Modelling

In [50]:
```
import pandas as pd
import numpy as np
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
from scipy.sparse import csr_matrix
from sklearn.metrics.pairwise import pairwise_distances
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.neighbors import NearestNeighbors
import ipywidgets as widgets
from IPython.display import display
from surprise import Reader, Dataset
from surprise.model_selection import train_test_split
from surprise.model_selection import cross_validate
from surprise.prediction_algorithms import SVD
from surprise.prediction_algorithms import KNNWithMeans, KNNBasic, KNNBaseline
from surprise.model_selection import GridSearchCV
from surprise.prediction_algorithms import knns
from surprise.similarities import cosine, msd, pearson
from surprise import accuracy
from surprise.model_selection.validation import cross_validate
import warnings
warnings.filterwarnings('ignore')


import matplotlib.pyplot as plt
import seaborn as sns
import matplotlib as mpl
import matplotlib.pylab as pylab
%matplotlib inline
pd.set_option('display.max_columns', 500)
mpl.style.use('ggplot')
sns.set_style('white')
pylab.rcParams['figure.figsize'] = 12,8
import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
df_anime = pd.read_csv("cleaned_anime.csv")
df_anime.drop("Unnamed: 0", axis = 1, inplace = True)
df_anime.head()
```

Out[2]:

	Anime_id	Title	Genre	Synopsis	Type	Producer	Studio	Rating	Scor
0	1	Cowboy Bebop	['Action', 'Adventure', 'Comedy', 'Drama', 'Sc...	In the year 2071, humanity has colonized sever...	TV	['Bandai Visual']	['Sunrise']	8.81	3638
1	5	Cowboy Bebop Tengoku no Tobira	['Action', 'Space', 'Drama', 'Mystery', 'Sci-Fi']	Another day, another bounty—such is the life o...	Movie	['Sunrise', 'Bandai Visual']	['Bones']	8.41	111
2	6	Trigun	['Action', 'Sci-Fi', 'Adventure', 'Comedy', 'D...	Vash the Stampede is the man with a $$60,000,0...	TV	['Victor Entertainment']	['Madhouse']	8.31	1974
3	7	Witch Hunter Robin	['Action', 'Magic', 'Police', 'Supernatural', ...	Witches are individuals with special powers li...	TV	['Bandai Visual']	['Sunrise']	7.34	318
4	8	Bouken Ou Beet	['Adventure', 'Fantasy', 'Shounen', 'Supernatu...	It is the dark century and the people are suff...	TV	Unknown	['Toei Animation']	7.04	47

In [3]:
```
!! pip install scikit-learn
```

Out[3]:
```
['Requirement already satisfied: scikit-learn in c:\\users\\musoo\\anaconda3\\e
nvs\\learn-env\\lib\\site-packages (0.23.2)',
 'Requirement already satisfied: joblib>=0.11 in c:\\users\\musoo\\anaconda3\\e
nvs\\learn-env\\lib\\site-packages (from scikit-learn) (0.17.0)',
 'Requirement already satisfied: scipy>=0.19.1 in c:\\users\\musoo\\anaconda3
\\envs\\learn-env\\lib\\site-packages (from scikit-learn) (1.5.0)',
 'Requirement already satisfied: threadpoolctl>=2.0.0 in c:\\users\\musoo\\anac
onda3\\envs\\learn-env\\lib\\site-packages (from scikit-learn) (2.1.0)',
 'Requirement already satisfied: numpy>=1.13.3 in c:\\users\\musoo\\anaconda3
\\envs\\learn-env\\lib\\site-packages (from scikit-learn) (1.18.5)']
```

In [4]:
```
df_anime['Synopsis'].isnull().sum()
```

Out[4]:
```
0
```

Metric Based Modelling

CONTENT BASED MODEL (First Model)

We will first build a content based model by using the `Synopsis` and `Title` column

In [5]:
```
df_anime['Synopsis'] = df_anime['Synopsis'].fillna('')
```

We will use the Term Frequency-Inverse Document Frequency on our model since it is content based.

In [6]:
```
# creating a sparse matrix for the synopsis column
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf = TfidfVectorizer(stop_words='english')

tfidf_matrix = tfidf.fit_transform(df_anime['Synopsis'])

tfidf_matrix.shape
```

Out[6]:
```
(12979, 40034)
```

In [7]:
```
# finding similar items
from sklearn.metrics.pairwise import linear_kernel

cosine_sim = linear_kernel(tfidf_matrix, tfidf_matrix)
cosine_sim
```

Out[7]:
```
array([[1.        , 0.23529586, 0.02127464, ..., 0.02133734, 0.00457067,
        0.02050118],
       [0.23529586, 1.        , 0.04320112, ..., 0.01628467, 0.01529336,
        0.        ],
       [0.02127464, 0.04320112, 1.        , ..., 0.00264197, 0.        ,
        0.        ],
       ...,
       [0.02133734, 0.01628467, 0.00264197, ..., 1.        , 0.        ,
        0.        ],
       [0.00457067, 0.01529336, 0.        , ..., 0.        , 1.        ,
        0.        ],
       [0.02050118, 0.        , 0.        , ..., 0.        , 0.        ,
        1.        ]])
```

In [8]:
```
#removing duplicate titles
indices = pd.Series(df_anime.index, index=df_anime['Title']).drop_duplicates()
indices
```

Out[8]:
```
Title
Cowboy Bebop                          0
Cowboy Bebop Tengoku no Tobira        1
Trigun                                2
Witch Hunter Robin                    3
Bouken Ou Beet                        4
                                   ...
Animagear                         12974
Magical Halloween MiracleQuartet  12975
Orbital Era                       12976
Akai Hana Shiroi Hana             12977
Arui Tekoteko                     12978
Length: 12979, dtype: int64
```

In [9]:
```
# creating a function to give the recommendations
def get_recommendations(title, cosine_sim=cosine_sim):

    idx = indices[title]

    sim_scores = list(enumerate(cosine_sim[idx]))

    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)

    # Get the scores of the 15 most similar movies
    sim_scores = sim_scores[1:16]

    anime_indices = [i[0] for i in sim_scores]

    return df_anime['Title'].iloc[anime_indices]
```

In [10]:
```
# testing our function
get_recommendations('Sen to Chihiro no Kamikakushi')
```

Out[10]:
```
9604                                      Shounen Maid
560                                 Kujibiki Unbalance
4521                               Suki Desu Suzukikun
5520                 Kami nomi zo Shiru Sekai 4nin to Idol
3283                       KerakuNoOH King of Pleasure
10156                              Kushimitama Samurai
5869                                      Sankarea OVA
2422                              ef A Tale of Memories
2816                                         Kiss yori
4199               ef A Tale of Memories  Recollections
11903                                   Mirai no Mirai
740                                         Jyu Oh Sei
2669                                    Hatenkou Yuugi
4822      Ultraman Kids Haha wo Tazunete 3000man Kounen
5113                                 Koguma no Korochan
Name: Title, dtype: object
```

In [11]:
```python
get_recommendations('Koe no Katachi')
```

Out[11]:
```
12407    Seishun Buta Yarou wa Yumemiru Shoujo no Yume ...
1254                                      Asatte no Houkou
5605                                          Hero Herokun
9159                                     Yuujou no Kickoff
1731                                                  Pops
6486                                          Danchi Tomoo
4730                          Fortune Arterial Akai Yakusoku
12702                                   Tomo Tabidachi no Toki
6611                           Kakumeiki Valvrave 2nd Season
6473                                        Harisu no Kaze
8974                           Gakuen Handsome The Animation
10173                                      Gakuen Handsome
10787                                           Tejina Shi
5429                                    VitaminX Addiction
5830                               Yajikita Gakuen Douchuuki
Name: Title, dtype: object
```

CONTENT BASED MODEL(Second Model)

Genres, Producer and Studio Based Recommender It goes without saying that the quality of our recommender would be increased with the usage of better metadata. That is exactly what we are going to do in this section. We are going to build a recommender based on the following metadata: the producer, related genres and the studio to see if we will get better recommendations

In [12]:
```python
#previewing data again
df_anime.head(2)
```

Out[12]:

	Anime_id	Title	Genre	Synopsis	Type	Producer	Studio	Rating	ScoredBy	Populi
0	1	Cowboy Bebop	['Action', 'Adventure', 'Comedy', 'Drama', 'Sc...	In the year 2071, humanity has colonized sever...	TV	['Bandai Visual']	['Sunrise']	8.81	363889.0	
1	5	Cowboy Bebop Tengoku no Tobira	['Action', 'Space', 'Drama', 'Mystery', 'Sci-Fi']	Another day, another bounty— such is the life o...	Movie	['Sunrise', 'Bandai Visual']	['Bones']	8.41	111187.0	4

In [13]:
```
# extracting the columns that we need
features = ['Genre','Producer', 'Studio']

print(df_anime[features].isnull().sum())
```

```
Genre       0
Producer    0
Studio      0
dtype: int64
```

In [14]:
```
df_anime[features] = df_anime[features].fillna('[' ']')
```

In [15]:
```
df_anime.Type.unique()
```

Out[15]:
```
array(['TV', 'Movie', 'OVA', 'Special', 'ONA', 'Music', 'Unknown'],
      dtype=object)
```

In [16]:
```
def clean_data(x):
    if isinstance(x, list):
        return [str.lower(i.replace(" ","")) for i in x]

    else:
        if isinstance(x, str):
            return str.lower(x.replace(" ",""))
        else:
            return ""
```

In [17]:
```
features = ['Genre','Producer', 'Studio', 'Type']

for feature in features:
    df_anime[feature] = df_anime[feature].apply(clean_data)
```

In [18]:
```
# previewing data
df_anime.head(2)
```

Out[18]:

	Anime_id	Title	Genre	Synopsis	Type	Producer
0	1	Cowboy Bebop	['action','adventure','comedy','drama','sci-fi...	In the year 2071, humanity has colonized sever...	tv	['bandaivisual']
1	5	Cowboy Bebop Tengoku no Tobira	['action','space','drama','mystery','sci-fi']	Another day, another bounty—such is the life o...	movie	['sunrise','bandaivisual']

In [19]:
```
# preview of features
features
```

Out[19]:
```
['Genre', 'Producer', 'Studio', 'Type']
```

In [20]:
```
# function to join the two dataframes
def create_soup(x):
    return " ".join(x['Genre']) + " " + x['Type'] + " " + " ".join(x['Producer'])
```

In [21]:
```
df_anime['soup'] = df_anime.apply(create_soup, axis=1)
```

In [22]:
```
df_anime['soup']
```

Out[22]:
```
0        [ ' a c t i o n ' , ' a d v e n t u r e ' , ' ...
1        [ ' a c t i o n ' , ' s p a c e ' , ' d r a m ...
2        [ ' a c t i o n ' , ' s c i - f i ' , ' a d v ...
3        [ ' a c t i o n ' , ' m a g i c ' , ' p o l i ...
4        [ ' a d v e n t u r e ' , ' f a n t a s y ' , ...
                              ...
12974    [ ' k i d s ' , ' m e c h a ' ] ona u n k n o ...
12975    [ ' c o m e d y ' , ' e c c h i ' , ' f a n t ...
12976    [ ' a c t i o n ' , ' a d v e n t u r e ' , ' ...
12977    [ ' k i d s ' , ' m u s i c ' ] music u n k n ...
12978    [ ' k i d s ' , ' m u s i c ' ] music u n k n ...
Name: soup, Length: 12979, dtype: object
```

The next steps are the same as what we did with our plot description based recommender. One important difference is that we use the CountVectorizer() instead of TF-IDF. This is because we do not want to down-weight the presence of a producer if he or she has acted or directed in relatively more movies. It doesn't make much intuitive sense.

In [23]:
```
# using a count vectorizer
count = CountVectorizer(stop_words='english')
count_matrix = count.fit_transform(df_anime['soup'])

# calculating cosine similarity

cosine_sim2 = cosine_similarity(count_matrix, count_matrix)
```

In [24]:
```
df_anime = df_anime.reset_index()
indices = pd.Series(df_anime.index, index=df_anime['Title'])
```

In [25]:
```
indices
```

Out[25]:
```
Title
Cowboy Bebop                            0
Cowboy Bebop Tengoku no Tobira          1
Trigun                                  2
Witch Hunter Robin                      3
Bouken Ou Beet                          4
                                     ...
Animagear                           12974
Magical Halloween MiracleQuartet    12975
Orbital Era                         12976
Akai Hana Shiroi Hana               12977
Arui Tekoteko                       12978
Length: 12979, dtype: int64
```

Now that we have the cosine similarities, we can use the fuction we created earlier to get recommendations.

In [26]:
```
# testing the function
get_recommendations('Cowboy Bebop', cosine_sim2)
```

Out[26]:
```
1                   Cowboy Bebop Tengoku no Tobira
3226                          Ginga Senpuu Braiger
11619                                       Hiromi
4563                                           KOn
5950                                Little Busters
1167                   Sol Bianca Taiyou no Fune
2533     Saraba Uchuu Senkan Yamato Ai no Senshitachi
2411               Code Geass Hangyaku no Lelouch R2
1338                           Uchuu Senkan Yamato
3841                         Uchuu Kuubo Blue Noah
11611                              Chibikko Cowboy
9948                                   Xiao Yeyou
6238                           Suisei no Gargantia
9016                             Dragon Ball Super
4964                    Rokushin Gattai GodMars 1982
Name: Title, dtype: object
```

In [27]:
```
get_recommendations('Sen to Chihiro no Kamikakushi', cosine_sim2)
```

Out[27]:
```
9604                                      Shounen Maid
740                                         Jyu Oh Sei
10156                               Kushimitama Samurai
2669                                     Hatenkou Yuugi
11903                                    Mirai no Mirai
6115                     Majokko Shimai no Yoyo to Nene
560                                  Kujibiki Unbalance
5113                                 Koguma no Korochan
6438             Ashita Genki ni Nare Hanbun no Satsumaimo
7374      Kero Kero Keroppi no Yowamushiouji no Daibouken
92                                  Chou Henshin Cosprayers
3124                                    Net Ghost Pipopa
5911                             Kyoto Animation Hoshihen
999                  Fushigi no Umi no Nadia Original Movie
10684          Death March kara Hajimaru Isekai Kyousoukyoku
Name: Title, dtype: object
```

In [28]:
```
get_recommendations('Mirai no Mirai', cosine_sim2)
```

Out[28]:
```
3478                               Umineko no Naku Koro ni
3809                                    Suteneko Torachan
9493                      Tenchi Muyou Ryououki 4th Season
8395                                           Hand Soap
4233                               Doubutsu Mura no Daisodou
2269                                        Umi no Triton
8323      Kyoukai no Kanata Movie 2 Ill Be Here  Miraihen
6529                                         Golden Time
10858                              Haruniwa Ie no 3 Ninme
5123                                               Oshin
3584                                     Minamike Okaeri
6035                               Kono Sekai no Katasumi ni
11288                                         Yao Shen Ji
298                                Mama wa Shougaku 4 Nensei
2993                                     Chis Sweet Home
Name: Title, dtype: object
```

In [29]:
```
display(df_anime[['Title', 'Rating', 'Producer', 'Studio']].loc[df_anime['Title']
```

	Title	Rating	Producer	Studio
178	Sen to Chihiro no Kamikakushi	8.92	['toho']	['studioghibli']

Our earlier model gives better recommendations in the context where we use `Cowboy Bebop` title as an example. The first function gives more recommendations of titles about `Cowboy Bebop` than the second model.

COLLABORATIVE FILTERING

Preprocessing for Collaborative Filtering

Preprocessing for Collaborative Filtering

In [30]:
```
# renaming the anime_id col
df_anime = df_anime.rename(columns={'MAL_ID':'anime_id'})
```

In [31]:
```
#looking at the shape
df_anime.shape
```

Out[31]:
```
(12979, 16)
```

In [32]:
```
#loading the cleaned ratings data
rating = pd.read_csv('cleaned_rating.csv')
rating.drop("Unnamed: 0", axis = 1, inplace = True)
rating.head()
```

Out[32]:

	user_id	anime_id	rating
0	1	20	-1
1	1	24	-1
2	1	79	-1
3	1	226	-1
4	1	241	-1

In [33]:
```
# confirming there are no duplicates
duplicates=rating.duplicated()
if duplicates.sum()>0:
    print('>{} duplicates'.format(duplicates.sum()))
    rating=rating[~duplicates]

print('>{} duplicates'.format(rating.duplicated().sum()))
```

```
>0 duplicates
```

In [34]:
```
# checking the entries
rating.shape
```

Out[34]:
```
(7813736, 3)
```

In [35]:
```
# finding unique users and items
unique_users = {int(x): i for i,x in enumerate(rating.user_id.unique())}
unique_items = {int(x): i for i,x in enumerate(rating.anime_id.unique())}

print(" No. of unique anime:",len(unique_items),'\n',"No. of Unique users ", len(
```

```
 No. of unique anime: 11200
 No. of Unique users  73515
```

In [36]:
```
# more wrangling
n_rating=rating['user_id'].value_counts()
rating=rating[rating['user_id'].isin(n_rating[n_rating>=500].index)].copy()
```

In [37]:
```
rating.shape
```

Out[37]:
```
(1384631, 3)
```

In [38]:
```
# preview of the ratings
rating.head()
```

Out[38]:

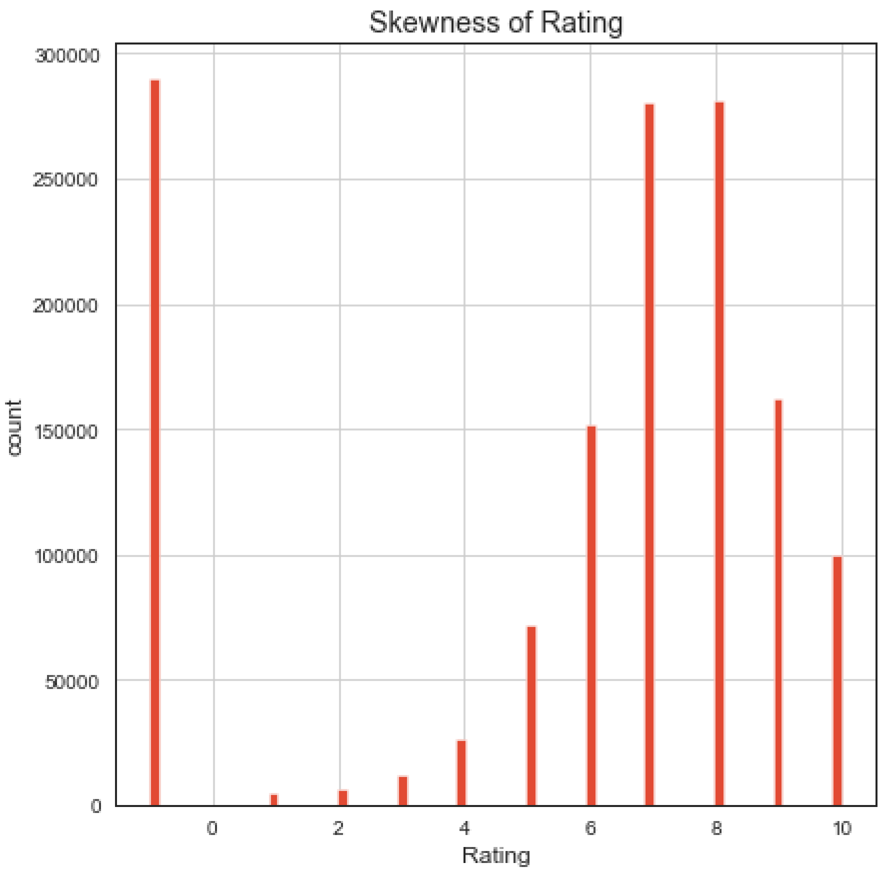
	user_id	anime_id	rating
1614	17	6	7
1615	17	19	10
1616	17	30	9
1617	17	32	10
1618	17	43	8

Distribution of ratings

- We will now investigate the distribution of ratings on both the datasets

In [39]:
```
    # ploting the skewness

    plt.figure(figsize = (15, 7))
    plt.subplot(1,2,1)
    rating["rating"].hist(bins=70)
    plt.xlabel("Rating")
    plt.ylabel("count")
    plt.title("Skewness of Rating")
    plt.show()
```



In [40]:
```
# picking numeric columns
new_data = df_anime[['Anime_id','Title', 'ScoredBy',"Members", 'Rating']]
```

In [41]:
```
# renaming some columns
rating=rating.rename(columns={'anime_id':'Anime_id'})
```

In [42]:
```
# merging the the two dataframes for modelling
new_data=pd.merge(new_data,rating,on='Anime_id')
```

In [43]:
```
new_data=new_data.sort_values(by='ScoredBy', ascending=False).query('Members>500'
```

In [44]:
```
new_data=new_data.sort_values(by='ScoredBy', ascending=False).query('Members>500'
```

In [45]:
```
new_data.shape
```

Out[45]: (1206012, 7)

In [46]:
```
total=new_data.isnull().sum().sort_values(ascending=False)
percent=(new_data.isnull().sum()/new_data.isnull().count()).sort_values(ascending
print(percent)
```

```
Title       0.000002
rating      0.000000
user_id     0.000000
Rating      0.000000
Members     0.000000
ScoredBy    0.000000
Anime_id    0.000000
dtype: float64
```

Pivot Table

We will now create a pivot table so as to make the modelling work much easier

In [47]:
```
anime_pivot=new_data.pivot_table(index='Title',columns='user_id',values='rating')
```

In [48]:
```
anime_pivot.head()
```

Out[48]:

user_id	17	54	201	226	271	294	342	392	446	478	661	741	771	786	804	917	940
Title																	
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0008	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
001	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
009 ReCyborg	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0091	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

5 rows × 1853 columns

Now that our data is ready for modelling, we will use the item based collaborative filtering model to get recommendations.

Item Based

In [51]:
```
# generating a sparse matrix from our pivot table
anime_matrix = csr_matrix(anime_pivot.values)

# initializing a base model
model_knn = NearestNeighbors(metric = 'cosine', algorithm = 'brute')
model_knn.fit(anime_matrix)
```

Out[51]:
```
NearestNeighbors(algorithm='brute', metric='cosine')
```

In [52]:
```
query_index = np.random.choice(anime_pivot.shape[0])
print(query_index)
distances, indices = model_knn.kneighbors(anime_pivot.iloc[query_index,:].values.
```

```
3177
```

In [53]:
```
# creating a recommendation function
def get_rec(name,it):
    distances, indices = model_knn.kneighbors(anime_pivot.loc[name].values.reshap
    d1=pd.DataFrame(columns=df_anime.columns)
    for i in range(0, len(distances.flatten())):
        if i == 0:
            print('Recommendations for',name)
        else:
            d2=df_anime.loc[df_anime["Title"]==anime_pivot.index[indices.flatten(
            d1=d1.append(d2)
    return d1[["Title","ScoredBy"]].sort_values(by="ScoredBy",ascending=False)[:i
```

In [54]:
```
# testing our function
get_rec('Shingeki no Kyojin',15)
```

```
Recommendations for Shingeki no Kyojin
```

Out[54]:

	Title	ScoredBy
1239	Death Note	1006242.0
5682	Sword Art Online	913806.0
3567	Fullmetal Alchemist Brotherhood	730784.0
8859	One Punch Man	687965.0
7300	Tokyo Ghoul	656039.0
10	Naruto	645672.0
4121	Angel Beats	640177.0
1274	Code Geass Hangyaku no Lelouch	625466.0
6891	No Game No Life	620456.0
5435	Mirai Nikki	591121.0
4921	SteinsGate	561405.0
3224	Toradora	556177.0
2411	Code Geass Hangyaku no Lelouch R2	541989.0
7252	Sword Art Online II	528708.0
5152	Ao no Exorcist	520377.0

In [55]:
```
get_rec('Fullmetal Alchemist Brotherhood',15)
```

```
Recommendations for Fullmetal Alchemist Brotherhood
```

Out[55]:

	Title	ScoredBy
1239	Death Note	1006242.0
6232	Shingeki no Kyojin	936784.0
5682	Sword Art Online	913806.0
8859	One Punch Man	687965.0
7300	Tokyo Ghoul	656039.0
10	Naruto	645672.0
4121	Angel Beats	640177.0
1274	Code Geass Hangyaku no Lelouch	625466.0
6891	No Game No Life	620456.0
5435	Mirai Nikki	591121.0
4921	SteinsGate	561405.0
3224	Toradora	556177.0
2411	Code Geass Hangyaku no Lelouch R2	541989.0
7252	Sword Art Online II	528708.0
5152	Ao no Exorcist	520377.0

In [56]:
```
get_rec('Doraemon Movie 07 Nobita to Tetsujin Heidan',10)
```

```
Recommendations for Doraemon Movie 07 Nobita to Tetsujin Heidan
```

Out[56]:

	Title	ScoredBy
9863	Dungeon ni Deai wo Motomeru no wa Machigatteir...	29421.0
2005	Digimon Adventure Bokura no War Game	29173.0
667	Digimon Savers	28566.0
1356	Bakuten Shoot Beyblade 2002	22666.0
2006	Digimon Adventure 02 Diablomon no Gyakushuu	21737.0
898	Medarot	21122.0
2447	Digimon Adventure 02 Movies	18619.0
1355	Bakuten Shoot Beyblade G Revolution	17211.0
9306	To LOVERu Darkness 2nd OVA	16698.0
1993	One Piece Jango no Dance Carnival	14493.0

In [57]:
```
get_rec('Naruto',10)
```

```
Recommendations for Naruto
```

Out[57]:

	Title	ScoredBy
1239	Death Note	1006242.0
6232	Shingeki no Kyojin	936784.0
5682	Sword Art Online	913806.0
3567	Fullmetal Alchemist Brotherhood	730784.0
8859	One Punch Man	687965.0
7300	Tokyo Ghoul	656039.0
4121	Angel Beats	640177.0
1274	Code Geass Hangyaku no Lelouch	625466.0
6891	No Game No Life	620456.0
5435	Mirai Nikki	591121.0

In [58]:
```
# saving our function
import pickle
filename='KNN_model_forCF.sav'
pickle.dump(model_knn,open(filename,'wb'))
```

HYBRID RECOMMENDATION

We will now build a hybrid recommendation system to get recommendations

In [59]:
```
data = pd.read_csv('cleaned_anime.csv')
data.drop('Unnamed: 0', axis = 1, inplace = True)
data.dropna(subset = ['Title'], inplace = True)
data.head()
```

Out[59]:

	Anime_id	Title	Genre	Synopsis	Type	Producer	Studio	Rating	Scor
0	1	Cowboy Bebop	['Action', 'Adventure', 'Comedy', 'Drama', 'Sc...	In the year 2071, humanity has colonized sever...	TV	['Bandai Visual']	['Sunrise']	8.81	3638
1	5	Cowboy Bebop Tengoku no Tobira	['Action', 'Space', 'Drama', 'Mystery', 'Sci-Fi']	Another day, another bounty—such is the life o...	Movie	['Sunrise', 'Bandai Visual']	['Bones']	8.41	111
2	6	Trigun	['Action', 'Sci-Fi', 'Adventure', 'Comedy', 'D...	Vash the Stampede is the man with a $$60,000,0...	TV	['Victor Entertainment']	['Madhouse']	8.31	197
3	7	Witch Hunter Robin	['Action', 'Magic', 'Police', 'Supernatural', ...	Witches are individuals with special powers li...	TV	['Bandai Visual']	['Sunrise']	7.34	318
4	8	Bouken Ou Beet	['Adventure', 'Fantasy', 'Shounen', 'Supernatu...	It is the dark century and the people are suff...	TV	Unknown	['Toei Animation']	7.04	47

We will then use a Term frequency - inverse document frequency to create a search engine so that it is easy to find an anime title and its anime id. The tfdif will enable the computer to find the title that is most similar to the title that we will enter.

In [60]:
```
# vector matrix for the anime titles

# finding groups of 2 words that are consecutive
vectorizer = TfidfVectorizer(ngram_range=(1,2))

tfidf = vectorizer.fit_transform(data["Title"])
```

We will then create a function that will use cosine similarity to compute the similarity between a term that we will enter in our search box and all the anime titles in our dataset.

In [61]:

```
# creating a search function
def search(title):
    """
    A function that takes in a search term of the title that we are
    looking for then cleans it and uses the vectorizer
    to turn it into a sparse matrix and then uses cosine similarity to find the
    most similar titles.
    """
    # cleaning the title entered
    title = title
    query_vec = vectorizer.transform([title])
    similarity = cosine_similarity(query_vec, tfidf).flatten()

    # find the 5 most similar titles to the searcch term
    indices = np.argpartition(similarity, -5)[-5:]

    # reverse results so that most similar result is at the top
    results = data.iloc[indices].iloc[::-1]

    return results
```

We will use the widgets library to build an interactive search box using our function above. The search box will enable one to type in the name of an anime and see the results.

In [62]:

```
# using the widgets library
movie_input = widgets.Text(
    # setting default value as Trigun
    value='Trigun',
    description='Movie Title:',
    disabled=False
)
movie_list = widgets.Output()

def on_type(data):
    with movie_list:
        movie_list.clear_output()
        title = data["new"]

        # display output when length of title is more than 5
        if len(title) > 5:
            display(search(title))

movie_input.observe(on_type, names='value')


display(movie_input, movie_list)
```

```
Text(value='Trigun', description='Movie Title:')

Output()
```

We will hardcode an anime id for the 'Cowoy Bebop' anime to find the users who watched and also liked same anime and gave it a rating greater than 6.

In [63]:
```
# hard coding an anime id to find the users who liked the same movie
movie_id = 1

#def find_similar_movies(movie_id):
anime = data[data["Anime_id"] == movie_id]
```

We will now use the ratings dataset to find the users who liked the same movie.

In [64]:
```
# loadind the clean rating data
ratings = pd.read_csv('cleaned_rating.csv')
ratings.drop("Unnamed: 0", axis = 1, inplace = True)
ratings.head()
```

Out[64]:

	user_id	anime_id	rating
0	1	20	-1
1	1	24	-1
2	1	79	-1
3	1	226	-1
4	1	241	-1

In [65]:
```
# finding similar users
similar_users = ratings[(ratings["anime_id"] == movie_id) & (ratings["rating"] >
similar_users
```

Out[65]:
```
array([   19,    21,    23, ..., 73507, 73513, 73515], dtype=int64)
```

We will then put it into a dataframe to return the records of the similar users.

In [66]:
```
# similar users dataframe
similar_user_recs = ratings[(ratings["user_id"].isin(similar_users)) & (ratings['
similar_user_recs
```

Out[66]:

	user_id	anime_id	rating
2240	19	1	10
2241	19	5	9
2242	19	121	8
2243	19	199	9
2244	19	578	10
...	...	...	...
7813729	73515	13659	8
7813730	73515	14345	7
7813731	73515	16512	7
7813732	73515	17187	9
7813733	73515	22145	10

1809322 rows × 3 columns

In [67]:
```
# getting the anime ids from the dataframe
similar_user_recs = ratings[(ratings["user_id"].isin(similar_users)) & (ratings['
similar_user_recs
```

Out[67]:
```
2240          1
2241          5
2242        121
2243        199
2244        578
           ...
7813729    13659
7813730    14345
7813731    16512
7813732    17187
7813733    22145
Name: anime_id, Length: 1809322, dtype: int64
```

We will then narrow down the dataframe so that we only get 10% or more of users who are similar and also liked the same movie

In [68]:
```
# get value counts for the movie ids
similar_user_recs.value_counts()
```

Out[68]:
```
1        12940
1535      8315
1575      7235
121       7011
2001      6785
         ...
30840        1
25173        1
11869        1
7771         1
30178        1
Name: anime_id, Length: 8239, dtype: int64
```

In [69]:
```
# Converting to percentage
similar_user_recs = similar_user_recs.value_counts() / len(similar_users)

# Taking records where only greater than 10% liked
similar_user_recs = similar_user_recs[similar_user_recs > .10]
similar_user_recs
```

Out[69]:
```
1        1.000000
1535     0.642581
1575     0.559119
121      0.541808
2001     0.524343
           ...
3002     0.101391
486      0.101159
239      0.100773
357      0.100696
15451    0.100077
Name: anime_id, Length: 326, dtype: float64
```

We now have a set of 326 animes where more than 10% of users liked the anime title. We will now find users who liked animes that are similar to the 'Cowboy' anime. This will be done by finding the users who have rated the animes in our 'similar_user_recs'.

In [70]:
```
# finding how much all users in the dataset like the animes
all_users = ratings[(ratings["anime_id"].isin(similar_user_recs.index)) & (rating
all_users
```

Out[70]:

	user_id	anime_id	rating
47	1	8074	10
81	1	11617	10
83	1	11757	10
101	1	15451	10
153	2	11771	10
...	...	...	...
7813722	73515	12413	9
7813729	73515	13659	8
7813730	73515	14345	7
7813734	73516	790	9
7813735	73516	8074	9

2337672 rows × 3 columns

We will then find the percentage of all users who recommended the animes that are in `similar_user_recs` .

In [71]:
```
# calculating percentage
all_user_recs = all_users["anime_id"].value_counts() / len(all_users["user_id"].u
all_user_recs
```

Out[71]:
```
1535     0.481432
16498    0.353054
1575     0.343044
11757    0.329717
6547     0.322566
           ...
166      0.034704
486      0.030753
239      0.030679
617      0.030473
329      0.028866
Name: anime_id, Length: 326, dtype: float64
```

We will then compare the percentages in `similar_user_recs` and `all_users_recs` by joining the two pandas series so as to see how users similar to us liked the anime and how much an average person liked the same animes.

The goal is to find the animes that have a big difference.

In [72]:
```
# joining the two pandas series
rec_percentages = pd.concat([similar_user_recs, all_user_recs], axis=1)
rec_percentages.columns = ["similar", "all"]
rec_percentages
```

Out[72]:

	similar	all
1	1.000000	0.190768
5	0.382767	0.081025
6	0.431453	0.130427
16	0.113524	0.044685
19	0.165533	0.057894
...	...	...
30276	0.263292	0.162522
30503	0.105564	0.079079
31043	0.168161	0.112839
31240	0.111283	0.073683
31964	0.109892	0.070322

326 rows × 2 columns

We will then calculate a score by dividing one percentage by the other.

In [73]:
```
# finding the ratio between the percentages
rec_percentages["score"] = rec_percentages["similar"] / rec_percentages["all"]

# sorting the recommendations based on the scores
rec_percentages = rec_percentages.sort_values("score", ascending=False)
rec_percentages
```

Out[73]:

	similar	all	score
1	1.000000	0.190768	5.241963
5	0.382767	0.081025	4.724062
400	0.142040	0.035721	3.976363
329	0.106105	0.028866	3.675799
801	0.184158	0.050125	3.673999
...	...	...	...
11757	0.339335	0.329717	1.029173
15451	0.100077	0.098170	1.019424
11617	0.147991	0.147484	1.003435
7054	0.165688	0.174109	0.951632
3457	0.129212	0.138639	0.932004

326 rows × 3 columns

From the above dataframe, the higher the score, the better the recommendation.

We will then get the top 10 recommendations and merge that with our `anime` dataset so as to get the title of the movies.

In [74]:
```python
# getting top 10 recommendations
rec_percentages.head(10).merge(data, left_index=True, right_on="Anime_id")
```

Out[74]:

	similar	all	score	Anime_id	Title	Genre	Synopsis	Type	Pr
0	1.000000	0.190768	5.241963	1	Cowboy Bebop	['Action', 'Adventure', 'Comedy', 'Drama', 'Sc...	In the year 2071, humanity has colonized sever...	TV	['Bandai
1	0.382767	0.081025	4.724062	5	Cowboy Bebop Tengoku no Tobira	['Action', 'Space', 'Drama', 'Mystery', 'Sci-Fi']	Another day, another bounty—such is the life o...	Movie	['S 'Bandai
294	0.106105	0.028866	3.675799	329	Planetes	['Drama', 'Romance', 'Sci-Fi', 'Seinen', 'Space']	In 2075, space travel is no longer just a drea...	TV	['Bandai
642	0.184158	0.050125	3.673999	801	Koukaku Kidoutai Stand Alone Complex 2nd GIG	['Action', 'Military', 'Sci-Fi', 'Mystery', 'P...	Following the closure of the "Laughing Man" ca...	TV	['Bandai 'Dentsu' Enter
3164	0.125502	0.035426	3.542633	4106	Trigun Badlands Rumble	['Action', 'Sci-Fi', 'Adventure', 'Comedy', 'D...	20 years after meddling into the bank heist o...	Movie	['flying 'Sun Up', .
398	0.263756	0.074877	3.522508	467	Koukaku Kidoutai Stand Alone Complex	['Action', 'Military', 'Sci-Fi', 'Police', 'Me...	In the not so distant future, mankind has adva...	TV	['Bandai 'Dentsu' Enter
518	0.106337	0.030473	3.489570	617	Juubee Ninpuuchou	['Adventure', 'Historical', 'Horror', 'Superna...	Jubei Kibagami wanders feudal Japan as an itin...	Movie	[
238	0.128130	0.037402	3.425768	267	Gungrave	['Action', 'Drama', 'Sci-Fi', 'Seinen', 'Super...	Brandon Heat and Harry MacDowel, two friends s...	TV	['Tohoku Corpo

We will then put all the code above into one function.

In [75]:
```
def find_similar_movies(movie_id):

    # finding recommendations for users that are similar
    similar_users = ratings[(ratings["anime_id"] == movie_id) & (ratings["rating"
    similar_user_recs = ratings[(ratings["user_id"].isin(similar_users)) & (ratin

    # narrowing them down to 10%
    similar_user_recs = similar_user_recs.value_counts() / len(similar_users)
    similar_user_recs = similar_user_recs[similar_user_recs > .10]

    # finding how common the recommendations are among all users
    all_users = ratings[(ratings["anime_id"].isin(similar_user_recs.index)) & (ra
    all_user_recs = all_users["anime_id"].value_counts() / len(all_users["user_id

    # concatenating all_user_recs and similar_user_recs
    rec_percentages = pd.concat([similar_user_recs, all_user_recs], axis=1)
    rec_percentages.columns = ["similar", "all"]

    # calculating the score
    rec_percentages["score"] = rec_percentages["similar"] / rec_percentages["all"
    # sorting values based on score
    rec_percentages = rec_percentages.sort_values("score", ascending=False)

    # return the top 10 recommendations
    return rec_percentages.head(10).merge(data, left_index=True, right_on="Anime_
```

We will now create an interactive recommendation widget where one gets to type in an anime title and get recommendations based on the anime title

In [76]:
```
# creating an input widget
movie_name_input = widgets.Text(
    value='Trigun',
    description='Movie Title:',
    disabled=False
)

# creating an output widget
recommendation_list = widgets.Output()

# creating an ontype function
def on_type(data):
    with recommendation_list:
        recommendation_list.clear_output()

        # grab title from input widget
        title = data["new"]

        # display output when length of title is more than 5
        if len(title) > 5:
            # search title using the search function
            results = search(title)
            movie_id = results.iloc[0]["Anime_id"]
            display(find_similar_movies(movie_id))

movie_name_input.observe(on_type, names='value')

display(movie_name_input, recommendation_list)
```

```
Text(value='Trigun', description='Movie Title:')
```

```
Output()
```

Model Based Modelling

In this section, we will use the knnwithmeans and SVD algorithms to build our recommendation model.

In [86]:
```
# reloading and preview of data
new_df = pd.read_csv("cleaned_anime.csv")
new_df.dropna(subset = ['Title'], inplace = True)
new_df = df_anime[['Anime_id', 'Title', 'Rating']]
new_df
```

Out[86]:

	Anime_id	Title	Rating
0	1	Cowboy Bebop	8.81
1	5	Cowboy Bebop Tengoku no Tobira	8.41
2	6	Trigun	8.31
3	7	Witch Hunter Robin	7.34
4	8	Bouken Ou Beet	7.04
...	...	...	...
12974	40033	Animagear	6.42
12975	40042	Magical Halloween MiracleQuartet	6.42
12976	40055	Orbital Era	6.42
12977	40057	Akai Hana Shiroi Hana	6.42
12978	40058	Arui Tekoteko	6.42

12979 rows × 3 columns

In [87]:
```
#Loading the data set
reader = Reader()
data = Dataset.load_from_df(new_df,reader)
```

In [88]:
```
# Print the number of users and items
dataset = data.build_full_trainset()
print('Number of users: ', dataset.n_users, '\n')
print('Number of items: ', dataset.n_items)
```

```
Number of users:  12943

Number of items:  12909
```

In [89]:
```
# Split the data into train and test set
trainset, testset = train_test_split(data, test_size=0.2, random_state=42)
```

In [90]:
```
# Print the number of users and items
print('Number of users_train: ', trainset.n_users, '\n')
print('Number of items_train: ', trainset.n_items, '\n')
```

```
Number of users_train:  10356

Number of items_train:  10341
```

In [91]:
```
# creating a class to get the top users
from collections import defaultdict

def get_top_n(predictions, n=10):
    # First map the predictions to each user.
    top_n = defaultdict(list)
    for uid, iid, true_r, est, _ in predictions:
        top_n[uid].append((iid, est))

    # Then sort the predictions for each user and retrieve the k highest ones.
    for uid, user_ratings in top_n.items():
        user_ratings.sort(key=lambda x: x[1], reverse=True)
        top_n[uid] = user_ratings[:n]

    return top_n
```

In [92]:
```
#function to use for collaborative Filtering
class collab_filtering_based_recommender_model():
    def __init__(self, model, trainset, testset, data):
        self.model = model
        self.trainset = trainset
        self.testset = testset
        self.data = data
        self.pred_test = None
        self.recommendations = None
        self.top_n = None
        self.recommenddf = None

    def fit_and_predict(self):
        print('**Fitting the train data...**')
        self.model.fit(self.trainset)

        print('**Predicting the test data...**')
        self.pred_test = self.model.test(self.testset)
        rmse = round(accuracy.rmse(self.pred_test), 3)
        print('**RMSE for the predicted result is ' + str(rmse) + '**')

        self.top_n = get_top_n(self.pred_test)
        self.recommenddf = pd.DataFrame(columns=['CustomerID', 'Description', 'ra
        for item in self.top_n:
            subdf = pd.DataFrame(self.top_n[item], columns=['Description', 'ratir
            subdf['CustomerID'] = item
            cols = subdf.columns.tolist()
            cols = cols[-1:] + cols[:-1]
            subdf = subdf[cols]
            self.recommenddf = pd.concat([self.recommenddf, subdf], axis = 0)
        return rmse

    def cross_validate(self):
        print('**Cross Validating the data...**')
        cv_result = cross_validate(self.model, self.data, n_jobs=-1)
        cv_result = round(cv_result['test_rmse'].mean(),3)
        print('**Mean CV RMSE is ' + str(cv_result)  + '**')
        return cv_result
```

In [93]:
```
# Finding the best collaborative Filtering model
from surprise.model_selection import RandomizedSearchCV

def find_best_model(model, parameters,data):
    clf = RandomizedSearchCV(model, parameters, n_jobs=-1, measures=['rmse'])
    clf.fit(data)
    print(clf.best_score)
    print(clf.best_params)
    print(clf.best_estimator)
    return clf
```

In [94]:
```
# finding the best model between cosine, pearson and pearson_baseline
sim_options = {
    "name": ["msd", "cosine", "pearson", "pearson_baseline"],
    "min_support": [3, 4, 5],
    "user_based": [True],
}
params = { 'k': range(30,50,1), 'sim_options': sim_options}
clf = find_best_model(KNNWithMeans, params, data)
```

```
{'rmse': 1.7064379632795088}
{'rmse': {'k': 44, 'sim_options': {'name': 'msd', 'min_support': 4, 'user_base
d': True}}}
{'rmse': <surprise.prediction_algorithms.knns.KNNWithMeans object at 0x00000192
D611FEE0>}
```

We have an RMSE of 1.7. We will go ahead and fit another model using our collaborative filtering function above to see if we get better results

In [95]:
```
#fit knnwithmeans model
knnwithmeans = clf.best_estimator['rmse']
col_fil_knnwithmeans = collab_filtering_based_recommender_model(knnwithmeans, tra
```

In [96]:
```
#fit and predict using the knnmeans model
knnwithmeans_rmse = col_fil_knnwithmeans.fit_and_predict()
```

```
**Fitting the train data...**
Computing the msd similarity matrix...
Done computing similarity matrix.
**Predicting the test data...**
RMSE: 1.6958
**RMSE for the predicted result is 1.696**
```

The rmse does not change much therefore we will go ahead and cross validate our model to see if we have rmse

In [97]:
```
#cross validate using knnwith means
knnwithmeans_cv_rmse = col_fil_knnwithmeans.cross_validate()
```

```
**Cross Validating the data...**
**Mean CV RMSE is 1.706**
```

The rmse still does not change much therefore we will use the SVD algorithm to build another model and then cross validate it to see if we have better results

In [98]:
```
#fitting the SVD model
params= {
    "n_epochs": [5, 10, 15, 20],
    "lr_all": [0.002, 0.005],
    "reg_all": [0.4, 0.6]
}
clf = find_best_model(SVD, params, data)
```

```
{'rmse': 1.7065532888090842}
{'rmse': {'n_epochs': 20, 'lr_all': 0.002, 'reg_all': 0.6}}
{'rmse': <surprise.prediction_algorithms.matrix_factorization.SVD object at 0x0
0000192F02F4B20>}
```

In [99]:
```
#fitting the SVD model to the train and test data
svd = clf.best_estimator['rmse']
col_fil_svd = collab_filtering_based_recommender_model(svd, trainset, testset, da
```

In [100]:
```
#fitting and prediciting the SVD model
svd_rmse = col_fil_svd.fit_and_predict()
```

```
**Fitting the train data...**
**Predicting the test data...**
RMSE: 1.6959
**RMSE for the predicted result is 1.696**
```

In [101]:
```
#cross validate using SVD
svd_cv_rmse = col_fil_svd.cross_validate()
```

```
**Cross Validating the data...**
**Mean CV RMSE is 1.707**
```

In [102]:
```
cross_validate(svd,data,measures=['RMSE', 'MAE'], cv=5, verbose=True)
```

```
Evaluating RMSE, MAE of algorithm SVD on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    1.6939  1.7235  1.7023  1.7022  1.7109  1.7066  0.0100
MAE (testset)     1.4767  1.4933  1.4795  1.4864  1.4917  1.4855  0.0066
Fit time          0.98    1.18    1.08    1.35    1.10    1.14    0.12
Test time         0.03    0.03    0.03    0.03    0.03    0.03    0.00
```

Out[102]:
```
{'test_rmse': array([1.693931  , 1.72354433, 1.70229858, 1.70219042, 1.7108591
]),
 'test_mae': array([1.4766718 , 1.49333975, 1.47948767, 1.48642527, 1.4916840
1]),
 'fit_time': (0.981809139251709,
  1.1813406944274902,
  1.0792567729949951,
  1.3485989570617676,
  1.0972683429718018),
 'test_time': (0.028460025787353516,
  0.031234264373779297,
  0.029009580612182617,
  0.02599954605102539,
  0.02600860595703125)}
```

EVALAUTION

We have built 2 types of recommendation model;

- Metric based
- Model based.

For the metric based model, we have the content-based and the hybrid model. The hybrid model gave better recommendations than than the content-based model.

For the Model based, we have used the `SVD` and `knnwithmeans` algorithms. Both models perform almost the same hence we opted to go with either of the model since they both satisfy our objective of attaining an RMSE of 2.0 and below.

In [103]:
```
Models=['SVD', 'KNNWithMeans']
```

In [105]:
```
# creating a dataframe to compare the algorithms
pd.DataFrame({'model':['SVD', 'KNNWithMeans'],'RMSE':[svd_cv_rmse,knnwithmeans_cv
```

Out[105]:

	model	RMSE
0	SVD	1.707
1	KNNWithMeans	1.706

DEPLOYMENT

We deployed the hybrid model on streamlit where the app prompts a user to select a title. After selecting the title, the user is given the top 10 recommendations.

- One can use the model through the following Link (https://anime-deployment.streamlit.app/)

RECOMMENDATIONS

- This app is beneficial to the anime lovers and anime producers.
- The recommendation system can be improved by using GridSearch cv. Grid search requires a lot of computational power therefore it is recommended that you have enough computational power so as to do a Grid search.