Picks a route, skips a street	Amap, Baidu Maps, DiDi	which streets people avoid, and when
Stays or leaves quickly	phone location data	where people choose to spend time
Overrides the thermostat	building logs, smart thermostats	where indoor comfort fails

CHALLENGE A · IMPLICIT FEEDBACK

Replace stars with “stayed longer than 20 minutes” (1) or “left” (0). Change the error from RMSE to log-loss. Does the ranking of models change?

CHALLENGE B · FAIRNESS

Split the 120 people into two groups with different tastes, and give one group three times fewer ratings. Measure RMSE per group. Who does the model serve worse? (See also Quintana et al., *Nature Cities*, 2025: perception of streets varies across demographics and personalities.)

CHALLENGE C · A GRAPH VIEW

Draw people and places as a graph, with an edge for every rating. Matrix factorisation is close to one round of message passing on that graph. Compare with the GNN guide of this series.

Pitfalls and Glossary

Read this before you use recommenders on real data.

Pitfalls checklist

- **Popularity bias.** Popular places get more ratings, so they get recommended more, so they get more ratings.
- **Feedback loops.** People can only rate what the system showed them. Mix in some exploration.
- **Who is missing.** Reviews come from people with phones and time. Older residents and visitors are under-represented.
- **Leakage.** Test only on ratings the model never saw, and split by time when the data has dates.
- **Privacy.** Location traces identify people. Aggregate, and never publish raw trajectories.
- **RMSE is not the goal.** In practice the top of the list matters most; also measure precision@k or NDCG.
- **Explain.** “Recommended because it is shaded and quiet” builds more trust than a score.

Glossary

Rating matrix
People × items; most cells unknown.

Explicit feedback
A rating the person gives on purpose.

Implicit feedback
Behaviour that reveals preference: clicks, stays, routes.

Baseline
A simple model that every new model must beat.

Shrinkage
Pulling averages from few ratings toward the global mean.

Collaborative filtering
Recommending from the ratings of similar people.

Matrix factorisation
Writing ratings as a product of person and item vectors.

Embedding
A learned vector for a person or an item.

Cold start
A new person or item with no ratings yet.

Content-based
Recommending from item features.

Hybrid
Combining collaborative and content-based models.

RMSE
Root mean squared error of predicted ratings.

One-Page Cheat Sheet

Everything in this guide on one page.

DATA

R: N people × M items, mostly unknown. Hold out ratings per person to test.

ERROR

$\text{RMSE} = \sqrt{\text{mean}(\ (\hat{r} - r)^2\ )}$ over held-out ratings

PLACE MEAN WITH SHRINKAGE

$(\Sigma\, r + \lambda\, \mu)\ /\ (n + \lambda)$

MATRIX FACTORISATION

$\hat{r}_{ij} = \mu + b_i + b_j + u_i \cdot v_j$

SGD STEP

$e = r - \hat{r} \cdot u \leftarrow u + lr\,(e\,v - reg\,u) \cdot v \leftarrow v + lr\,(e\,u - reg\,v)$

CONTENT-BASED

$\hat{r}_{ij} = f_j \cdot (w_0 + d_i)$, ridge on d_i

COLD START

Ask 2–5 diverse items; start from the average taste.

NEW ITEM

Only content-based models can score it before anyone rates it.

LAB RESULTS (SYNTHETIC)

Global 0.82 · place mean 0.72 · MF 0.69 · city features 0.65. Cold start 2.4 → 3.9.

RANKING METRICS

precision@k · recall@k · NDCG.

GRAPH VIEW

People and places as nodes, ratings as edges: see the GNN guide.

LIBRARIES

numpy to learn · Surprise, implicit, LightFM, PyTorch for real projects.

Further Reading

The papers behind each idea.

1. Koren, Y., Bell, R., & Volinsky, C. (2009). Matrix factorization techniques for recommender systems. *Computer*, 42(8), 30–37.
2. Resnick, P., Iacovou, N., Suchak, M., Bergstrom, P., & Riedl, J. (1994). GroupLens: an open architecture for collaborative filtering of netnews. *CSCW*.
3. Sarwar, B., Karypis, G., Konstan, J., & Riedl, J. (2001). Item-based collaborative filtering recommendation algorithms. *WWW*.
4. Hu, Y., Koren, Y., & Volinsky, C. (2008). Collaborative filtering for implicit feedback datasets. *ICDM*.
5. Covington, P., Adams, J., & Sargin, E. (2016). Deep neural networks for YouTube recommendations. *RecSys*.
6. He, X., Liao, L., Zhang, H., Nie, L., Hu, X., & Chua, T.-S. (2017). Neural collaborative filtering. *WWW*.
7. Schein, A. I., Popescul, A., Ungar, L. H., & Pennock, D. M. (2002). Methods and metrics for cold-start recommendations. *SIGIR*.
8. Ricci, F., Rokach, L., & Shapira, B. (Eds.) (2022). *Recommender Systems Handbook* (3rd ed.). Springer.
9. Jayathissa, P., Quintana, M., Abdelrahman, M., & Miller, C. (2020). Humans-as-a-sensor for buildings: intensive longitudinal indoor comfort models. *Buildings*, 10(10), 174.
10. Quintana, M., et al. (2025). Global urban visual perception varies across demographics and personalities. *Nature Cities*.

Files · `rec_lab.py` (all labs) · `rec_lab.ipynb` (the same code as a notebook)

Part of the Fundamentals 101 series · mahmoud.work