Recommendation Systems :-

1) Content Based

- Given an item, other items having similar features are recommended.

- uses distance metrics
  a) euclidean
  b) manhattan
  c) cosine - similarity.

2) Collaborative

- Given past history of similar users & history of items bought together, recommend new items to the user.

- uses pearson correlation.

:- Collaborative Recommendation System :-

a) user based recommendation :- - Based on the purchase history of user, similar users are identified.
- Items, that are not present in the target user's list but are present in other user's list are recommended.

b) Item based recommendation :- - Based on the items bought by target user.
- Items that are bought together with the user's current selection are recommended.

Cold start problem :- When there is no user data, for user based a new user.
- Collaborative recommendations are useless since there is no collaboration yet.
- use hybrid model to counter this.

h.w → write fn to recommend top 5 similar movies.
- Input → (movie, similarity)

LightFM → Library for recommendations.
- uses matrix factorization & limited deep learning.
- Fast & good with sparse data.
- Handles cold-start problem well.

```
dataset = lightfm.data.Dataset()
dataset.fit(users, movies)  → Internally indexes users & movies.
```

```
train_data.itertuples()
```
- Iterates the pandas data frame as (index, row) → Contains the columns.
- Can be slow but readable.
  - for large dataframe.
  - for small–medium → Very readable & intuitive.

itertuples() → faster version of itertuples().

Storing a Sparse Matrix →

a) **Coo format →** Co-ordinate format.

Sparse matrix → (row, col, data)
only for non-zero entries.

b) **CSR format →** Compressed sparse row format.
- Converts a sparse matrix into three rows.
  a) data.
  b) column index.
  c) Row index.
- Very optimal for matrix slicing & vector matrix multiplication.

LightFM → - uses csr format internally for calculations.
- use coo for easy insertions.

```
interactions, weights = dataset.build_interactions(
    [(user_id, item_id, rating) for user_id, item_id,
     rating in my_dataset])

fn recommend_movie_title_for_user(model, dataset, user_id,
                                  movies_df, n=5)

    // get mappings
    users_map, _, items_map, _ = dataset.mapping()

    internal_user_id = users_map[user_id]
    total_items = len(items_map)

    // get scores
    scores = model.predict(internal_user_id, np.arange(total_
                                                       items))

// get top n items
    top_items = np.argsort(-scores)[:n]

// get external movie ids
    inv_items_map = {val: key for key, val in items_map.items()}
                     external_id  internal_id
    recommended_ids = [inv_items_map[internal_id] for internal_id in
                       top_items]

    recommends = movies_df[movies_df['movie_id'].isin(recommended_
                                                      movie_ids)]
    r_movies = recommends.set_index('movie_id').loc[recommended_ids].
                                                    reset_index()
```

Hybrid Recommendation

schema for train data

user, item, rating

By Content based

Requires
similarity matrix
based on embeddings.

By Collaborative

similarity matrix
learnt by the light fm
model.

```
fn(itemid, dataset, model, cmat, df, n=5):      # model → lightfm model.
    imap = dataset.mapping()
    rimap = { --- }
    c2 = model.embeddings()
    c2 = cosine_similarity(c2)
    s1 = cmat[itemid]  ]
    s2 = c2[itemid]    ] → convert both of them to pandas series.
    top1 = np.argsort(-s1)[1:n+1]
    top2 = np.argsort(-s2)[1:n+1]
    items1 = df[df['p_id'].isin(top1)]
    items2 =   "   "      "      "  top2 " "
    // arrange in correct order of scores →
      i1 = items1.set_index('pid').loc[top1].reset_index()
      i2 =   "  2    "     "      "    top2   "   "   "
      return [i1, i2]
```