In [5]:
```python
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

HW 5

In [6]:
```python
train = pd.read_csv("MusicRatingsTrain.csv")
test = pd.read_csv("MusicRatingsTest.csv")
vala = pd.read_csv("MusicRatingsValidationA.csv")
valb = pd.read_csv("MusicRatingsValidationB.csv")
train.head()
```

Out[6]:

	userID	songID	rating	songName	year	artist	genre
0	2071	581	2.484754	Show You How	2004	The Killers	Rock
1	1208	55	1.000000	Luvstruck	1999	Southside Spinners	Electronic
2	1484	510	1.000000	In One Ear	2008	Cage The Elephant	Rock
3	453	299	1.000000	Gardenhead / Leave Me Alone	1992	Neutral Milk Hotel	Rock
4	2238	113	1.000000	Hustler	2006	Simian Mobile Disco	Electronic

Question 1

a

In [7]:
```python
songs = len(pd.unique(train['songID']))
songs
```

Out[7]: 807

In [8]:
```python
users = len(pd.unique(train['userID']))
users
```

Out[8]: 2421

In [9]:
```
stats = train['rating'].describe()
stats
```

Out[9]:
```
count    245997.000000
mean          1.321962
std           0.459894
min           1.000000
25%           1.000000
50%           1.000000
75%           1.494918
max           4.768656
Name: rating, dtype: float64
```

In [10]:
```
median = train['rating'].median()
median
```

Out[10]:
```
1.0
```

b) i.

There are a total of 807 + 2421 = 3228 parameters included in the model. We have to train the model with 245997 observations because there are no null ratings in the training dataset, so the number of observations is the same as the number of ratings.

b) ii.

In [11]:
```
from fancyimpute import BiScaler
```

In [12]:
```
train.sort_index()
train_df = train.pivot_table(index="userID", columns = "songID", values = "
train_mat = train_df.to_numpy()
train_mat
```

Out[12]:
```
array([[       nan,        nan,        nan, ...,        nan,        nan,
               nan],
       [       nan, 1.98983574,        nan, ...,        nan,        nan,
               nan],
       [       nan,        nan,        nan, ...,        nan,        nan,
               nan],
       ...,
       [       nan,        nan,        nan, ...,        nan,        nan,
               nan],
       [       nan,        nan,        nan, ...,        nan,        nan,
               nan],
       [       nan, 1.49491787, 1.49491787, ...,        nan,        nan,
               nan]])
```

In [13]:
```
song_biscaler = BiScaler(scale_rows=False, scale_columns=False, verbose=Tru
train_mat_centered = song_biscaler.fit_transform(train_mat)
```

```
[BiScaler] Initial log residual value = 8.635876
[BiScaler] Iter 1: log residual = -1.278081, log improvement ratio=9.9139
58
[BiScaler] Iter 2: log residual = -2.464318, log improvement ratio=1.1862
37
[BiScaler] Iter 3: log residual = -3.599288, log improvement ratio=1.1349
70
[BiScaler] Iter 4: log residual = -4.728969, log improvement ratio=1.1296
81
[BiScaler] Iter 5: log residual = -5.851105, log improvement ratio=1.1221
36
[BiScaler] Iter 6: log residual = -6.961307, log improvement ratio=1.1102
02
[BiScaler] Iter 7: log residual = -8.052791, log improvement ratio=1.0914
84
[BiScaler] Iter 8: log residual = -9.115833, log improvement ratio=1.0630
43
[BiScaler] Iter 9: log residual = -10.137984, log improvement ratio=1.022
150
[BiScaler] Iter 10: log residual = -11.105953, log improvement ratio=0.96
7969
[BiScaler] Iter 11: log residual = -12.009465, log improvement ratio=0.90
3512
[BiScaler] Iter 12: log residual = -12.845445, log improvement ratio=0.83
5981
[BiScaler] Iter 13: log residual = -13.619498, log improvement ratio=0.77
4052
[BiScaler] Iter 14: log residual = -14.343386, log improvement ratio=0.72
3888
[BiScaler] Iter 15: log residual = -15.030646, log improvement ratio=0.68
7260
[BiScaler] Iter 16: log residual = -15.693214, log improvement ratio=0.66
2568
[BiScaler] Iter 17: log residual = -16.340088, log improvement ratio=0.64
6874
[BiScaler] Iter 18: log residual = -16.977415, log improvement ratio=0.63
7327
[BiScaler] Iter 19: log residual = -17.609141, log improvement ratio=0.63
1726
[BiScaler] Iter 20: log residual = -18.237697, log improvement ratio=0.62
8555
[BiScaler] Iter 21: log residual = -18.864535, log improvement ratio=0.62
6838
[BiScaler] Iter 22: log residual = -19.490503, log improvement ratio=0.62
5968
[BiScaler] Iter 23: log residual = -20.116082, log improvement ratio=0.62
5579
[BiScaler] Iter 24: log residual = -20.741536, log improvement ratio=0.62
5455
[BiScaler] Iter 25: log residual = -21.367004, log improvement ratio=0.62
5468
[BiScaler] Iter 26: log residual = -21.992551, log improvement ratio=0.62
5547
[BiScaler] Iter 27: log residual = -22.618202, log improvement ratio=0.62
```

```
5651
[BiScaler] Iter 28: log residual = -23.243962, log improvement ratio=0.62
5760
[BiScaler] Iter 29: log residual = -23.869823, log improvement ratio=0.62
5861
[BiScaler] Iter 30: log residual = -24.495775, log improvement ratio=0.62
5951
[BiScaler] Iter 31: log residual = -25.121804, log improvement ratio=0.62
6029
[BiScaler] Iter 32: log residual = -25.747898, log improvement ratio=0.62
6094
[BiScaler] Iter 33: log residual = -26.374046, log improvement ratio=0.62
6148
[BiScaler] Iter 34: log residual = -27.000238, log improvement ratio=0.62
6192
[BiScaler] Iter 35: log residual = -27.626466, log improvement ratio=0.62
6228
[BiScaler] Iter 36: log residual = -28.252723, log improvement ratio=0.62
6257
[BiScaler] Iter 37: log residual = -28.879004, log improvement ratio=0.62
6281
[BiScaler] Iter 38: log residual = -29.505303, log improvement ratio=0.62
6299
[BiScaler] Iter 39: log residual = -30.131618, log improvement ratio=0.62
6314
[BiScaler] Iter 40: log residual = -30.757944, log improvement ratio=0.62
6326
[BiScaler] Iter 41: log residual = -31.384280, log improvement ratio=0.62
6336
[BiScaler] Iter 42: log residual = -32.010624, log improvement ratio=0.62
6344
[BiScaler] Iter 43: log residual = -32.636973, log improvement ratio=0.62
6350
[BiScaler] Iter 44: log residual = -33.263328, log improvement ratio=0.62
6354
[BiScaler] Iter 45: log residual = -33.889686, log improvement ratio=0.62
6358
[BiScaler] Iter 46: log residual = -34.516047, log improvement ratio=0.62
6361
[BiScaler] Iter 47: log residual = -35.142411, log improvement ratio=0.62
6364
[BiScaler] Iter 48: log residual = -35.768777, log improvement ratio=0.62
6366
[BiScaler] Iter 49: log residual = -36.395144, log improvement ratio=0.62
6367
[BiScaler] Iter 50: log residual = -37.021512, log improvement ratio=0.62
6368
[BiScaler] Iter 51: log residual = -37.647882, log improvement ratio=0.62
6369
[BiScaler] Iter 52: log residual = -38.274252, log improvement ratio=0.62
6370
[BiScaler] Iter 53: log residual = -38.900623, log improvement ratio=0.62
6370
[BiScaler] Iter 54: log residual = -39.526994, log improvement ratio=0.62
6371
[BiScaler] Iter 55: log residual = -40.153366, log improvement ratio=0.62
6372
```

```
[BiScaler] Iter 56: log residual = -40.779738, log improvement ratio=0.62
6372
[BiScaler] Iter 57: log residual = -41.406109, log improvement ratio=0.62
6372
[BiScaler] Iter 58: log residual = -42.032481, log improvement ratio=0.62
6372
[BiScaler] Iter 59: log residual = -42.658853, log improvement ratio=0.62
6372
[BiScaler] Iter 60: log residual = -43.285226, log improvement ratio=0.62
6373
[BiScaler] Iter 61: log residual = -43.911602, log improvement ratio=0.62
6376
[BiScaler] Iter 62: log residual = -44.537966, log improvement ratio=0.62
6364
[BiScaler] Iter 63: log residual = -45.164347, log improvement ratio=0.62
6381
[BiScaler] Iter 64: log residual = -45.790719, log improvement ratio=0.62
6373
[BiScaler] Iter 65: log residual = -46.417087, log improvement ratio=0.62
6368
[BiScaler] Iter 66: log residual = -47.043472, log improvement ratio=0.62
6385
[BiScaler] Iter 67: log residual = -47.669846, log improvement ratio=0.62
6373
[BiScaler] Iter 68: log residual = -48.296203, log improvement ratio=0.62
6357
[BiScaler] Iter 69: log residual = -48.922611, log improvement ratio=0.62
6409
[BiScaler] Iter 70: log residual = -49.548918, log improvement ratio=0.62
6307
[BiScaler] Iter 71: log residual = -50.175297, log improvement ratio=0.62
6378
[BiScaler] Iter 72: log residual = -50.801673, log improvement ratio=0.62
6376
[BiScaler] Iter 73: log residual = -51.428247, log improvement ratio=0.62
6575
[BiScaler] Iter 74: log residual = -52.054331, log improvement ratio=0.62
6084
[BiScaler] Iter 75: log residual = -52.680838, log improvement ratio=0.62
6507
[BiScaler] Iter 76: log residual = -53.306945, log improvement ratio=0.62
6106
[BiScaler] Iter 77: log residual = -53.933222, log improvement ratio=0.62
6278
[BiScaler] Iter 78: log residual = -54.559726, log improvement ratio=0.62
6504
[BiScaler] Iter 79: log residual = -55.187109, log improvement ratio=0.62
7383
[BiScaler] Iter 80: log residual = -55.813326, log improvement ratio=0.62
6217
[BiScaler] Iter 81: log residual = -56.439541, log improvement ratio=0.62
6215
[BiScaler] Iter 82: log residual = -57.064246, log improvement ratio=0.62
4705
[BiScaler] Iter 83: log residual = -57.694122, log improvement ratio=0.62
9876
[BiScaler] Iter 84: log residual = -58.320642, log improvement ratio=0.62
```

```
6519
[BiScaler] Iter 85: log residual = -58.942397, log improvement ratio=0.62
1756
[BiScaler] Iter 86: log residual = -59.567736, log improvement ratio=0.62
5339
[BiScaler] Iter 87: log residual = -60.192510, log improvement ratio=0.62
4774
[BiScaler] Iter 88: log residual = -60.820491, log improvement ratio=0.62
7981
[BiScaler] Iter 89: log residual = -61.426252, log improvement ratio=0.60
5760
[BiScaler] Iter 90: log residual = -62.053421, log improvement ratio=0.62
7169
[BiScaler] Iter 91: log residual = -62.688779, log improvement ratio=0.63
5359
[BiScaler] Iter 92: log residual = -63.261398, log improvement ratio=0.57
2619
[BiScaler] Iter 93: log residual = -63.846200, log improvement ratio=0.58
4802
[BiScaler] Iter 94: log residual = -64.449178, log improvement ratio=0.60
2978
[BiScaler] Iter 95: log residual = -64.830932, log improvement ratio=0.38
1755
[BiScaler] Iter 96: log residual = -65.351248, log improvement ratio=0.52
0316
[BiScaler] Iter 97: log residual = -65.519441, log improvement ratio=0.16
8192
[BiScaler] Iter 98: log residual = -65.800609, log improvement ratio=0.28
1169
[BiScaler] Iter 99: log residual = -65.967886, log improvement ratio=0.16
7276
[BiScaler] Iter 100: log residual = -66.089223, log improvement ratio=0.1
21337
```

In [14]:

```
# Access the alpha (row_mean) and beta (column) values
alpha = song_biscaler.row_means
alpha
```

Out[14]:

```
array([ 0.23597732,  0.37836219, -0.12202861, ..., -0.00099498,
       -0.09292935, -0.07297946])
```

In [15]:
```
beta = song_biscaler.column_means
beta
```

Out[15]:
```
array([1.1085277 , 1.28189302, 1.45665654, 1.15972853, 1.28211814,
       1.12983062, 1.47101127, 1.28577961, 1.35033994, 1.43749681,
       1.16378442, 1.21511576, 1.4097627 , 1.23517437, 1.25240525,
       1.18324648, 1.57097994, 1.24492046, 1.50054534, 1.21816032,
       1.19678   , 1.45653583, 1.23245392, 1.1058456 , 1.41882857,
       2.1370632 , 1.40920159, 1.12776456, 1.32721814, 1.21159835,
       1.11964289, 1.20034543, 1.17693884, 1.58421436, 1.45240405,
       1.27537058, 1.23174929, 1.15390764, 1.29276939, 1.11644992,
       1.37531285, 1.14228572, 1.21817148, 1.26869206, 1.20196407,
       1.18522847, 1.2132076 , 1.20720276, 1.12599004, 1.17277644,
       1.26637501, 1.19294522, 1.16589595, 2.15784144, 1.7676486 ,
       1.19068097, 1.12156866, 1.13149055, 1.85976324, 1.17344651,
       1.41308493, 1.10785718, 1.09833489, 1.35995412, 1.31486195,
       1.2667538 , 1.37337888, 1.11536594, 1.24306441, 1.45978565,
       1.16395292, 1.21780274, 1.26713043, 1.20356712, 1.20973988,
       1.4430925 , 1.23889642, 1.3110657 , 1.39755168, 1.35581768,
       1.15864186, 1.19324072, 1.13430115, 1.14487043, 1.30399132,
       1.13730439, 1.15994019, 1.33331344, 1.27431343, 1.31721697,
       1.28752372, 1.30361275, 1.58147094, 1.24530107, 1.2061634 ,
```

In [16]:
```
train.sort_index()
train_n = train.pivot_table(index="songID")
train_n['beta']= beta
sorted = train_n.sort_values('beta', ascending=False)
sorted
```

Out[16]:

songID	rating	userID	year	beta
54	2.157240	1180.916137	1990	2.157841
26	2.134060	1196.513648	2001	2.137063
439	2.016957	1222.037037	2009	2.038437
637	1.950094	1221.344209	2002	1.937541
600	1.877886	1182.569444	2008	1.889610
...	...	...	...	...
317	1.077181	1119.037879	1995	1.073712
157	1.144404	1181.421769	2009	1.073472
165	1.069733	1316.726619	1996	1.072993
96	1.197060	1242.075188	2005	1.071808
481	1.132071	1186.690476	1978	1.062140

807 rows × 4 columns

In [17]:
```
train[train['songID'].isin([54, 26, 439])]
```

Out[17]:

	userID	songID	rating	songName	year	artist	genre
11	2352	439	1.784426	Secrets	2009	OneRepublic	Rock
229	1025	26	1.494918	Undo	2001	Bjork	Rock
231	695	54	2.149164	You're The One	1990	Dwight Yoakam	Country
268	357	54	3.022958	You're The One	1990	Dwight Yoakam	Country
762	1284	439	1.784426	Secrets	2009	OneRepublic	Rock
...	...	...	...	...	...	...	...
245482	1034	26	1.000000	Undo	2001	Bjork	Rock
245558	650	54	1.494918	You're The One	1990	Dwight Yoakam	Country
245731	1873	439	1.000000	Secrets	2009	OneRepublic	Rock
245739	1605	54	2.149164	You're The One	1990	Dwight Yoakam	Country
245790	833	26	4.073169	Undo	2001	Bjork	Rock

2619 rows × 7 columns

songID songName Year Artist Genre Beta
54 You're The One 1990 Dwight Yoakam Country 2.157841
26 Undo 2001 Bjork Rock 2.137063
439 Secrets 2009 OneRepublic Rock 2.038437

The three largest beta's are the three most popular songs after removing for the bias due to how particular users rate certain songs. So here the 3 largest Bj values equal the 3 most popular songs

b) iii.

In [18]:
```
train.sort_index()
train_a = train.pivot_table(index="userID")
train_a['alpha']= alpha
# train_a
sorted_a = train_a.sort_values('alpha', ascending=False)
sorted_a
```

Out[18]:

	rating	songID	year	alpha
userID				
1540	2.353383	395.373494	2003.945783	0.967014
838	2.105060	398.661157	2005.669421	0.814727
1569	2.077348	391.358491	2005.858491	0.794237
950	2.112970	426.039604	2003.782178	0.691602
345	2.109287	420.044248	2003.017699	0.674504
...	...	...	...	...
2076	1.059971	426.614035	2004.421053	-0.404973
1550	1.063416	373.462687	2003.925373	-0.405059
1595	1.014345	403.652174	2004.086957	-0.408122
241	1.006971	390.014085	2005.450704	-0.409727
2413	1.079893	413.809524	2003.174603	-0.431814

2421 rows × 4 columns

userID alpha
1540 0.967014
838 0.814727
1569 0.794237

b) iv.

In [19]:
```
user_df = train_df.reset_index()[['userID']]
```

In [20]:
```
test.sort_index()
test_df = test.pivot_table(index="userID", columns = "songID", values = "ra
```

In [21]:
```
test_df=pd.merge(user_df,test_df.reset_index(),how='outer',on='userID')
test_df=test_df.set_index('userID')
test_mat = test_df.to_numpy()
test_mat
```

Out[21]:
```
array([[nan, nan, nan, ..., nan, nan, nan],
       [nan, nan, nan, ..., nan, nan, nan],
       [nan, nan, nan, ..., nan, nan, nan],
       ...,
       [nan, nan, nan, ..., nan,  1., nan],
       [nan, nan, nan, ..., nan, nan, nan],
       [nan, nan, nan, ..., nan, nan, nan]])
```

In [22]:
```
test_mask = ~np.isnan(test_mat)
print(np.sum(test_mask))
```

```
14471
```

In [23]:
```
import copy
song_centered_0 = copy.copy(train_mat_centered)
song_centered_0[np.isnan(song_centered_0)]=0
song_centered_0
```

Out[23]:
```
array([[0.        , 0.        , 0.        , ..., 0.        , 0.        ,
        0.        ],
       [0.        , 0.32958053, 0.        , ..., 0.        , 0.        ,
        0.        ],
       [0.        , 0.        , 0.        , ..., 0.        , 0.        ,
        0.        ],
       ...,
       [0.        , 0.        , 0.        , ..., 0.        , 0.        ,
        0.        ],
       [0.        , 0.        , 0.        , ..., 0.        , 0.        ,
        0.        ],
       [0.        , 0.28600431, 0.11124078, ..., 0.        , 0.        ,
        0.        ]])
```

In [24]:
```
song_BiScaler_filled = song_biscaler.inverse_transform(song_centered_0)
song_BiScaler_filled
```

Out[24]:
```
array([[1.34450503, 1.51787035, 1.69263387, ..., 1.55709616, 1.41842513,
        1.40129053],
       [1.4868899 , 1.98983574, 1.83501874, ..., 1.69948103, 1.56081   ,
        1.5436754 ],
       [0.98649909, 1.15986441, 1.33462793, ..., 1.19909023, 1.0604192 ,
        1.0432846 ],
       ...,
       [1.10753272, 1.28089804, 1.45566156, ..., 1.32012386, 1.18145283,
        1.16431823],
       [1.01559836, 1.18896367, 1.3637272 , ..., 1.22818949, 1.08951846,
        1.07238386],
       [1.03554825, 1.49491787, 1.49491787, ..., 1.24813938, 1.10946835,
        1.09233375]])
```

In [25]:
```
def masked_mae(X_true, X_pred, mask):
    masked_diff = X_true[mask] - X_pred[mask]
    return np.mean(np.abs(masked_diff))

def masked_mse(X_true, X_pred, mask):
    masked_diff = X_true[mask] - X_pred[mask]
    return np.mean(masked_diff ** 2)

def OSR2(mse_model, mse_baseline):
    return 1 - mse_model/mse_baseline
```

In [26]:
```
test_mae = masked_mae(test_mat, song_BiScaler_filled, test_mask)
print("Biscale MAE %s " % (test_mae/4)) #Note that we normalize MAE and RMS

test_mse = masked_mse(test_mat, song_BiScaler_filled, test_mask)
print("Biscale RMSE %s " % (np.sqrt(test_mse)/4))

baseline_pred = np.mean(train)[2]
baseline_model = baseline_pred*np.ones((2421, 807))
baseline_mse = masked_mse(test_mat, baseline_model, test_mask)
print("Biscale OSR2 %s" % OSR2(test_mse, baseline_mse))
```

```
Biscale MAE 0.0747571775133017
Biscale RMSE 0.0985693901114982
Biscale OSR2 0.2702373901935097
```

c) i.

There are a total of 807 + 2421 +807*k* + *2421*k= 3228(k+1) parameters included in the model. We have to train the model with 245997 observations.

c) ii.

In [27]:
```
vala.sort_index()
vala_df = vala.pivot_table(index="userID", columns = "songID", values = "ra
```

In [28]:
```
vala_df=pd.merge(user_df,vala_df.reset_index(),how='outer',on='userID')
vala_df=vala_df.set_index('userID')
vala_mat = vala_df.to_numpy()
vala_mat
```

Out[28]:
```
array([[nan, nan, nan, ..., nan, nan, nan],
       [nan, nan, nan, ..., nan, nan, nan],
       [nan, nan, nan, ..., nan, nan, nan],
       ...,
       [nan, nan, nan, ..., nan, nan, nan],
       [nan, nan, nan, ..., nan, nan, nan],
       [nan, nan, nan, ..., nan, nan, nan]])
```

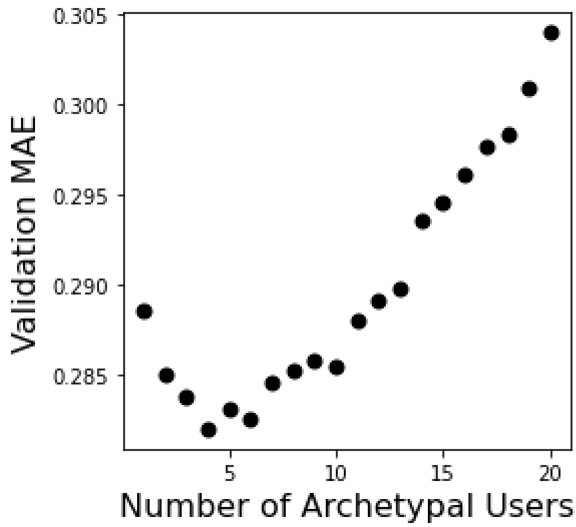
In [29]:
```
vala_mask = ~np.isnan(vala_mat)
print(np.sum(vala_mask))
```

```
14470
```

In [30]:
```
from fancyimpute import SoftImpute
songs_valA_mae_lst = []
for i in range(20):
    param_cv = i+1
    songs_soft_imputer_cv = SoftImpute(max_rank=param_cv, verbose=False)
    songs_centered_filled_cv = songs_soft_imputer_cv.fit_transform(train_ma
    songs_filled_cv = song_biscaler.inverse_transform(songs_centered_filled
    songs_filled_cv = np.clip(songs_filled_cv, 1, 5)
    songs_valA_mae_cv = masked_mae(vala_mat, songs_filled_cv, vala_mask)
    songs_valA_mae_lst.append(songs_valA_mae_cv)
    print('iter %s - Validation MAE %s' % (param_cv, songs_valA_mae_cv))
```

```
iter 1 - Validation MAE 0.2885634518667774
iter 2 - Validation MAE 0.28500121297797326
iter 3 - Validation MAE 0.2837867835747125
iter 4 - Validation MAE 0.2820231031496958
iter 5 - Validation MAE 0.2831439714205493
iter 6 - Validation MAE 0.2826094102603166
iter 7 - Validation MAE 0.28458407348601406
iter 8 - Validation MAE 0.2851844490181127
iter 9 - Validation MAE 0.2858040476209231
iter 10 - Validation MAE 0.28547984285183964
iter 11 - Validation MAE 0.2879752234507606
iter 12 - Validation MAE 0.28910102265746274
iter 13 - Validation MAE 0.2898415855585168
iter 14 - Validation MAE 0.2935896748594369
iter 15 - Validation MAE 0.2945718362743434
iter 16 - Validation MAE 0.29618740759125856
iter 17 - Validation MAE 0.2976931208274731
iter 18 - Validation MAE 0.2983513563550037
iter 19 - Validation MAE 0.30094743761703024
iter 20 - Validation MAE 0.30400269235062105
```

```
In [31]: import matplotlib.pyplot as plt
x = range(1,21)
y = songs_valA_mae_lst
plt.figure(figsize=(4, 4))
plt.scatter(x, y, linewidth=2, color='black')
plt.xlabel('Number of Archetypal Users', fontsize=16)
plt.ylabel('Validation MAE', fontsize=16)
plt.show()
```



I trained the models with k from 1-20 and evaluated the performance on validation set A. I chose k=4 as the number of archetypes because it has the lowest Validation MAE on the validation set A.

c) iii.

```
In [32]: songs_soft_imputer = SoftImpute(max_rank=4, verbose=False) #use the best
songs_centered_filled = songs_soft_imputer.fit_transform(train_mat_centered
songs_filled_matrix = song_biscaler.inverse_transform(songs_centered_filled
songs_filled_matrix = np.clip(songs_filled_matrix, 1, 5)
```

```
In [33]: test_mae = masked_mae(test_mat, songs_filled_matrix, test_mask)
print("MAE %s " % (test_mae/4)) #Note that we normalize MAE and RMSE by the

test_mse = masked_mse(test_mat, songs_filled_matrix, test_mask)
print("RMSE %s " % (np.sqrt(test_mse)/4))

baseline_pred = np.mean(train_df)[2]
baseline_model = baseline_pred*np.ones((2421, 807))
baseline_mse = masked_mse(test_mat, baseline_model, test_mask)
print("OSR2 %s" % OSR2(test_mse, baseline_mse))
```

```
MAE 0.07063690502316015
RMSE 0.09598278651787194
OSR2 0.30874774832022855
```

d) i.

Linear Regression

In [34]:
```
train['genre'] = train.genre.astype('category')
train['year'] = train.year.astype('category')
test['genre'] = test.genre.astype('category')
test['year'] = test.year.astype('category')
test
```

Out[34]:

	userID	songID	rating	songName	year	artist	genre
0	853	54	1.989836	You're The One	1990	Dwight Yoakam	Country
1	608	34	1.000000	Nah!	2002	Shania Twain	Country
2	9	54	3.969507	You're The One	1990	Dwight Yoakam	Country
3	1862	54	2.279344	You're The One	1990	Dwight Yoakam	Country
4	329	80	1.494918	Tim McGraw	2006	Taylor Swift	Country
...	...	...	...	...	...	...	...
14466	443	615	1.494918	Daylight	2002	Coldplay	Rock
14467	1061	657	1.000000	Kissy Kissy	2003	The Kills	Rock
14468	610	653	1.000000	The Laws Have Changed	2003	The New Pornographers	Rock
14469	690	657	1.000000	Kissy Kissy	2003	The Kills	Rock
14470	230	657	1.000000	Kissy Kissy	2003	The Kills	Rock

14471 rows × 7 columns

In [35]:
```python
import statsmodels.formula.api as smf
ols = smf.ols(formula='rating ~ genre+year', data=train)
res = ols.fit()
print(res.summary())
```

```
                            OLS Regression Results
==============================================================================
=====
Dep. Variable:                 rating   R-squared:
0.034
Model:                            OLS   Adj. R-squared:
0.034
Method:                 Least Squares   F-statistic:
261.6
Date:                Wed, 28 Apr 2021   Prob (F-statistic):
0.00
Time:                        13:56:00   Log-Likelihood:                -1.537
3e+05
No. Observations:              245997   AIC:                             3.07
5e+05
Df Residuals:                  245963   BIC:                             3.07
9e+05
Df Model:                          33
Covariance Type:            nonrobust
==============================================================================
==============
                          coef    std err          t      P>|t|      [0.0
25      0.975]
------------------------------------------------------------------------------
--------------
Intercept               2.0937      0.024     87.425      0.000       2.0
47       2.141
genre[T.Electronic]    -0.0777      0.011     -6.813      0.000      -0.1
00      -0.055
genre[T.Folk]          -0.0974      0.012     -7.881      0.000      -0.1
22      -0.073
genre[T.Pop]           -0.1351      0.011    -12.056      0.000      -0.1
57      -0.113
genre[T.Rap]           -0.1118      0.012     -9.211      0.000      -0.1
36      -0.088
genre[T.RnB]           -0.2781      0.012    -23.772      0.000      -0.3
01      -0.255
genre[T.Rock]          -0.1704      0.011    -15.423      0.000      -0.1
92      -0.149
year[T.1976]           -0.4801      0.027    -17.484      0.000      -0.5
34      -0.426
year[T.1978]           -0.7912      0.041    -19.351      0.000      -0.8
71      -0.711
year[T.1979]           -0.1324      0.032     -4.110      0.000      -0.1
96      -0.069
year[T.1985]           -0.5300      0.043    -12.281      0.000      -0.6
15      -0.445
year[T.1986]           -0.4073      0.027    -15.162      0.000      -0.4
60      -0.355
year[T.1987]           -0.5412      0.028    -19.487      0.000      -0.5
96      -0.487
year[T.1988]           -0.3531      0.026    -13.483      0.000      -0.4
```

```
04      -0.302
year[T.1990]          -0.0660      0.027     -2.480      0.013      -0.1
18      -0.014
year[T.1991]          -0.6384      0.025    -25.474      0.000      -0.6
88      -0.589
year[T.1992]          -0.5042      0.024    -20.807      0.000      -0.5
52      -0.457
year[T.1993]          -0.5394      0.024    -22.354      0.000      -0.5
87      -0.492
year[T.1995]          -0.4788      0.026    -18.487      0.000      -0.5
30      -0.428
year[T.1996]          -0.8069      0.023    -35.725      0.000      -0.8
51      -0.763
year[T.1997]          -0.5502      0.024    -23.338      0.000      -0.5
96      -0.504
year[T.1998]          -0.8162      0.027    -30.583      0.000      -0.8
69      -0.764
year[T.1999]          -0.5511      0.022    -24.644      0.000      -0.5
95      -0.507
year[T.2000]          -0.6126      0.022    -28.162      0.000      -0.6
55      -0.570
year[T.2001]          -0.5470      0.022    -24.951      0.000      -0.5
90      -0.504
year[T.2002]          -0.5946      0.022    -27.436      0.000      -0.6
37      -0.552
year[T.2003]          -0.6362      0.021    -29.633      0.000      -0.6
78      -0.594
year[T.2004]          -0.7157      0.022    -32.678      0.000      -0.7
59      -0.673
year[T.2005]          -0.6430      0.022    -29.902      0.000      -0.6
85      -0.601
year[T.2006]          -0.6379      0.021    -29.753      0.000      -0.6
80      -0.596
year[T.2007]          -0.6421      0.021    -29.924      0.000      -0.6
84      -0.600
year[T.2008]          -0.6450      0.021    -30.112      0.000      -0.6
87      -0.603
year[T.2009]          -0.5772      0.021    -26.930      0.000      -0.6
19      -0.535
year[T.2010]          -0.6543      0.022    -30.022      0.000      -0.6
97      -0.612
=========================================================================
=====
Omnibus:                    63019.216   Durbin-Watson:
2.003
Prob(Omnibus):                  0.000   Jarque-Bera (JB):           14508
5.217
Skew:                           1.457   Prob(JB):
0.00
Kurtosis:                       5.380   Cond. No.
153.
=========================================================================
=====

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is co
rrectly specified.
```

In [36]:
```
test_pred_ols = res.predict(test)
test_mae_ols = np.mean(np.abs(test['rating'] - test_pred_ols))
print("MAE %s " % (test_mae_ols/4)) #Note that we normalize MAE and RMSE by

test_mse_ols = np.mean((test['rating'] - test_pred_ols)**2)
print("RMSE %s " % (np.sqrt(test_mse_ols)/4))

print("OSR2 %s " % OSR2(test_mse_ols, baseline_mse))
```

```
MAE 0.0923556815359249
RMSE 0.11312545968408343
OSR2 0.039780403459793834
```

Random Forest

In [41]:
```
from sklearn.ensemble import RandomForestRegressor

X_train = train.drop(columns = ["genre","songName","artist"])
y_train = train["rating"]
rf = RandomForestRegressor(max_features=len(X_train.columns), min_samples_l
                           n_estimators = 500, random_state=88, verbose=2)
rf.fit(X_train, y_train)
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent w
orkers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.1s remaining:
0.0s
```

In [42]:
```
test_pred_rf=rf.predict(test.drop(columns = ["genre","songName","artist"]))
test_mae_rf = np.mean(np.abs(test['rating'] - test_pred_rf))
print("MAE %s " % (test_mae_rf/4)) #Note that we normalize MAE and RMSE by

test_mse_rf = np.mean((test['rating'] - test_pred_rf)**2)
print("RMSE %s " % (np.sqrt(test_mse_rf)/4))

print("OSR2 %s " % OSR2(test_mse_rf, baseline_mse))
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent w
orkers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:
0.0s

MAE 3.0931830823804327e-06
RMSE 0.0002774117114856834
OSR2 0.9999942256987917

[Parallel(n_jobs=1)]: Done 500 out of 500 | elapsed:    0.3s finished
```

d) ii.

In [ ]:
```
val_pred_cf = train_filled_matrix[valB_mask]

blend_valB_df = songs_valB.reset_index()[['userID','movieID','rating']]

blend_valB_df['val_pred_cf']=val_pred_cf
blend_valB_df
```

In [ ]:
```
val_pred_ols = res.predict(valB)
blend_valB_df['val_pred_ols']=val_pred_ols
blend_valB_df
```

In [ ]:
```
blend_valB_df['val_pred_rf']=val_pred_rf
blend_valB_df
```

In [ ]:
```
val_pred_blended =blending_res.predict(blend_valB_df)
```

In [ ]:
```
test_mae_blended = np.mean(np.abs(test_mat[test_mask] - val_pred_blended))
print("Test_blended MAE  %s " % (test_mae_blended/4)) #Note that we normali

test_mse_blended = np.mean((movie_lens_test_mat[movie_lens_test_mask] - tes
print("Test_blended RMSE %s " % (np.sqrt(test_mse_blended)/4))

print("Test_blended OSR2 %s " % OSR2(test_mse_blended, baseline_mse))
```