Amazon Recommendation System

Intro to Data Science:
Final Project

Jason Cai | Matthew Eng | Amol Srivastava

INTRODUCTION

We were assigned with creating a recommendation system based on one of the two provided datasets. Based on concepts that we learned in class and other researched topics, we had to complete 3 tasks:

1. Data selection and preprocessing
2. Rating Prediction
3. Item Recommendation

We also had to measure how well our models and algorithms performed through multiple metrics.

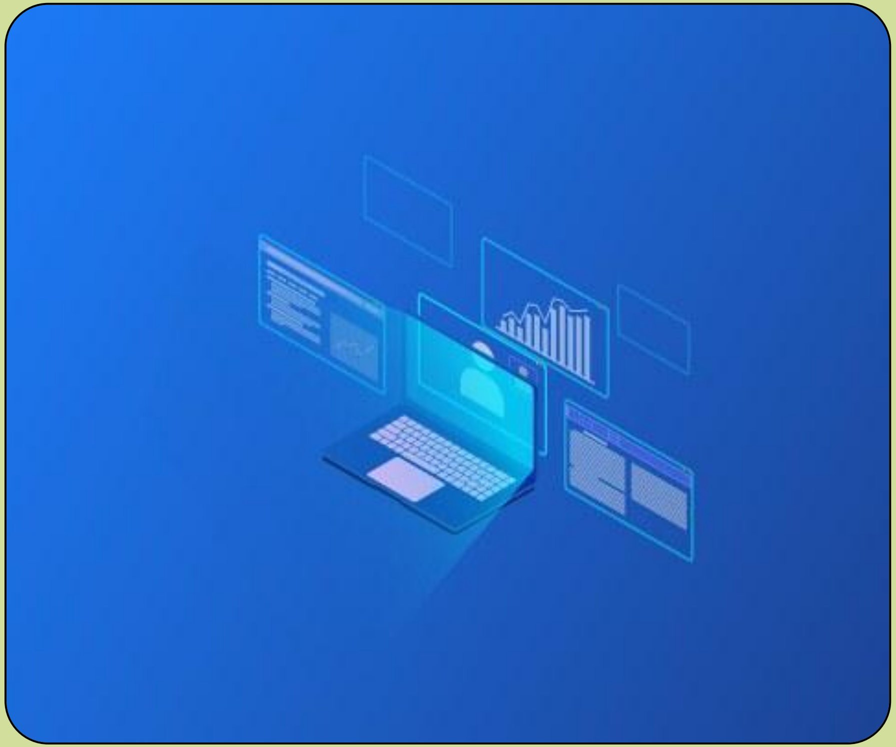
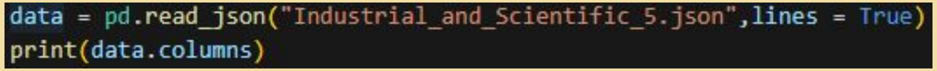


TABLE OF CONTENTS

Data selection and preprocessing

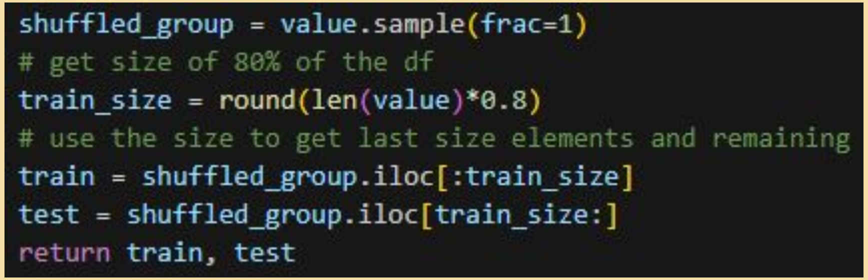
Task 1

We first read the "Industrial and Scientific" JSON file with the Pandas library.



```
data = pd.read_json("Industrial_and_Scientific_5.json",lines = True)
print(data.columns)
```

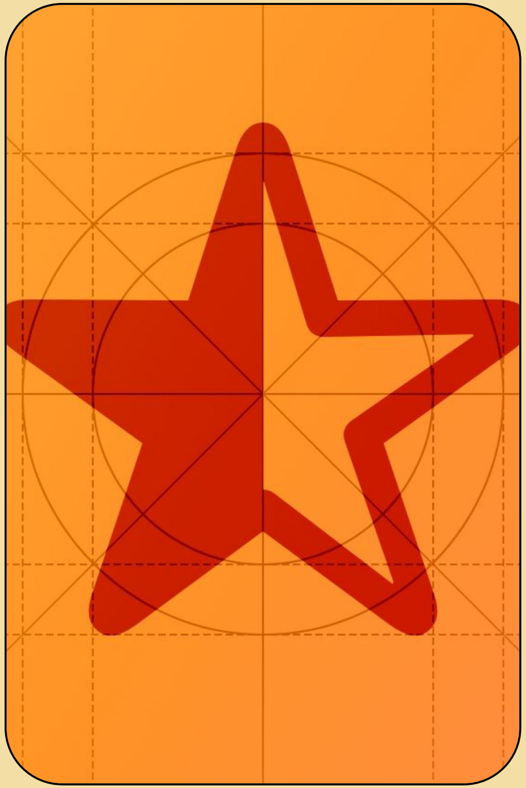
Then we sampled and split the data into training and testing data through .sample and .iloc, according to the 80/20 conventions.



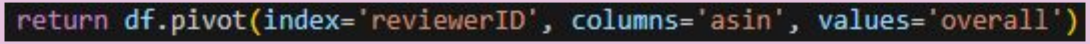
```
shuffled_group = value.sample(frac=1)
# get size of 80% of the df
train_size = round(len(value)*0.8)
# use the size to get last size elements and remaining
train = shuffled_group.iloc[:train_size]
test = shuffled_group.iloc[train_size:]
return train, test
```

Task 2

Rating Prediction

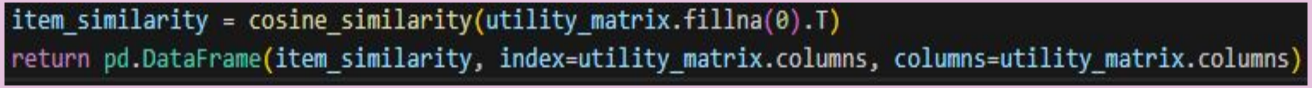


To start, we generated a utility matrix to facilitate the rest of the steps that were necessary to accurately predict the ratings of the testing data through the ratings of the training data.



```
return df.pivot(index='reviewerID', columns='asin', values='overall')
```

From there, we then generated a similarity matrix based on our item-item implementation.

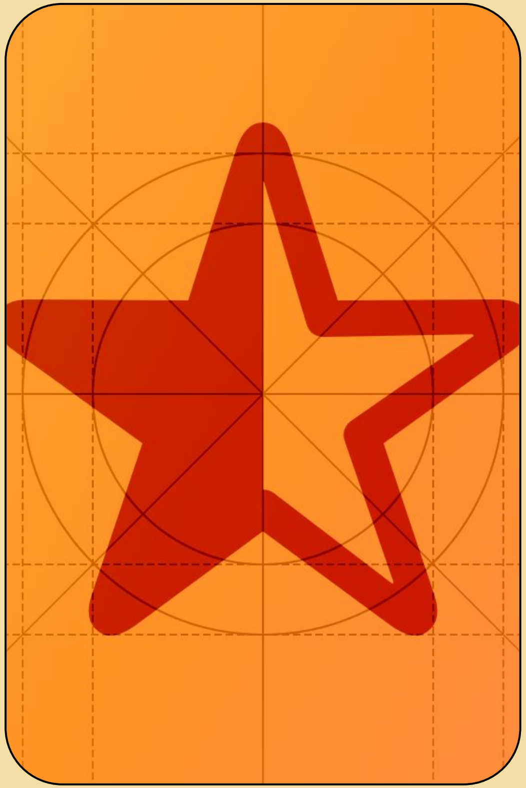


```
item_similarity = cosine_similarity(utility_matrix.fillna(0).T)
return pd.DataFrame(item_similarity, index=utility_matrix.columns, columns=utility_matrix.columns)
```

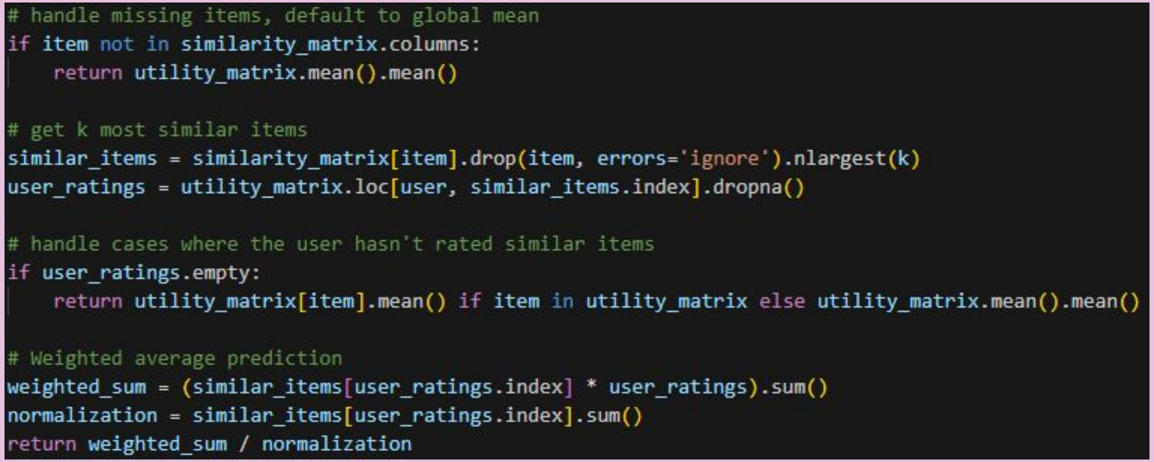
Next, we actually predicted the previously mentioned ratings through item-item collaborative filtering.....

Task 2

Rating Prediction (cont.)(1)



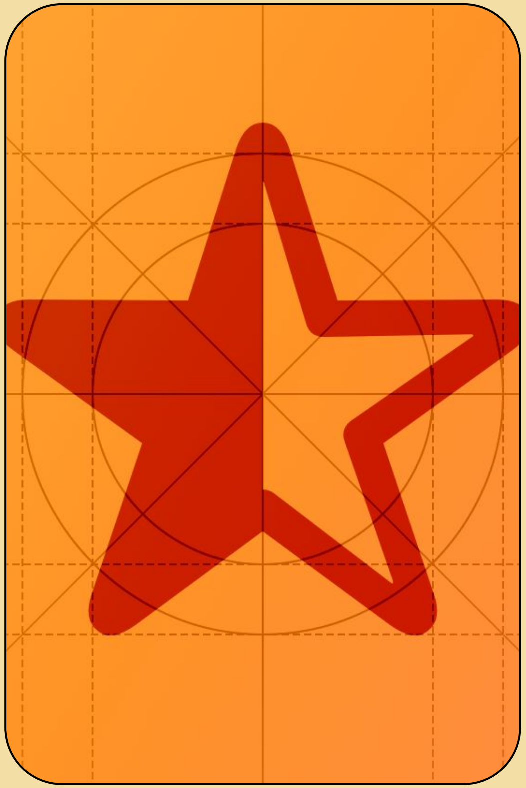
After experimenting, we landed on using a k value of 5. We also included code to handle edge cases (missing items, no rated similar items, etc.). For the actual predictions, we used weighted averages.



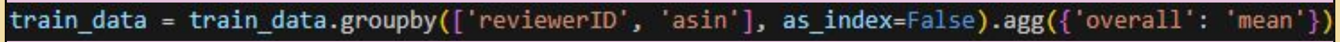
```
# handle missing items, default to global mean
if item not in similarity_matrix.columns:
    return utility_matrix.mean().mean()

# get k most similar items
similar_items = similarity_matrix[item].drop(item, errors='ignore').nlargest(k)
user_ratings = utility_matrix.loc[user, similar_items.index].dropna()

# handle cases where the user hasn't rated similar items
if user_ratings.empty:
    return utility_matrix[item].mean() if item in utility_matrix else utility_matrix.mean().mean()

# Weighted average prediction
weighted_sum = (similar_items[user_ratings.index] * user_ratings).sum()
normalization = similar_items[user_ratings.index].sum()
return weighted_sum / normalization
```

Task 2

Rating Prediction (cont.)(2)

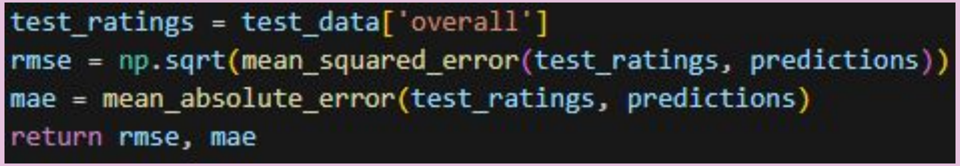


Finally, when we actually implemented these functions, we first grouped the training data by “reviewerID” and “asin”. To get rid of duplicates and provide more accurate results, we calculated the means of each item and user’s “overall” values.



```
train_data = train_data.groupby(['reviewerID', 'asin'], as_index=False).agg({'overall': 'mean'})
```

Lastly, we used a mix of NumPy and scikit-learn to calculate RMSE and MAE

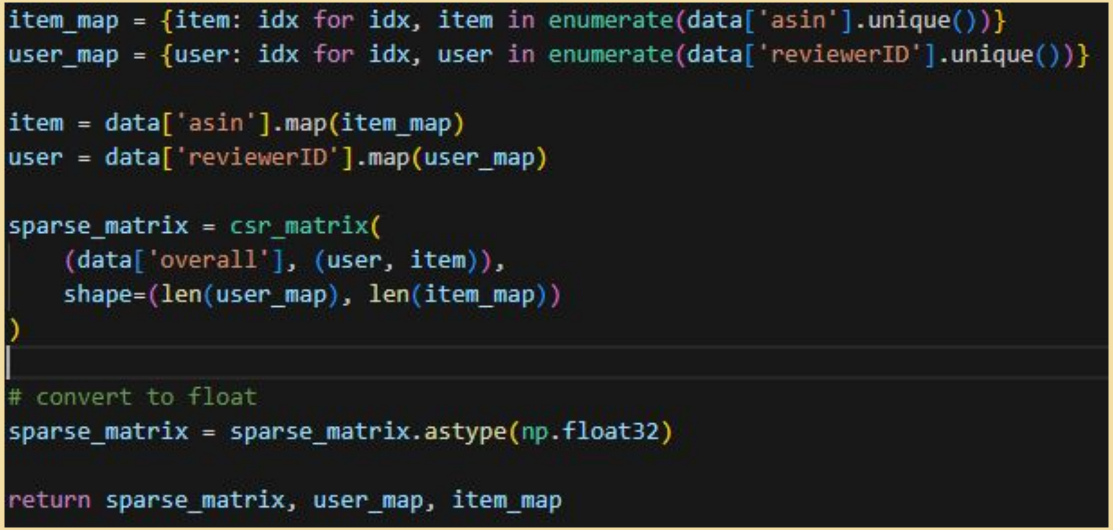


```
test_ratings = test_data['overall']
rmse = np.sqrt(mean_squared_error(test_ratings, predictions))
mae = mean_absolute_error(test_ratings, predictions)
return rmse, mae
```

Task 3

Item Recommendation

To start, we generated a user-item matrix of unique values. To help with this, we incorporated a sparse matrix.

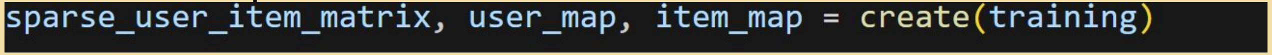


```
item_map = {item: idx for idx, item in enumerate(data['asin'].unique())}
user_map = {user: idx for idx, user in enumerate(data['reviewerID'].unique())}

item = data['asin'].map(item_map)
user = data['reviewerID'].map(user_map)

sparse_matrix = csr_matrix(
    (data['overall'], (user, item)),
    shape=(len(user_map), len(item_map))
)

# convert to float
sparse_matrix = sparse_matrix.astype(np.float32)

return sparse_matrix, user_map, item_map
```

Item Recommendation (cont.)(1)

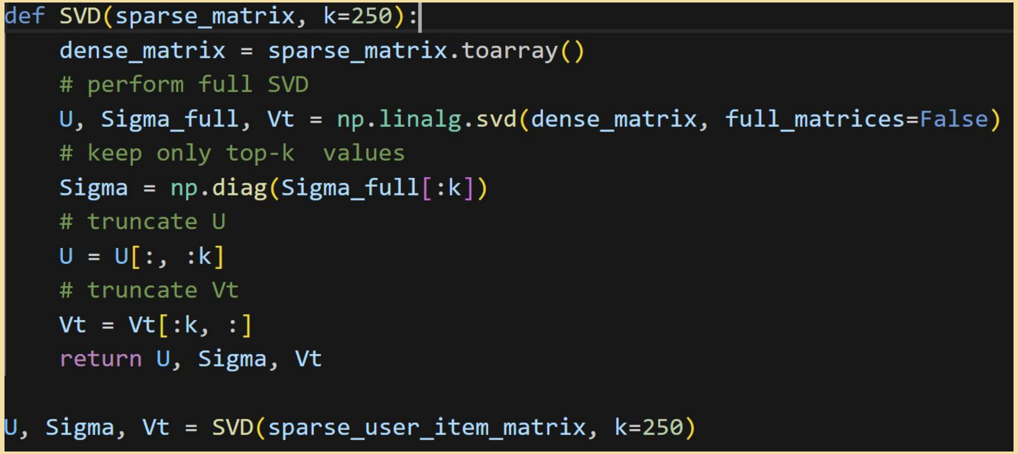
Task 3

We run the previous method to obtain our needed values



```
sparse_user_item_matrix, user_map, item_map = create(training)
```

Next we need to implement and run a full SVD on some k values we choose and resultantly have truncate everything in the SVD to account for it.

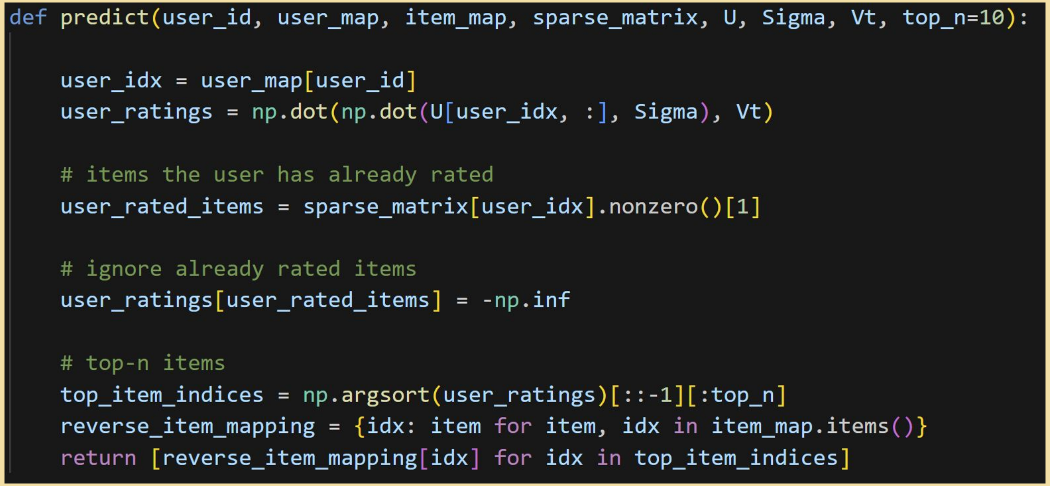


```
def SVD(sparse_matrix, k=250):
    dense_matrix = sparse_matrix.toarray()
    # perform full SVD
    U, Sigma_full, Vt = np.linalg.svd(dense_matrix, full_matrices=False)
    # keep only top-k  values
    Sigma = np.diag(Sigma_full[:k])
    # truncate U
    U = U[:, :k]
    # truncate Vt
    Vt = Vt[:k, :]
    return U, Sigma, Vt

U, Sigma, Vt = SVD(sparse_user_item_matrix, k=250)
```

Task 3

Item Recommendation (cont.)(2)

Now with the adjusted U, Sigma and Vt values we can use them to filter out already rated items and obtain the top-n recommendation for each user.

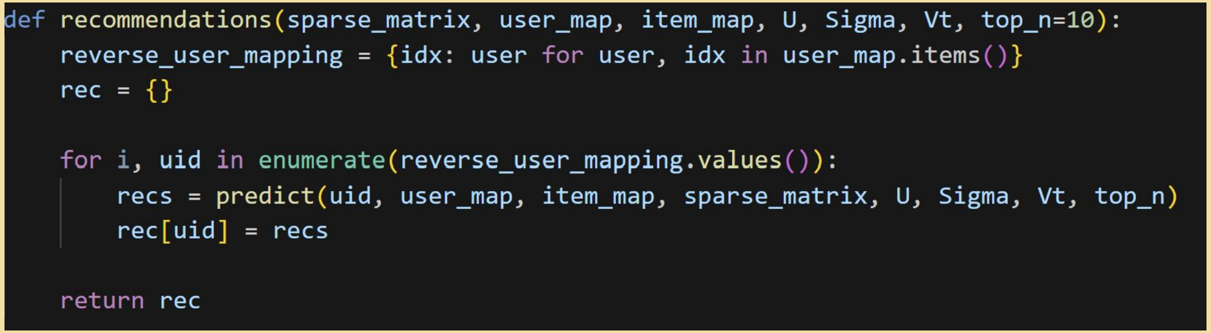


```
def predict(user_id, user_map, item_map, sparse_matrix, U, Sigma, Vt, top_n=10):

    user_idx = user_map[user_id]
    user_ratings = np.dot(np.dot(U[user_idx, :], Sigma), Vt)

    # items the user has already rated
    user_rated_items = sparse_matrix[user_idx].nonzero()[1]

    # ignore already rated items
    user_ratings[user_rated_items] = -np.inf

    # top-n items
    top_item_indices = np.argsort(user_ratings)[::-1][:top_n]
    reverse_item_mapping = {idx: item for item, idx in item_map.items()}
    return [reverse_item_mapping[idx] for idx in top_item_indices]
```

Item Recommendation (cont.)(3)

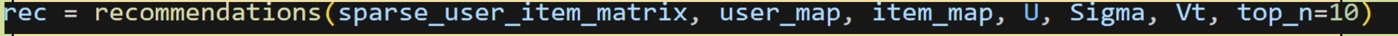
Task 3

Since we have a way to calculate individual recommendations we now have to apply it to all of the users by mapping through them all.



```
def recommendations(sparse_matrix, user_map, item_map, U, Sigma, Vt, top_n=10):
    reverse_user_mapping = {idx: user for user, idx in user_map.items()}
    rec = {}

    for i, uid in enumerate(reverse_user_mapping.values()):
        recs = predict(uid, user_map, item_map, sparse_matrix, U, Sigma, Vt, top_n)
        rec[uid] = recs

    return rec
```

After that we just need to call the methods.

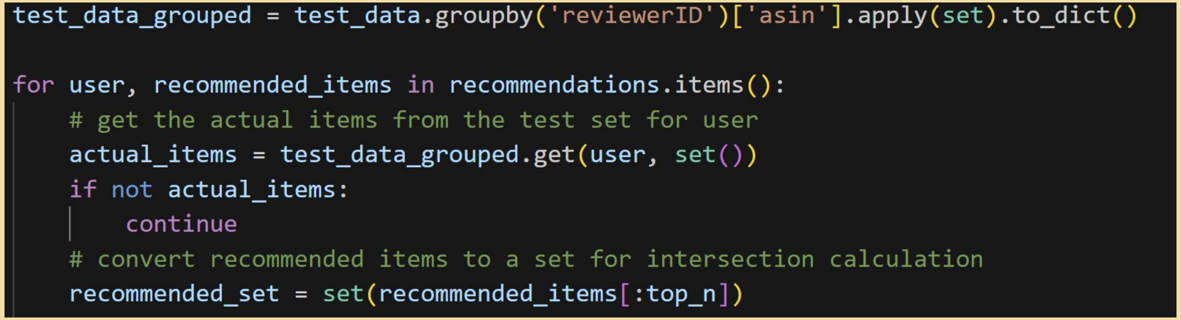


```
rec = recommendations(sparse_user_item_matrix, user_map, item_map, U, Sigma, Vt, top_n=10)
```

Item Recommendation (cont.)(4)

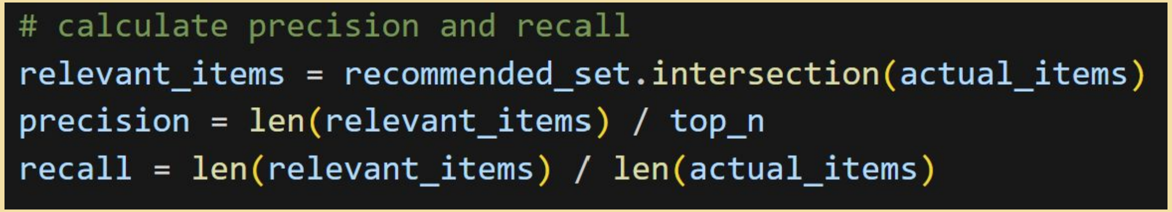
Task 3

All that is left is to get our metrics. We first group our users and convert recommended items to a set.



```
test_data_grouped = test_data.groupby('reviewerID')['asin'].apply(set).to_dict()

for user, recommended_items in recommendations.items():
    # get the actual items from the test set for user
    actual_items = test_data_grouped.get(user, set())
    if not actual_items:
        continue
    # convert recommended items to a set for intersection calculation
    recommended_set = set(recommended_items[:top_n])
```

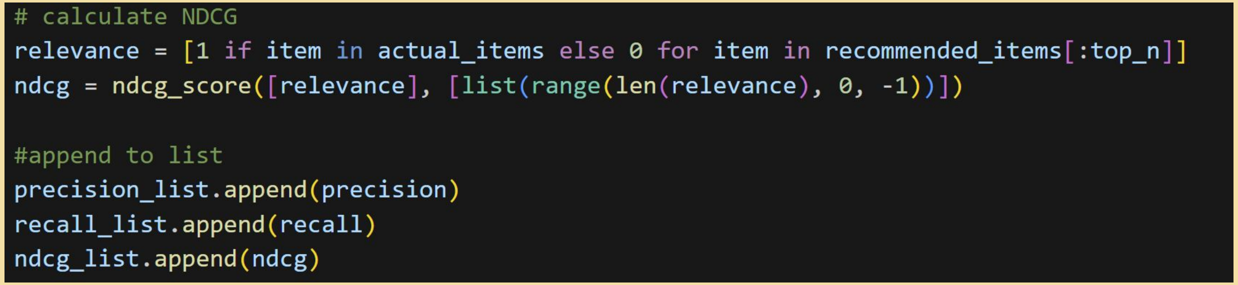
We calculate precision and recall



```
# calculate precision and recall
relevant_items = recommended_set.intersection(actual_items)
precision = len(relevant_items) / top_n
recall = len(relevant_items) / len(actual_items)
```

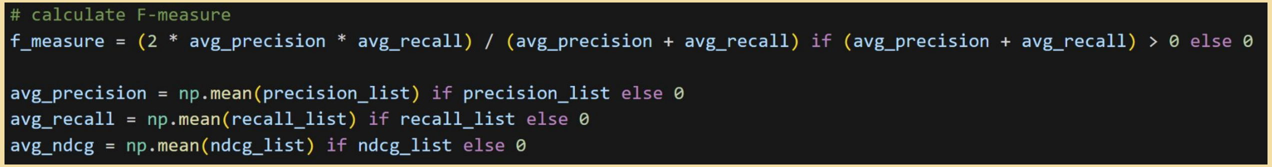
Item Recommendation (cont.)(5)

We calculate NDCG and append all three of them to a a list



```
# calculate NDCG
relevance = [1 if item in actual_items else 0 for item in recommended_items[:top_n]]
ndcg = ndcg_score([relevance], [list(range(len(relevance), 0, -1))])

#append to list
precision_list.append(precision)
recall_list.append(recall)
ndcg_list.append(ndcg)
```

All that's left is calculating F-measure and getting the averages of the other three metrics



```
# calculate F-measure
f_measure = (2 * avg_precision * avg_recall) / (avg_precision + avg_recall) if (avg_precision + avg_recall) > 0 else 0

avg_precision = np.mean(precision_list) if precision_list else 0
avg_recall = np.mean(recall_list) if recall_list else 0
avg_ndcg = np.mean(ndcg_list) if ndcg_list else 0
```

Task 3

Results

1

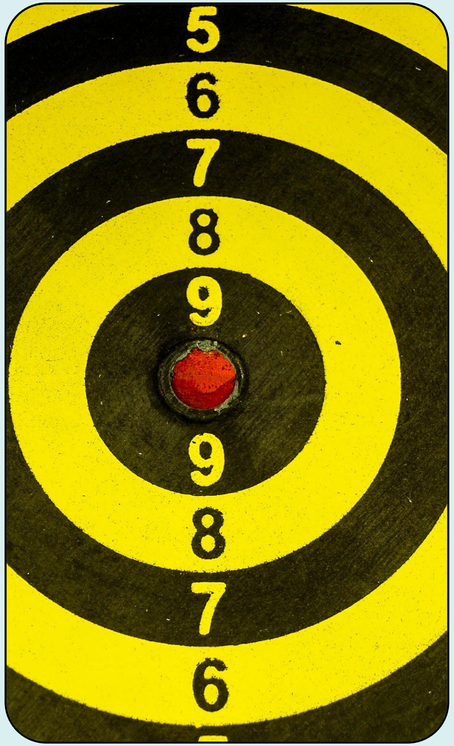
Training: 62307 rows (80.8%)
Testing: 14764 rows (19.2%)

2

MAE (Mean Absolute Error): 0.5796
RMSE (Root Mean-square Deviation): 0.9146

3.1

Precision: 0.01 (1%)



Results (cont.)

3.2

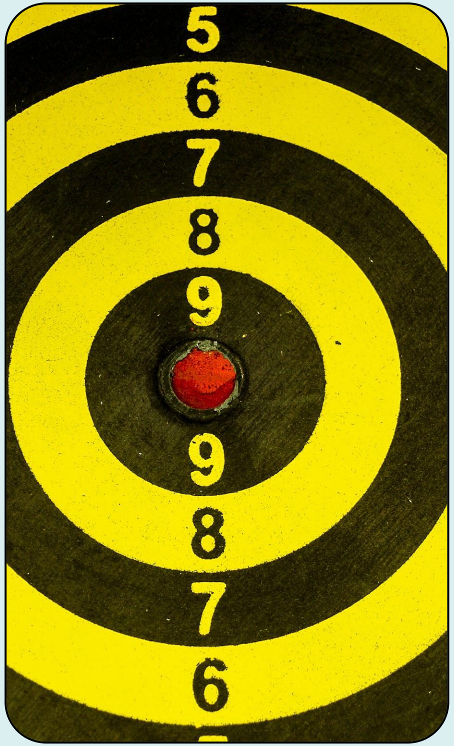
Recall: 0.074 (7.4%)

3.3

F-measure: 0.017 (1.7%)

3.4

NDCG: 0.067 (6.7%)



THANK YOU