In [1]:

```
from surprise import Dataset
from surprise import Reader
from surprise import SVD
from surprise import Dataset
import os
from surprise.model_selection import cross_validate
from surprise.model_selection import KFold
from surprise import KNNBasic
from surprise import NMF
import surprise.accuracy
import pandas as pd
import numpy as np
```

In [2]:

```
#load data from a file
file_path = os.path.expanduser('restaurant_ratings.txt')
reader = Reader(line_format='user item rating timestamp', sep='\t')
data = Dataset.load_from_file(file_path, reader=reader)
```

In [3]:

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

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

                  Fold 1  Fold 2  Fold 3  Mean    Std
RMSE (testset)    0.9499  0.9427  0.9446  0.9457  0.0031
MAE (testset)     0.7497  0.7440  0.7453  0.7463  0.0024
Fit time          9.89    8.98    9.17    9.35    0.39
Test time         0.57    0.58    0.53    0.56    0.02
```

Out[3]:

```
{'test_rmse': array([0.94989913, 0.94268487, 0.94460542]),
 'test_mae': array([0.74966624, 0.74404881, 0.74528417]),
 'fit_time': (9.893847227096558, 8.983985424041748, 9.165119171142578),
 'test_time': (0.5659093856811523, 0.5811896324157715, 0.5325772762298584)}
```

In [4]:

```
# PMF
algo2 = SVD(biased=False)
cross_validate(algo2, data, measures=['RMSE', 'MAE'], cv=3, verbose=True)
```

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

                  Fold 1  Fold 2  Fold 3  Mean    Std
RMSE (testset)    0.9657  0.9648  0.9675  0.9660  0.0011
MAE (testset)     0.7608  0.7613  0.7625  0.7616  0.0007
Fit time          8.62    8.21    10.49   9.11    0.99
Test time         0.50    0.49    0.72    0.57    0.11
```

Out[4]:

```
{'test_rmse': array([0.96573508, 0.96482593, 0.96747769]),
 'test_mae': array([0.76084124, 0.76129585, 0.76251572]),
 'fit_time': (8.62000036239624, 8.209943771362305, 10.488538265228271),
 'test_time': (0.49788594245910645, 0.49398159980773926, 0.722059488296508
8)}
```

In [ ]:

In [5]:

```
algo3 = NMF()
cross_validate(algo3, data, measures=['RMSE', 'MAE'], cv=3, verbose=True)
```

```
Evaluating RMSE, MAE of algorithm NMF on 3 split(s).

                  Fold 1  Fold 2  Fold 3  Mean    Std
RMSE (testset)    0.9778  0.9773  0.9720  0.9757  0.0026
MAE (testset)     0.7689  0.7688  0.7644  0.7674  0.0021
Fit time          9.92    8.63    8.78    9.11    0.58
Test time         0.48    0.48    0.41    0.45    0.03
```

Out[5]:

```
{'test_rmse': array([0.97781482, 0.97726288, 0.97198659]),
 'test_mae': array([0.76893172, 0.76877083, 0.76436599]),
 'fit_time': (9.922745943069458, 8.63115930557251, 8.782884120941162),
 'test_time': (0.4806191921234131, 0.47755002975463867, 0.4058911800384521
5)}
```

In [ ]:

In [6]:

```
# User based
algo4 = KNNBasic(sim_options = {'user_based': True })
cross_validate(algo4, data, measures=['RMSE', 'MAE'], cv=3, verbose=True)
```

```
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Evaluating RMSE, MAE of algorithm KNNBasic on 3 split(s).

                  Fold 1  Fold 2  Fold 3  Mean    Std
RMSE (testset)    0.9895  0.9859  0.9878  0.9877  0.0014
MAE (testset)     0.7832  0.7797  0.7797  0.7808  0.0016
Fit time          0.75    0.76    0.78    0.76    0.01
Test time         10.40   10.46   10.19   10.35   0.11
```

Out[6]:

```
{'test_rmse': array([0.98947729, 0.98592916, 0.98777981]),
 'test_mae': array([0.78315787, 0.77966616, 0.779668  ]),
 'fit_time': (0.7476444244384766, 0.7597825527191162, 0.7799127101898193),
 'test_time': (10.3979651927948, 10.45864462852478, 10.191497564315796)}
```

In [7]:

```
# Item based
algo5 = KNNBasic(sim_options = {'user_based': False })
cross_validate(algo5, data, measures=['RMSE', 'MAE'], cv=3, verbose=True)
```

```
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Evaluating RMSE, MAE of algorithm KNNBasic on 3 split(s).

                  Fold 1  Fold 2  Fold 3  Mean    Std
RMSE (testset)    0.9914  0.9806  0.9840  0.9853  0.0045
MAE (testset)     0.7840  0.7763  0.7818  0.7807  0.0032
Fit time          1.08    1.06    1.01    1.05    0.03
Test time         11.30   11.56   11.39   11.42   0.11
```

Out[7]:

```
{'test_rmse': array([0.9914325 , 0.98062422, 0.98397833]),
 'test_mae': array([0.78395824, 0.77626149, 0.78177725]),
 'fit_time': (1.0849952697753906, 1.0550470352172852, 1.0078012943267822),
 'test_time': (11.297596454620361, 11.56394362449646, 11.39256763458252)}
```

In [ ]:

In [8]:

```
kf = KFold(n_splits=3)


# for  i in range(4):
#     exec(f'RMSE_{i}=[]')
#     exec(f'MAE_{i}=[]')
RMSE = []
MAE = []


algo_list = []
algo_list.append(SVD())
algo_list.append(SVD(biased=False))
algo_list.append(NMF())
algo_list.append(KNNBasic(sim_options = {'user_based': True}))
algo_list.append(KNNBasic(sim_options = {'user_based': False}))
```

In [9]:

```
i = 0
for trainset, testset in kf.split(data):
    i += 1
    # train and test algorithm.

    for algo in algo_list:
        algo.fit(trainset)
        predictions = algo.test(testset)

        RMSE.append(surprise.accuracy.rmse(predictions, verbose=False))
        MAE.append(surprise.accuracy.mae(predictions, verbose=False))
```

```
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
```

In [63]:

```
RMSE_mat = np.asarray(RMSE)
MAE_mat = np.asarray(MAE)
RMSE_mat = RMSE_mat.reshape((3,5))
MAE_mat = MAE_mat.reshape((3,5))

algolist=["  SVD  ", "  PMF  ", "  NMF  ","User based","Item based"]


# print(MAE_mat)
for i in range(3):
    print("\n Rmse for fold : "+str(i+1)+" = ")
    print(algolist)
    [print(RMSE_mat[i])]
    print("\n Mae for fold : "+str(i+1)+" = ")
    print(algolist)

    print(RMSE_mat[i])

avg_RMSE = np.mean(RMSE_mat, axis=0)
avg_MAE = np.mean(MAE_mat, axis=0)


print()
print("              ",algolist)
print ("Average RSME is :",avg_RMSE)
print ("Average MAE is  :",avg_MAE)
```

```
 Rmse for fold : 1 =
['  SVD  ', '  PMF  ', '  NMF  ', 'User based', 'Item based']
[0.94359996 0.96298722 0.97415664 0.98773502 0.98724828]

 Mae for fold : 1 =
['  SVD  ', '  PMF  ', '  NMF  ', 'User based', 'Item based']
[0.94359996 0.96298722 0.97415664 0.98773502 0.98724828]

 Rmse for fold : 2 =
['  SVD  ', '  PMF  ', '  NMF  ', 'User based', 'Item based']
[0.94476946 0.96648624 0.97637795 0.99151436 0.98308464]

 Mae for fold : 2 =
['  SVD  ', '  PMF  ', '  NMF  ', 'User based', 'Item based']
[0.94476946 0.96648624 0.97637795 0.99151436 0.98308464]

 Rmse for fold : 3 =
['  SVD  ', '  PMF  ', '  NMF  ', 'User based', 'Item based']
[0.94753781 0.97254006 0.97813709 0.99048192 0.98829049]

 Mae for fold : 3 =
['  SVD  ', '  PMF  ', '  NMF  ', 'User based', 'Item based']
[0.94753781 0.97254006 0.97813709 0.99048192 0.98829049]

               ['  SVD  ', '  PMF  ', '  NMF  ', 'User based', 'Item based']
Average RSME is : [0.94530241 0.96733784 0.9762239  0.98991043 0.9862078 ]
Average MAE is  : [0.7456942  0.76241704 0.76607208 0.78237705 0.78070182]
```

In [11]:

```
metrics =['MSD','cosine','pearson']
for metric in metrics:
    algo = KNNBasic(sim_options = {'name':metric,'user_based': True })
    exec(f'perf_user_{metric} = cross_validate(algo, data, measures=["RMSE","MAE"], cv=3, v
    algo2 = KNNBasic(sim_options = {'name':metric,'user_based': False })
    exec(f'perf_item_{metric} = cross_validate(algo, data, measures=["RMSE","MAE"], cv=3, v
```

```
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
```

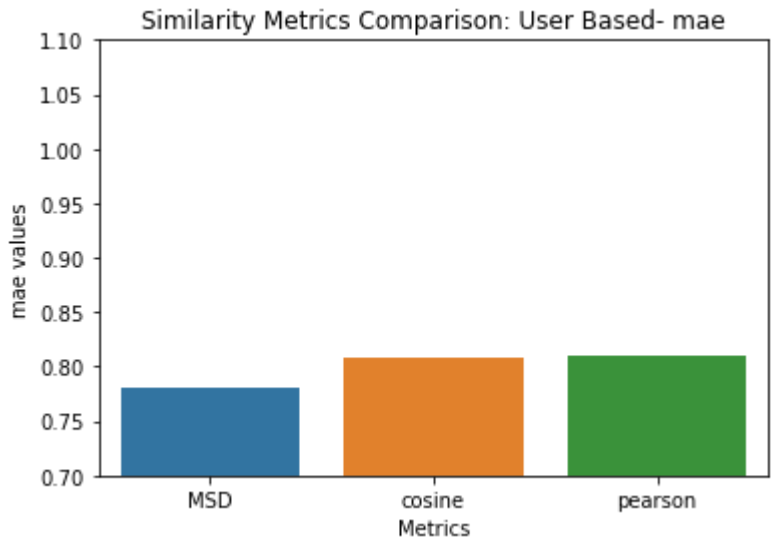
In [12]:

```
print(perf_user_MSD)
import seaborn as sns
import matplotlib.pyplot as plt
```

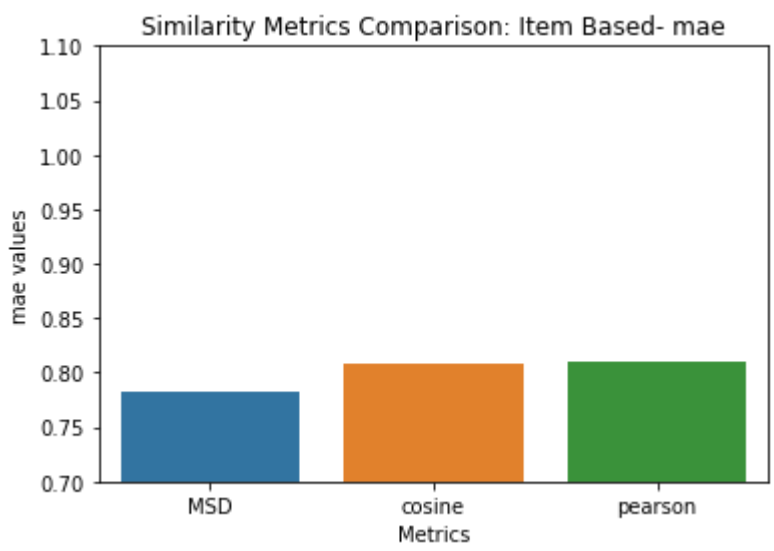
```
{'test_rmse': array([0.98222762, 0.99365745, 0.98703151]), 'test_mae': array
([0.77536781, 0.78607831, 0.7804483 ]), 'fit_time': (0.7297124862670898, 0.7
066473960876465, 0.8134076595306396), 'test_time': (10.288438320159912, 9.87
8553628921509, 10.002488374710083)}
```

In [13]:

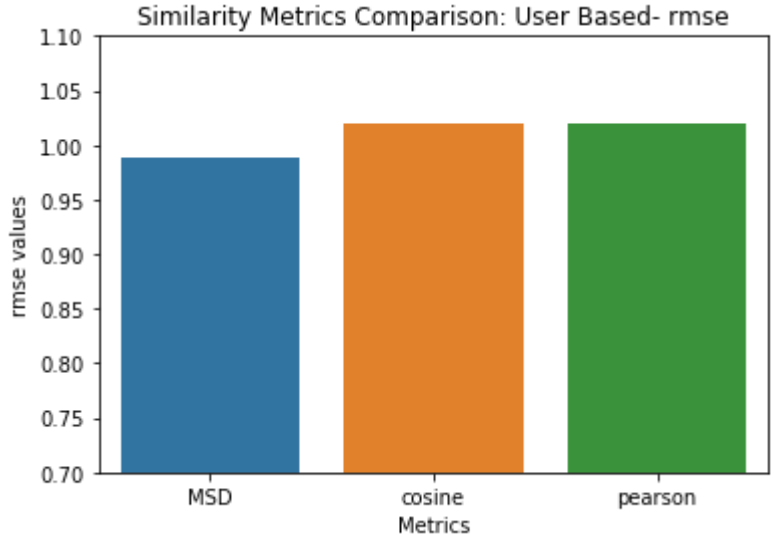
```
metrics = np.array(['MSD', 'cosine', 'pearson'])
def fun(err):
    vals = [np.mean(perf_user_MSD['test_{}'.format(err)]), np.mean(perf_user_cosine['test_{
    series = pd.Series(name='{}'.format(err), data=vals)
    ax = sns.barplot(metrics, series.values)
    ax.set_title("Comparision of metric:{} on User Based".format(err))
    ax.set_ylabel('{} values'.format(err))
    ax.set_xlabel('Metrics')
    ax.set_ylim(.7, 1.1)
    plt.show()
    print(vals)

    vals = [np.mean(perf_item_MSD['test_{}'.format(err)]), np.mean(perf_item_cosine['test_{
    series = pd.Series(name='{}'.format(err), data=vals)
    ax = sns.barplot(metrics, series.values)
    ax.set_title("Comparision of metric:{} on Item Based ".format(err))
    ax.set_ylabel('{} values'.format(err))
    ax.set_xlabel('Metrics')
    ax.set_ylim(.7, 1.1)
    plt.show()
    print(vals)

error= ['mae','rmse']
for err in error:
    fun(err)
```



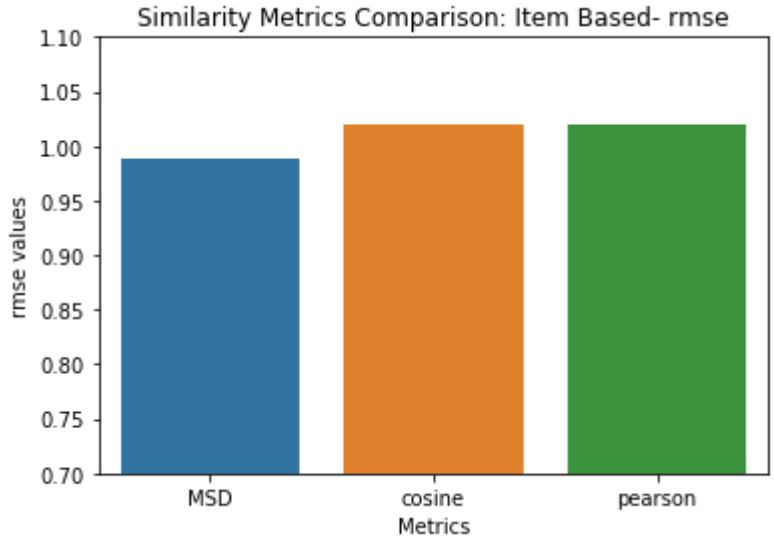
[0.780631474629455, 0.8077269742201704, 0.8098229145198877]




```
[0.7816253564438235, 0.8079645194550412, 0.8095608379028657]
```




```
[0.9876388594462657, 1.0197429978226313, 1.020362490527477]
```




```
[0.9889441745328998, 1.020269346378727, 1.0195388728066044]
```

In [31]:

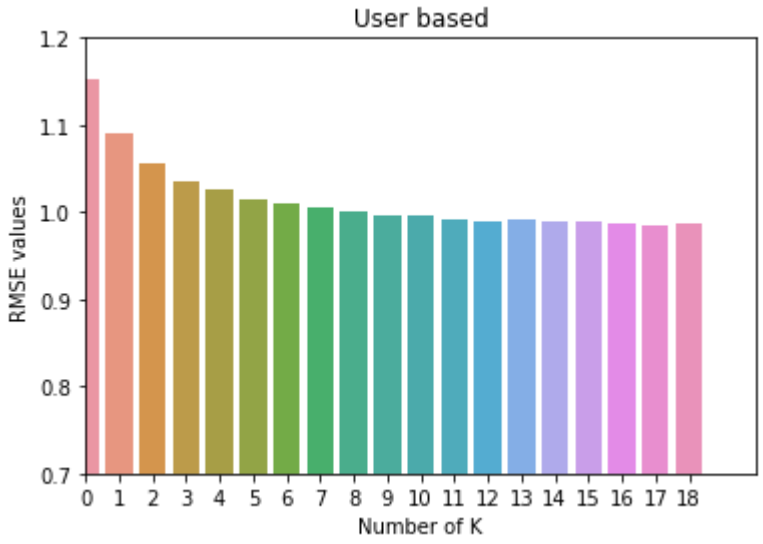
```
user_Based_RMSE_of_different_k = []
item_Based_RMSE_of_different_k = []
for i in range(1,40):
    algo = KNNBasic(k=i, sim_options = {
            'name': 'MSD',
            'user_based': True
            })
    algo2 =KNNBasic(k=i, sim_options = {
            'name': 'MSD',
            'user_based': False
            })


    perf_UserBased_MSD = cross_validate(algo, data, measures=['RMSE'],cv=3, verbose=True)
    perf_ItemBased_MSD = cross_validate(algo2, data, measures=['RMSE'],cv=3, verbose=True)

    print("_____________________________________")
    print("K= ", i)
    user_Based_RMSE_of_different_k.append(np.mean(perf_UserBased_MSD['test_rmse']))
    item_Based_RMSE_of_different_k.append(np.mean(perf_ItemBased_MSD['test_rmse']))
```

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

                  Fold 1  Fold 2  Fold 3  Mean    Std
RMSE (testset)    0.9932  0.9860  0.9860  0.9884  0.0034
Fit time          0.27    0.28    0.28    0.28    0.01
Test time         3.58    3.73    3.64    3.65    0.06
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Evaluating RMSE of algorithm KNNBasic on 3 split(s).

                  Fold 1  Fold 2  Fold 3  Mean    Std
RMSE (testset)    0.9865  0.9915  0.9812  0.9864  0.0042
Fit time          0.39    0.40    0.41    0.40    0.01
Test time         4.13    4.17    4.17    4.16    0.02

_____________________________________
K=  39
```

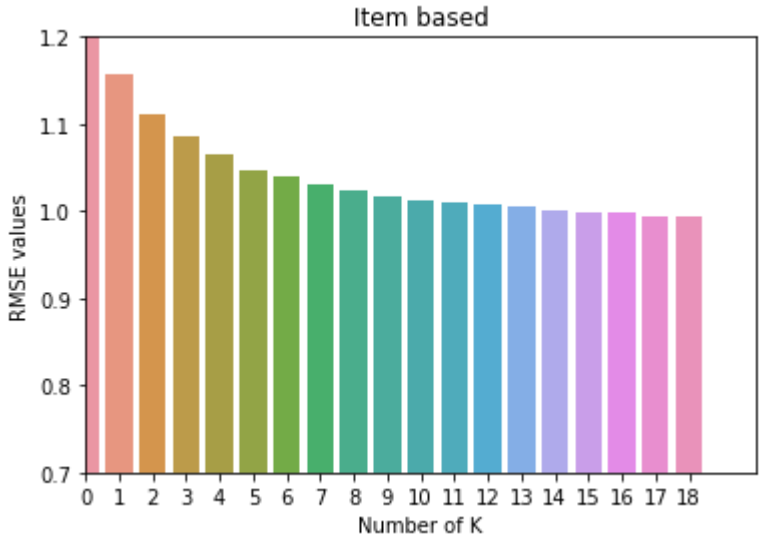
In [53]:

```
based = ['user','item']
def fun2(i,j):
    us = [user_Based_RMSE_of_different_k[1:j],item_Based_RMSE_of_different_k[1:j]]
    min_RMSE_index = np.argmin(us[i])
    print("Best K= ", min_RMSE_index)
    print("Best RMSE= ", us[i][min_RMSE_index])

    series = pd.Series(name='rmse', data=us[i])

    ax = sns.barplot(series.index, series.values)
    ax.set_ylabel('RMSE values')
    ax.set_xlabel('Number of K')
    ax.set_xlim(0, j)
    if i==0:
        ax.set_title("User based")
    else:
        ax.set_title("Item based")

    ax.set_ylim(.7, 1.2)
    plt.show()

for i,base in enumerate(based):
    fun2(i,20)
```

```
Best K=  17
Best RMSE=  0.98542366886906
```



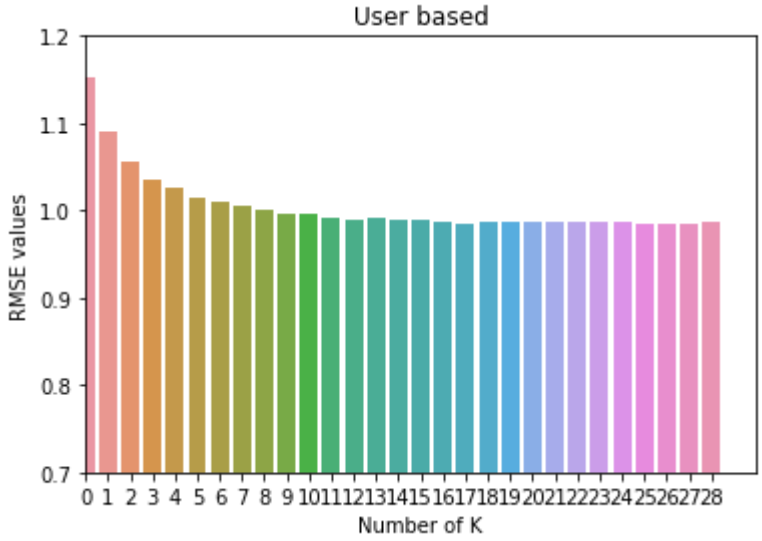

```
Best K=  18
Best RMSE=  0.9932969907656043
```


Item based
RMSE values
Number of K
1.2
1.1
1.0
0.9
0.8
0.7
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18

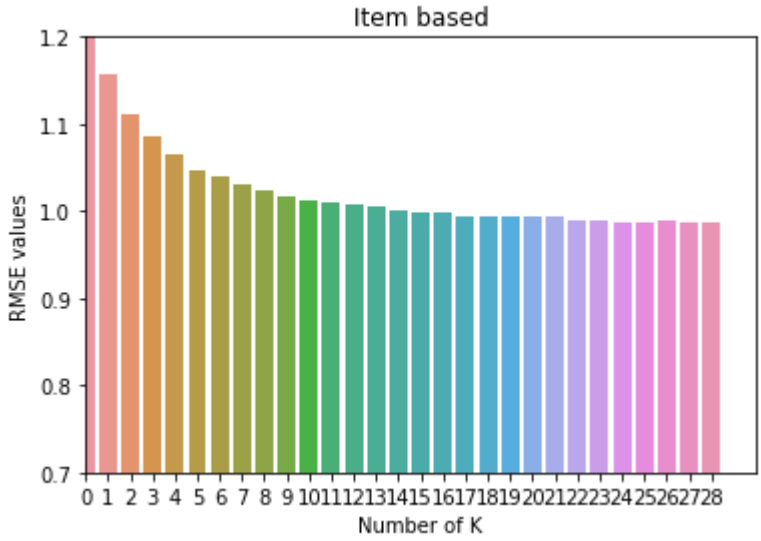
In [54]:

```
for i,base in enumerate(based):
    fun2(i,30)
```

```
Best K=  25
Best RMSE=  0.9852557209329196
```



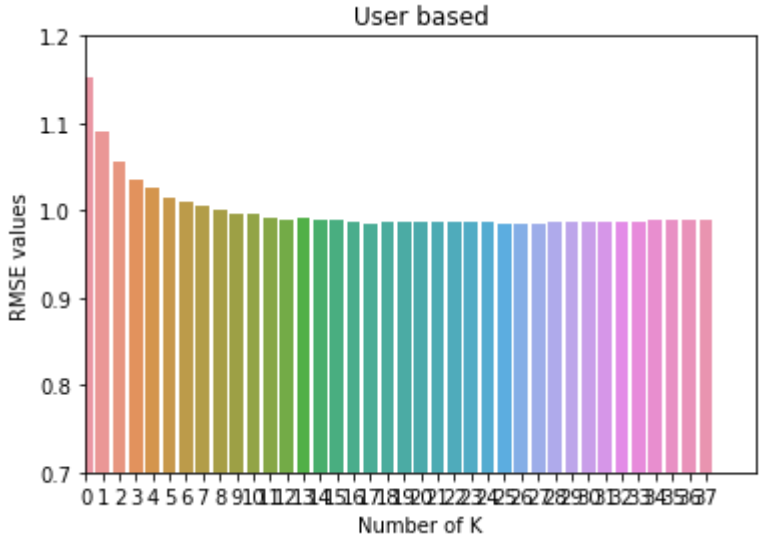

```
Best K=  28
Best RMSE=  0.9868287532562047
```



In [55]:

```
for i,base in enumerate(based):
    fun2(i,40)
```

```
Best K=  25
Best RMSE=  0.9852557209329196
```




```
Best K=  36
Best RMSE=  0.9852857069828861
```