# Data 612 Assignment 1

## Introduction

Our recommender system recommends data science books to readers based on their historical ratings. By predicting ratings when a new reader–book pair is encountered, the system can suggest books the reader is likely to enjoy.

## Code

```python
import numpy as np
import pandas as pd
from sklearn.metrics import mean_squared_error

data = {
    'user': [
        'A', 'A', 'A',
        'B', 'B', 'B', 'B',
        'C', 'C', 'C', 'C',
        'D', 'D', 'D',
        'E', 'E', 'E', 'E'
    ],
    'item': [
        'book1', 'book2', 'book3',
        'book1', 'book3', 'book4', 'book5',
        'book2', 'book3', 'book4', 'book5',
        'book1', 'book2', 'book3',
        'book1', 'book2', 'book4', 'book5'
    ],
    'rating': [
        5, 3, 4,
        4, 2, 1, 3,
        5, 4, 3, 3,
        3, 3, 4,
        2, 5, 4, 3
    ]
}

ratings_df = pd.DataFrame(data)
print("Original Ratings:")
print(ratings_df)
```

```
Original Ratings DataFrame:
   user   item  rating
0     A  book1       5
1     A  book2       3
2     A  book3       4
```

```
3      B  book1       4
4      B  book3       2
5      B  book4       1
6      B  book5       3
7      C  book2       5
8      C  book3       4
9      C  book4       3
10     C  book5       3
11     D  book1       3
12     D  book2       3
13     D  book3       4
14     E  book1       2
15     E  book2       5
16     E  book4       4
17     E  book5       3
```

```
user_item_matrix = ratings_df.pivot(index='user', columns='item',
values='rating')
print("\nUser-Item Matrix:")
print(user_item_matrix)
```

```
User-Item Matrix:
item  book1  book2  book3  book4  book5
user
A       5.0    3.0    4.0    NaN    NaN
B       4.0    NaN    2.0    1.0    3.0
C       NaN    5.0    4.0    3.0    3.0
D       3.0    3.0    4.0    NaN    NaN
E       2.0    5.0    NaN    4.0    3.0
```

```
np.random.seed(612)

mask = np.random.rand(len(ratings_df)) < 0.8
train_df = ratings_df[mask].reset_index(drop=True)
test_df = ratings_df[~mask].reset_index(drop=True)

print("Training Data:")
print(train_df)
print("\nTest Data:")
print(test_df)
```

```
Training Data:
   user   item  rating
0      A  book1       5
1      A  book2       3
2      B  book1       4
3      B  book4       1
4      B  book5       3
5      C  book3       4
```

```
6     C   book4       3
7     D   book2       3
8     D   book3       4
9     E   book2       5
10    E   book4       4
11    E   book5       3

Test Data:
  user   item  rating
0    A  book3       4
1    B  book3       2
2    C  book2       5
3    C  book5       3
4    D  book1       3
5    E  book1       2
```

```python
global_avg = train_df['rating'].mean()
print(f"\nGlobal Average Rating: {global_avg:.2f}")

train_df['pred_raw'] = global_avg
test_df['pred_raw'] = global_avg

rmse_train_raw = np.sqrt(mean_squared_error(train_df['rating'],
train_df['pred_raw']))
rmse_test_raw = np.sqrt(mean_squared_error(test_df['rating'],
test_df['pred_raw']))
print(f"Raw Average RMSE (Training): {rmse_train_raw:.2f}")
print(f"Raw Average RMSE (Test): {rmse_test_raw:.2f}")
```

```
Global Average Rating: 3.50
Raw Average RMSE (Training): 1.04
Raw Average RMSE (Test): 1.12
```

```python
user_bias = train_df.groupby('user')['rating'].mean() - global_avg

item_bias = train_df.groupby('item')['rating'].mean() - global_avg

print("\nUser Biases:")
print(user_bias)
print("\nItem Biases:")
print(item_bias)
```

```
User Biases:
user
A    0.500000
B   -0.833333
C    0.000000
D    0.000000
E    0.500000
```

```
Name: rating, dtype: float64

Item Biases:
item
book1    1.000000
book2    0.166667
book3    0.500000
book4   -0.833333
book5   -0.500000
Name: rating, dtype: float64
```

```python
def baseline_predict(row):
    u_bias = user_bias.get(row['user'], 0)
    i_bias = item_bias.get(row['item'], 0)
    return global_avg + u_bias + i_bias

train_df['pred_baseline'] = train_df.apply(baseline_predict, axis=1)
test_df['pred_baseline'] = test_df.apply(baseline_predict, axis=1)

print("\nTraining predictions:")
print(train_df[['user', 'item', 'rating', 'pred_baseline']])
print("\nTest predictions:")
print(test_df[['user', 'item', 'rating', 'pred_baseline']])
```

```
Training predictions (first few rows):
   user   item  rating  pred_baseline
0     A  book1       5       5.000000
1     A  book2       3       4.166667
2     B  book1       4       3.666667
3     B  book4       1       1.833333
4     B  book5       3       2.166667
5     C  book3       4       4.000000
6     C  book4       3       2.666667
7     D  book2       3       3.666667
8     D  book3       4       4.000000
9     E  book2       5       4.166667
10    E  book4       4       3.166667
11    E  book5       3       3.500000

Test predictions (first few rows):
  user   item  rating  pred_baseline
0    A  book3       4       4.500000
1    B  book3       2       3.166667
2    C  book2       5       3.666667
3    C  book5       3       3.000000
4    D  book1       3       4.500000
5    E  book1       2       5.000000
```

```
rmse_train_bl = np.sqrt(mean_squared_error(train_df['rating'],
train_df['pred_baseline']))
rmse_test_bl = np.sqrt(mean_squared_error(test_df['rating'],
test_df['pred_baseline']))
print(f"\nBaseline Predictor RMSE (Training): {rmse_train_bl:.2f}")
print(f"Baseline Predictor RMSE (Test): {rmse_test_bl:.2f}")
```

```

Baseline Predictor RMSE (Training): 0.65
Baseline Predictor RMSE (Test): 1.56
```

## Conclusion

In conclusion a simple model was built that predicts every rating as the global average. It gives RMSEs of about 1.04 (training) and 1.12 (test).

Incorporating user and item biases improve training performances, RMSE improving to 0.65, this is because it adjusts for known tendencies. However, on the test set, the RMSE increased to 1.56. This means that the biases capture the training data well, but they overfit on the test dataset. However this is to be expected since we are working with a toy dataset with very few data.