Case Study 4.1 - Movies

Note: If you close this notebook at any time, you will have to run all cells again upon re-opening it.

Note: You may get different numerical results running the notebook different times. This is to be expected, you can just report whatever results you get.

BEGINNER PYTHON

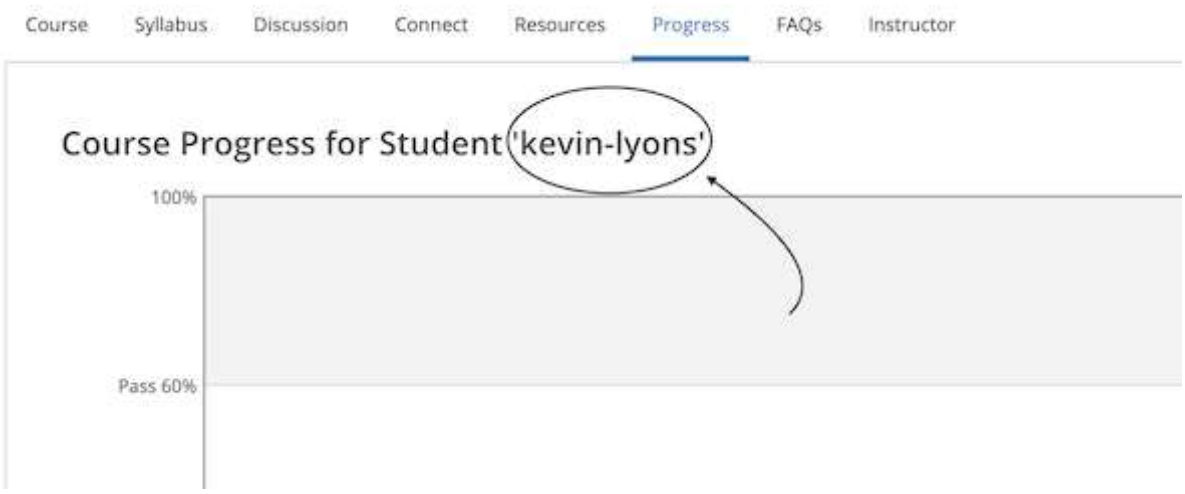
As this is a beginner version, we include a lot of code here to help you along the way.

First, fill in your identification information below. Then, you only have to type in the answers to the questions we ask you. For the rest of the cells, just Run them by pressing the "Run" button above.

Identification Information

You can find your MITxPro username on this page and copying the username in quotations after it says "Student". For example, Kevin's username is "kevin-lyons".

Please use the correct username and the e-mail address right next to it as it makes it much easier for course staff to identify your notebook.



YOUR NAME = Leonardo Pacheco

YOUR MITX PRO USERNAME = leonardo-pacheco

YOUR MITX PRO E-MAIL = wleonardop@gmail.com

Setup

Run these cells to install all the packages you need to complete the remainder of the case study. This may take a few minutes, so please be patient.

```
!pip install --upgrade pip
!pip install surprise==0.1
```

```
Collecting pip
  Downloading https://files.pythonhosted.org/packages/fe/ef/60d7ba03b5c442309ef42e7d
     |████████████████████████████████| 1.5MB 5.7MB/s
Installing collected packages: pip
  Found existing installation: pip 19.3.1
    Uninstalling pip-19.3.1:
      Successfully uninstalled pip-19.3.1
Successfully installed pip-21.0.1
Collecting surprise==0.1
  Downloading surprise-0.1-py2.py3-none-any.whl (1.8 kB)
Collecting scikit-surprise
  Downloading scikit-surprise-1.1.1.tar.gz (11.8 MB)
     |████████████████████████████████| 11.8 MB 5.1 MB/s
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.6/dist-package
Requirement already satisfied: numpy>=1.11.2 in /usr/local/lib/python3.6/dist-packag
Requirement already satisfied: scipy>=1.0.0 in /usr/local/lib/python3.6/dist-package
Requirement already satisfied: six>=1.10.0 in /usr/local/lib/python3.6/dist-packages
Building wheels for collected packages: scikit-surprise
  Building wheel for scikit-surprise (setup.py) ... done
  Created wheel for scikit-surprise: filename=scikit_surprise-1.1.1-cp36-cp36m-linux
  Stored in directory: /root/.cache/pip/wheels/de/9a/41/6a57bf37eb7b50de7f8c7ca9d705
Successfully built scikit-surprise
Installing collected packages: scikit-surprise, surprise
Successfully installed scikit-surprise-1.1.1 surprise-0.1
```

If you do not see any red text, then the install was successful. Yellow text is just warnings, not errors.

Import

Import the required tools into the notebook.

```
import pandas as pd
```

```
import pandas as pd
import matplotlib
from surprise import Dataset, SVD, NormalPredictor, BaselineOnly, KNNBasic, NMF
from surprise.model_selection import cross_validate, KFold
%matplotlib inline
print('Imports successful!')
```

```
Imports successful!
```

Data

Load the MovieLens data. A dialog may pop up saying "**Dataset ml-100k could not be found. Do you want to download it? [Y/n]**" Type Y and hit Enter to start the download process.

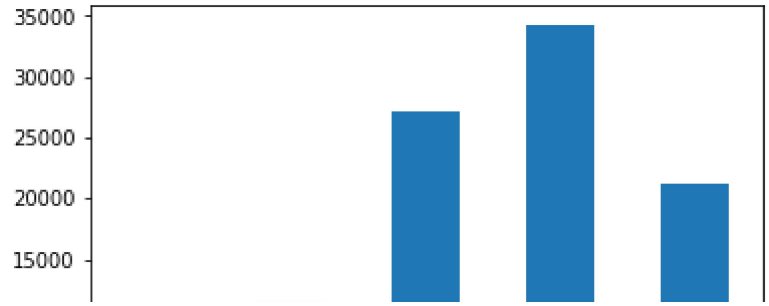
```
data = Dataset.load_builtin('ml-100k')
print('\n\nData load successful!')
```

```
Dataset ml-100k could not be found. Do you want to download it? [Y/n] y
Trying to download dataset from http://files.grouplens.org/datasets/movielens/ml-100
Done! Dataset ml-100k has been saved to /root/.surprise_data/ml-100k


Data load successful!
```

We also want to get a sense of what the data looks like. Let's create a histogram of all the ratings we have in the dataset.

```
# 1. Get the ratings file from the data object
# This is just a filename that has all the data stored in it
ratings_file = data.ratings_file

# 2. Load that table using pandas, a commmon python data loading tool
# We set the column names manually here
col_names = ['user_id', 'item_id', 'rating', 'timestamp']
raw_data = pd.read_table(ratings_file, names=col_names)

# 3. Get the rating column
ratings = raw_data.rating

# 4. Generate a bar plot/histogram of that data
ratings.value_counts().sort_index().plot.bar()

print('\n\nHistogram generation successful!')
```

```
Histogram generation successful!
```



QUESTION 1: DATA ANALYSIS

Describe the dataset. How many ratings are in the dataset? How would you describe the distribution of ratings? Is there anything else we should observe? Make sure the histogram is visible in the notebook.

```
#the following code calculates the number of ratings
print(len(ratings))
```

```
100000
```

Type your response here...

- There are 100000 raitings.
- The distribution of the ratings is single-peaked and skewed-right. The center of the distribution must be some number between 3 and 4. The min and max values of the distribution are 1 and 5, respectively. There are not outliers in this distribution.

Model 1: Random

```
# Create model object
model_random = NormalPredictor()
print('Model creation successful!')
```

```
Model creation successful!
```

```
# Train on data using cross-validation with k=5 folds, measuring the RMSE
model_random_results = cross_validate(model_random, data, measures=['RMSE'], cv=5, verbose
print('\n\nModel training successful!')
```

```
Evaluating RMSE of algorithm NormalPredictor on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    1.5229  1.5180  1.5156  1.5161  1.5200  1.5185  0.0027
```

```
Fit time          0.12    0.16    0.16    0.15    0.15    0.15    0.01
Test time         0.23    0.16    0.22    0.14    0.19    0.19    0.04


Model training successful!
```

Model 2: User-Based Collaborative Filtering

```
# Create model object
model_user = KNNBasic(sim_options={'user_based': True})
print('Model creation successful!')
```

```
Model creation successful!
```

```
# Train on data using cross-validation with k=5 folds, measuring the RMSE
# Note, this may have a lot of print output
# You can set verbose=False to prevent this from happening
model_user_results = cross_validate(model_user, data, measures=['RMSE'], cv=5, verbose=Tru
print('\n\nModel training successful!')
```

```
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Evaluating RMSE of algorithm KNNBasic on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    0.9822  0.9923  0.9793  0.9687  0.9709  0.9787  0.0085
Fit time          0.29    0.32    0.33    0.33    0.34    0.32    0.02
Test time         3.27    3.24    3.35    3.30    3.34    3.30    0.04


Model training successful!
```

Model 3: Item-Based Collaborative Filtering

```
# Create model object
model_item = KNNBasic(sim_options={'user_based': False})
print('Model creation successful!')
```

```
Model creation successful!
```

```
# Train on data using cross-validation with k=5 folds, measuring the RMSE
# Note, this may have a lot of print output
# You can set verbose=False to prevent this from happening
```

```
# You can set verbose=False to prevent this from happening
model_item_results = cross_validate(model_item, data, measures=['RMSE'], cv=5, verbose=Tru
print('\n\nModel training successful!')
```

```
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Evaluating RMSE of algorithm KNNBasic on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    0.9767  0.9803  0.9738  0.9703  0.9723  0.9747  0.0035
Fit time          0.60    0.58    0.57    0.56    0.54    0.57    0.02
Test time         3.88    3.88    3.93    3.91    3.95    3.91    0.03


Model training successful!
```

QUESTION 2: COLLABORATIVE FILTERING MODELS

Compare the results from the user-user and item-item models. How do they compare to each other? How do they compare to our original "random" model? Can you provide any intuition as to why the results came out the way they did?

Type your response here...

- The user-user and the item-item models gave a similar mean RMSE 0.9787 and 0.9747, respectively. Although the item-item performance is slightly better.
- The user-user and the item-item models are more accurated than the random model, in fact the mean RMSE of the latest is 1.5185.
- Collaborative filtering give us a better prediction than the random model, it captures the structure of the data more prescicely

Model 4: Matrix Factorization

```
# Create model object
model_matrix = SVD()
print('Model creation successful!')
```

```
Model creation successful!
```

```
# Train on data using cross-validation with k=5 folds, measuring the RMSE
# Note, this may take some time (2-3 minutes) to train, so please be patient
model_matrix_results = cross_validate(model_matrix, data, measures=['RMSE'], cv=5, verbose
print('\n\nModel training successful!')
```

```
Evaluating RMSE of algorithm SVD on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    0.9339  0.9305  0.9449  0.9356  0.9366  0.9363  0.0048
Fit time          4.67    4.70    4.66    4.70    4.70    4.69    0.01
Test time         0.24    0.14    0.23    0.13    0.24    0.20    0.05


Model training successful!
```

QUESTION 3: MATRIX FACTORIZATION MODEL

The matrix factorization model is different from the collaborative filtering models. Briefly describe this difference. Also, compare the RMSE again. Does it improve? Can you offer any reasoning as to why that might be?

Type your response here...

- To make the prediction the user-user and the item-item models only use the rows and the columns of the matrix, respectively. On the other hand, The matrix factorization model decompose the data in blocks using outer products where both, rows and columns are involved.
- The RMSE of the Matrix factorization model is 0.9363, a better result compared with the other models, indeed the best result was 0.9747, correspondig to theitem-item model.

Precision and Recall @ k

We now want to compute the precision and recall for 2 values of `k`: 5 and 10. We have provided some code here to help you do that.

First, we define a function that takes in some predictions, a value of `k` and a threshold parameter. This code is adapted from here. **Make sure you run this cell.**

```
def precision_recall_at_k(predictions, k=10, threshold=3.5):
    '''Return precision and recall at k metrics for each user.'''

    # First map the predictions to each user.
    user_est_true = dict()
    for uid, _, true_r, est, _ in predictions:
```

```
            current = user_est_true.get(uid, list())
            current.append((est, true_r))
            user_est_true[uid] = current

        precisions = dict()
        recalls = dict()
        for uid, user_ratings in user_est_true.items():

            # Sort user ratings by estimated value
            user_ratings.sort(key=lambda x: x[0], reverse=True)

            # Number of relevant items
            n_rel = sum((true_r >= threshold) for (_, true_r) in user_ratings)

            # Number of recommended items in top k
            n_rec_k = sum((est >= threshold) for (est, _) in user_ratings[:k])

            # Number of relevant and recommended items in top k
            n_rel_and_rec_k = sum(((true_r >= threshold) and (est >= threshold))
                                  for (est, true_r) in user_ratings[:k])

            # Precision@K: Proportion of recommended items that are relevant
            precisions[uid] = n_rel_and_rec_k / n_rec_k if n_rec_k != 0 else 1

            # Recall@K: Proportion of relevant items that are recommended
            recalls[uid] = n_rel_and_rec_k / n_rel if n_rel != 0 else 1

        return precisions, recalls

    print('Function creation successful!')
```

```
Function creation successful!
```

Next, we compute the precision and recall at `k` = 5 and 10 for each of our 4 models. We use 5-fold cross validation again to average the results across the entire dataseat.

Please note that this will take some time to compute.

QUESTION 4: PRECISION/RECALL

Compute the precision and recall, for each of the 4 models, at `k` = 5 and 10. This is 2 x 2 x 4 = 16 numerical values. Do you note anything interesting about these values? Anything differerent from the RMSE values you computed above?

```
#The next function returns the name of the model
def  modelName(model):
    if model == model_random:
        return 'random'
    elif model == model_item:
        return 'item-item'
```

```
        elif model == model_user:
            return 'user-user'
        elif model == model_matrix:
            return 'matrix'
    #definition of the folds and the cycle that is going to compute the precisions al recalls
    kf = KFold(n_splits=5)
    lst = [5, 10]
    mls = [model_random, model_user, model_item, model_matrix]
    for i in mls:
        for j in lst:
            sumPre = 0
            sumRec = 0
            for trainset, testset in kf.split(data):
                i.fit(trainset)
                predictions = i.test(testset)
                precisions, recalls = precision_recall_at_k(predictions, k=j, threshold=3.5)
                sumPre = sumPre + sum(pr for pr in precisions.values()) / len(precisions)
                sumRec = sumRec + sum(rec for rec in recalls.values()) / len(recalls)
                # Precision and recall can then be averaged over all users
            meanPre = sumPre/5
            meanRec = sumRec/5
            print('The  average precision for ',  modelName(i),  'with k=' + str(j) + ' is ',
            print('The  average recall for ', modelName(i),  'with k=' + str(j) + ' is ', mear
```

```
The  average precision for  random with k=5 is  0.5902557942932753
The  average recall for  random with k=5 is  0.34204352684906525
The  average precision for  random with k=10 is  0.5883178770829411
The  average recall for  random with k=10 is  0.43722422925961507
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
The  average precision for  user-user with k=5 is  0.7632670426557325
The  average recall for  user-user with k=5 is  0.45430400319219383
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
The  average precision for  user-user with k=10 is  0.7349051345777127
The  average recall for  user-user with k=10 is  0.593299313507478
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
```

```
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
The  average precision for  item-item with k=5 is  0.8179426066786226
The  average recall for  item-item with k=5 is  0.3912312746562175
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
The  average precision for  item-item with k=10 is  0.7877888662768179
The  average recall for  item-item with k=10 is  0.5322608511224156
The  average precision for  matrix with k=5 is  0.7825708021238966
The  average recall for  matrix with k=5 is  0.43319187495749495
The  average precision for  matrix with k=10 is  0.7595582677447872
The  average recall for  matrix with k=10 is  0.5604525854596626
```

Type your response here...

- The Precision and Recall minimum values in all the posible scenaries were obtained with the random model.
- For k=5 the best Precision was obtained by item-item model. On the other hand the best Recall was given by user-user model.
- For k=10 best Precision was obtained by item-item model. On the other hand the best Recall was given by user-user model.
- It is interesting that the matrix factorization model does not have the best Presicion and Recall, it has the lowest RMSE though

```
# Make list of k values
K = [5, 10]

# Make list of models
models = [model_random, model_user, model_item, model_matrix]
model_names = ['model_random', 'model_user', 'model_item', 'model_matrix']

# Create k-fold cross validation object
kf = KFold(n_splits=5)

for k in K:
    for i, model in enumerate(models):
        print(f'>>> k={k}, model={model_names[i]}')
        # Run folder and take average
        p = []
        r = []
        for trainset, testset in kf.split(data):
            model.fit(trainset)
```

```
            predictions = model.test(testset, verbose=False)
            precisions, recalls = precision_recall_at_k(predictions, k=k, threshold=3.5)

            # Precision and recall can then be averaged over all users
            p.append(sum(prec for prec in precisions.values()) / len(precisions))
            r.append(sum(rec for rec in recalls.values()) / len(recalls))

        print('>>> precision:', round(sum(p) / len(p), 3))
        print('>>> reccall  :', round(sum(r) / len(r), 3))
        print('\n')

print('\n\nPrecision and recall computation successful!')
```

```
>>> k=5, model=model_random
>>> precision: 0.584
>>> reccall  : 0.342


>>> k=5, model=model_user
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
>>> precision: 0.765
>>> reccall  : 0.456


>>> k=5, model=model_item
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
>>> precision: 0.815
>>> reccall  : 0.388


>>> k=5, model=model_matrix
>>> precision: 0.784
>>> reccall  : 0.435


>>> k=10, model=model_random
>>> precision: 0.586
>>> reccall  : 0.429


>>> k=10, model=model_user
```

```
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
>>> precision: 0.734
>>> reccall  : 0.591
```

Top-n Predictions

Finally, we can see what some of the actual movie ratings are for particular users, as outputs of our model.

Again, we define a helpful function.

```
def get_top_n(predictions, n=5):
    '''Return the top-N recommendation for each user from a set of predictions.

    Args:
        predictions(list of Prediction objects): The list of predictions, as
            returned by the test method of an algorithm.
        n(int): The number of recommendation to output for each user. Default
            is 10.

    Returns:
    A dict where keys are user (raw) ids and values are lists of tuples:
        [(raw item id, rating estimation), ...] of size n.
    '''

    # First map the predictions to each user.
    top_n = dict()
    for uid, iid, true_r, est, _ in predictions:
        current = top_n.get(uid, [])
        current.append((iid, est))
        top_n[uid] = current

    # Then sort the predictions for each user and retrieve the k highest ones.
    for uid, user_ratings in top_n.items():
        user_ratings.sort(key=lambda x: x[1], reverse=True)
        top_n[uid] = user_ratings[:n]

    return top_n

print('Function creation successful!')
```

```
Function creation successful!
```

Then, we call this function on each of our models, first training on **all** the data we have available, then predicting on the remaining, missing data. We use `n`=5 here, but you can pick any reasonable value of `n` you would like.

This may take some time to compute, so be patient.

```
trainset = data.build_full_trainset()
testset = trainset.build_anti_testset()
print('Trainset and testset creation successful!')
```

```
Trainset and testset creation successful!
```

```
for i in mls:
   i.fit(trainset)
   predictions = i.test(testset)
   topN = get_top_n(predictions, n=5)
   usr = list(topN.keys())[0]
   print('Model:', modelName(i))
   print('User', usr)
   print('Predictions',topN[usr])
```

```
Model: random
User 196
Predictions [('377', 5), ('40', 5), ('1184', 5), ('768', 5), ('88', 5)]
Computing the msd similarity matrix...
Done computing similarity matrix.
Model: user-user
User 196
Predictions [('1189', 5), ('1500', 5), ('814', 5), ('1536', 5), ('1599', 5)]
Computing the msd similarity matrix...
Done computing similarity matrix.
Model: item-item
User 196
Predictions [('1414', 4.666666666666667), ('1309', 4.5), ('1310', 4.5), ('1675', 4.3
Model: matrix
User 196
Predictions [('318', 4.5879274978167235), ('408', 4.489381493575415), ('427', 4.4844
```

QUESTION 5: TOP N PREDICTIONS

Do the top n predictions that you received make sense? What is the rating value (1-5) of these predictions? How could you use these predictions in the real-world if you were trying to build a generic content recommender system for a company?

Type your response here...

- The top n predictions do make sense.

- The values of the predictions are numbers in the interval [4,5].
- These predictions could be used to recommend to the corresponding users the products involved in the list obtained (by the model). There is a high probability that the clients would like these products, then is a good idea.

```
for i, model in enumerate(models):
    model.fit(trainset)
    predictions = model.test(testset)
    top_n = get_top_n(predictions, n=5)
    # Print the first one
    user = list(top_n.keys())[0]
    print(f'model name: {model_names[i]}')
    print(f'user ID: {user}')
    print(f'top 5 movie ID\'s this user would like, sorted by rating highest to lowest: {t

print('\n\nTop N computation successful! YOU ARE DONE WITH THE CODE!')
```

```
model name: model_random
user ID: 196
top 5 movie ID's this user would like, sorted by rating highest to lowest: [('387',
Computing the msd similarity matrix...
Done computing similarity matrix.
model name: model_user
user ID: 196
top 5 movie ID's this user would like, sorted by rating highest to lowest: [('1189',
Computing the msd similarity matrix...
Done computing similarity matrix.
model name: model_item
user ID: 196
top 5 movie ID's this user would like, sorted by rating highest to lowest: [('1414',
model name: model_matrix
user ID: 196
top 5 movie ID's this user would like, sorted by rating highest to lowest: [('169',


Top N computation successful! YOU ARE DONE WITH THE CODE!
```

Great job! Now, make sure you check out the **Conclusion** section of the instruction manual to wrap up this case study properly.