CF ASSIGNMENT

Name: Aayush Gakhar
Roll no: 2020006

Step 1: Loading required libraries

```
import numpy as np
import pandas as pd
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.metrics import mean_absolute_error
import seaborn as sns
import matplotlib.pyplot as plt
```

Step 2: Loading data

```
# Load the data
data = pd.read_csv('ml-100k/u.data', sep='\t', header=None,
                   names=['user_id', 'item_id', 'rating', 'timestamp'])

folds = []
n_users = 943
n_items = 1682
for i in range(1, 6):
    train = pd.read_csv('ml-100k/u' + str(i) + '.base', sep='\t', header=None,
                        names=['user_id', 'item_id', 'rating', 'timestamp'])
    test = pd.read_csv('ml-100k/u' + str(i) + '.test', sep='\t', header=None,
                       names=['user_id', 'item_id', 'rating', 'timestamp'])
    ratings = np.zeros((n_users, n_items))

    for row in train.itertuples():
        ratings[row[1]-1, row[2]-1] = row[3]
    folds.append({'train': train, 'test': test, 'ratings': ratings})
```

Step 3: Made functions to do ALS

```python
def als(ratings, K=20, lambda_=0.1, n_iterations=10):
    m, n = ratings.shape
    U = np.random.rand(m, K)
    V = np.random.rand(n, K)
    for i in range(n_iterations):
        # Update user matrix
        for u in range(m):
            V_u = V[ratings[u, :] > 0, :]
            A = np.dot(V_u.T, V_u) + lambda_ * np.eye(K)
            b = np.dot(V_u.T, ratings[u, ratings[u, :] > 0])
            U[u, :] = np.linalg.solve(A, b)
        # Update item matrix
        for v in range(n):
            U_v = U[ratings[:, v] > 0, :]
            A = np.dot(U_v.T, U_v) + lambda_ * np.eye(K)
            b = np.dot(U_v.T, ratings[ratings[:, v] > 0, v])
            V[v, :] = np.linalg.solve(A, b)
    return U, V
```

Step 4: Iterated over all folds and K values

```python
def lfm(ratings, test_data, param):
    # Run ALS
    U, V = als(ratings, K=param[0], lambda_=param[1], n_iterations=param[2])

    # Impute missing values
    ratings_imputed = np.dot(U, V.T)

    y_true = test_data['rating'].values
    y_pred = ratings_imputed[test_data['user_id'] -
                             1, test_data['item_id'] - 1]

    errors = np.array(y_true) - np.array(y_pred)
    nmae = np.mean(np.abs(errors)) / np.mean(np.abs(y_true))
    return nmae
```

s Pyt

```python
def cross_validation(folds, param):
    # Initialize the list of NMAEs
    nmaes = []

    # For each fold
    for fold in folds:
        # Compute the MAE for the algorithm and the fold
        nmae = lfm(fold['ratings'], fold['test'], param)

        # Add the MAE to the list of NMAEs
        nmaes.append(nmae)

    avg_nmae = np.mean(nmaes)
    nmaes.append(avg_nmae)

    # Return the list of NMAEs
    return nmaes
```

```
nmaes = []
params = [(10, 0.1, 10), (10, 0.1, 50), (20, 0.5, 20),
          (7, 0.5, 50), (10, 1, 10), (10, 0.1, 50), (20, 0.5, 20)]
for param in params:
    nmaes.append(cross_validation(folds, param))
```

TABLE for nmae with different parameters

	params(K, lambda, iterations)	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Average
0	(10, 0.1, 10)	0.255809	0.245550	0.249534	0.250585	0.255597	0.251415
1	(10, 0.1, 50)	0.266411	0.251052	0.256275	0.258541	0.267461	0.259948
2	(20, 0.5, 20)	0.271542	0.267112	0.272208	0.271557	0.271301	0.270744
3	(7, 0.5, 50)	0.226252	0.220204	0.221460	0.223701	0.226554	0.223634
4	(10, 1, 10)	0.230087	0.224742	0.226455	0.225682	0.227312	0.226855
5	(10, 0.1, 50)	0.267488	0.258221	0.258304	0.252114	0.263533	0.259932
6	(20, 0.5, 20)	0.271207	0.270691	0.272216	0.272727	0.272760	0.271920