Anime Ratings Analysis

Importing the necessary libraries datasets

In [ ]:
```
# importing all the required libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split

import warnings
warnings.filterwarnings('ignore')
```

In [ ]:
```
#loading the datasets
anime_details = pd.read_csv('Data/anime.csv')
anime_ratings = pd.read_csv('Data/rating.csv')
```

Creating dictionaries to be used for styling the index, data and the caption

In [ ]:
```
# Creating dictionaries for different aspects to be used for styling in the whol
headers = { #header styling
    'selector': 'th:not(.index_name)',
    'props': [('background-color', '#474440'), ('color', 'white'),('text-align',
}

caption = { #caption styling
    'selector': 'caption',
    'props': [('font-weight', 'bold'), ('color', '#363445'),('text-align', 'left
}

cells = {  # row styling
    'selector': 'tr',
    'props': [('background-color', '#d7e1ee'), ('color', 'black'),('text-align',
}
```

In [ ]:
```
#creating a list of colors for Seaborn color palet
sns.set_style("white")
sns.set_context("poster",font_scale = .7)

colors = ["#ee2e31", "#f85e00" ,"#1d7874", "#fd7f6f", "#db8eca", "#7eb0d5", "#b2
sns.palplot(sns.color_palette(colors))
```

Descriptive Analysis

Summarizing and cleaning anime_data

In [ ]:
```
# taking a look at the dataset by sorting the data frame on anime_id
anime_details.sort_values(by='anime_id').head().style.set_table_styles([headers,
    .format(precision=2).set_caption("Let's take a look at the anime dataset")
```

Out[ ]:

Let's take a look at the anime dataset

anime_id	name	genre	type	episodes	rating	total_members
1	Cowboy Bebop	Action, Adventure, Comedy, Drama, Sci-Fi, Space	TV	26	8.82	486824
5	Cowboy Bebop: Tengoku no Tobira	Action, Drama, Mystery, Sci-Fi, Space	Movie	1	8.40	137636
6	Trigun	Action, Comedy, Sci-Fi	TV	26	8.32	283069
7	Witch Hunter Robin	Action, Drama, Magic, Mystery, Police, Supernatural	TV	26	7.36	64905
8	Beet the Vandel Buster	Adventure, Fantasy, Shounen, Supernatural	TV	52	7.06	9848

In [ ]:
```
# get a shape of the data frame
anime_details.shape
```

Out[ ]:
```
(12294, 7)
```

In [ ]:
```
#information regarding the columns in the data frame
anime_details.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 12294 entries, 0 to 12293
Data columns (total 7 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   anime_id       12294 non-null  int64  
 1   name           12294 non-null  object 
 2   genre          12232 non-null  object 
 3   type           12269 non-null  object 
 4   episodes       12294 non-null  object 
 5   rating         12064 non-null  float64
 6   total_members  12294 non-null  int64  
dtypes: float64(1), int64(2), object(4)
memory usage: 672.5+ KB
```

In [ ]:
```
# looking at the episodes column to determine why it's type is object instead of
anime_details['episodes'].value_counts().sort_index(ascending=False).head(10).to
    .style.set_table_styles([headers,cells,caption]) #checking the data in episo
```

Out[ ]:

	Unknown	99	98	97	96	95	94	93	92	91
episodes	106	2	1	3	4	2	3	1	2	2

In [ ]:
```
anime_details['episodes']= anime_details['episodes'].replace('Unknown',np.nan) #
anime_details['episodes']= anime_details['episodes'].astype(float) #Converting e
```

In [ ]:
```
#Checking the number of duplicate values
print(f"No.of duplicate values in anime_data: {anime_details[anime_details.dupli
```

```
No.of duplicate values in anime_data: 0
```

In [ ]:
```
#Checking for null values and eliminating them
display(anime_details.isna().sum().to_frame('count').sort_values(by='count',asce
    .style.set_table_styles([headers,cells,caption]).set_caption("No. of null va

anime_details.dropna(inplace = True)

display(anime_details.isna().sum().to_frame('count').sort_values(by='count',asce
    .style.set_table_styles([headers,cells,caption]).set_caption("No. of null va
```

No. of null values

	rating	episodes	genre	type	anime_id	name	total_members
count	230	106	62	25	0	0	0

No. of null values after dropping them

	anime_id	name	genre	type	episodes	rating	total_members
count	0	0	0	0	0	0	0

In [ ]:
```
#Statistical glimpse of anime_details data
anime_details.describe(include='all').style.set_table_styles([headers,cells,capt
```

Out[ ]:
Summary of anime data

	anime_id	name	genre	type	episodes	rating	total_members
count	11942.00	11942	11942	11942	11942.00	11942.00	11942.00
unique	nan	11940	3224	6	nan	nan	nan
top	nan	Shi Wan Ge Leng Xiaohua	Hentai	TV	nan	nan	nan
freq	nan	2	810	3618	nan	nan	nan
mean	13566.41	nan	nan	nan	12.70	6.48	18406.31
std	11198.00	nan	nan	nan	47.43	1.02	55343.06
min	1.00	nan	nan	nan	1.00	1.67	12.00
25%	3375.50	nan	nan	nan	1.00	5.89	230.00
50%	9920.50	nan	nan	nan	2.00	6.57	1573.50
75%	23614.50	nan	nan	nan	12.00	7.18	9649.75
max	34519.00	nan	nan	nan	1818.00	10.00	1013917.00

Summarizing and cleaning anime_ratings

In [ ]:
```
# taking a look at the dataset by sorting the data frame on anime_id

anime_ratings.sort_values(by='anime_id').head(10).style.set_table_styles([header
    .format(precision=2).set_caption("Let's take a look at the ratings dataset")
```

Out[ ]: **Let's take a look at the ratings dataset**

user_id	anime_id	rating
47485	1	8
10115	1	9
31698	1	10
52930	1	7
5315	1	7
41370	1	-1
68570	1	10
58344	1	10
36606	1	7
65723	1	7

In [ ]:
```
#information regarding the columns in the data frame

anime_ratings.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7813737 entries, 0 to 7813736
Data columns (total 3 columns):
 #   Column    Dtype
---  ------    -----
 0   user_id   int64
 1   anime_id  int64
 2   rating    int64
dtypes: int64(3)
memory usage: 178.8 MB
```

In [ ]:
```
#Checking for duplicate values and eliminating them

print(f"No.of duplicate values in anime_ratings: {anime_ratings[anime_ratings.du
anime_ratings.drop_duplicates(keep='first',inplace=True)
print(f"After removing duplicate values there are {anime_ratings.shape[0]} rows"
```

```
No.of duplicate values in anime_ratings: 1
After removing duplicate values there are 7813736 rows
```

In [ ]:
```
#Checking the range of ratings
anime_ratings['rating'].value_counts().to_frame('count').sort_index().T\
    .style.set_table_styles([headers,cells,caption])
```

Out[ ]:

	-1	1	2	3	4	5	6	7	8	9	10
count	1476496	16649	23150	41453	104291	282806	637775	1375287	1646018	1254096	95

In [ ]:
```
#Converting -1 to NaN. -1 is assigned to those anime which users have watched bu
anime_ratings['rating'].replace(to_replace = -1 , value = np.nan ,inplace=True)

#Checking for null values and eliminating them
display(anime_ratings.isna().sum().to_frame('count').sort_values(by='count',asce
```

```
    .style.set_table_styles([headers,cells,caption]).set_caption("No. of null va

anime_ratings.dropna(inplace = True)
display(anime_ratings.isna().sum().to_frame('count').sort_values(by='count',asce
    .style.set_table_styles([headers,cells,caption]).set_caption("No. of null va
```

No. of null values

	rating	user_id	anime_id
count	1476496	0	0

No. of null values after dropping them

	user_id	anime_id	rating
count	0	0	0

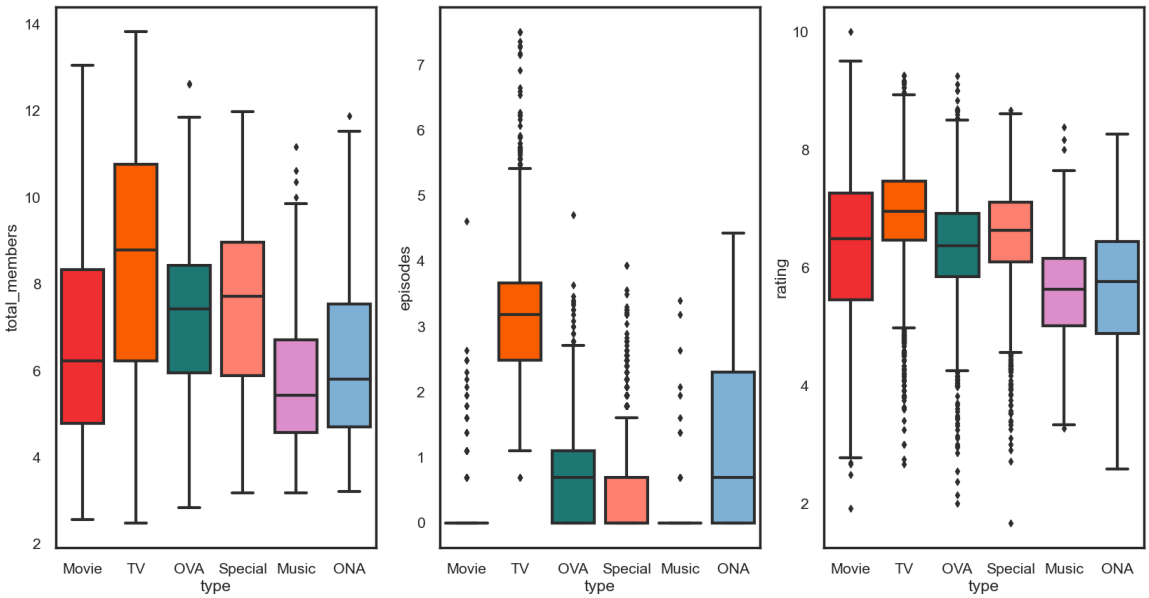
In [ ]:
```
#Statistical glimpse of anime_ratings data
anime_ratings.describe(include='all').style.set_table_styles([headers,cells,capt
```

Out[ ]: **Summary of anime data**

	user_id	anime_id	rating
count	6337240.00	6337240.00	6337240.00
mean	36747.91	8902.87	7.81
std	21013.40	8882.00	1.57
min	1.00	1.00	1.00
25%	18984.00	1239.00	7.00
50%	36815.00	6213.00	8.00
75%	54873.00	14075.00	9.00
max	73516.00	34475.00	10.00

In [ ]:
```
#Checking for outliers
fig, axes = plt.subplots(1, 3, figsize = (20, 10))
sns.boxplot(x=anime_details['type'],y=np.log(anime_details['total_members']),ax=
sns.boxplot(x=anime_details['type'],y=np.log(anime_details['episodes']),ax=axes[
sns.boxplot(x=anime_details['type'],y=anime_details['rating'],ax=axes[2],palette
```

Out[ ]:
```
<AxesSubplot:xlabel='type', ylabel='rating'>
```



In [ ]:
```
#Verifying if the outliers are actually that or if we can use the data without e
anime_details.query("(type == 'Movie' or type == 'Music') & episodes>1").sort_va
    .style.set_table_styles([headers,cells,caption]).set_caption("Checking the p
```

Out[ ]:

Checking the possible outliers

anime_id	name	genre	type	episodes	rating	total_members
33108	Anime Douyou	Kids, Music	Music	30.000000	5.750000	38
32633	Nowisee	Music	Music	24.000000	7.640000	560
5016	Fluximation	Music	Music	14.000000	6.740000	2240
4705	Tengen Toppa Gurren Lagann: Parallel Works	Music	Music	8.000000	7.270000	22213
8348	Tengen Toppa Gurren Lagann: Parallel Works 2	Mecha, Music	Music	7.000000	7.090000	13361
14359	Vocaloid China Project Senden Animation	Fantasy, Music	Music	5.000000	6.190000	1582
1998	Amazing Nuts!	Adventure, Music, Police, Romance, Sci-Fi	Music	4.000000	6.740000	6650
3642	Shina Dark: Kuroki Tsuki no Ou to Souheki no Tsuki no Himegimi	Ecchi, Fantasy, Harem, Music	Music	4.000000	6.230000	3730
30903	Children Record	Music	Music	2.000000	7.060000	745
33911	Gakuen Handsome: Legend of Sexy	Comedy, Music	Music	2.000000	6.000000	363
33912	Gakuen Handsome: Haitoku no Lesson	Comedy, Music	Music	2.000000	5.790000	316
9650	Dream C Club Pure Songs Clips	Music	Music	2.000000	5.680000	358
33578	Hungry Zombie Francesca!!	Fantasy, Music	Music	2.000000	5.570000	242
29924	Goman-hiki	Kids	Movie	100.000000	7.000000	56
13817	Kamiusagi Rope	Comedy, Slice of Life	Movie	14.000000	6.000000	137
18755	Donyatsu	Comedy, Sci-Fi, Seinen	Movie	12.000000	6.270000	2168
13819	Kamiusagi Rope 2	Comedy, Slice of Life	Movie	12.000000	5.410000	100
30150	Kamiusagi Rope 3	Comedy, Slice of Life	Movie	12.000000	4.670000	104
31020	Norasco: Cinema Point Card-hen	Comedy, Slice of Life	Movie	10.000000	6.860000	57
8928	Visions of Frank: Short Films by Japan's Most Audacious Animators	Dementia	Movie	9.000000	6.220000	267
30617	Animegatari	Comedy	Movie	8.000000	5.360000	1308

anime_id	name	genre	type	episodes	rating	total_members
3508	Genius Party	Action, Dementia, Fantasy, Mecha, Music, Psychological, Romance, Sci-Fi	Movie	7.000000	7.390000	18612
23697	Kara no Kyoukai: Manner Movies	Action, Comedy	Movie	7.000000	6.490000	5367
7420	Byulbyul Iyagi	Drama, Psychological	Movie	6.000000	6.100000	235
9447	Byulbyul Iyagi 2	Drama, Psychological	Movie	6.000000	5.860000	122
6795	Genius Party Beyond	Dementia, Fantasy, Music, Sci-Fi	Movie	5.000000	7.390000	10660
8205	Norabbits' Minutes	Comedy, Fantasy, Kids	Movie	5.000000	6.130000	102
22675	Peeping Life: Gekijou Original-ban	Comedy, Slice of Life	Movie	5.000000	4.670000	161
23831	Mahou Shoujo Madoka★Magica Movie 3: Hangyaku no Monogatari - Magica Quartet x Nisioisin	Comedy	Movie	4.000000	6.520000	6946
7616	Michi	Drama	Movie	4.000000	8.000000	187
1689	Byousoku 5 Centimeter	Drama, Romance, Slice of Life	Movie	3.000000	8.100000	324035
1462	Memories	Drama, Horror, Psychological, Sci-Fi	Movie	3.000000	7.840000	38643
27539	Pikmin Short Movies	Fantasy, Kids	Movie	3.000000	7.270000	406
1951	Manie-Manie: Meikyuu Monogatari	Adventure, Fantasy, Horror, Sci-Fi, Supernatural	Movie	3.000000	7.040000	9568
6189	Baton	Adventure, Sci-Fi	Movie	3.000000	6.010000	1482
7809	3-tsu no Kumo	Dementia	Movie	3.000000	5.100000	918
32736	Pepsi Nex x 009 Re:Cyborg	Action, Comedy	Movie	3.000000	5.270000	135
21899	Gintama: Yorinuki Gintama-san on Theater 2D	Action, Comedy, Historical, Parody, Samurai, Sci-Fi, Shounen	Movie	2.000000	8.600000	11104

anime_id	name	genre	type	episodes	rating	total_members
1911	Top wo Nerae! & Top wo Nerae 2! Gattai Movie!!	Comedy, Mecha, Shounen	Movie	2.000000	7.570000	8079
2962	Digimon Adventure 02 Movies	Adventure, Fantasy, Kids, Sci-Fi	Movie	2.000000	7.230000	26543
2611	Panda Kopanda	Comedy, Kids	Movie	2.000000	6.910000	4922
7626	Umi no Triton (1979)	Adventure, Fantasy, Shounen	Movie	2.000000	6.520000	235
31923	Mini Hama: Minimum Hamatora Movies	Comedy, Mystery, School, Super Power	Movie	2.000000	6.350000	833
9087	Mobile Suit SD Gundam Musha, Knight, Commando	Action, Comedy, Fantasy, Mecha, Parody	Movie	2.000000	6.020000	860
32397	Sagaken wo Meguru Animation	Slice of Life	Movie	2.000000	6.240000	535

Merging the datasets

In [ ]:
```
# merging anime and ratings data frame on 'anime_id' and giving a suffix to colu
anime_merged = pd.merge(anime_details,anime_ratings,on="anime_id",suffixes= [Nor
anime_merged = anime_merged.rename(columns={"rating_user": "user_rating"}) #renc

anime_merged.head().style.set_table_styles([headers,cells,caption]).set_caption(
```

Out[ ]: **Let's take a look at the merged Dataset**

	anime_id	name	genre	type	episodes	rating	total_members	user_id	user_ratin
0	32281	Kimi no Na wa.	Drama, Romance, School, Supernatural	Movie	1.000000	9.370000	200630	99	5.00000
1	32281	Kimi no Na wa.	Drama, Romance, School, Supernatural	Movie	1.000000	9.370000	200630	152	10.00000
2	32281	Kimi no Na wa.	Drama, Romance, School, Supernatural	Movie	1.000000	9.370000	200630	244	10.00000
3	32281	Kimi no Na wa.	Drama, Romance, School, Supernatural	Movie	1.000000	9.370000	200630	271	10.00000
4	32281	Kimi no Na wa.	Drama, Romance, School, Supernatural	Movie	1.000000	9.370000	200630	322	10.00000

In [ ]:
```
#getting the shape of the new data frame and creating a copy of it
print(anime_merged.shape)
anime_merged_temp = anime_merged.copy()
```

```
(6337144, 9)
```

In [ ]:
```
#grouping the dataset by taking mean of the user_ratings and sum of the number o
anime_merged_temp = anime_merged_temp.groupby(by= ['anime_id','name','genre','ty
.rename(columns={'user_id':'users_count','user_rating':'average_rating'})\
.reset_index()

anime_merged_temp.head().head().head().style.set_table_styles([headers,cells,cap
```

Out[ ]:

	anime_id	name	genre	type	episodes	rating	total_members	users_count	a
0	1	Cowboy Bebop	Action, Adventure, Comedy, Drama, Sci-Fi, Space	TV	26.000000	8.820000	486824	13449	
1	5	Cowboy Bebop: Tengoku no Tobira	Action, Drama, Mystery, Sci-Fi, Space	Movie	1.000000	8.400000	137636	5790	
2	6	Trigun	Action, Comedy, Sci-Fi	TV	26.000000	8.320000	283069	9385	
3	7	Witch Hunter Robin	Action, Drama, Magic, Mystery, Police, Supernatural	TV	26.000000	7.360000	64905	2169	
4	8	Beet the Vandel Buster	Adventure, Fantasy, Shounen, Supernatural	TV	52.000000	7.060000	9848	308	

Data Visualization

In [ ]:
```
# creating a new data frame by sorting the values on 'total_members'
set1 = anime_merged_temp.sort_values(["total_members"],ascending=False)
set1.head().head().style.set_table_styles([headers,cells,caption])
```

Out[ ]:

	anime_id	name	genre	type	episodes	rating	total_members	users_c
1388	1535	Death Note	Mystery, Police, Psychological, Supernatural, Thriller	TV	37.000000	8.710000	1013917	34
7058	16498	Shingeki no Kyojin	Action, Drama, Fantasy, Shounen, Super Power	TV	25.000000	8.540000	896229	25
6322	11757	Sword Art Online	Action, Adventure, Fantasy, Game, Romance	TV	25.000000	7.830000	893100	26
3936	5114	Fullmetal Alchemist: Brotherhood	Action, Adventure, Drama, Fantasy, Magic, Military, Shounen	TV	64.000000	9.260000	793665	21
4567	6547	Angel Beats!	Action, Comedy, Drama, School, Supernatural	TV	13.000000	8.390000	717796	23

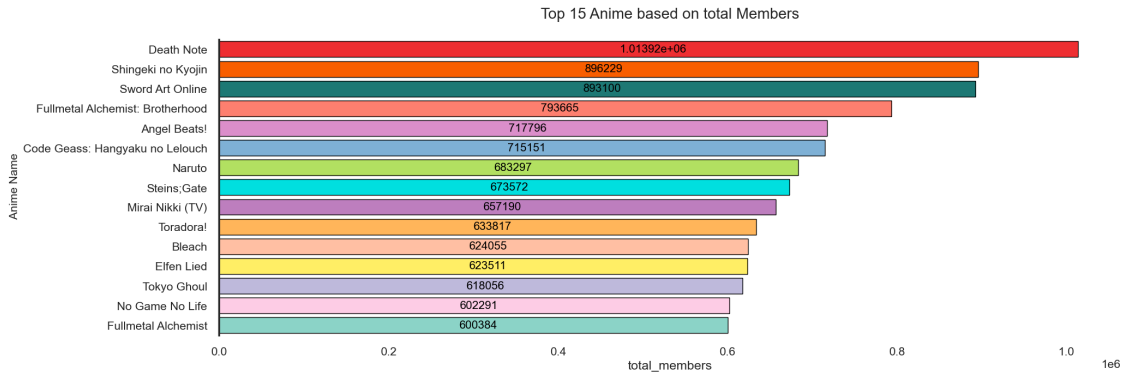
In [ ]:
```
# creating a new data frame by sorting the values on 'users_count' and 'average
set2 = anime_merged_temp.sort_values(["users_count", "average_rating"],ascending
set2.head().head().style.set_table_styles([headers,cells,caption])
```

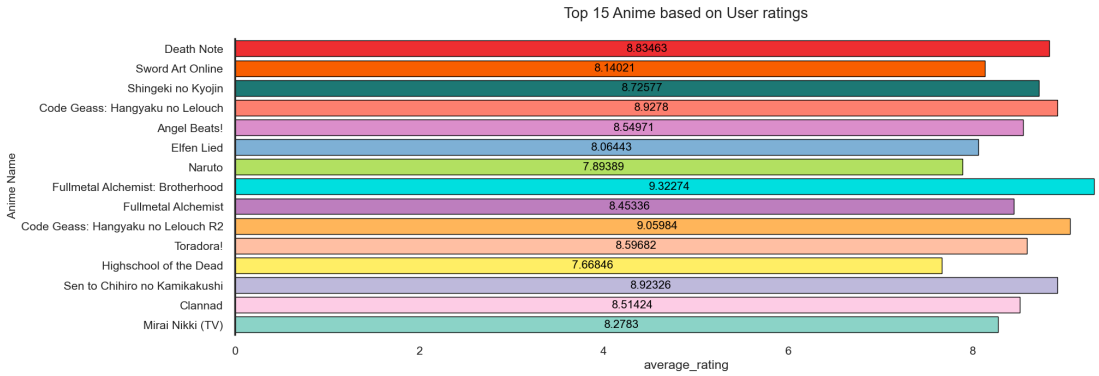
Out[ ]:

	anime_id	name	genre	type	episodes	rating	total_members	users_cou
1388	1535	Death Note	Mystery, Police, Psychological, Supernatural, Thriller	TV	37.000000	8.710000	1013917	3422
6322	11757	Sword Art Online	Action, Adventure, Fantasy, Game, Romance	TV	25.000000	7.830000	893100	2631
7058	16498	Shingeki no Kyojin	Action, Drama, Fantasy, Shounen, Super Power	TV	25.000000	8.540000	896229	2528
1426	1575	Code Geass: Hangyaku no Lelouch	Action, Mecha, Military, School, Sci-Fi, Super Power	TV	25.000000	8.830000	715151	2412
4567	6547	Angel Beats!	Action, Comedy, Drama, School, Supernatural	TV	13.000000	8.390000	717796	2356

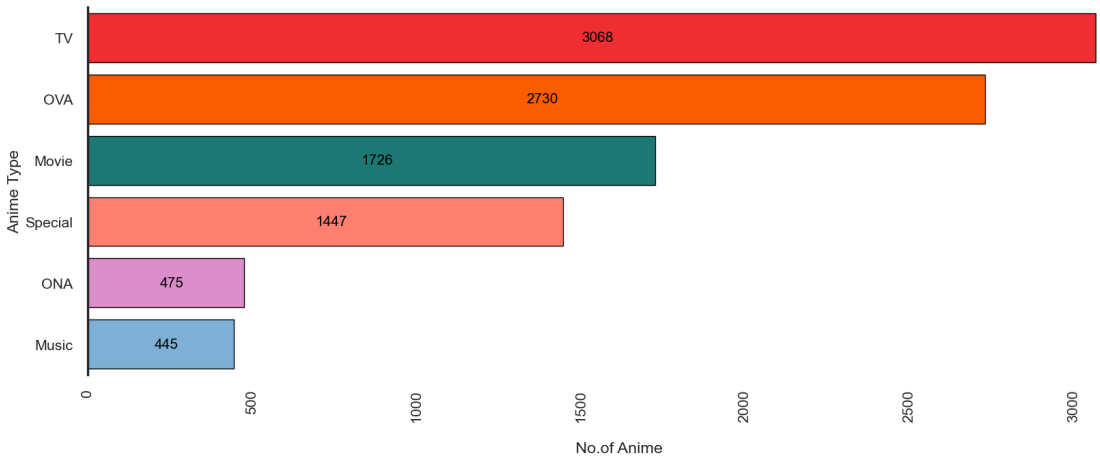
In [ ]:

```
_, axs = plt.subplots(2,1,figsize=(20,16),sharex=False,sharey=False)
plt.tight_layout(pad=6.0) #setting the figure size and properties of subplots

plot_1 = sns.barplot(x=set1["total_members"], y=set1["name"][:15], ax=axs[0], pa
axs[0].set_ylabel("\n Anime Name", fontsize = 15)
axs[0].set_title("\nTop 15 Anime based on total Members\n",fontsize = 20)
sns.despine(bottom=True) #creating a bar plot of top 15 Anime based on total_mem

for container in plot_1.containers: #creating bar labels
    plot_1.bar_label(container,label_type = "center",padding = 6,size = 15,color

plot_2 = sns.barplot(x=set2["average_rating"], y=set2["name"][:15], ax=axs[1], p
axs[1].set_ylabel("\n Anime Name", fontsize = 15)
axs[1].set_title("\nTop 15 Anime based on User ratings\n",fontsize = 20)
sns.despine(bottom=True) #creating a bar plot of top 15 Anime based on user_rati

for container in plot_2.containers: #creating bar labels
    plot_2.bar_label(container,label_type = "center",padding = 6,size = 15,color

plt.show()
```



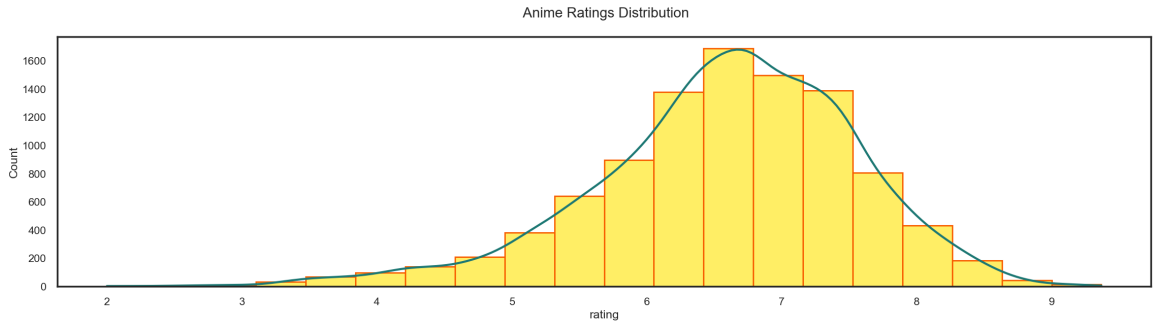


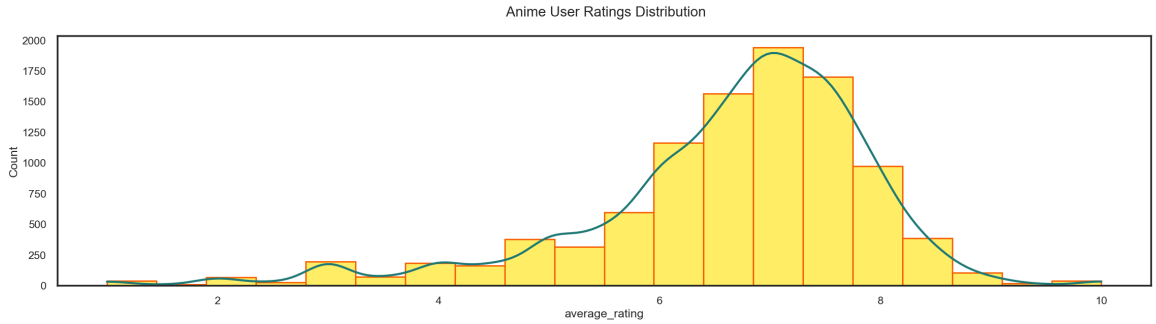
In [ ]:
```
f, ax = plt.subplots(figsize=(20, 7))
plot_2 = sns.countplot(y=set1['type'],order = set1["type"].value_counts().index,
plt.xticks(rotation = 90) #creating a bar plot of anime by type

plt.xlabel("\n No.of Anime")
plt.ylabel("Anime Type")
sns.despine(bottom=True)

for container in plot_2.containers: #creating bar labels
    plot_2.bar_label(container,label_type = "center",padding = 6,size = 15,color
plt.show()
```



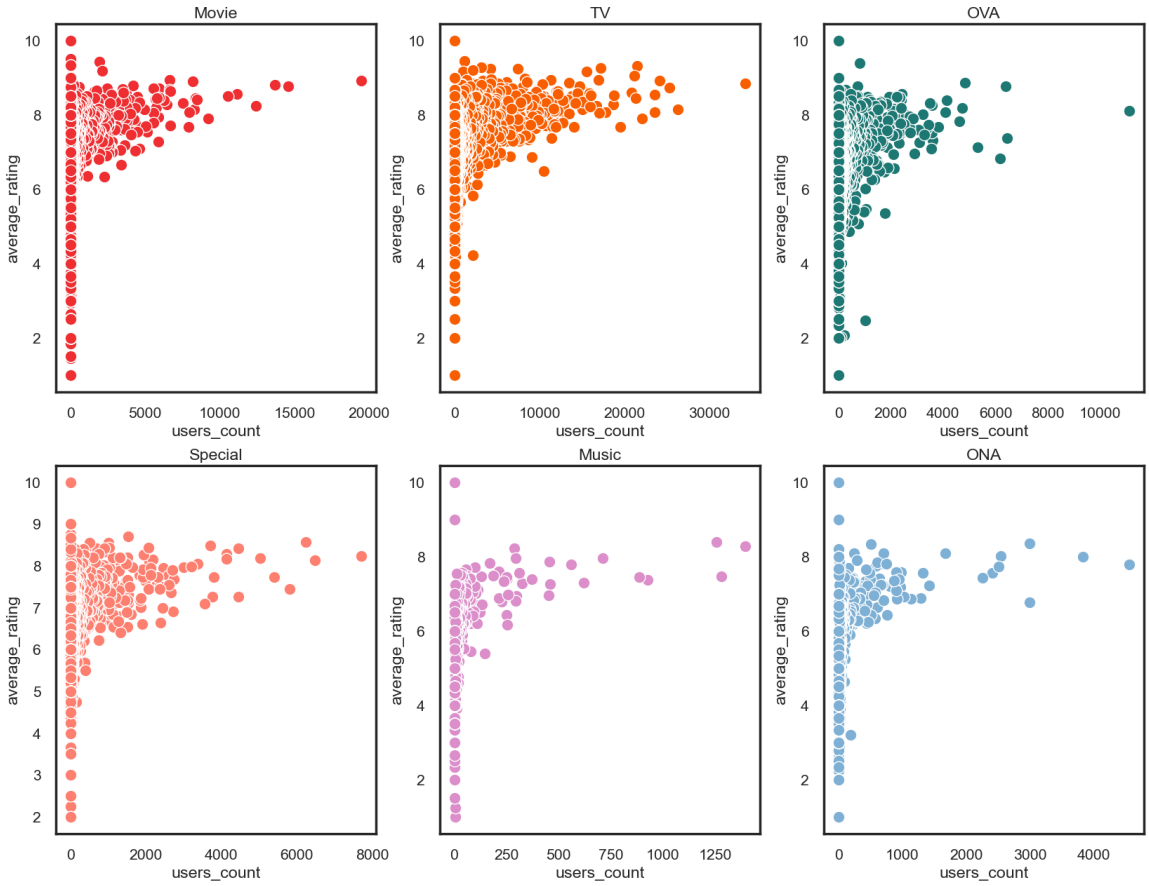
In [ ]:
```
_, axs = plt.subplots(2,1,figsize=(25,15),sharex=False,sharey=False)
plt.tight_layout(pad=6.0) #setting the figure size and properties of subplots

sns.histplot(set2["rating"],color=colors[11],kde=True,ax=axs[0],bins=20,alpha=1,
axs[0].lines[0].set_color(colors[2])
axs[0].set_title("\n Anime Ratings Distribution\n",fontsize = 20) #creating a hi

sns.histplot(set2["average_rating"],color=colors[11],kde=True,ax=axs[1],bins=20,
```

```
axs[1].lines[0].set_color(colors[2])
axs[1].set_title("\n Anime User Ratings Distribution\n",fontsize = 20); #creatir
```

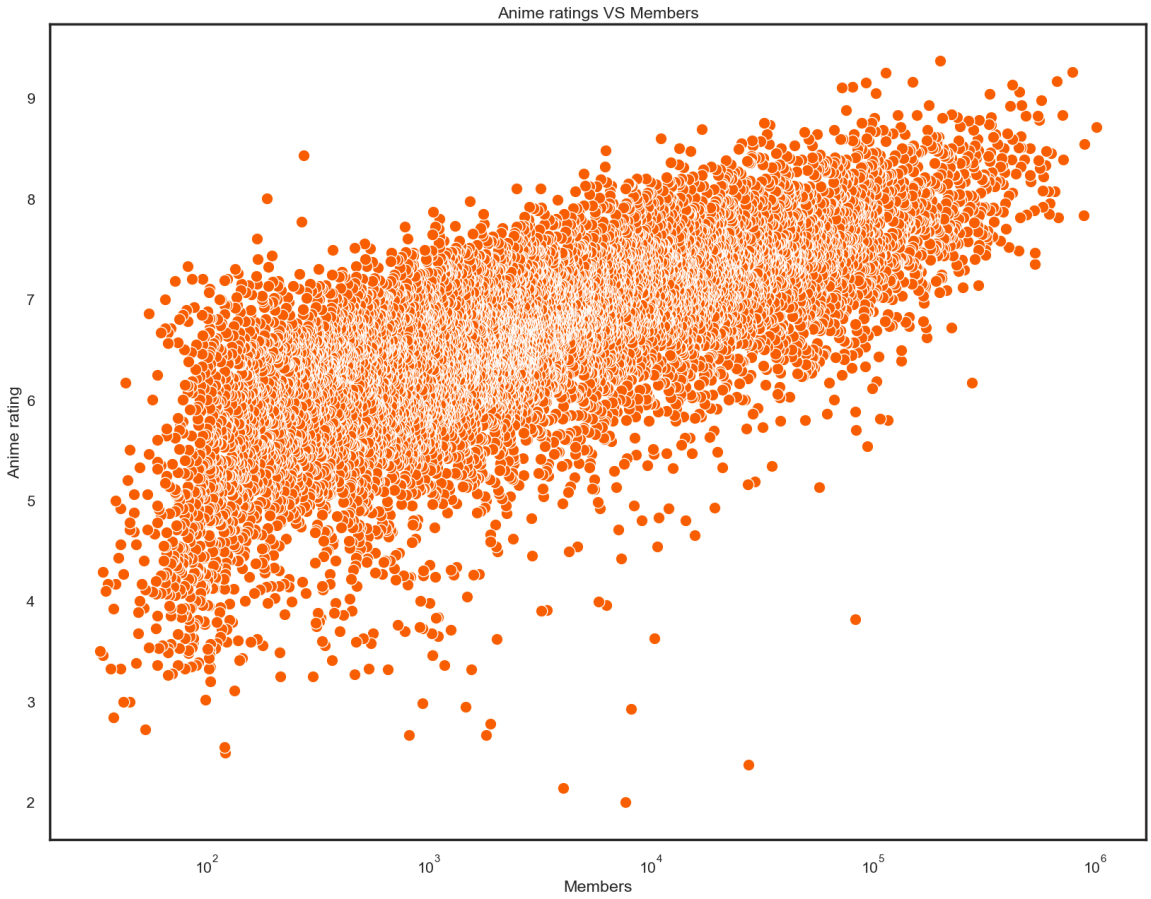




In [ ]:
```
plt.figure(figsize=(20,15))

anime_type=anime_details['type'].unique() #creating a variable that contains all

for x,y in enumerate(anime_type):
    plt.subplot(2,3,x+1)
    types= set1[set1['type']==y]
    sns.scatterplot(x=types['users_count'],y=types['average_rating'],color=color
    plt.title(f'{y}') #plotting subplot on average_rating vs users_count for dif
```



In [ ]:
```
#plotting a scatter plot between total_members and rating of the anime
plt.figure(figsize=(20,15))
sns.scatterplot(x=set1['total_members'],y=set1['rating'],color=colors[1])
ax=plt.gca(projection='polar')
ax.set_xscale('log')
plt.title('Anime ratings VS Members')
plt.xlabel('Members')
plt.ylabel('Anime rating')
plt.show()
```



Final Preprocessing

In [ ]:
```
set3 = anime_merged_temp.copy()

#using labelencoder to conver the categorical variable 'type' to numerical varia
le = LabelEncoder()

type_label = le.fit_transform(anime_merged_temp['type']) #
type_mappings = {index: label for index, label in enumerate(le.classes_)}

print(type_mappings)

set3['type'] = type_label
```

```
{0: 'Movie', 1: 'Music', 2: 'ONA', 3: 'OVA', 4: 'Special', 5: 'TV'}
```

In [ ]:
```
set3.head().style.set_table_styles([headers,cells,caption]).set_caption("Let's t
```

Out[ ]: **Let's take a look at the dataframe with encoded values**

	anime_id	name	genre	type	episodes	rating	total_members	users_count	av
0	1	Cowboy Bebop	Action, Adventure, Comedy, Drama, Sci-Fi, Space	5	26.000000	8.820000	486824	13449	
1	5	Cowboy Bebop: Tengoku no Tobira	Action, Drama, Mystery, Sci-Fi, Space	0	1.000000	8.400000	137636	5790	
2	6	Trigun	Action, Comedy, Sci-Fi	5	26.000000	8.320000	283069	9385	
3	7	Witch Hunter Robin	Action, Drama, Magic, Mystery, Police, Supernatural	5	26.000000	7.360000	64905	2169	
4	8	Beet the Vandel Buster	Adventure, Fantasy, Shounen, Supernatural	5	52.000000	7.060000	9848	308	

In [ ]:
```
#Onehot Encoding the genre column using get_dummies
genre_temp = set3['genre'].str.get_dummies(sep=', ') #using (sep=', ') since the
set3 = pd.concat([set3, genre_temp], axis = 1)
set3 = set3.drop(["rating","genre"],axis=1)
```

In [ ]:
```
set3['rating_bracket'] = round(set3['average_rating']) #rouding off average_rati
```

In [ ]:
```
#Eliminating anime where <30 users have rated the anime
print(f'shape of set3 before removing Anime with less that 30 user ratings: {set
set3 = set3.loc[(set3['users_count'] >= 30)]
print(f'shape of set3 after removing Anime with less that 30 user ratings: {set3
set3.head().style.set_table_styles([headers,cells,caption]).set_caption("Let's t
```

```
shape of set3 before removing Anime with less that 30 user ratings: (9891, 51)
shape of set3 after removing Anime with less that 30 user ratings: (5942, 51)
```

Out[ ]: **Let's take a look at the final dataframe for model training**

	anime_id	name	type	episodes	total_members	users_count	average_rating	Action	A
0	1	Cowboy Bebop	5	26.000000	486824	13449	8.869433	1	
1	5	Cowboy Bebop: Tengoku no Tobira	0	1.000000	137636	5790	8.439724	1	
2	6	Trigun	5	26.000000	283069	9385	8.419393	1	
3	7	Witch Hunter Robin	5	26.000000	64905	2169	7.533426	1	
4	8	Beet the Vandel	5	52.000000	9848	308	7.198052	0	

Treatinig the dataset as Classification problem

In [ ]:
```
#importing the required libraries for moedl fitting and evaluation
from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_squared_error
from sklearn.metrics import accuracy_score
from sklearn.metrics import plot_confusion_matrix
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
from sklearn.model_selection import ShuffleSplit,cross_val_score
from sklearn.model_selection import cross_val_score,RepeatedKFold
from sklearn.pipeline import Pipeline
```

In [ ]:
```
# Applying scaler() to all the columns except the 'dummy' variables
num_vars = ['type', 'episodes', 'total_members', 'users_count']
set3[num_vars] = scaler.fit_transform(set3[num_vars])
set3.head().style.set_table_styles([headers,cells,caption])
```

Out[ ]:

	anime_id	name	type	episodes	total_members	users_count	average_rating	Action
0	1	Cowboy Bebop	1.000000	0.013998	0.479805	0.392414	8.869433	1
1	5	Cowboy Bebop: Tengoku no Tobira	0.000000	0.000000	0.135187	0.168441	8.439724	1
2	6	Trigun	1.000000	0.013998	0.278717	0.273570	8.419393	1
3	7	Witch Hunter Robin	1.000000	0.013998	0.063408	0.062551	7.533426	1
4	8	Beet the Vandel Buster	1.000000	0.028555	0.009072	0.008130	7.198052	0

In [ ]:
```
#creating features and labels variables
c_features = set3.drop(['name','rating_bracket','average_rating','anime_id'],axi
c_labels = set3['rating_bracket']

# splitting the data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(c_features, c_labels, test_s
```

In [ ]:
```
#checking for the best parameters for Random forest classifier
# for our data using GridsearchCV method
rfc = RandomForestClassifier()
parameters = {
'n_estimators': [10,50,100,150],
'max_depth': [10,20,None]
}
rfc_cv = GridSearchCV(rfc, parameters, cv=5)
rfc_cv.fit(X_train, y_train.values.ravel())

print(rfc_cv.best_params_)
```

```
{'max_depth': 20, 'n_estimators': 150}
```

In [ ]:
```
#fitting our data using randomforestclassifier model
rfc_model = RandomForestClassifier(n_estimators=150, max_depth=20,random_state=1
rfc_model.fit(X_train, y_train)
rfc_predicted = rfc_model.predict(X_test)
rfc_score = accuracy_score(y_test,rfc_predicted)
print(rfc_score)
```

```
0.6417157275021026
```

In [ ]:
```
#checking for the features with major contribution
cols = ['name', 'importance']
lst1 = []
lst2 = []
for name, importance in zip(c_features.columns, rfc_model.feature_importances_):
    lst1.append([name,importance])
```

```
imp_df = pd.DataFrame(lst1,columns=cols)
imp_df.sort_values(by='importance',ascending=False).head()
```

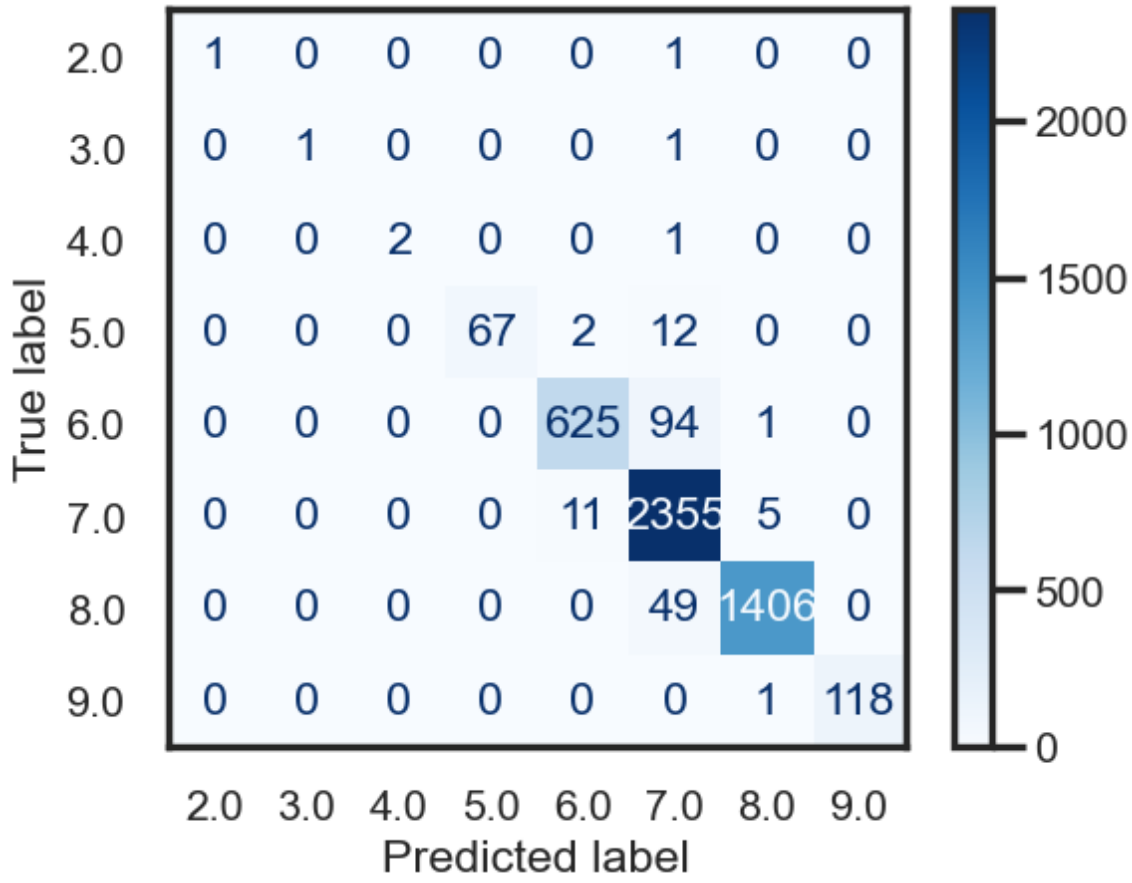
Out[ ]:

	name	importance
2	total_members	0.230905
3	users_count	0.198882
1	episodes	0.084778
0	type	0.049959
7	Comedy	0.022458

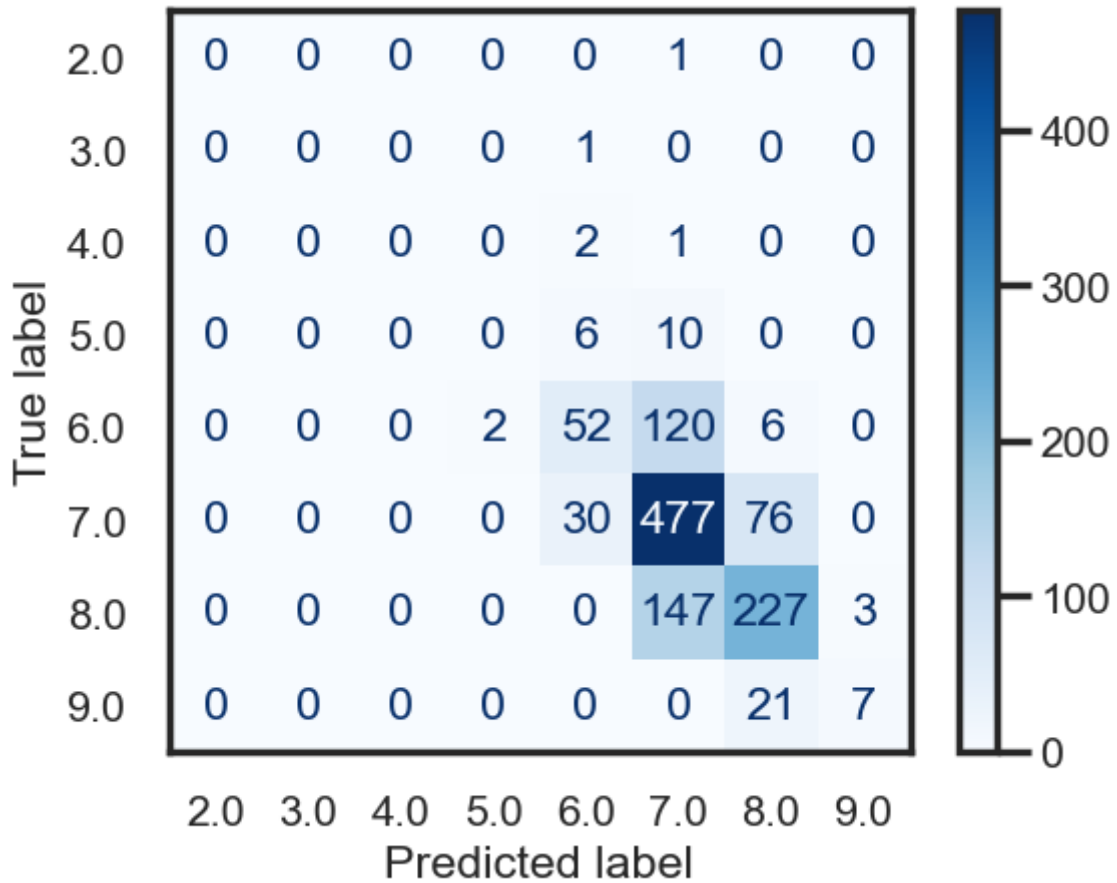
In [ ]:
```
#cross validating to check for variance
pipe = Pipeline([
    ('model', RandomForestClassifier(n_estimators=100, max_depth=20,random_state
])

result_kf = cross_val_score(estimator=pipe, X=X_train, y=y_train, scoring='accur
display(result_kf)
print(result_kf.std())
```

```
array([0.64984227, 0.64984227, 0.64353312, 0.64210526, 0.64842105,
       0.66456362, 0.66351209, 0.60778128, 0.65263158, 0.63578947,
       0.63932702, 0.62881178, 0.65299685, 0.62842105, 0.66315789,
       0.65404837, 0.64773922, 0.66351209, 0.62631579, 0.64421053,
       0.65299685, 0.63617245, 0.64773922, 0.62947368, 0.64631579])
0.013354081567894563
```

In [ ]:
```
#plotting confusion matrix for training data
plot_confusion_matrix(rfc_model, X_train, y_train, cmap=plt.cm.Blues)
plt.show()
```



In [ ]:
```
#plotting confusion matrix for testing data
plot_confusion_matrix(rfc_model, X_test, y_test, cmap=plt.cm.Blues)
plt.show()
```



Treatinig the dataset as Regression problem

In [ ]:
```
#creating features and labels variables

r_features = set3.drop(['name','rating_bracket','average_rating','anime_id'],axi
r_labels = set3['average_rating']

# splitting the data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(r_features, r_labels, test_s
```

In [ ]:
```
#checking for the best parameters for Random forest regressor
# for our data using GridsearchCV method
rfr = RandomForestRegressor()
parameters = {
'n_estimators': [50,100,150,200,250,300,350,400],
'max_depth': [10,20,None]
}
rfr_cv = GridSearchCV(rfr, parameters, cv=5)
rfr_cv.fit(X_train, y_train.values.ravel())

print(rfr_cv.best_params_)
```

```
{'max_depth': 20, 'n_estimators': 200}
```

In [ ]:
```
#fitting our data using RandomForestRegressor model

rfr_model = RandomForestRegressor(n_estimators=300, max_depth=20, random_state=4
rfr_model.fit(X_train, y_train)
rf_predicted_values = rfr_model.predict(X_test)
mse = mean_squared_error(y_test, rf_predicted_values)
rmse = mse**.5
print(mse)
print(rmse)
```

```
0.28963192095231555
0.5381746193869751
```

In [ ]:
```
#checking for the features with major contribution
cols = ['name', 'importance']
lst1 = []
lst2 = []
for name, importance in zip(r_features.columns, rfr_model.feature_importances_):
    lst1.append([name,importance])

imp_df = pd.DataFrame(lst1,columns=cols)
imp_df.sort_values(by='importance',ascending=False).head()
```

Out[ ]:

	name	importance
2	total_members	0.494963
3	users_count	0.103227
1	episodes	0.075584
0	type	0.049741
11	Ecchi	0.028615

In [ ]:
```
#using monte carlo cross validation
shuffle_split=ShuffleSplit(test_size=0.3,train_size=0.5,n_splits=10)
scores=cross_val_score(estimator=rfr_model, X=X_train, y=y_train,cv=shuffle_spli
print(scores)
print(scores.std())
```

```
[0.48674726 0.50914306 0.48265814 0.48346153 0.49328578 0.48216708
 0.46653209 0.48278743 0.50333745 0.51050214]
0.0132166312054912
```

In [ ]:
```
#comparing the predicted values with test values
data = {'Test': y_test,'Predicted':rf_predicted_values}

df = pd.DataFrame(data)
df.head()
```

Out[ ]:

	Test	Predicted
7196	7.257261	7.235997
2394	6.123288	6.427557
1465	7.413570	7.248583
7953	7.394531	7.056332
3654	7.401709	7.056382