In [1]:
```
import numpy as np
import pandas as pd
from scipy import spatial
import re
import os
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
anime = pd.read_csv('anime.csv')
anime = anime.dropna()
anime = anime[anime['episodes'] != 'Unknown']
rating = pd.read_csv('rating.csv')
```

In [3]:
```
anime_full=pd.merge(anime,rating,on='anime_id',suffixes= ['', '_user'])
```

In [4]:
```
rating.shape
```

Out[4]:
```
(7813737, 3)
```

In [5]:
```
rating.head()
```

Out[5]:

	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 [6]:
```
ratings = rating[rating['rating'] != -1]
```

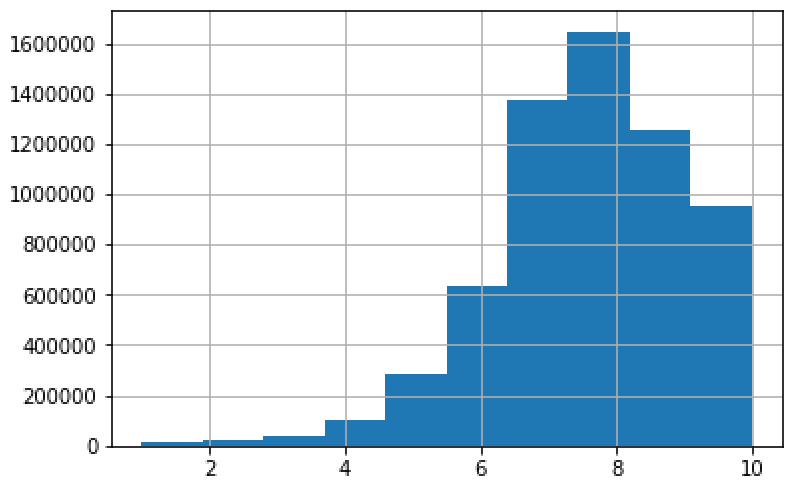
In [7]:
```
ratings.head()
```

Out[7]:

	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

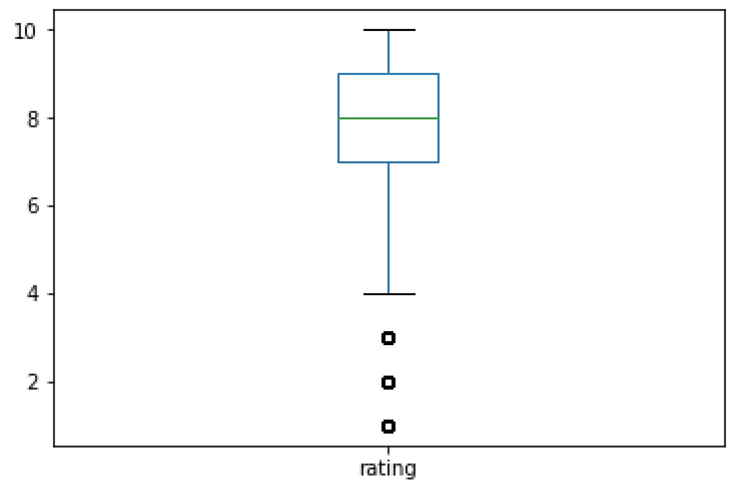
In [8]:
```
ratings['rating'].hist()
```

Out[8]: <matplotlib.axes._subplots.AxesSubplot at 0x7fb7c3cc2240>



In [9]:
```
ratings['rating'].plot(kind = 'box', subplots = True)
```

Out[9]:
```
rating    AxesSubplot(0.125,0.125;0.775x0.755)
dtype: object
```



Anime ratings aggregated by user

In [10]:
```
uRatings = ratings.groupby(['user_id']).agg({'rating' : [np.size, np.mean]})
```

In [11]:
```
uRatings.reset_index(inplace = True)
```

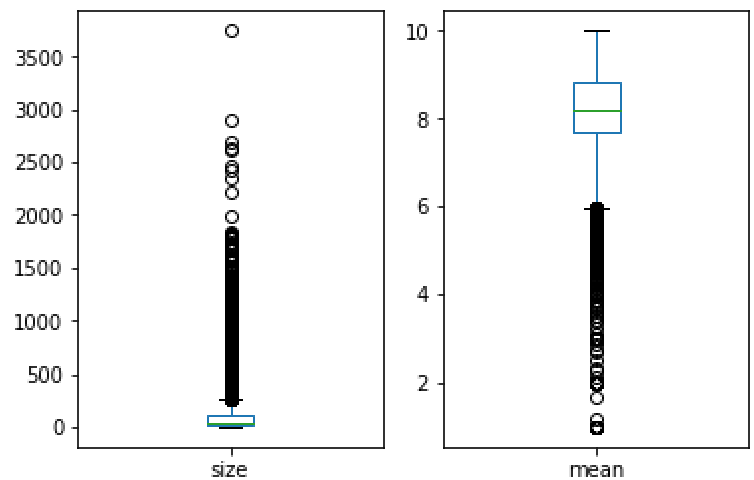
In [12]:
```
1 uRatings['rating'].describe()
```

Out[12]:

	size	mean
count	69600.000000	69600.000000
mean	91.052313	8.227761
std	135.764253	0.902856
min	1.000000	1.000000
25%	13.000000	7.666667
50%	45.000000	8.193548
75%	114.000000	8.815789
max	3747.000000	10.000000

In [13]:
```
1 uRatings['rating'].plot(kind = 'box', subplots = True)
```

Out[13]:
```
size        AxesSubplot(0.125,0.125;0.352273x0.755)
mean    AxesSubplot(0.547727,0.125;0.352273x0.755)
dtype: object
```



Anime ratings aggregated by anime

In [14]:
```
mRatings = ratings.groupby(['anime_id']).agg({'rating' : [np.size, np.mean]})
mRatings.reset_index(inplace = True)
```

In [15]:
```
mRatings['rating'].describe()
```

Out[15]:

	size	mean
count	9927.000000	9927.000000
mean	638.384305	6.637940
std	1795.865541	1.298863
min	1.000000	1.000000
25%	9.000000	6.066667
50%	57.000000	6.897959
75%	395.000000	7.491484
max	34226.000000	10.000000

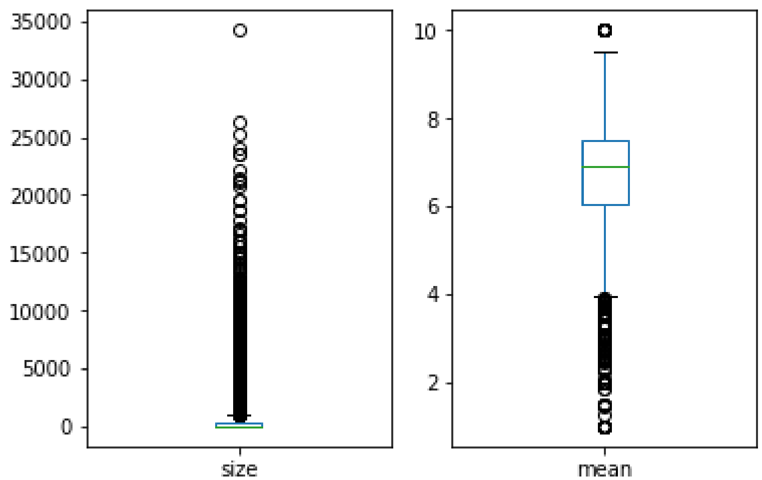
In [16]:
```
mRatings.head()
```

Out[16]:

	anime_id	rating	
		size	mean
0	1	13449	8.869433
1	5	5790	8.439724
2	6	9385	8.419393
3	7	2169	7.533426
4	8	308	7.198052

In [17]:
```
mRatings['rating'].plot(kind = 'box', subplots = True)
```

Out[17]:
```
size        AxesSubplot(0.125,0.125;0.352273x0.755)
mean    AxesSubplot(0.547727,0.125;0.352273x0.755)
dtype: object
```



Ger genre Matrix

In [18]:
```
animes = anime.merge(mRatings, on = 'anime_id', how = 'left')
animes.columns = ['anime_id', 'name', 'genre', 'type', 'episodes', 'rating', 'members', 'rating_cou
```

```
/Users/boris/anaconda3/envs/python/lib/python3.7/site-packages/pandas/core/reshape/merge.py:522: UserW
arning: merging between different levels can give an unintended result (1 levels on the left, 2 on the
right)
  warnings.warn(msg, UserWarning)
/Users/boris/anaconda3/envs/python/lib/python3.7/site-packages/pandas/core/generic.py:3812: Performanc
eWarning: dropping on a non-lexsorted multi-index without a level parameter may impact performance.
  new_axis = axis.drop(labels, errors=errors)
```

In [19]:
```
animes.fillna(0,inplace = True)
```

Top 10 Animes based on rating count

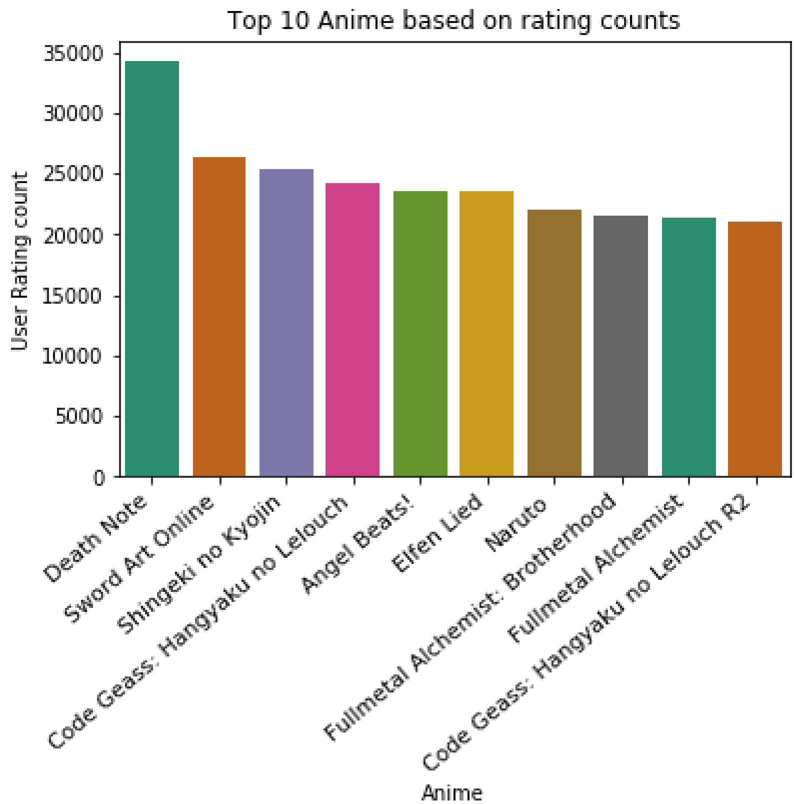
In [20]:
```
top10_animerating=animes[['name', 'rating_count']].sort_values(by = 'rating_count',ascending = Fals
top10_animerating
```

Out[20]:

	name	rating_count
40	Death Note	34226.0
801	Sword Art Online	26310.0
85	Shingeki no Kyojin	25290.0
19	Code Geass: Hangyaku no Lelouch	24126.0
158	Angel Beats!	23565.0
757	Elfen Lied	23528.0
838	Naruto	22071.0
1	Fullmetal Alchemist: Brotherhood	21494.0
199	Fullmetal Alchemist	21332.0
13	Code Geass: Hangyaku no Lelouch R2	21124.0

In [21]:
```
ax=sns.barplot(x="name", y="rating_count", data=top10_animerating, palette="Dark2")
ax.set_xticklabels(ax.get_xticklabels(), fontsize=11, rotation=40, ha="right")
ax.set_title('Top 10 Anime based on rating counts',fontsize = 12)
ax.set_xlabel('Anime',fontsize = 10)
ax.set_ylabel('User Rating count', fontsize = 10)
```

Out[21]:
```
Text(0, 0.5, 'User Rating count')
```



Top 10 Animes based on Community size

In [22]:

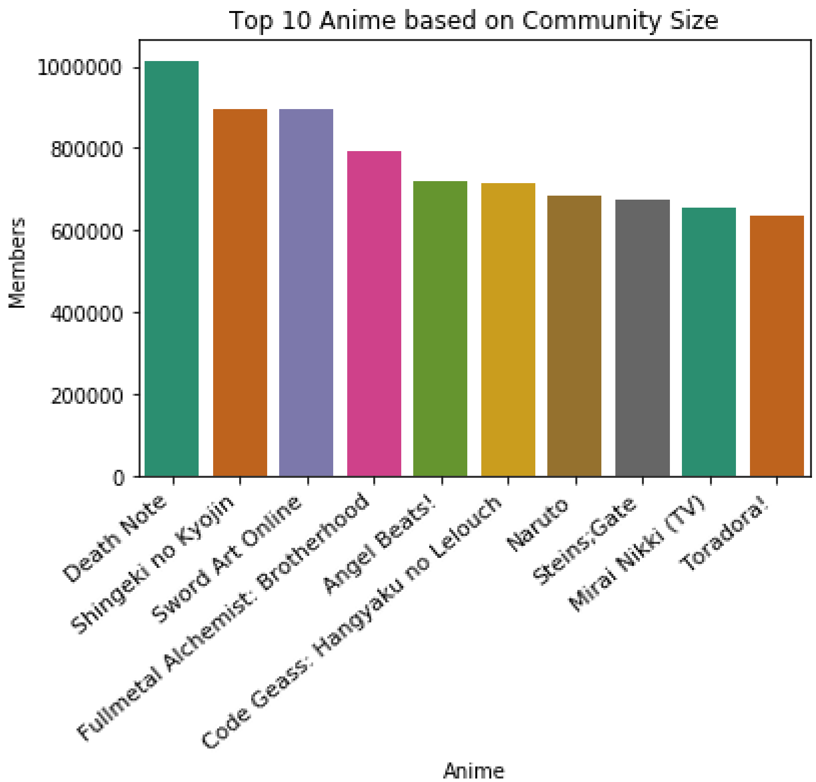
```
top10_animemember=animes[['name', 'members']].sort_values(by = 'members',ascending = False).head(10
top10_animemember
```

Out[22]:

	name	members
40	Death Note	1013917
85	Shingeki no Kyojin	896229
801	Sword Art Online	893100
1	Fullmetal Alchemist: Brotherhood	793665
158	Angel Beats!	717796
19	Code Geass: Hangyaku no Lelouch	715151
838	Naruto	683297
3	Steins;Gate	673572
443	Mirai Nikki (TV)	657190
130	Toradora!	633817

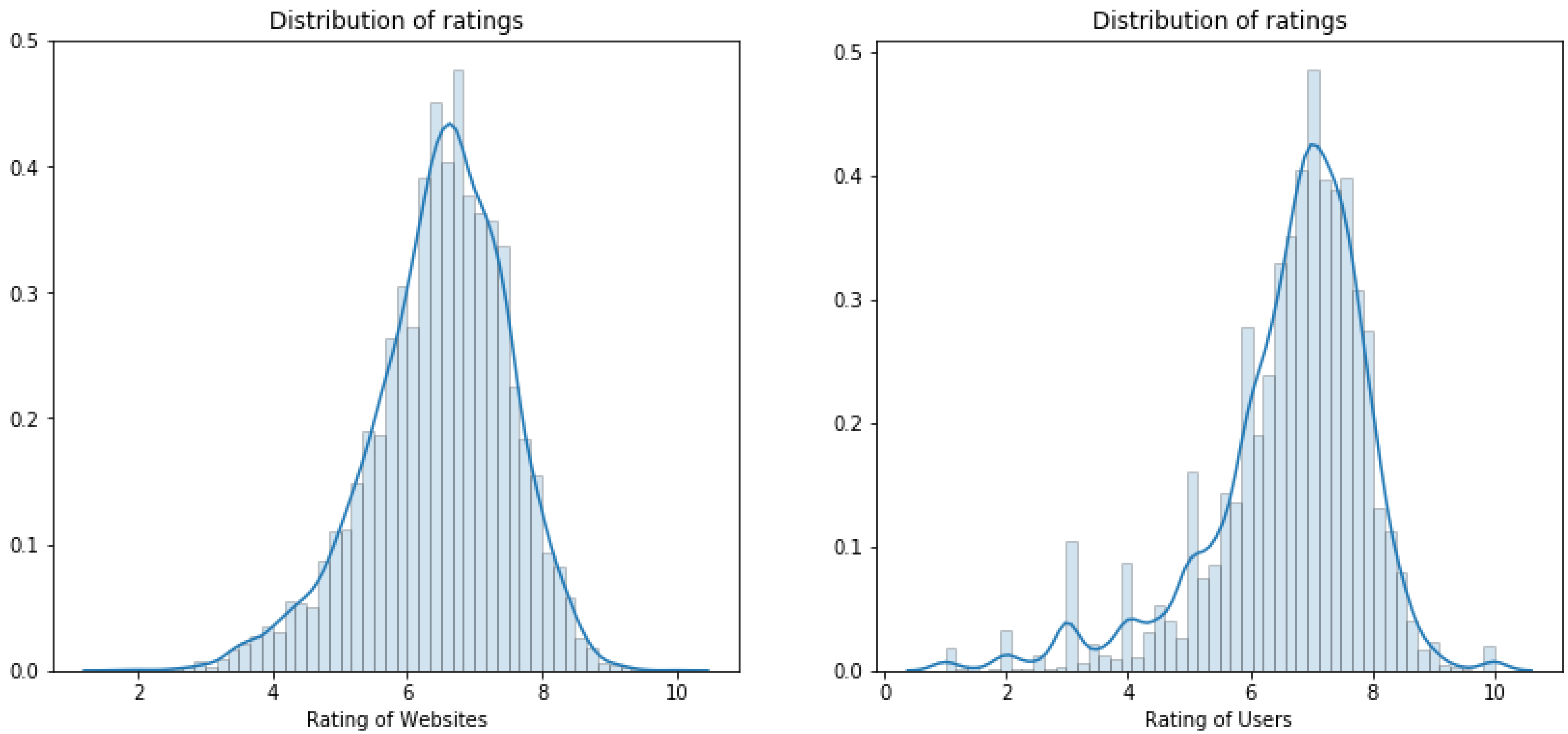
In [23]:
```
ax=sns.barplot(x="name", y="members", data=top10_animemember, palette="Dark2")
ax.set_xticklabels(ax.get_xticklabels(), fontsize=11, rotation=40, ha="right")
ax.set_title('Top 10 Anime based on Community Size',fontsize = 12)
ax.set_xlabel('Anime',fontsize = 10)
ax.set_ylabel('Members', fontsize = 10)
```

Out[23]:
```
Text(0, 0.5, 'Members')
```



Distribution of ratings

In [24]:
```
hist_kws = {'histtype':'bar', 'edgecolor':'black', 'alpha':0.2}
fig, ax = plt.subplots(nrows = 1, ncols = 2, figsize = (14,6))
sns.distplot(animes['rating'], hist_kws=hist_kws, ax = ax[0])
sns.distplot(animes[animes['rating_avg'] != 0]['rating_avg'], hist_kws=hist_kws, ax = ax[1])
ax[0].set_title('Distribution of ratings')
ax[0].set_xlabel('Rating of Websites')
ax[1].set_title('Distribution of ratings')
ax[1].set_xlabel('Rating of Users')
plt.show()
```



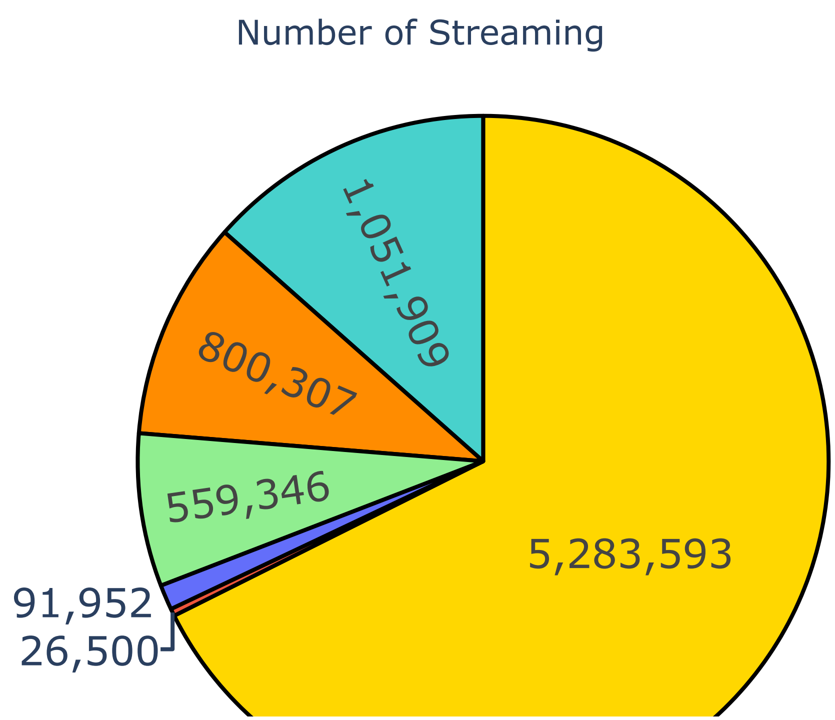
Streaming Type

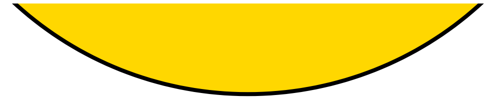
In [25]:

```
import plotly.graph_objects as go
labels = anime_full['type'].value_counts().index
values = anime_full['type'].value_counts().values
colors = ['gold', 'mediumturquoise', 'darkorange', 'lightgreen']
fig = go.Figure(data=[go.Pie(labels=labels,
                             values=values)])
fig.update_traces(hoverinfo='label+percent', textinfo='value', textfont_size=20,
                  marker=dict(colors=colors, line=dict(color='#000000', width=2)))

fig.update_layout(
    title={
        'text': "Number of Streaming",
        'y':0.9,
        'x':0.5,
        'xanchor': 'center',
        'yanchor': 'top'})

fig.show()
```





In [26]:

```
nonull_anime=anime_full.copy()
nonull_anime.dropna(inplace=True)
from collections import defaultdict

all_genres = defaultdict(int)

for genres in nonull_anime['genre']:
    for genre in genres.split(','):
        all_genres[genre.strip()] += 1

from wordcloud import WordCloud

genres_cloud = WordCloud(width=800, height=400, background_color='white', colormap='gnuplot').genera
plt.imshow(genres_cloud, interpolation='bilinear')
plt.axis('off')
```

Out[26]: (-0.5, 799.5, 399.5, -0.5)

Content Based Recommendation system

Clean Anime name

In [27]:
```
anime_df = pd.read_csv('anime.csv')
anime_df.dropna(inplace=True)
def text_cleaning(text):
    text = re.sub(r'&quot;', '', text)
    text = re.sub(r'.hack//', '', text)
    text = re.sub(r'&#039;', '', text)
    text = re.sub(r'A&#039;s', '', text)
    text = re.sub(r'I&#039;', 'I\'', text)
    text = re.sub(r'&amp;', 'and', text)

    return text

anime_df['name'] = anime_df['name'].apply(text_cleaning)
```

Term Frequency (TF) and Inverse Document Frequency (IDF)

In [28]:
```
from sklearn.feature_extraction.text import TfidfVectorizer

genres_str = anime_df['genre'].str.split(',').astype(str)

tfidf = TfidfVectorizer(analyzer='word', ngram_range=(1, 4), min_df=0)
tfidf_matrix = tfidf.fit_transform(genres_str)
```

In [29]:
```
from sklearn.metrics.pairwise import linear_kernel

cosine_sim = linear_kernel(tfidf_matrix, tfidf_matrix)
```

In [30]:
```
indices = pd.Series(anime_df.index, index=anime_df['name'])
```

In [31]:
```python
def genre_recommendations(title, highest_rating=False, similarity=False):

    if highest_rating == False:
        if similarity == False:

            idx = indices[title]
            sim_scores = list(enumerate(cosine_sim[idx]))
            sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)
            sim_scores = sim_scores[1:11]

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

            return pd.DataFrame({'Anime name': anime_df['name'].iloc[anime_indices].values,
                                 'Type': anime_df['type'].iloc[anime_indices].values})

        elif similarity == True:

            idx = indices[title]
            sim_scores = list(enumerate(cosine_sim[idx]))
            sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)
            sim_scores = sim_scores[1:11]

            anime_indices = [i[0] for i in sim_scores]
            similarity_ = [i[1] for i in sim_scores]

            return pd.DataFrame({'Anime name': anime_df['name'].iloc[anime_indices].values,
                                 'Similarity': similarity_,
                                 'Type': anime_df['type'].iloc[anime_indices].values})
    elif highest_rating == True:
        if similarity == False:

            idx = indices[title]
            sim_scores = list(enumerate(cosine_sim[idx]))
            sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)
            sim_scores = sim_scores[1:11]

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

            result_df = pd.DataFrame({'Anime name': anime_df['name'].iloc[anime_indices].values,
                                 'Type': anime_df['type'].iloc[anime_indices].values,
                                 'Rating': anime_df['rating'].iloc[anime_indices].values})

```

```
            return result_df.sort_values('Rating', ascending=False)

        elif similarity == True:

            idx = indices[title]
            sim_scores = list(enumerate(cosine_sim[idx]))
            sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)
            sim_scores = sim_scores[1:11]

            anime_indices = [i[0] for i in sim_scores]
            similarity_ = [i[1] for i in sim_scores]

            result_df = pd.DataFrame({'Anime name': anime_df['name'].iloc[anime_indices].values,
                                     'Similarity': similarity_,
                                     'Type': anime_df['type'].iloc[anime_indices].values,
                                     'Rating': anime_df['rating'].iloc[anime_indices].values})

            return result_df.sort_values('Rating', ascending=False)
```

In [32]:

```
genre_recommendations('One Punch Man', highest_rating=True, similarity=True)
```

Out[32]:

	Anime name	Similarity	Type	Rating
5	Gungrave	0.536635	TV	7.97
0	One Punch Man Specials	1.000000	Special	7.86
1	One Punch Man: Road to Hero	1.000000	OVA	7.85
7	Darker than Black: Kuro no Keiyakusha Special	0.483113	Special	7.65
8	Haiyore! Nyaruko-san W	0.483113	TV	7.43
9	Haiyore! Nyaruko-san: Yasashii Teki no Shitome...	0.483113	OVA	7.27
2	Genji Tsuushin Agedama	0.604524	TV	6.58
6	Bobobo-bo Bo-bobo Recap	0.535226	Special	6.54
3	Oh! Super Milk-chan	0.604524	TV	6.07
4	Super Milk-chan	0.604524	TV	5.88

Collaborative filtering Recommendation system

In [33]:
```
rating = pd.read_csv('rating.csv')
```

In [34]:
```
rating = rating[rating['rating'] >= 6]
```

In [35]:
```
def change_rating(rating):
    if rating == 6:
        return 1
    elif rating == 7:
        return 2
    elif rating == 8:
        return 3
    elif rating == 9:
        return 4
    elif rating == 10:
        return 5

a = rating['rating'].apply(change_rating)
rating['rating'] = a
```

In [36]:
```
from sklearn.preprocessing import LabelEncoder

user_enc = LabelEncoder()
rating['user_id'] = user_enc.fit_transform(rating['user_id'])

anime_enc = LabelEncoder()
rating['anime_id'] = anime_enc.fit_transform(rating['anime_id'])
```

In [37]:
```
import tensorflow as tf
from tensorflow.keras import Model
from tensorflow.keras.layers import Input, Embedding, Reshape, Dot, Flatten, concatenate, Dense, Dr
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import ModelCheckpoint

from tensorflow.keras.utils import model_to_dot
from IPython.display import SVG
```

In [38]:
```
def RecommenderV2(n_users, n_movies, n_dim):

    user = Input(shape=(1,))
    U = Embedding(n_users, n_dim)(user)
    U = Flatten()(U)

    movie = Input(shape=(1,))
    M = Embedding(n_movies, n_dim)(movie)
    M = Flatten()(M)

    merged_vector = concatenate([U, M])
    dense_1 = Dense(128, activation='relu')(merged_vector)
    dropout = Dropout(0.5)(dense_1)
    final = Dense(1)(dropout)

    model = Model(inputs=[user, movie], outputs=final)

    model.compile(optimizer=Adam(0.001),
                  loss='mean_squared_error')

    return model
```

In [39]:
```
userid_nunique = rating['user_id'].nunique()
anime_nunique = rating['anime_id'].nunique()
```

In [40]:
```
model = RecommenderV2(userid_nunique, anime_nunique, 100)
```

In [41]:
```
#To decrease computation time, I only select first 100 users
rating = rating[rating['user_id'] <= 100]
```

In [42]:
```
from sklearn.model_selection import train_test_split

X = rating.drop(['rating'], axis=1)
y = rating['rating']

X_train, X_val, y_train, y_val = train_test_split(X, y,
                                                  test_size=.1,
                                                  stratify=y,
                                                  random_state=2020)

X_train.shape, X_val.shape, y_train.shape, y_val.shape
```

Out[42]:
```
((5230, 2), (582, 2), (5230,), (582,))
```

In [43]:
```
checkpoint = ModelCheckpoint('model1.h5', monitor='val_loss', verbose=0, save_best_only=True)
```

In [44]:

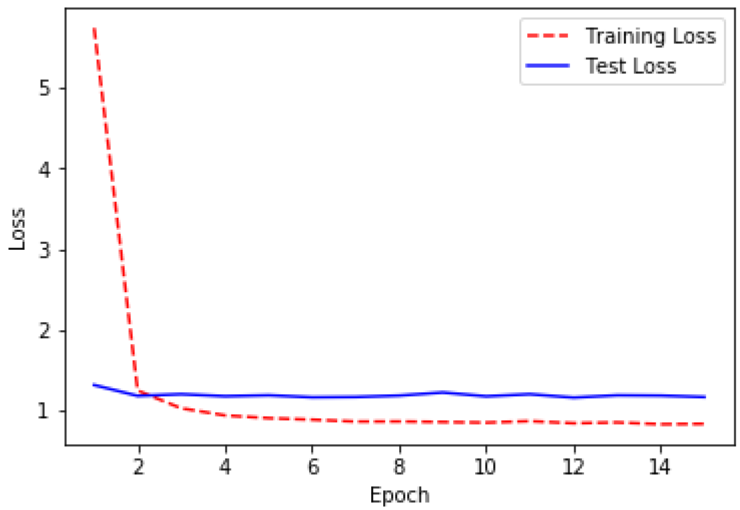
```
history = model.fit(x=[X_train['user_id'], X_train['anime_id']],
                    y=y_train,
                    batch_size=64,
                    epochs=15,
                    verbose=1,
                    validation_data=([X_val['user_id'], X_val['anime_id']], y_val),
                    callbacks=[checkpoint])
```

```
Train on 5230 samples, validate on 582 samples
Epoch 1/15
5230/5230 [==============================] - 6s 1ms/sample - loss: 5.7367 - val_loss: 1.3177
Epoch 2/15
5230/5230 [==============================] - 6s 1ms/sample - loss: 1.2491 - val_loss: 1.1833
Epoch 3/15
5230/5230 [==============================] - 6s 1ms/sample - loss: 1.0292 - val_loss: 1.2041
Epoch 4/15
5230/5230 [==============================] - 6s 1ms/sample - loss: 0.9409 - val_loss: 1.1791
Epoch 5/15
5230/5230 [==============================] - 6s 1ms/sample - loss: 0.9070 - val_loss: 1.1906
Epoch 6/15
5230/5230 [==============================] - 6s 1ms/sample - loss: 0.8883 - val_loss: 1.1658
Epoch 7/15
5230/5230 [==============================] - 6s 1ms/sample - loss: 0.8650 - val_loss: 1.1695
Epoch 8/15
5230/5230 [==============================] - 6s 1ms/sample - loss: 0.8661 - val_loss: 1.1870
Epoch 9/15
5230/5230 [==============================] - 6s 1ms/sample - loss: 0.8591 - val_loss: 1.2253
Epoch 10/15
5230/5230 [==============================] - 6s 1ms/sample - loss: 0.8519 - val_loss: 1.1778
Epoch 11/15
5230/5230 [==============================] - 6s 1ms/sample - loss: 0.8704 - val_loss: 1.2044
Epoch 12/15
5230/5230 [==============================] - 6s 1ms/sample - loss: 0.8438 - val_loss: 1.1621
Epoch 13/15
5230/5230 [==============================] - 6s 1ms/sample - loss: 0.8544 - val_loss: 1.1906
Epoch 14/15
5230/5230 [==============================] - 6s 1ms/sample - loss: 0.8318 - val_loss: 1.1867
Epoch 15/15
5230/5230 [==============================] - 6s 1ms/sample - loss: 0.8367 - val_loss: 1.1691
```

In [45]:
```python
# Get training and test loss histories
training_loss2 = history.history['loss']
test_loss2 = history.history['val_loss']

# Create count of the number of epochs
epoch_count = range(1, len(training_loss2) + 1)

# Visualize
plt.plot(epoch_count, training_loss2, 'r--')
plt.plot(epoch_count, test_loss2, 'b-')
plt.legend(['Training Loss', 'Test Loss'])
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.show()
```



In [46]:
```python
from tensorflow.keras.models import load_model

model = load_model('model1.h5')

```

In [47]:
```python
def make_pred(user_id, anime_id, model):
    return model.predict([np.array([user_id]), np.array([anime_id])])[0][0]
```

In [48]:
```
def get_topN_rec(user_id, model):

    user_id = int(user_id) - 1
    user_ratings = rating[rating['user_id'] == user_id]
    recommendation = rating[~rating['anime_id'].isin(user_ratings['anime_id'])][['anime_id']].drop_d
    recommendation['rating_predict'] = recommendation.apply(lambda x: make_pred(user_id, x['anime_id

    final_rec = recommendation.sort_values(by='rating_predict', ascending=False).merge(anime_df[['an
                                                                                       on='anime_id'

    return final_rec.sort_values('rating_predict', ascending=False)[['name', 'type', 'rating_predict
```

In [49]:
```
#Recommend 10 movies to user 26
get_topN_rec(26, model)
```

Out[49]:

	name	type	rating_predict
0	Ace wo Nerae!: Final Stage	OVA	4.804463
1	Koisuru Tenshi Angelique: Chibi Character Adve...	Special	4.639820
2	Onegai♪My Melody Kirara★	TV	4.602197
3	Canvas: Sepia-iro no Motif	OVA	4.567598
4	Touka Gettan	TV	4.562448
5	Ashita no Yukinojou	OVA	4.483605
6	Prince of Tennis	TV	4.448203
7	Working!!	TV	4.422463
8	Choujuu Kishin Dancougar: Hakunetsu no Shuushou	OVA	4.386375
9	Hellsing: Psalm of Darkness	Special	4.351424

In [ ]:
```

```