Collaborative Recommendation system

some imports

In [4]:

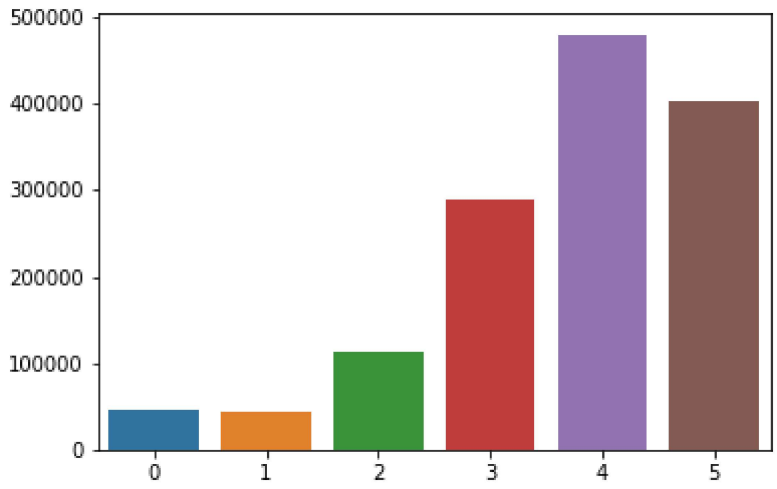
```
import pandas as pd
from google.colab import drive
# drive.mount('/content/gdrive')

#remove the below hashtag if you don't have the required library
# !pip install surprise
from surprise import Reader,Dataset
from surprise import SVD,SlopeOne, NMF, NormalPredictor, BaselineOnly, CoCluster
from surprise.model_selection import cross_validate
import seaborn as sns
jsonFileName= "/content/gdrive/MyDrive/goodreads_reviews_spoiler.csv"
```

Reading the Dataset

In [2]:
```
main_df = pd.read_csv(jsonFileName)
rate_keys = list(main_df['rating'].unique())
rate_keys = sorted(rate_keys, reverse=True)
ratings_dict = {}
for i in rate_keys: ratings_dict[i] = 0
enum = enumerate(pd.read_csv(jsonFileName,chunksize=10000),start=1)
for i in enum:
  for rate in i[1]['rating']:
    ratings_dict[rate]+=1
keys = list(ratings_dict.keys())
# # get values in the same order as keys, and parse percentage values
vals = [(ratings_dict[k]) for k in keys]
sns.barplot(x=keys, y=vals)
```

Out[2]: <matplotlib.axes._subplots.AxesSubplot at 0x7f71dacf3710>



In [5]:
```
reader = Reader(rating_scale=(0, 5))
data = Dataset.load_from_df(main_df[['user_id', 'book_id', 'rating']], reader)
data.df.sample(5)
```

Out[5]:

	user_id	book_id	rating
1333527	621e07a81873c07f1093ce551a39a3f6	17210470	3
1243450	4d19bb85549390ca40c45f9c3894298e	5093	2
1027905	171350ed6c59a8c75f23b289e10596fa	13253276	3
494786	151bd68492cdea380d1b816aa01ece88	18134249	5
319741	958afdf8dd56da62298b04321be0d827	22055262	5

Test the algorithms

- in this section we're going to test some algoritms
- and evaluate the best algorithm using Root Mean Squared Error metric

In [7]:
```
# SVD ,NormalPredictor, CoClustering,BaselineOnly , SlopeOne, NMF
result = []

for algo in [SVD(),NormalPredictor(), CoClustering(),BaselineOnly()]:

    results = cross_validate(algo, data, measures=['RMSE'], cv=3, verbose=True)

    t1 = pd.DataFrame.from_dict(results).mean(axis=0)

    t1 = t1.append(pd.Series([str(algo).split(' ')[0].split('.')[-1]], index=['A

    result.append(t1)

res = pd.DataFrame(result).set_index('Algorithm').sort_values('test_rmse')
res
```

```
Evaluating RMSE of algorithm SVD on 3 split(s).

                  Fold 1  Fold 2  Fold 3  Mean    Std
RMSE (testset)    1.1173  1.1148  1.1160  1.1160  0.0010
Fit time          58.59   59.37   63.40   60.46   2.11
Test time         6.53    7.26    6.59    6.80    0.33
Evaluating RMSE of algorithm NormalPredictor on 3 split(s).

                  Fold 1  Fold 2  Fold 3  Mean    Std
RMSE (testset)    1.6631  1.6669  1.6676  1.6659  0.0020
Fit time          1.87    2.76    2.70    2.44    0.41
Test time         7.33    5.79    7.08    6.73    0.67
Evaluating RMSE of algorithm CoClustering on 3 split(s).

                  Fold 1  Fold 2  Fold 3  Mean    Std
RMSE (testset)    1.1361  1.1267  1.1260  1.1296  0.0046
Fit time          40.38   40.49   43.85   41.58   1.61
Test time         6.90    6.67    5.91    6.49    0.42
Estimating biases using als...
Estimating biases using als...
Estimating biases using als...
Evaluating RMSE of algorithm BaselineOnly on 3 split(s).

                  Fold 1  Fold 2  Fold 3  Mean    Std
RMSE (testset)    1.0932  1.0945  1.0932  1.0937  0.0006
Fit time          8.38    8.87    9.58    8.94    0.50
Test time         7.02    6.63    5.81    6.48    0.51
```

Out[7]:

Algorithm	test_rmse	fit_time	test_time
BaselineOnly	1.093671	8.943585	6.484528
SVD	1.116033	60.455054	6.795038
CoClustering	1.129602	41.575189	6.491079
NormalPredictor	1.665865	2.444099	6.734090

Hyper parameters tuning

- we found the best algorithm for our task
- now we gonna find the best parameters for the algorithm

In [8]:
```
from surprise.model_selection import GridSearchCV
from surprise.model_selection import train_test_split,cross_validate

"""reg_u : the The regularization parameter for users (default is 15)"""
"""reg_i : the The regularization parameter for items(default is 10)"""
"""n_epochs : The number of iteration of the ALS procedure (default is 10)"""

bsl_options = {
    "method": ["als", "sgd"],
    'n_epochs': [5,10],
    'reg_u': [12,20],
    'reg_i': [5,10]
}

param_grid = {"bsl_options": bsl_options}

gs = GridSearchCV(BaselineOnly, param_grid, measures=["rmse"], cv=3)
fit_res = gs.fit(data)

print(gs.best_score["rmse"])
print(gs.best_params["rmse"])
```

```
Estimating biases using als...
Estimating biases using als...
Estimating biases using als...
Estimating biases using als...
Estimating biases using als...
Estimating biases using als...
Estimating biases using als...
Estimating biases using als...
Estimating biases using als...
Estimating biases using als...
Estimating biases using als...
Estimating biases using als...
Estimating biases using als...
Estimating biases using als...
Estimating biases using als...
Estimating biases using als...
Estimating biases using als...
Estimating biases using als...
Estimating biases using als...
Estimating biases using als...
Estimating biases using als...
Estimating biases using als...
Estimating biases using als...
Estimating biases using als...
Estimating biases using als...
Estimating biases using als...
Estimating biases using als...
Estimating biases using als...
Estimating biases using als...
Estimating biases using sgd...
Estimating biases using sgd...
Estimating biases using sgd...
Estimating biases using sgd...
Estimating biases using sgd...
Estimating biases using sgd...
Estimating biases using sgd...
Estimating biases using sgd...
Estimating biases using sgd...
Estimating biases using sgd...
```

```
Estimating biases using sgd...
Estimating biases using sgd...
Estimating biases using sgd...
Estimating biases using sgd...
Estimating biases using sgd...
Estimating biases using sgd...
Estimating biases using sgd...
Estimating biases using sgd...
Estimating biases using sgd...
Estimating biases using sgd...
Estimating biases using sgd...
Estimating biases using sgd...
Estimating biases using sgd...
Estimating biases using sgd...
Estimating biases using sgd...
1.091807244545283
{'bsl_options': {'method': 'als', 'n_epochs': 10, 'reg_u': 12, 'reg_i': 5}}
```

Train the final module

- After we figured out the best parameters for the algorithm
- Finally we are going to train the module

In [20]:
```
#the BaselineOnly Model is the best model from the tested models

from surprise.model_selection import train_test_split,cross_validate
from surprise import accuracy,BaselineOnly

print('Using ALS')
bsl_options = {'method': 'als',
               'n_epochs': 10,
               'reg_u': 12,
               'reg_i': 5
               }

algo = BaselineOnly(bsl_options=bsl_options)

cross_validate(algo, data, measures=['RMSE'], cv=3, verbose=True)

trainset, testset = train_test_split(data, test_size=0.25)

algo = BaselineOnly(bsl_options=bsl_options)

predictions = algo.fit(trainset).test(testset)

accuracy.rmse(predictions)
```

```
Using ALS
Estimating biases using als...
Estimating biases using als...
Estimating biases using als...
Evaluating RMSE of algorithm BaselineOnly on 3 split(s).

                  Fold 1  Fold 2  Fold 3  Mean    Std
RMSE (testset)    1.0918  1.0921  1.0912  1.0917  0.0004
Fit time          8.52    9.66    9.57    9.25    0.52
Test time         5.44    4.66    4.68    4.93    0.37
Estimating biases using als...
RMSE: 1.0889
```

Out[20]:
```
1.0889207739016644
```

Show some results

In [21]:
```
def get_Iu(uid):
    """returns the number of items rated by the user """
    try:
        return len(trainset.ur[trainset.to_inner_uid(uid)])
    except ValueError: # user was not part of the trainset
        return 0

def get_Ui(iid):
    """returns the number of users that have rated the item """
    try:
        return len(trainset.ir[trainset.to_inner_iid(iid)])
    except ValueError:
        return 0
```

In [22]:
```
df = pd.DataFrame(predictions, columns=['uid', 'iid', 'rui', 'est', 'details'])
df['Iu'] = df.uid.apply(get_Iu)
df['Ui'] = df.iid.apply(get_Ui)
df['err'] = abs(df.est - df.rui)

#top ten best predictions
best_predictions = df.sort_values(by='err')[:10]
best_predictions
```

Out[22]:

	uid	iid	rui	est	details	Iu	Ui	err
18688	ff448f7e29c82edfdf9fde5d4be8901f	18168902	5.0	5.0	{'was_impossible': False}	96	33	0.0
250135	17f1730775ed8fd9bd4c3cb3066cddd2	17927182	5.0	5.0	{'was_impossible': False}	139	41	0.0
74100	d033cacfec2b2d012bf5dd38131ab2f4	77523	5.0	5.0	{'was_impossible': False}	102	76	0.0
121404	7f76b4e6eeffbc251b1174dc7ecf4352	17202452	5.0	5.0	{'was_impossible': False}	346	154	0.0
297778	272ed5d6ea8581d6c9871aa96ebf5931	17756559	5.0	5.0	{'was_impossible': False}	201	154	0.0
44782	d132ed28aed3457cbcbfa5803a4a1918	16169533	0.0	0.0	{'was_impossible': False}	338	8	0.0
144806	def079f6192273cb2b3dfb896520c18a	24574656	0.0	0.0	{'was_impossible': False}	267	18	0.0
93655	2d8d2e5f9ca86670024142030f073bd4	32066878	5.0	5.0	{'was_impossible': False}	225	35	0.0
171397	e7d6f6549cbf7d4cd6a5a34bea357758	8621462	5.0	5.0	{'was_impossible': False}	230	725	0.0
297662	7ac68d284be15768081bd3e86a1431c9	11710373	5.0	5.0	{'was_impossible': False}	250	79	0.0

In [23]:
```
#top ten worst predictions
worst_predictions = df.sort_values(by='err')[-10:]
worst_predictions
```

Out[23]:

	uid	iid	rui	est	details	Iu	Ui	err
139482	53643c79f99eb10a8f0343cb082169f9	23389993	0.0	5.0	{'was_impossible': False}	367	41	5.0
252143	11525d39b06fcc7a4b41f068f34f3bb7	30826152	0.0	5.0	{'was_impossible': False}	332	19	5.0
84410	30047196afd5b95176e1a8a935c99acf	16060716	0.0	5.0	{'was_impossible': False}	205	156	5.0
21958	4ad272c41af8a81a381278a7587d14a9	17237477	0.0	5.0	{'was_impossible': False}	564	42	5.0
325968	e27409822e45be1a4c5c848783db0e24	30633337	0.0	5.0	{'was_impossible': False}	84	67	5.0
130922	2d6e0ea93a403354261dc77771359b0a	16006	0.0	5.0	{'was_impossible': False}	269	29	5.0
223566	3ffe42a4ad71cb54553f0e2976aa22c2	26148181	0.0	5.0	{'was_impossible': False}	203	55	5.0
178460	087ae3f0d82f3b66c1d2e788527d5cfd	6178648	0.0	5.0	{'was_impossible': False}	243	87	5.0
162367	0655c9bc5331d3abc3a23a64fc2cc5dc	29283884	0.0	5.0	{'was_impossible': False}	153	277	5.0
114366	9c237e22706f608960e45c41128af976	12649718	0.0	5.0	{'was_impossible': False}	112	210	5.0

In [ ]:
```
```