In [1]:
```
import pandas as pd
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
#from sklearn.cross_validation import cross_val_score
from sklearn.model_selection import cross_val_score
from surprise import SVD
from surprise import Dataset
from surprise.reader import Reader
from surprise.model_selection import cross_validate
from sklearn.model_selection import train_test_split
from surprise.model_selection.search import GridSearchCV
import matplotlib.pyplot as plt
```

In [2]:
```
#read the anime file
#anime = pd.read_csv('C:\\Users\\przem\\Documents\\anime.csv', sep=',')
anime = pd.read_csv('anime.csv', sep=',')
anime.head(5)
len(anime)
```

Out[2]: 12294

In [3]:
```
#read the rating file
#rating = pd.read_csv('C:\\Users\\przem\\Documents\\rating.csv', sep
=',')
rating = pd.read_csv('rating.csv', sep=',')
#train and test split
train,test = train_test_split(rating,shuffle = True, test_size = 0.2)
rating.describe()

#len(rating)
```

Out[3]:

	user_id	anime_id	rating

	user_id	anime_id	rating
count	7.813737e+06	7.813737e+06	7.813737e+06
mean	3.672796e+04	8.909072e+03	6.144030e+00
std	2.099795e+04	8.883950e+03	3.727800e+00
min	1.000000e+00	1.000000e+00	-1.000000e+00
25%	1.897400e+04	1.240000e+03	6.000000e+00
50%	3.679100e+04	6.213000e+03	7.000000e+00
75%	5.475700e+04	1.409300e+04	9.000000e+00
max	7.351600e+04	3.451900e+04	1.000000e+01

Filtering

In [4]:

```
#Filter Data. Return df where column name == vale
def FilterByColumnValue(data, columnName, value):
    filtered=data.loc[data[columnName].isin(value)]
    return filtered

#Return df which contains string 'value'
def FilterByColumnHas(data, columnName, value):
    filtered=data.loc[data[columnName].str.contains(value)]
    return filtered
```

Count and make list of users that have made less than 100 reviews

In [5]:

```
from collections import Counter

#counter=Counter()

#Count users and sum of their reviews
```

```
counter=Counter(rating['user_id'])


#Filter all reviewers with more than 99 reviews
filt={x : counter[x] for x in counter if counter[x] <= 200}
reviewers=sorted(filt,key=filt.get,reverse=True)
```

Filter users:

In [6]:
```
#filter all but movies
anime_data=FilterByColumnValue(anime,'type',['Movie'])
#make list of anime_id's in Movies
anime_data=anime_data['anime_id'].values

#filter rating data with reviwer<99 list
rating_x=FilterByColumnValue(rating,'user_id',reviewers)

#Filter rating data with previously made list of movies
rating_x=FilterByColumnValue(rating_x,'anime_id',anime_data)

#filter all unrated movies
rating_x=rating_x[rating_x['rating'] >0]


len((rating_x['user_id'].unique()))
print(len(rating_x))

anime_id=rating_x['anime_id'].values
anime_x=FilterByColumnValue(anime,'anime_id',anime_id)
print(len(anime_data))
```

```
418536
2348
```

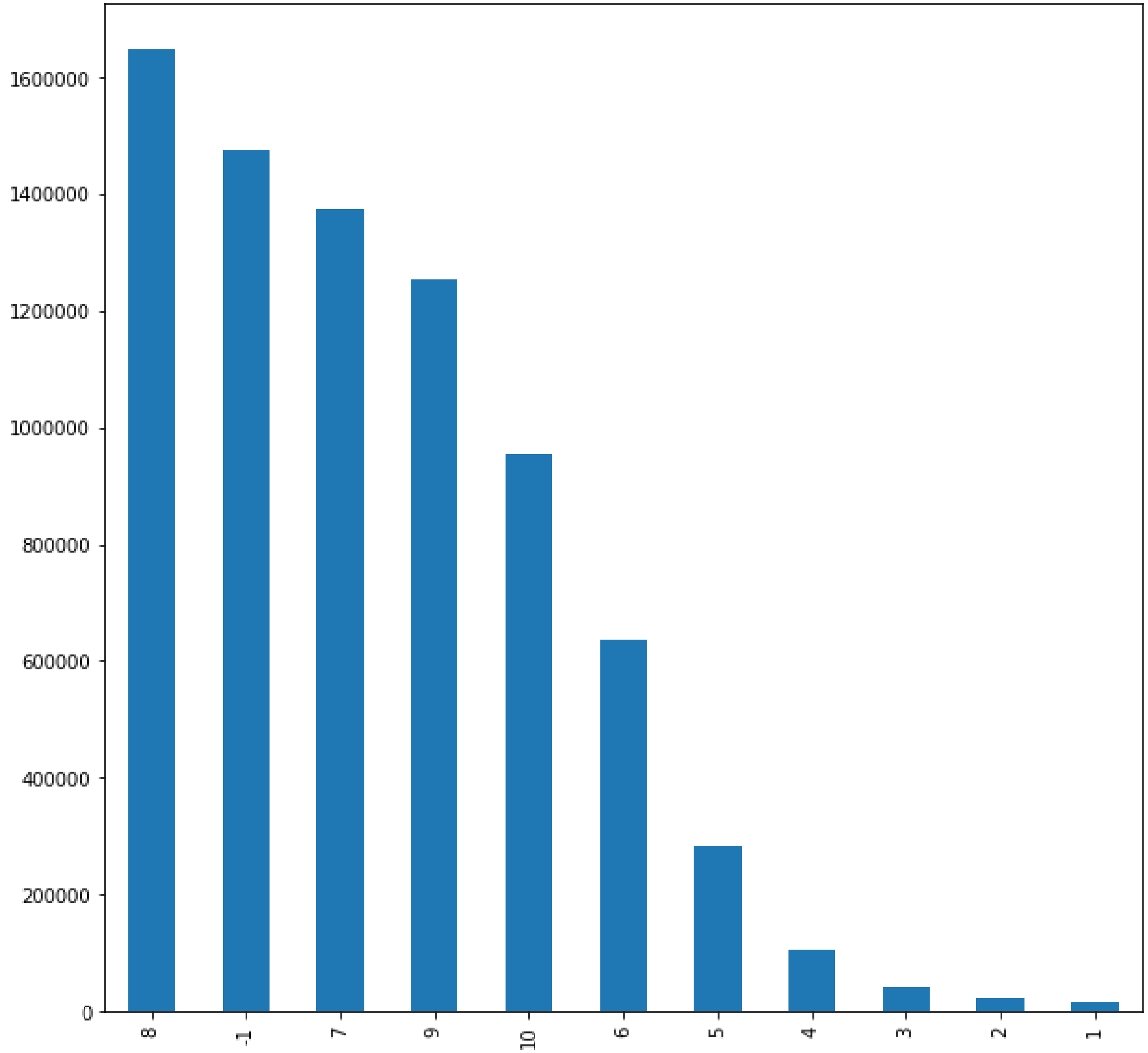
In [60]:
```
def OptionalFilter():
    #remove all rating < 5
    anime_data=anime[anime['rating'] >4.0]
```

```
    #remove all members < 150k
    anime_data=anime_data[anime_data['members'] >1000]

    #remove all but movies
    anime_data=filterByColumnValue(anime_data,'type',['Movie'])
    anime_filterdata=anime_data['anime_id'].values


    len(anime_data)

    #filter review with anime_filtered
    rating_train=filterByColumnValue(rating,'anime_id',anime_filterdata
)
    rating_train=rating_train[rating_train['rating'] >0]
    len(rating_train)

    len(anime_data['genre'].unique())
    len(anime_data)
```

Plot review amounts

Review amount - raw data (all types)

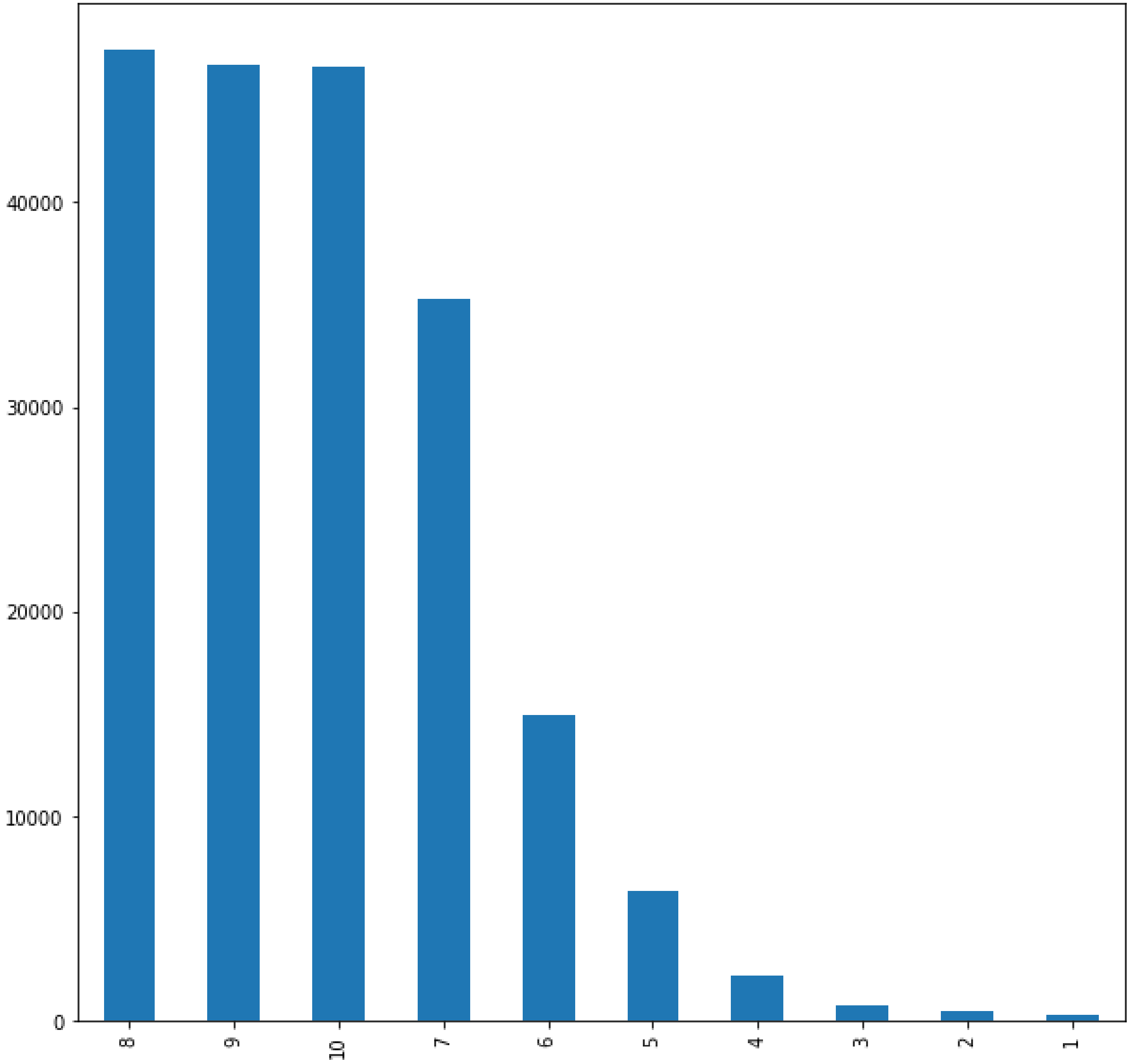
In [11]:
```
rating.rating.value_counts().plot(kind='bar',figsize=(10,10))
plt.show()
len(rating)
```




```
Out[11]: 7813737
```

Review amount filtered

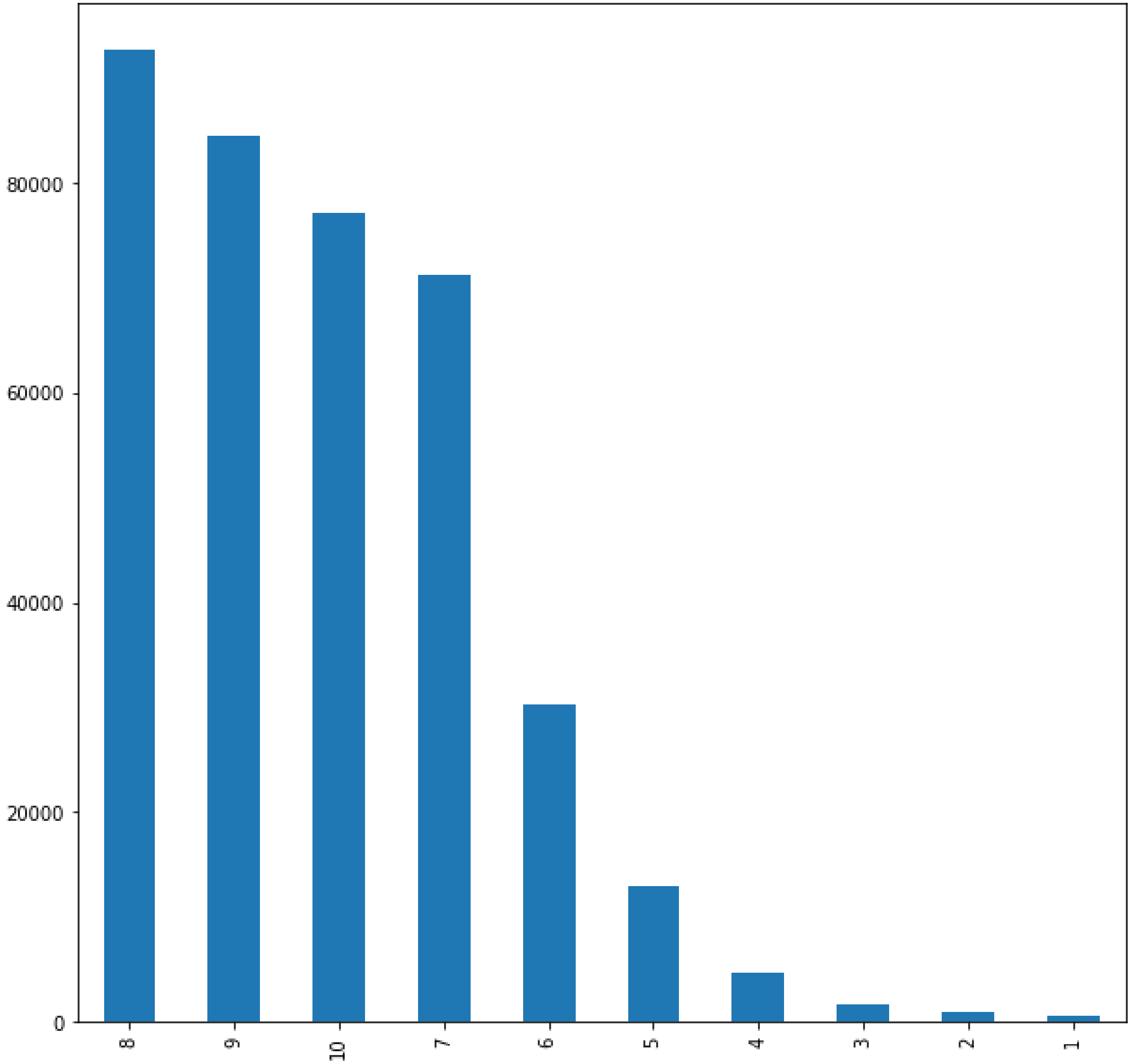
In [12]:
```
rating_x.rating.value_counts().plot(kind='bar',figsize=(10,10))
plt.show()
len(rating)
```



Out[12]: 7813737

Review amount - train data

In [7]:
```
train,test = train_test_split(rating_x,shuffle = True, test_size = 0.1)
train.rating.value_counts().plot(kind='bar',figsize=(10,10))
plt.show()
```



Model

Cross validating KNN and SVD algorithms and RMSE & MSE score

In [9]:
```
from surprise.model_selection.search import GridSearchCV
from surprise.model_selection import cross_validate
from surprise.prediction_algorithms import knns
from surprise import SVDpp, SlopeOne, NMF, CoClustering, NormalPredicto
r, BaselineOnly
reader = Reader(rating_scale=(1, 10))
recom_ratings=Dataset.load_from_df(train[['user_id','anime_id','rating'
]],reader)

crossval_results = []
sim_options = {'name': 'pearson_baseline', 'user_based': False}
# Iterate over all algorithms
for algorithm in [ knns.KNNBaseline(sim_options=sim_options), knns.KNNB
asic(sim_options=sim_options), knns.KNNWithMeans(sim_options=sim_option
s), knns.KNNWithZScore(sim_options=sim_options),SVD()]:

    # Perform cross validation
    results = cross_validate(algorithm, recom_ratings, measures=['RMSE'
,'mae'], cv=3, verbose=False)

    # Get results & append algorithm name
    tmp_results = pd.DataFrame.from_dict(results).mean(axis=0)
    tmp_results = tmp_results.append(pd.Series([str(algorithm).split('
 ')[0].split('.')[-1]], index=['Algorithm']))
    crossval_results.append(tmp_results)

pd.DataFrame(crossval_results).set_index('Algorithm').sort_values('test
_rmse')
```

```
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
```

```
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...

Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
```

Out[9]:

Algorithm	test_rmse	test_mae	fit_time	test_time
SVD	1.244319	0.947927	13.743789	0.998259
KNNBaseline	1.245537	0.905085	1.627665	3.732582
KNNWithMeans	1.254503	0.912217	1.720928	3.034991
KNNWithZScore	1.261182	0.915743	1.855850	3.235263

KNNBasic	1.335605	0.961420	1.633246	2.666646

In [174]:
```
#Save model scores to xlxs
#pd.DataFrame(crossval_results).set_index('Algorithm').sort_values('tes
t_rmse').to_excel('crossvalidation_results.xlsx')
```

In [10]:
```
from surprise import accuracy
recom_ratings=Dataset.load_from_df(train[['user_id','anime_id','rating'
]],reader)
trainset = recom_ratings.build_full_trainset()
#Make model
algo = knns.KNNBaseline(sim_options=sim_options)
predictions = algo.fit(trainset)
#accuracy.rmse(predictions)
```

```
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
```

In [71]:
```
predictions.predict(uid= 1012, iid= 5)
```

Out[71]:
```
Prediction(uid=1012, iid=5, r_ui=None, est=9.862862032385529, details=
{'actual_k': 5, 'was_impossible': False})
```

Get recommendations for choosen movie

In [11]:
```
#get inner id of choosen movie
movieID=5

#get inner id of choosen movie from the model
recom_inner_id = algo.trainset.to_inner_iid(movieID)

#get top 10 recommendations for the choosen movie
recommendation = algo.get_neighbors(recom_inner_id, k=10)

#get real ID's of recommended movies from original data
```

```
recommendation_neighbors = (algo.trainset.to_raw_iid(inner_id)for inner
_id in recommendation)

#Print info of choosen movie
FilterByColumnValue(anime_x,'anime_id',[movieID])
```

Out[11]:

	anime_id	name	genre	type	episodes	rating	members
152	5	Cowboy Bebop: Tengoku no Tobira	Action, Drama, Mystery, Sci-Fi, Space	Movie	1	8.4	137636

In [12]:

```
#Top 10 recommendations for person who watched choosen movie
FilterByColumnValue(anime_x,'anime_id',recommendation_neighbors)
```

Out[12]:

	anime_id	name	genre	type	episodes	rating	members
307	1430	Lupin III: Cagliostro no Shiro	Adventure, Comedy, Shounen	Movie	1	8.20	32732
358	47	Akira	Action, Adventure, Horror, Military, Sci-Fi, S...	Movie	1	8.15	215897
441	4106	Trigun: Badlands Rumble	Action, Comedy, Sci-Fi	Movie	1	8.08	68181
484	793	xxxHOLiC Movie: Manatsu no Yoru no Yume	Comedy, Drama, Mystery, Psychological, Superna...	Movie	1	8.04	41547
624	468	Ghost in the Shell 2: Innocence	Mecha, Military, Police, Psychological, Sci-Fi	Movie	1	7.93	85714
728	2175	Kino no Tabi: The Beautiful World - Byouki no ...	Adventure, Drama, Fantasy	Movie	1	7.87	23453
783	1462	Memories	Drama, Horror, Psychological, Sci-Fi	Movie	3	7.84	38643
1208	8100	Mardock Scramble: The First Compression	Action, Psychological, Sci-Fi	Movie	1	7.63	40698

	anime_id	name	genre	type	episodes	rating	members
1959	713	Air Movie	Drama, Romance, Supernatural	Movie	1	7.39	44179
2655	2490	One Piece: Mezase! Kaizoku Yakyuu Ou	Comedy, Fantasy, Shounen, Sports	Movie	1	7.20	20054

Conclusion

1. What kind of preprocessing is necessary for the ratings dataset?

  - Filtering data because there is too much to process
  - In our case we decided to build recommendation engine for anime movies.
  - Filtered out all users that had made over 100 reviews because we wanted input from diffrent kinds of people and decided buil recommendation system for common people.
  - Need run data through reader to be able to parse dataset
  - Remove unrated information.

2. How do the recommendation algorithms (e.g. KNN and SVD) perform with a data set of this magnitude? Do you encounter hardware limitations? If yes, how can you circumvent some of the limitations to be able to carry on with the experiment?

  - SVD and KNN algorithms faced problem when we tried run the whole data scikit learn crashed due to the memory errors.
    - That is why we proceed filtering steps explained in previous section
  - SVD was much faster than KNN (how much)
    - It wasnt not far behind KNNBaseline in metrics but because it is 3-4 times faster than KNN it might be good option for huge datasets
  - Therer were many different KNN algorithms to choose from, but every one of them were much slower that SVD, but KNNBaseline had smallest RMSE so it was most valid choice for accurate model
  - When we set the parameter "user_base = True" the KNN algorithms were faster than SVD but SVD was actually able to process large dataset. Never the less, the computation time was few times longer than in case of "user_base = False"

3. Can you combine the information in the two files in a meaningful way to have the recommender display the titles of the recommended movies?

    - Files can be combined easly through the anime_id parameter but it needed couple of tricks to get real id's for movies from model that can see from "get recommendations..." part which is above this cell.

References

https://github.com/NicolasHug/Surprise/blob/master/examples/k_nearest_neighbors.py

In [ ]: