```
# === 1) LOAD ===
import pandas as pd
import numpy as np

movies = pd.read_csv("movies.csv")
ratings = pd.read_csv("ratings.csv")

print(movies.shape)   # e.g., (62423, 3)
print(ratings.shape)  # e.g., (453712, 4)
display(movies.head())
display(ratings.head())

# --- Join ratings with movie text (title + genres) ---
mov = movies.copy()
mov["text"] = (mov["title"].fillna("") + " | " + mov["genres"].fillna("")).str.lower()
df_full = ratings.merge(mov[["movieId","text","title","genres"]], on="movieId", how="inner")

# For per-user training we'll define like/dislike later (y) using >=4 as like, <=2.5 as dislike (drop neutrals)
```

```
(62423, 3)
(43149, 4)
```

	movieId	title	genres
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy
1	2	Jumanji (1995)	Adventure|Children|Fantasy
2	3	Grumpier Old Men (1995)	Comedy|Romance
3	4	Waiting to Exhale (1995)	Comedy|Drama|Romance
4	5	Father of the Bride Part II (1995)	Comedy

	userId	movieId	rating	timestamp
0	1	296	5.0	1.147880e+09
1	1	306	3.5	1.147869e+09
2	1	307	5.0	1.147869e+09
3	1	665	5.0	1.147879e+09
4	1	899	3.5	1.147869e+09

```
# === 2) A HELPER to build a per-user dataset (like your spam PREPARE DATA) ===
from sklearn.model_selection import train_test_split

def make_user_dataset(user_id, neutral_cut=3.0):
    """Return X_train, X_test, y_train, y_test, pool_unseen for a given user.
       Likes: rating >= 4.0; Dislikes: rating <= 2.5; Drop neutrals (around 3.0).
       pool_unseen = all movies the user hasn't rated yet (to rank & recommend)."""
    user_df = df_full[df_full["userId"] == user_id].copy()
    if user_df.empty:
        raise ValueError(f"User {user_id} not found.")

    # label: 1 = like, 0 = dislike; drop neutrals near 3
    user_df = user_df.assign(
        y = np.where(user_df["rating"] >= 4.0, 1,
             np.where(user_df["rating"] <= 2.5, 0, np.nan))
    ).dropna(subset=["y"])
    user_df["y"] = user_df["y"].astype(int)

    # Text features (title+genres string)
    X = user_df["text"].astype(str)
    y = user_df["y"].astype(int)

    # Keep a pool of unseen movies for recommendation
    seen_ids = set(user_df["movieId"].unique())
    pool_unseen = mov[~mov["movieId"].isin(seen_ids)].copy()

    # Guard: need at least some positives and negatives
    pos, neg = (y==1).sum(), (y==0).sum()
    if pos < 5 or neg < 5:
        print(f"⚠️ User {user_id} has limited labels (pos={pos}, neg={neg}). Results may be weak.")

    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.25, stratify=y, random_state=42
    )
    return X_train, X_test, y_train, y_test, pool_unseen
```

```
# === 3) PREPARE MODELS (very similar to your spam project) ===
from sklearn.pipeline import Pipeline
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression, SGDClassifier
from sklearn.naive_bayes import MultinomialNB, ComplementNB
from sklearn.svm import LinearSVC
from sklearn.calibration import CalibratedClassifierCV

pipelines = {
    "LogisticRegression": Pipeline([
        ("tfidf", TfidfVectorizer(ngram_range=(1,2), min_df=2)),
        ("clf", LogisticRegression(max_iter=200, class_weight="balanced", random_state=42)),
    ]),
    "MultinomialNB": Pipeline([
        ("tfidf", TfidfVectorizer(ngram_range=(1,2), min_df=2)),
        ("clf", MultinomialNB()),
    ]),
    "ComplementNB": Pipeline([
        ("tfidf", TfidfVectorizer(ngram_range=(1,2), min_df=2)),
        ("clf", ComplementNB()),
    ]),
    # LinearSVC is strong; wrap for predict_proba
    "LinearSVC+Calibrated": Pipeline([
        ("tfidf", TfidfVectorizer(ngram_range=(1,2), min_df=2)),
        ("clf", CalibratedClassifierCV(LinearSVC(random_state=42), cv=3)),
    ]),
    "SGD (log loss)": Pipeline([
        ("tfidf", TfidfVectorizer(ngram_range=(1,2), min_df=2)),
        ("clf", SGDClassifier(loss="log_loss", class_weight="balanced", random_state=42)),
    ]),
}
```

```
# === 4) TRAIN & EVALUATE for a chosen user (like spam training loop) ===
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

user_id = 1  # <-- change to any existing userId
X_train, X_test, y_train, y_test, pool_unseen = make_user_dataset(user_id)

results = []
best_name, best_acc = None, -1.0
for name, pipe in pipelines.items():
    pipe.fit(X_train, y_train)
    pred = pipe.predict(X_test)
    acc = accuracy_score(y_test, pred)
    results.append((name, acc))
    if acc > best_acc:
        best_acc, best_name = acc, name
    print(f"\n=== {name} ===")
    print("Accuracy:", round(acc, 3))
    print(classification_report(y_test, pred, target_names=["dislike","like"], digits=3))

print("\nSUMMARY:", sorted(results, key=lambda x: x[1], reverse=True))
best_pipe = pipelines[best_name]
print(f"\nBest model: {best_name} | Accuracy: {best_acc:.3f}")

cm = confusion_matrix(y_test, best_pipe.predict(X_test))
print("Confusion matrix [[TN, FP],[FN, TP]]:\n", cm)

import matplotlib.pyplot as plt
plt.figure(figsize=(4,4))
plt.imshow(cm, interpolation='nearest')
plt.title("Confusion Matrix")
plt.xticks([0,1], ["dislike","like"])
plt.yticks([0,1], ["dislike","like"])
for i in range(2):
    for j in range(2):
        plt.text(j, i, cm[i, j], ha="center", va="center")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.tight_layout()
plt.show()
```

```
=== LogisticRegression ===
Accuracy: 0.615
              precision    recall  f1-score   support

     dislike      0.000     0.000     0.000         3
        like      0.727     0.800     0.762        10

    accuracy                          0.615        13
   macro avg      0.364     0.400     0.381        13
weighted avg      0.559     0.615     0.586        13


=== MultinomialNB ===
Accuracy: 0.769
              precision    recall  f1-score   support
```

```
# === 5) RECOMMEND (score all unseen movies for this user and show Top-N) ===
def predict_proba_safe(pipe, texts):
    if hasattr(pipe, "predict_proba"):
        try:
            return pipe.predict_proba(texts)[:,1]
        except Exception:
            pass
    if hasattr(pipe, "decision_function"):
        from scipy.special import expit
        return expit(pipe.decision_function(texts))
    # fallback: hard predictions only -> map to {0,1} as "prob"
    preds = pipe.predict(texts)
    return preds.astype(float)

def recommend_for_user(pipe, user_id, pool_unseen, top_n=10):
    texts = pool_unseen["text"].astype(str)
    probs = predict_proba_safe(pipe, texts)
    pool_unseen = pool_unseen.assign(p_like=probs)
    top = pool_unseen.sort_values("p_like", ascending=False).head(top_n)
    return top[["title","genres","p_like"]]

top10 = recommend_for_user(best_pipe, user_id, pool_unseen, top_n=10)
display(top10)
```

	title	genres	p_like
26000	The Helen Morgan Story (1957)	Drama|Romance	0.945439
25879	Sea Wife (1957)	Drama|Romance	0.944567
36293	Cinderella (1957)	Drama|Romance	0.944567
7232	Peyton Place (1957)	Drama|Romance	0.944567
31482	Bombers B-52 (1957)	Drama|Romance	0.944567
15155	Kisses (Kuchizuke) (1957)	Drama|Romance	0.944567
54751	Berlin, Schoenhauser Corner (1957)	Drama|Romance	0.944567
38248	Rot ist die Liebe (1957)	Drama|Romance	0.944567
11052	Notti bianche, Le (White Nights) (1957)	Drama|Romance	0.944567
11108	Raintree County (1957)	Drama|Romance	0.944567

```
    accuracy                          0.769        13
   macro avg      0.385     0.500     0.435        13

              precision    recall  f1-score   support

    accuracy                          0.692        13
weighted avg      0.577     0.692     0.629        13
                                                    brated', 0.7692307692307693), ('ComplementNB', 0.692307692307692
Best model: MultinomialNB | Accuracy: 0.769
Confusion matrix [[TN, FP],[FN, TP]]:
[[ 0  3]
 [ 0 10]]
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defined
```

Next steps: Generate code with top10 New interactive sheet

```
# === 6) TINY GUI (like your spam GUI) ===
import ipywidgets as widgets
from IPython.display import display, clear_output

user_in   = widgets.IntText(value=1, description="User ID:")
model_dd  = widgets.Dropdown(options=list(pipelines.keys()), value=best_name, description="Model:")
train_btn = widgets.Button(description="(Re)Train", button_style="primary")
topn_in   = widgets.IntSlider(value=10, min=5, max=30, step=1, description="Top-N:")
pred_btn  = widgets.Button(description="Recommend", button_style="")
out       = widgets.Output(layout={'border': '1px solid #ddd'})

state = {"pipe": best_pipe, "name": best_name, "acc": best_acc,
         "X_train": X_train, "X_test": X_test, "y_train": y_train, "y_test": y_test,
         "pool": pool_unseen, "user": user_id}
```

```
def retrain_for_user(uid, name):
    X_tr, X_te, y_tr, y_te, pool = make_user_dataset(uid)
    pipe = pipelines[name]
    pipe.fit(X_tr, y_tr)
    acc = accuracy_score(y_te, pipe.predict(X_te))
    return pipe, acc, X_tr, X_te, y_tr, y_te, pool

def on_train_clicked(_):
    with out:
        clear_output(wait=True)
        uid  = int(user_in.value)
        name = model_dd.value
        try:
            pipe, acc, X_tr, X_te, y_tr, y_te, pool = retrain_for_user(uid, name)
            state.update({"pipe": pipe, "name": name, "acc": acc,
                          "X_train": X_tr, "X_test": X_te, "y_train": y_tr, "y_test": y_te,
                          "pool": pool, "user": uid})
            print(f"✅ Trained {name} for User {uid} | Test Acc: {acc:.3f} | Train size: {X_tr.shape[0]}")
        except Exception as e:
            print("Error:", e)

def on_recommend_clicked(_):
    with out:
        if state["pipe"] is None:
            print("Please train a model first.")
            return
        topn = int(topn_in.value)
        recs = recommend_for_user(state["pipe"], state["user"], state["pool"], top_n=topn)
        print(f"Top-{topn} recommendations for User {state['user']} (Model: {state['name']}, Acc {state['acc']:.3f}):")
        display(recs)

train_btn.on_click(on_train_clicked)
pred_btn.on_click(on_recommend_clicked)

ui = widgets.VBox([
    widgets.HBox([user_in, model_dd, train_btn]),
    widgets.HBox([topn_in, pred_btn]),
    out
])
display(ui)

with out:
    print(f"Ready. Current user={state['user']} | Model={state['name']} (Acc {state['acc']:.3f}). Click Recommend.")
```

User ID: 1   Model: MultinomialNB   (Re)Train

Top-N: 10   Recommend

```
✅ Trained MultinomialNB for User 1 | Test Acc: 0.769 | Train size: 36
```