In [30]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
import numpy as np
from sklearn.metrics import root_mean_squared_error
```

# Project 1 | Global Baseline Predictors and RMSE

This jupyter notebook uses a global baseline bias recommender system of Van Leeuwen Ice Cream flavors ratings from coworkers to predict if they will like flavors they havent tasted (and rated) before using user and item biases

## Find a dataset, or build out your own toy dataset. As a minimum requirement for complexity, please include numeric ratings for at least five users, across at least five items, with some missing data.

In [31]:
```
df = pd.read_csv('VanLeeuwen.csv')
```

In [32]:
```
df
```

Out[32]:

| | Tasters | Black Cherry Chip | Cookies & Cream | Marionberry Cheesecake | Honeycomb | Mint Chip |
|---|---|---|---|---|---|---|
| **0** | Sam | 5 | NaN | 4.0 | NaN | 4 |
| **1** | Nina | 4 | 3.0 | 5.0 | 3.0 | 4 |
| **2** | Nancy | 4 | 2.0 | NaN | NaN | 3 |
| **3** | Aditi | 2 | 2.0 | 3.0 | 1.0 | 2 |
| **4** | Yerlene | 4 | NaN | 5.0 | 4.0 | 5 |
| **5** | Jose | 4 | 2.0 | 5.0 | 4.0 | 4 |

In [33]:
```
df['Mint Chip'] = df['Mint Chip'].astype(float)
df['Black Cherry Chip'] = df['Black Cherry Chip'].astype(float)
```

In [34]:
```
df.dtypes
```

Out[34]:
```
Tasters                  object
Black Cherry Chip       float64
Cookies & Cream         float64
Marionberry Cheesecake  float64
Honeycomb               float64
Mint Chip               float64
dtype: object
```

## Load your data into (for example) an R or pandas dataframe, a Python dictionary or list of lists, (or another data structure of your choosing). From there, create a user-item matrix.

In [35]:
```
#tidy-ify
user_item_matrix = pd.melt(df, id_vars=['Tasters'], value_vars=['Black Cherry Chip','Cookies & Cream', 'Marionberry Cheesecak
user_item_matrix.head()
```

Out[35]:

| | Tasters | variable | value |
|---|---|---|---|
| **0** | Sam | Black Cherry Chip | 5.0 |
| **1** | Nina | Black Cherry Chip | 4.0 |
| **2** | Nancy | Black Cherry Chip | 4.0 |
| **3** | Aditi | Black Cherry Chip | 2.0 |
| **4** | Yerlene | Black Cherry Chip | 4.0 |

## Break your ratings into separate training and test datasets.

In [36]:
```
train_data, test_data = train_test_split(user_item_matrix, test_size=0.2, random_state=30)
```

## Using your training data, calculate the raw average (mean) rating for every user-item combination.

In [37]:
```
train_data.head()
```

Out[37]:

| | Tasters | variable | value |
|---|---|---|---|
| **8** | Nancy | Cookies & Cream | 2.0 |
| **6** | Sam | Cookies & Cream | NaN |
| **25** | Nina | Mint Chip | 4.0 |
| **4** | Yerlene | Black Cherry Chip | 4.0 |
| **11** | Jose | Cookies & Cream | 2.0 |

In [38]:
```
raw_avg = train_data['value'].copy().mean()
raw_avg
```

Out[38]:
```
np.float64(3.4)
```

In [39]:
```
raw_avg_train = train_data.value[train_data['value'].notnull()].copy()
raw_avg_train[:] = raw_avg
raw_avg_train = np.array(raw_avg_train, dtype = 'float')
```

In [40]:
```
raw_avg_test = test_data.value[test_data['value'].notnull()].copy()
raw_avg_test[:] = raw_avg
raw_avg_test = np.array(raw_avg_test, dtype = 'float')
```

In [41]:
```
raw_avg_train
```

Out[41]:
```
array([3.4, 3.4, 3.4, 3.4, 3.4, 3.4, 3.4, 3.4, 3.4, 3.4, 3.4, 3.4, 3.4,
       3.4, 3.4, 3.4, 3.4, 3.4, 3.4, 3.4])
```

In [42]:
```
raw_avg_test
```

Out[42]:
```
array([3.4, 3.4, 3.4, 3.4, 3.4])
```

## Calculate the RMSE for raw average for both your training data and your test data.

The RMSE (Root Mean Squared Error) measures the average difference between the observed values, and the predicted ones. For this case we are using 3.4 as our global average prediction. Our RMSE being ~1 for our training and test data suggests that our prediction of 3.4 is on average within 1 rating point from the actual scores. Our test set RMSE of 1.07 ratings are slighty closer to the average score than the RMSE of 1.15 of the train data.

In [43]:
```
root_mean_squared_error(train_data.value[train_data['value'].notnull()],raw_avg_train)
```

Out[43]:
```
1.1575836902790226
```

In [44]:
```
root_mean_squared_error(test_data.value[test_data['value'].notnull()],raw_avg_test)
```

Out[44]:
```
1.077032961426901
```

## Using your training data, calculate the bias for each user and each item.

### Taster (User) Bias:

is how different that user tends to rate the ice cream versus the average user. An "ice cream lover" would probably have a positive Taster Bias number. For example Yerlene below has a 0.93 user bias, suggesting she rates all ice cream flavors almost ~1 point on average higher than the average taster.

In [45]:
```
#taster bias
taster_bias = train_data.groupby('Tasters')['value'].mean()
taster_bias = pd.DataFrame(taster_bias)
taster_bias.value = taster_bias.value - raw_avg
taster_bias
```

Out[45]:

| Tasters | value |
|---|---|
| **Aditi** | -1.650000 |
| **Jose** | 0.400000 |

| | |
|---|---|
| **Nancy** | -0.400000 |
| **Nina** | 0.400000 |
| **Sam** | 0.600000 |
| **Yerlene** | 0.933333 |

### Flavor (Item) Bias:

is how different that flavor tends to be rated versus the average flavor. A "tasty flavor" would probably have a positive Flavor Bias number. For example Marionberry Cheesecake below has a 1.35 item bias, suggesting it's being rated by everyone 1.35 points on average higher than the average flavor.

In [46]:
```
#flavor bias
flavor_bias = train_data.groupby('variable')['value'].mean()
flavor_bias = pd.DataFrame(flavor_bias)
flavor_bias.value = flavor_bias.value - raw_avg
flavor_bias
```

Out[46]:

| variable | value |
|---|---|
| **Black Cherry Chip** | 0.200000 |
| **Cookies & Cream** | -1.150000 |
| **Honeycomb** | -0.400000 |
| **Marionberry Cheesecake** | 1.350000 |
| **Mint Chip** | -0.066667 |

## From the raw average, and the appropriate user and item biases, calculate the baseline predictors for every user-item combination.

Our baseline predictor starts with the 3.4 average rating, and adds the individual item bias, and then the taster bias to end up with a rating that takes into account how tasty an ice cream flavor is and if our raters are more strict or forgiving raters.

In [47]:
```
#append avg
base_line_predictors = user_item_matrix.copy()
base_line_predictors['raw_avg'] = raw_avg
base_line_predictors
```

Out[47]:

| | Tasters | variable | value | raw_avg |
|---|---|---|---|---|
| **0** | Sam | Black Cherry Chip | 5.0 | 3.4 |
| **1** | Nina | Black Cherry Chip | 4.0 | 3.4 |
| **2** | Nancy | Black Cherry Chip | 4.0 | 3.4 |
| **3** | Aditi | Black Cherry Chip | 2.0 | 3.4 |
| **4** | Yerlene | Black Cherry Chip | 4.0 | 3.4 |
| **5** | Jose | Black Cherry Chip | 4.0 | 3.4 |
| **6** | Sam | Cookies & Cream | NaN | 3.4 |
| **7** | Nina | Cookies & Cream | 3.0 | 3.4 |
| **8** | Nancy | Cookies & Cream | 2.0 | 3.4 |
| **9** | Aditi | Cookies & Cream | 2.0 | 3.4 |
| **10** | Yerlene | Cookies & Cream | NaN | 3.4 |
| **11** | Jose | Cookies & Cream | 2.0 | 3.4 |
| **12** | Sam | Marionberry Cheesecake | 4.0 | 3.4 |
| **13** | Nina | Marionberry Cheesecake | 5.0 | 3.4 |
| **14** | Nancy | Marionberry Cheesecake | NaN | 3.4 |
| **15** | Aditi | Marionberry Cheesecake | 3.0 | 3.4 |
| **16** | Yerlene | Marionberry Cheesecake | 5.0 | 3.4 |
| **17** | Jose | Marionberry Cheesecake | 5.0 | 3.4 |
| **18** | Sam | Honeycomb | NaN | 3.4 |
| **19** | Nina | Honeycomb | 3.0 | 3.4 |

| | | | | |
|---|---|---|---|---|
| **20** | Nancy | Honeycomb | NaN | 3.4 |
| **21** | Aditi | Honeycomb | 1.0 | 3.4 |
| **22** | Yerlene | Honeycomb | 4.0 | 3.4 |
| **23** | Jose | Honeycomb | 4.0 | 3.4 |
| **24** | Sam | Mint Chip | 4.0 | 3.4 |
| **25** | Nina | Mint Chip | 4.0 | 3.4 |
| **26** | Nancy | Mint Chip | 3.0 | 3.4 |
| **27** | Aditi | Mint Chip | 2.0 | 3.4 |
| **28** | Yerlene | Mint Chip | 5.0 | 3.4 |
| **29** | Jose | Mint Chip | 4.0 | 3.4 |

In [48]:
```
#append biases
base_line_predictors = base_line_predictors.merge(flavor_bias, left_on='variable', right_on='variable', suffixes=(None, '_fla
base_line_predictors = base_line_predictors.merge(taster_bias, left_on='Tasters', right_on='Tasters',suffixes=(None,'_taster_
```

In [49]:
```
#create predicted
base_line_predictors['predicted'] = base_line_predictors['raw_avg'] + base_line_predictors['value_flavor_bias'] + base_line_p
```

In [50]:
```
#above 5 or below 1, assumed to be 5 or 1
base_line_predictors.predicted = base_line_predictors.predicted.clip(1, 5)
```

In [64]:
```
base_line_predictors = pd.DataFrame(base_line_predictors)
```

In [65]:
```
base_line_predictors
```

Out[65]:

| | Tasters | variable | value | raw_avg | value_flavor_bias | value_taster_bias | predicted |
|---|---|---|---|---|---|---|---|
| **0** | Sam | Black Cherry Chip | 5.0 | 3.4 | 0.200000 | 0.600000 | 4.200000 |
| **1** | Nina | Black Cherry Chip | 4.0 | 3.4 | 0.200000 | 0.400000 | 4.000000 |
| **2** | Nancy | Black Cherry Chip | 4.0 | 3.4 | 0.200000 | -0.400000 | 3.200000 |
| **3** | Aditi | Black Cherry Chip | 2.0 | 3.4 | 0.200000 | -1.650000 | 1.950000 |
| **4** | Yerlene | Black Cherry Chip | 4.0 | 3.4 | 0.200000 | 0.933333 | 4.533333 |
| **5** | Jose | Black Cherry Chip | 4.0 | 3.4 | 0.200000 | 0.400000 | 4.000000 |
| **6** | Sam | Cookies & Cream | NaN | 3.4 | -1.150000 | 0.600000 | 2.850000 |
| **7** | Nina | Cookies & Cream | 3.0 | 3.4 | -1.150000 | 0.400000 | 2.650000 |
| **8** | Nancy | Cookies & Cream | 2.0 | 3.4 | -1.150000 | -0.400000 | 1.850000 |
| **9** | Aditi | Cookies & Cream | 2.0 | 3.4 | -1.150000 | -1.650000 | 1.000000 |
| **10** | Yerlene | Cookies & Cream | NaN | 3.4 | -1.150000 | 0.933333 | 3.183333 |
| **11** | Jose | Cookies & Cream | 2.0 | 3.4 | -1.150000 | 0.400000 | 2.650000 |
| **12** | Sam | Marionberry Cheesecake | 4.0 | 3.4 | 1.350000 | 0.600000 | 5.000000 |
| **13** | Nina | Marionberry Cheesecake | 5.0 | 3.4 | 1.350000 | 0.400000 | 5.000000 |
| **14** | Nancy | Marionberry Cheesecake | NaN | 3.4 | 1.350000 | -0.400000 | 4.350000 |
| **15** | Aditi | Marionberry Cheesecake | 3.0 | 3.4 | 1.350000 | -1.650000 | 3.100000 |
| **16** | Yerlene | Marionberry Cheesecake | 5.0 | 3.4 | 1.350000 | 0.933333 | 5.000000 |
| **17** | Jose | Marionberry Cheesecake | 5.0 | 3.4 | 1.350000 | 0.400000 | 5.000000 |
| **18** | Sam | Honeycomb | NaN | 3.4 | -0.400000 | 0.600000 | 3.600000 |
| **19** | Nina | Honeycomb | 3.0 | 3.4 | -0.400000 | 0.400000 | 3.400000 |
| **20** | Nancy | Honeycomb | NaN | 3.4 | -0.400000 | -0.400000 | 2.600000 |
| **21** | Aditi | Honeycomb | 1.0 | 3.4 | -0.400000 | -1.650000 | 1.350000 |
| **22** | Yerlene | Honeycomb | 4.0 | 3.4 | -0.400000 | 0.933333 | 3.933333 |
| **23** | Jose | Honeycomb | 4.0 | 3.4 | -0.400000 | 0.400000 | 3.400000 |
| **24** | Sam | Mint Chip | 4.0 | 3.4 | -0.066667 | 0.600000 | 3.933333 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| **25** | Nina | Mint Chip | 4.0 | 3.4 | -0.066667 | 0.400000 | 3.733333 |
| **26** | Nancy | Mint Chip | 3.0 | 3.4 | -0.066667 | -0.400000 | 2.933333 |
| **27** | Aditi | Mint Chip | 2.0 | 3.4 | -0.066667 | -1.650000 | 1.683333 |
| **28** | Yerlene | Mint Chip | 5.0 | 3.4 | -0.066667 | 0.933333 | 4.266667 |
| **29** | Jose | Mint Chip | 4.0 | 3.4 | -0.066667 | 0.400000 | 3.733333 |

## Calculate the RMSE for the baseline predictors for both your training data and your test data.

The RMSE (Root Mean Squared Error) measures the average difference between the observed values, and the predicted ones. For this case we are using our baseline predictors (3.4 + user_bias + item_bias). Our RMSE being ~0.5 for our training and test data suggests that our baseline predictions are on average within 0.5 rating points from the actual scores. Our model has gotten more accurate as we added the user and item biases. Our training set RMSE of 0.47 ratings are slighty closer to the average score than the RMSE of 0.49 of the train data.

In [52]:
```
train_data1, test_data1 = train_test_split(base_line_predictors, test_size=0.2, random_state=30)
```

In [53]:
```
predicted_train1 = train_data1.predicted[train_data1['value'].notnull()].copy()
```

In [54]:
```
predicted_test1 = test_data1.predicted[test_data1['value'].notnull()].copy()
```

Our testing RMSE went down to 0.47 from the global avg RMSE of 1.08

In [55]:
```
RMSE_train = root_mean_squared_error(train_data1.value[train_data1['value'].notnull()],predicted_train1)
RMSE_train
```

Out[55]:
```
0.468182063351902
```

Our training RMSE went down to 0.49 from the global avg RMSE of 1.16

In [56]:
```
RMSE_test = root_mean_squared_error(test_data1.value[test_data1['value'].notnull()],predicted_test1)
RMSE_test
```

Out[56]:
```
0.4892170615721954
```

## Using our recommender to get predicted scores for untasted flavors

Our RMSE tells us the recommender system is relatively accurate, we see the predicted scores within 0.5 of the actual scores. This hopefully should mean that the scores for flavors our tasters haven't tasted yet should also fall within 0.5 our of predicted scores. These are the flavors that our tasters hadnt tried. It looks like we should really recommend Nancy to try Marionberry Cheesecake with a predicted score of 4.35. Inversely, Nancy should probably avoid Honeycomb with a predicted score of 2.6.

In [68]:
```
base_line_predictors[base_line_predictors.value.isnull()]
```

Out[68]:

| | Tasters | variable | value | raw_avg | value_flavor_bias | value_taster_bias | predicted |
|---|---|---|---|---|---|---|---|
| **6** | Sam | Cookies & Cream | NaN | 3.4 | -1.15 | 0.600000 | 2.850000 |
| **10** | Yerlene | Cookies & Cream | NaN | 3.4 | -1.15 | 0.933333 | 3.183333 |
| **14** | Nancy | Marionberry Cheesecake | NaN | 3.4 | 1.35 | -0.400000 | 4.350000 |
| **18** | Sam | Honeycomb | NaN | 3.4 | -0.40 | 0.600000 | 3.600000 |
| **20** | Nancy | Honeycomb | NaN | 3.4 | -0.40 | -0.400000 | 2.600000 |

## Summarize your results.

I created a global baseline bias recommender system of Van Leeuwen Ice Cream flavors ratings from coworkers to predict if they will like flavors they havent tasted (and rated) before using user and item biases.

The average rating was 3.4. Yerlene was our most positive taster with a +0.93 bias. Aditi was our most negative taster with a bias of -1.65. Marionberry Cheesecake seems like an excellent flavor with a +1.35 bias. Cookies & Cream is least loved with a bias of -1.15.

Our RMSE being ~0.5 for our training and test data suggests that our baseline predictions are on average within 0.5 rating points from the actual scores. Our RMSE tells us the recommender system is relatively accurate, we see the predicted scores within 0.5 of the actual scores.

Using our recommender to get predicted scores for untasted flavors. We should really recommend Nancy to try Marionberry Cheesecake with a predicted score of 4.35.