The Github link is as follows for both parts 1 and 2: https://github.com/aaelim/CSCA-5632-Week-4-Project.

1. The purpose of this notebook is to fill in the missing MovieLens-1M ratings, to report the RMSE that results, and to discuss why this is not an effective way compared to results obtained in Homework 3. To run the notebook for yourself, make sure to have downloaded the movie data files, have all relevant python librairies installed, and that all paths are set to your actual file locations. The link to the MovieLen-1M ratings is provided as https://www.kaggle.com/datasets/odedgolden/movielens-1m-dataset. Iterations can be reduced to 300 on slower machines and still result in RMSE around those with this notebook's 1000 iterations.

•[1]:
```
from pathlib import Path
import pandas as pd, numpy as np
from sklearn.model_selection import train_test_split
from sklearn.decomposition import NMF
from sklearn.metrics import mean_squared_error

# Folder that contains BBC files + movies.dat / ratings.dat / users.dat
# Set The path to your personal path to run the notebook
DATA_DIR = (
    Path.home()
    / "Documents"
    / "University Degrees"
    / "University of Colorado Boulder"
    / "Current Courses"
    / "CSCA 5632 Unsupervised Algorithms in Machine Learning (IN PROGRESS - Active - Assignments Remain)"
    / "Week 4"
)

ratings_file = DATA_DIR / "ratings.dat"
movies_file  = DATA_DIR / "movies.dat"
users_file   = DATA_DIR / "users.dat"

print("Path exists:", DATA_DIR.exists())
print("ratings.dat present:", ratings_file.exists())
```

```
Path exists: True
ratings.dat present: True
```

This next cell acts to confirm that the data is correctly loaded.

•[2]:
```
cols = ["userId", "movieId", "rating", "timestamp"]
ratings = pd.read_csv(
    ratings_file,
    sep="::",
    names=cols,
    engine="python",
    usecols=[0, 1, 2]
)

print(ratings.head())
print("Shape:", ratings.shape, "Unique users:", ratings.userId.nunique(),
      "Unique items:", ratings.movieId.nunique())
```

```
   userId  movieId  rating
0       1     1193       5
1       1      661       3
2       1      914       3
3       1     3408       4
4       1     2355       5
Shape: (1000209, 3) Unique users: 6040 Unique items: 3706
```

The metadata is not used by non-negative matrix factorization, but is included for hybrid models and tuning.

•[3]:
```
cols_mov = ["movieId", "title", "genres"]
movies   = pd.read_csv(
    movies_file,
    sep="::",
    names=cols_mov,
    engine="python",
    encoding="latin-1"      #fixes UnicodeDecodeError
)

cols_usr = ["userId", "gender", "age", "occupation", "zip"]
users    = pd.read_csv(
    users_file,
    sep="::",
    names=cols_usr,
    engine="python",
    encoding="latin-1"      # same safeguard as above for UnicodeDecodeError
)

print("movies:", movies.shape, "users:", users.shape)
print(movies.head(3))
```

```
movies: (3883, 3) users: (6040, 5)
   movieId                    title                        genres
0        1         Toy Story (1995)   Animation|Children's|Comedy
1        2           Jumanji (1995)  Adventure|Children's|Fantasy
2        3  Grumpier Old Men (1995)                Comedy|Romance
```

This is a custom 80/20 split per user to ensure all users appear in train.

•[4]:
```
def stratified_holdout(df, test_size=0.2, seed=42):
    np.random.seed(seed)
    test_idx = []
    for u, grp in df.groupby("userId"):
        # ensures at least one rating in the train
        n_test = max(1, int(len(grp) * test_size))
        test_idx += list(grp.sample(n_test, random_state=seed).index)
    test = df.loc[test_idx]
    train = df.drop(index=test_idx)
    return train.reset_index(drop=True), test.reset_index(drop=True)

train_ratings, test_ratings = stratified_holdout(ratings, 0.2)
print(train_ratings.shape, test_ratings.shape)
```

```
(802553, 3) (197656, 3)
```

Unobserved entries get forced to zero.

•[5]:
```
n_users  = ratings.userId.nunique()
n_items  = ratings.movieId.nunique()

user_map  = {u:i for i,u in enumerate(ratings.userId.unique())}
item_map  = {m:i for i,m in enumerate(ratings.movieId.unique())}

def df_to_matrix(df):
    mat = np.zeros((n_users, n_items), dtype=np.float32)
    for row in df.itertuples(index=False):
        mat[user_map[row.userId], item_map[row.movieId]] = row.rating
    return mat

R_train = df_to_matrix(train_ratings)
```

Sets parameters and leaves all other parameters at default to provide a straight baseline.

•[6]:
```
k = 20                    # latent factors
nmf = NMF(n_components=k, init="random",
          solver="mu",     # multiplicative update
          beta_loss="frobenius",
          max_iter=1000, random_state=0)
W = nmf.fit_transform(R_train)   # shape (users, k)
H = nmf.components_              # shape (k, items)
R_pred_full = np.dot(W, H)       # reconstructed ratings
```

Shows that the RMSE is much worse than the RMSE values obtained in Homework 3, showing that applying scikit-learn's NMF to sparse ratings matrix does not work well.

[7]:
```
def predict_row(row):
    u = user_map[row.userId]; i = item_map[row.movieId]
    return R_pred_full[u, i]

test_ratings["pred"] = test_ratings.apply(predict_row, axis=1)
rmse = np.sqrt(
    ((test_ratings["rating"] - test_ratings["pred"]) ** 2).mean()
)
print(f"RMSE (sklearn NMF, k={k}): {rmse:.4f}")
```

```
RMSE (sklearn NMF, k=20): 2.7461
```

2. The results show a NMF giving a RMSE of 2.7461, which is significantly higher than the values around 1.0 obtained from simple baseline or similarity-based methods.

The reasons for this are due to this algorithm not being optimised for large recommender matrics, the lack of bias terms, all entries having equal weightings, defaults overfitting sparse data, and predictions getting biased downwards due to scikit-learn minimizing errors over entries of zeroes.

There are multiple potential ways to improve this, such as but not limited to adding global and user and item bias terms then factorizing the residuals, using recommender specific libraries that treat missing values as unknown, or potentially combining MF with k-NN similarity. These fixes would likely result in RMSE at similar levels to those obtained in Homework 3.