Q1. (15%) How do you calculate the similarity? Do you use *Jaccard similarity? Pearson coefficient?* or *Cosine similarity?* Or maybe, the similarity measure you used is none of the above, then what is it? Please explain why you choose the metric. Also, describe how you calculate the similarity between any 2 users, as well as how you implement it in your code.

A1. I try the Cosine similarity and Pearson's coefficient, but in the end I choose the Pearson's coefficient is because of the bias adjustment of mean-centering, so it's more suitable for users-based collaborative filtering, following is an explanation for mine similarity calculating.

Steps1: Reduced the data by 'Users-rating points' and 'Items-rating counts', since I don't want inside the data have too many users are giving 0 point and choose books count at least have more than 1 time.

```
count1 = train_df['User-ID'].value_counts(ascending=False) #User-counts，User的評分分數
matrix_df = train_df[train_df['User-ID'].isin(count1[count1>0].index)]
count2 = train_df.groupby('unique_isbn')['Book_Rating'].count()#書的評分次數
matrix_df = train_df[train_df['unique_isbn'].isin(count2[count2>1].index)]
```

Steps2: Convert source data to new data-frame which index is 'User ID' the columns is 'ISBN' and the values are 'Book rating', since I choose the Pearson's coefficient, so we need to calculating the mean then subtract it, also there have many NA values inside the data, so filled the 0 **after subtract mean.**

```
final_matrix = matrix_df.pivot_table(index='User-ID', columns='unique_isbn', values='Book_Rating')
# Pearson's
mean_ratings = final_matrix.mean(axis=1)
final_matrix = final_matrix.sub(mean_ratings, axis=0)
final_matrix.fillna(0,inplace=True)
# final_matrix = final_matrix.astype(np.int64)
```

Steps3: Put the data-frame we subtract the mean into consie_similarity() function, then will convert the data structure is np.array, then we can through it to create the new data-frame, In this data-frame the index/columns we all use 'User-ID', this features can help us to review the user-user similarity as figure.

```
cosine_similarity_array = cosine_similarity(final_matrix) #這裡會用Consine
set_pandas_display_options()
# print(cosine_similarity_array)
cosine_similarity_df = pd.DataFrame(cosine_similarity_array,index=final_matrix.index,columns=final_matrix.index)
# print(cosine_similarity_df) #稀鬆矩陣
cosine_similarity_series = cosine_similarity_df.loc[26] #填入要預測的對象
order = cosine_similarity_series.sort_values(ascending=False) #排出User-User之間的correlation
```

User-ID	26	91	114	243	244	254	256	...	278535	278541	278543	278554	278633	278698	278843
User-ID								...							
26	1.0	0.0	0.0	0.0	0.0	0.0	0.000000	...	0.0	0.0	0.0	0.000000	0.00000	0.0	0.000000
91	0.0	1.0	0.0	0.0	0.0	0.0	0.000000	...	0.0	0.0	0.0	0.000000	0.00000	0.0	0.000000
114	0.0	0.0	1.0	0.0	0.0	0.0	0.000000	...	0.0	0.0	0.0	0.000000	0.00000	0.0	0.000000
243	0.0	0.0	0.0	1.0	0.0	0.0	-0.180803	...	0.0	0.0	0.0	0.000000	0.00000	0.0	0.000000
244	0.0	0.0	0.0	0.0	1.0	0.0	0.000000	...	0.0	0.0	0.0	0.000000	0.00000	0.0	0.000000
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
278543	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	...	0.0	0.0	1.0	0.000000	0.00000	0.0	0.000000
278554	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	...	0.0	0.0	0.0	1.000000	-0.30026	0.0	0.309782
278633	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	...	0.0	0.0	0.0	-0.300260	1.00000	0.0	-0.969260
278698	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	...	0.0	0.0	0.0	0.000000	0.00000	1.0	0.000000
278843	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	...	0.0	0.0	0.0	0.309782	-0.96926	0.0	1.000000

Q2. (15%) Aside from the ratings, do you use other features to help deriving the similarity? If you do, how do you use them and how do you implement in your code? And if you don't, what features do you think might be helpful in calculating similarity? How could you fit the features in the procedure?

A2. Beside using the ratings, I think we also can filter by the user location, split the location to [Country, State, City] then limit our user data in only **eg,.US or Canada area**. And then combine user data with the rating data and total rating count data, this feature might be helpful for calculating similarity as well

Q3. (25%) Please briefly explain the implementation regarding the procedures mentioned above in your code. That is, how you pick out the most similar *k* users to a certain user? How you filter out those items that the user has not experienced but the other similar *k* users have tried? Finally, among these items, how you make the final recommendation?

A2.

Steps1: Use the pervious Pearson's similarity data-frame we build and input the user whom we want to predict, then will show the most similar user by ascending=False as following

```
predicted_user = 11676
cosine_similarity_series = cosine_similarity_df.loc[predicted_user]
order = cosine_similarity_series.sort_values(ascending=False) #排出User-User之間的correlation
# print(order[(order>0.3)].head(15))
print(order.head(15))
```

```
User-ID
11676     1.000000
158295    0.589199
217740    0.483235
104665    0.478299
194614    0.467949
95932     0.463290
167517    0.457639
174596    0.455165
37904     0.453208
137151    0.453208
175703    0.453208
2041      0.453208
123544    0.453208
246470    0.453208
133682    0.453208
```

Steps2: Create the new data-frame is because I want to sort the 'book rating' by ascending = False, the reason why need to decreasing the rating order is because I want the books recommend to user is always high rating first, and create two additional lists, one list is for books of predicted user already read, another list is for the top K similar users with predicted user.

```
Recommend_df = matrix_df.sort_values(by = ['Book_Rating'],ascending=False) #Rating先由大到小排序，才可推薦由高分到低分的新書
Recommend_df.to_csv(r'.\\Project1\\Book-Crossing Dataset\\Recommend_df.csv',index=False)
similar_user_list = []
predicted_user_list = []
for i in range(0,len(Recommend_df[Recommend_df['User-ID'] == predicted_user]['unique_isbn'])):
    predicted_user_list.append(Recommend_df[Recommend_df['User-ID'] == predicted_user]['unique_isbn'].values[i]) #預測對象已讀過的書

for i in range(0,5):
    similar_user = order.iloc[1:6,].index[i] #int64index轉int要有.index[0]才可用，因int64index是為list
    similar_user_list.append(similar_user)
```

Steps3: Start to build the recommendation books for predicted user, use two for loop, one is for top 5 similar users, another is for these 5 similar users recommended books (noted: 'book ratings ascending = False, so always recommend the high rating books first), also we know there have some books the predicted user have already read it, so I use condition sentence if recommend books are in the list of predicted user already read then will **return 0**, else will **return true**.

```
def get_isbn_from_index(title, number, predicted_user_list):
    return Recommend_df[Recommend_df['unique_isbn'] == title]['User-ID'].values[0]
def get_index_from_isbn(index, number, predicted_user_list):
    # return matrix_df[matrix_df['User-ID'] == index] #All
    re_books = Recommend_df[Recommend_df['User-ID'] == index]['unique_isbn'].values[number] #第[number]本出現的書
    if re_books in predicted_user_list:
        # print('Not match:',re_books)
        return 0
    return re_books
for j in similar_user_list:
    print('Top similarity users:',j)
    for i in range(0,len(Recommend_df[Recommend_df['User-ID'] == j]['unique_isbn'])):
        if get_index_from_isbn(j, i, predicted_user_list) == 0:
            pass
        else:
            print('Recommend books ISBN:',get_index_from_isbn(j, i, predicted_user_list))
```

Q4. (15%) Please describe how you estimate the ratings, given a user and an item, such as explaining it by formula. Also, briefly explain how you accomplish the estimations in your code.

A4. The final method I choose for predicted rating is SVD (singular value decomposition).

Steps1: Need to create the new data-frame that columns sequence are 'User-ID, ISBN, Book rating', then set rating scale is 1 to 10 because of our rating is 1 to 10, after all settling done, split the 80% train data and 20% test data.

```
# --SVD--
    filter_df = matrix_df.drop(columns=['ISBN','Country','State','City','Book_Title','Book-Author','Year-Of-Publication','Publisher','Age'],axis=1)
    filter_df = filter_df.reindex(columns=['User-ID','unique_isbn','Book_Rating'])
    #使用surpise的train_test_split要reader
    reader = Reader(rating_scale=(1, 10)) #Rating range 1~10
    # Load the data into a 'Dataset' object directly from the pandas df.
    # Note: The fields must be in the order: user, item, rating
    data = Dataset.load_from_df(filter_df, reader)
    train_set, test_set = train_test_split(data, test_size=0.2)
```

Steps2: There have two ways I use for training model, first use the K-Fold to split dataset and look which accuracy are the best, second one is the GridSearchCV function, give the different parameter into SVD function and print out the scores, then can know what parameter we adjust will be the best for 'rmse', after I know the best parameter then bring into the SVD function to start training the model.

Now the model is done training, then can input the user-id and item-id into model, will output the predicted rating.

```
# 調RMSE方法1
# model = SVD()
# kf = KFold(n_splits=3)
# for train_set, test_set in kf.split(data):
#     model.fit(train_set)
#     predictions = model.test(test_set)
#     accuracy.rmse(predictions)

# 調RMSE方法2
param_grid = {'n_factors': [80, 100, 120], 'lr_all': [0.001, 0.005, 0.01], 'reg_all': [0.01, 0.02, 0.04]}
gs = GridSearchCV(SVD, param_grid, measures=['rmse', 'mae'], cv=3)
gs.fit(data)
# print(gs.best_score['rmse'])
# print(gs.best_params['rmse']) #Result: {'n_factors': 80, 'lr_all': 0.005, 'reg_all': 0.02}
model = SVD(n_factors=80, lr_all=0.005, reg_all=0.04)
model.fit(train_set) # re-fit on only the training data using the best hyperparameters
test_pred = model.test(test_set)
accuracy.rmse(test_pred, verbose=True)
uid = 188593
iid = 553148001
pred1 = model.predict(uid, iid, verbose=True)
```

About the explaining for SVD, I have reference following web:

https://medium.com/data-scientists-playground/svd-%E6%8E%A8%E8%96%A6%E7%B3%BB%E7%B5%B1-%E5%8E%9F%E7%90%86-c72c2e35af9c

就我對 SVD 的基礎理解大概是 SVD 會做矩陣分解，將一個原本完整的矩陣，拆解成三個矩陣相乘。在分解的過程中，因為我們用了更少的資料來代表原本的矩陣，因此必定會造成資訊損失，但帶來的優勢便是更小更快的計算量，以及可以避免因為資料稀疏，=而造成協同過濾演算法難以計算的困境。

透過 SVD 的轉換，我們可以獲得這些資料的隱含特徵，並以這些特徵來計算個別的特徵。接著，我們透過將 U、Sigma、Vt 來計算特定的值，以減少整個矩陣的稀疏性， 整體來說，**SVD 雖然解決了儲存稀疏矩陣的問題**；但在計算時的空間複雜度是呈現 Big O(n^3) 非常大。

假設有 n 個 users，t 個 items，則用戶評分矩陣利用下列公式預測:

$$\hat{r}_{ui} = p_u q_i^T$$

針對user矩陣偏微分：

$$\frac{df}{dp_u} = \frac{d}{dp_u}(r_{ui} - p_u \times q_i)^2 = -2q_i(r_{ui} - p_u \times q_i)$$

dPu偏微分

針對item矩陣偏微分：

$$\frac{df}{dq_i} = \frac{d}{dq_i}(r_{ui} - p_u \times q_i)^2 = -2p_u(r_{ui} - p_u \times q_i)$$

dQi偏微分

最後會給定初始矩陣 p,q，針對每個已知評分計算更新公式直接結束:

$$p_u = p_u - \alpha \times q_i(r_{ui} - p_u \times q_i)$$
$$q_i = q_i - \alpha \times p_u(r_{ui} - p_u \times q_i)$$

Q5. (10%) To overcome the aforementioned downside of user-based collaborative filtering, what measures will you take? Do you implement the measure in your code? If so, how do you do that?

A5. 以此題為例子的話, cold start 的問題我認為可以透過新加入的 user 的**國籍/城市/年齡**對 books 的**作者/出版社**作為判斷依據，先預測一個模型是專門針對某個國家、某個城市的各年齡層的 user 都是看某個作者/出版社的書以及如何給予評分的，當有新 user 加入時再透過這個模型做推薦，看他是哪個國家/城市的人去比對這個國家/城市的人曾經看過最高分的書，這是我的想法但並未實做出來。

若是新書加入我認為以此 dataset 來看給的 feature 太少了，因為沒有書的種類，故很難去判斷新書加入該如何歸類，用作者/出版社都很難取得書的共通性

Q6. (10%) of your score will be based on your **performance ranking**, compared with others using the same dataset. The higher-ranked 50% get a full 10%, while the lower-ranked 50% get 5% in this section.

A6. Refer the attach 110209058.csv