Name: Rajesh Bisht (Batch – 8)

Email Id : rbisht.india@gmail.com

Git: https://github.com/RajeshBisht28/UnsupervisedAssignments

a) Explain the Eigenvalue and eigenvector in detail along with some examples. Its role in PCA.

Eigenvalue:

Eigenvalues can be any number, imaginary, real, whole any number can be eigenvalue.

Eigenvalues are the special set of scalars associated with the system of linear equations. It is mostly used in matrix equations. ‘Eigen’ is a German word that means ‘proper’ or ‘characteristic’. Therefore, the term eigenvalue can be termed as characteristic value, characteristic root, proper values or latent roots as well. In simple words, the eigenvalue is a scalar that is used to transform the eigenvector. The basic equation is

$$Ax = \lambda x$$

The number or scalar value “**λ”** is an eigenvalue of A.

Eigen Vectors?

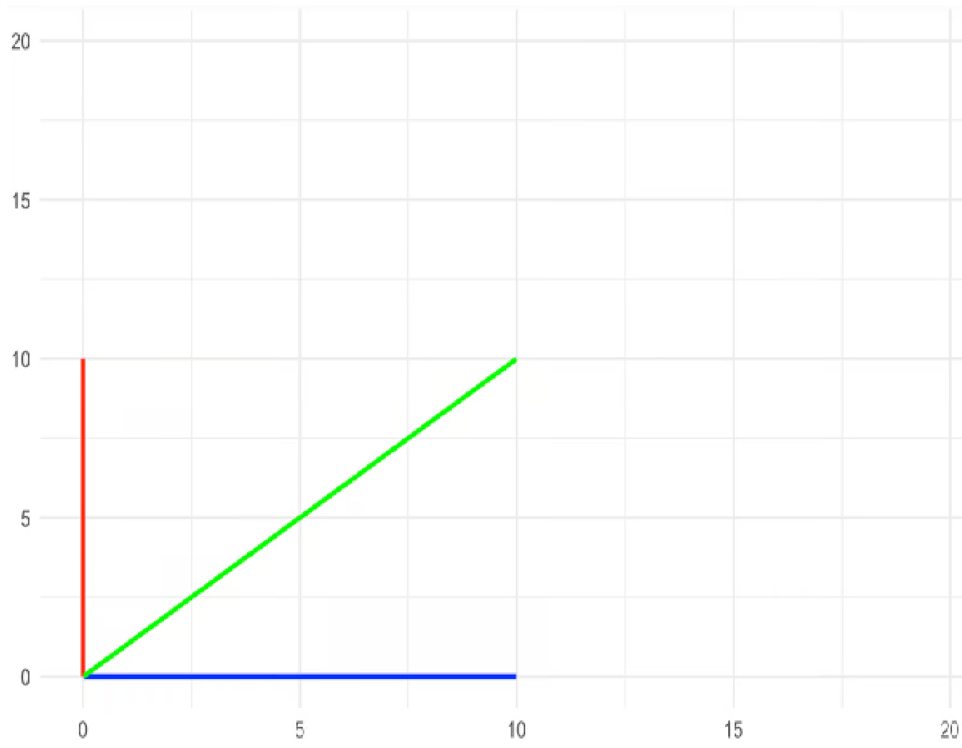
Eigenvectors are the vectors (non-zero) that do not change the direction when any linear transformation is applied. It changes by only a scalar factor. In a brief, we can say, if A is a linear transformation from a vector space V and **x** is a vector in V, which is not a zero vector, then v is an eigenvector of A if A(X) is a scalar multiple of **x**.

Note:

- There could be infinitely many Eigenvectors, corresponding to one eigenvalue.
- For distinct eigenvalues, the eigenvectors are linearly dependent.

Understanding Eigenvectors and Eigenvalues:

As see below Imagine a two-dimensional space, in which we have three vectors, a blue one that rests on the x-axis, a red one that rests on the y-axis, and a green one that rests diagonally between the two.



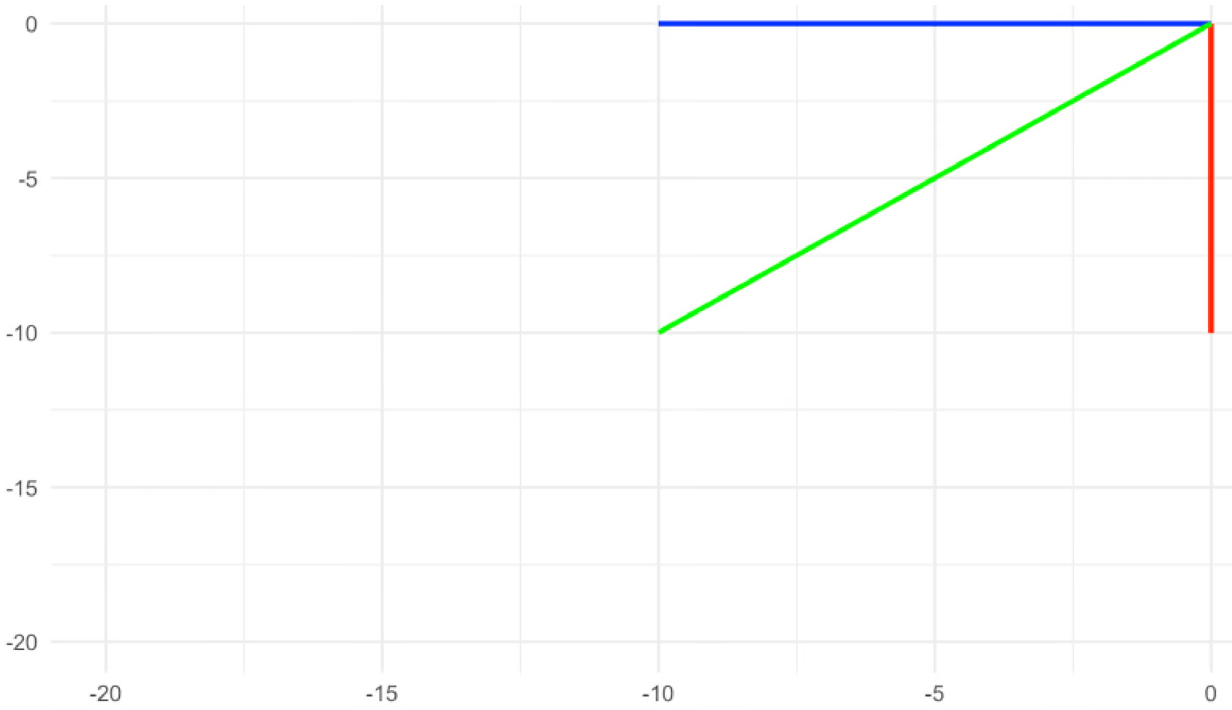
Lets see in below image, The red and blue ones did not change their direction. But Green change its direction.
But Green and Red both changes magnitude, But Blue not change magnitude.

The blue eigenvector did not change its magnitude, while the red one doubled.

But Green changes its magnitude and as well direction.

Properties of eigenvalues:

Now check below image, magnitude are same but direction of vectors, Red, Blue and Green.



We see that the sign of the number (the eigenvalue) can tell us additional information about the transformation that took place. A positive eigenvalue indicates that the transformation did not include a rotation, while a negative eigenvalue indicates that the transformation included a rotation of the eigenvector around its origin.

Eigenvalues of a 2x2 Matrix

Let us see the process of finding the eigenvalues of a 2x2 matrix with an example where we will find the eigenvalues of A = $\begin{bmatrix}5 & 4\\ 1 & 2\end{bmatrix}$. Let λ represents its eigenvalue(s). The identity matrix of order 2x2 is, I = $\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}$. Then

$\lambda I = \lambda \begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix} = \begin{bmatrix}\lambda & 0\\ 0 & \lambda\end{bmatrix}$

$A - \lambda I = \begin{bmatrix}5 & 4\\ 1 & 2\end{bmatrix} - \begin{bmatrix}\lambda & 0\\ 0 & \lambda\end{bmatrix}$

$= \begin{bmatrix}5-\lambda & 4\\ 1 & 2-\lambda\end{bmatrix}$

Its determinant is,

$|A - \lambda I| = (5 - \lambda)(2 - \lambda) - (1)(4)$

$= 10 - 5\lambda - 2\lambda + \lambda^2 - 4$

$= \lambda^2 - 7\lambda + 6$

The characteristic equation is,

$|A - \lambda I| = 0$

$\lambda 2 - 7\lambda + 6 = 0$

$(\lambda - 6)(\lambda - 1) = 0$

$\lambda - 6 = 0; \lambda - 1 = 0$

$\lambda = 6; \lambda = 1$

Thus, the eigenvalues of matrix A are 1 and 6.

Eigen value and Eigen vector use in PCA:

In Principal Component Analysis (PCA), eigenvalues and eigenvectors are used to identify the directions of maximum variance within a dataset, essentially defining the new axes of the transformed data where the most important information is captured, with the eigenvectors representing these directions and the corresponding eigenvalues indicating how much variance lies along each eigenvector; this allows for dimensionality reduction by selecting the principal components (eigenvectors with the largest eigenvalues) to represent the data in a lower-dimensional space while retaining most of the relevant information.

How it used in PCA:

1. **Calculate the covariance matrix:**

   Compute the covariance matrix of the standardized data.

2. **Find eigenvalues and eigenvectors:**

   Perform eigen decomposition on the covariance matrix to obtain the eigenvalues and eigenvectors.

3. **Sort by variance:**

   Sort the eigenvectors based on their corresponding eigenvalues in descending order.

4. **Select principal components:**

   Choose the top few eigenvectors (with the largest eigenvalues) to form the projection matrix.

5. **Transform data:**
   Project the original data onto the new subspace defined by the selected eigenvectors.

b) Use the Students' Social Network Profile Clustering dataset from below Kaggle link and create an end-to-end project on Jupyter/Colab. https://www.kaggle.com/datasets/zabihullah18/students-social-network-profile-clustering/data i. Download the dataset from above link and load it into your Python environment. ii. Perform the EDA and do the visualizations. iii. Check the distributions/skewness in the variables and do the transformations if required. iv. Check/Treat the outliers and do the feature scaling if required. v. Create a ML model to segment the students with similar interests, demographic profiling, and trend analysis over time. vi. Try out all the 3 clustering methods (K-Mean, Hierarchical, DBSCAN) and compare their silhoutte scores.

School student profile (Data Set) Overview:

Data set provides, 15,000 high school students who maintained profiles on a popular social networking platform during the period spanning 2006 to 2009. The data was collected uniformly over these years and provides valuable insights into the online activities and interests of adolescents during that time frame.
The dataset was obtained by crawling profiles from the social network and subsequently processed using text mining techniques to extract information about student interests. Specifically, the dataset includes counts of the 37 most dominant words found in the profiles, such as "football" and "shopping," indicating the prevalence of various topics among the student population.

In addition to interest-related data, the dataset also includes essential demographic information about each student, including their graduation year (gradyear), gender, age at the time of the survey, and the number of contacts or friends they had on the social network (NumberOffriends).

Load data set :

In [166…

```
import pandas as pd
import numpy as np
# Load dataset
df_original = pd.read_csv(r'Clustering_Marketing.csv')
df_original.head()
```

Out[166...

	gradyear	gender	age	NumberOffriends	basketball	football	soccer	softball	volleyball	swimming	...	blonde	mall
0	2007	NaN	NaN	0	0	0	0	0	0	0	...	0	0
1	2007	F	17.41	49	0	0	1	0	0	1	...	0	0
2	2007	F	17.511	41	0	0	0	0	0	0	...	0	1
3	2006	F	NaN	36	0	0	0	0	0	0	...	0	0
4	2008	F	16.657	1	0	0	0	0	0	1	...	0	0

5 rows × 40 columns

In [167...

```
print(f"Age total null : {df_original['age'].isnull().sum()}")
print(f"Gender total null : {df_original['gender'].isnull().sum()}")
```

```
Age total null : 2496
Gender total null : 1337
```

================== Exploratory Data Analysis (EDA) =======================

impute 'age' missing values with mean

In [169...

```
df_original['age'] = df_original['age'].replace(['unknown'], np.nan)
df_original['age'] = pd.to_numeric(df_original['age'], errors='coerce')  # Coerce invalid values to NaN
df_original['age'] = df_original['age'].fillna(df_original['age'].mean())
print(f"Age total null after mean replace : {df_original['age'].isnull().sum()}")
print(f"Gender total null : {df_original['gender'].isnull().sum()}")
```

```
Age total null after mean replace : 0
Gender total null : 1337
```

In [ ]:

```
```

Impute missing values of gender using RandomForestRegressor.

In [170…

```
import pandas as pd
from sklearn.ensemble import RandomForestRegressor
from sklearn.preprocessing import LabelEncoder
df_encode_gender = df_original.copy()
# Assuming df_original is already defined
# Step 1: Separate rows with and without missing values in 'gender'
train_data = df_original[df_original["gender"].notna()].copy()  # Explicitly make a copy
test_data = df_original[df_original["gender"].isna()].copy()    # Explicitly make a copy

print(f"Train data missing gender: {train_data['gender'].isna().sum()}")
print(f"Test data missing gender: {test_data['gender'].isna().sum()}")

# Encode 'gender' as numeric
label_encoder = LabelEncoder()
train_data['gender'] = label_encoder.fit_transform(train_data['gender'])

# Prepare training data
X_train = train_data.drop("gender", axis=1)
y_train = train_data["gender"]

# Fit the model
model = RandomForestRegressor(random_state=42)
model.fit(X_train, y_train)

# Prepare test data
X_test = test_data.drop("gender", axis=1)

# Predict missing genders
predicted_genders = model.predict(X_test)

# Convert predictions back to the original gender labels
predicted_genders_labels = label_encoder.inverse_transform(predicted_genders.round().astype(int))

# Step 4: Push predictions into the original dataset
df_encode_gender.loc[df_encode_gender["gender"].isna(), "gender"] = predicted_genders_labels

# Verify the updated dataset
print("Updated dataset with imputed 'gender':")
print(f"Gender Null values after impute: {df_encode_gender['gender'].isnull().sum()}")
```

```
Train data missing gender: 0
Test data missing gender: 1337
Updated dataset with imputed 'gender':
Gender Null values after impute: 0
```

In [171…

```python
gender_counts = df_encode_gender['gender'].value_counts()
print(f"Gender counts: {gender_counts}")
```

```
Gender counts: gender
F    12279
M     2721
Name: count, dtype: int64
```

Apply SMOTE for resampling: Due to the class imbalancing in gender feature - F=12279 and M=2721,

In [172…

```python
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from imblearn.over_sampling import SMOTE
df_smote = df_encode_gender.copy()

# Step 1: Label Encode the gender column
label_encoder = LabelEncoder()
df_smote['gender'] = label_encoder.fit_transform(df_encode_gender['gender'])  # F -> 0, M -> 1

# Step 2: Separate features (X) and target (y)
X = df_smote.drop('gender', axis=1)
y = df_smote['gender']

# Step 3: Apply SMOTE
smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X, y)

# Step 4: Convert back to DataFrame
df_resampled = pd.concat([pd.DataFrame(X_resampled, columns=X.columns),
                          pd.DataFrame(y_resampled, columns=['gender'])], axis=1)

# Decode the gender column back to original labels if needed
df_resampled['gender'] = label_encoder.inverse_transform(df_resampled['gender'])

## Label encode of gender
df_resampled['gender'] = label_encoder.fit_transform(df_resampled['gender'])  # F -> 0, M -> 1
```

```
# Display gender counts the balanced dataset
print(f"After Class balance for Gender : {df_resampled['gender'].value_counts()}")
```

```
After Class balance for Gender : gender
0    12279
1    12279
Name: count, dtype: int64
```

========================== Feature Engineering ============================

Remove gender and grapear feature which is not much important.

In [173…

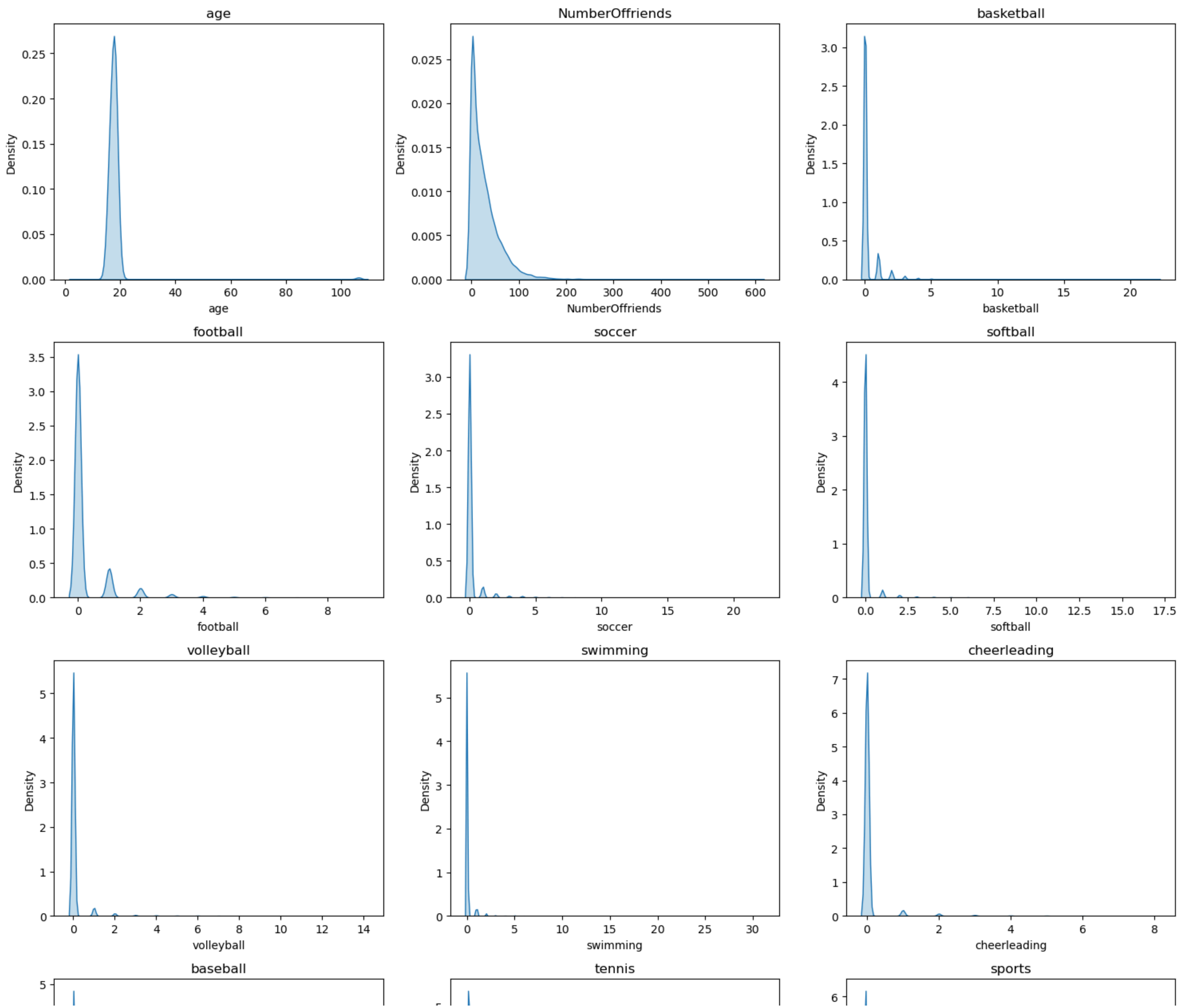
```
df_resampled = df_resampled.drop(columns=['gender','gradyear'])
```

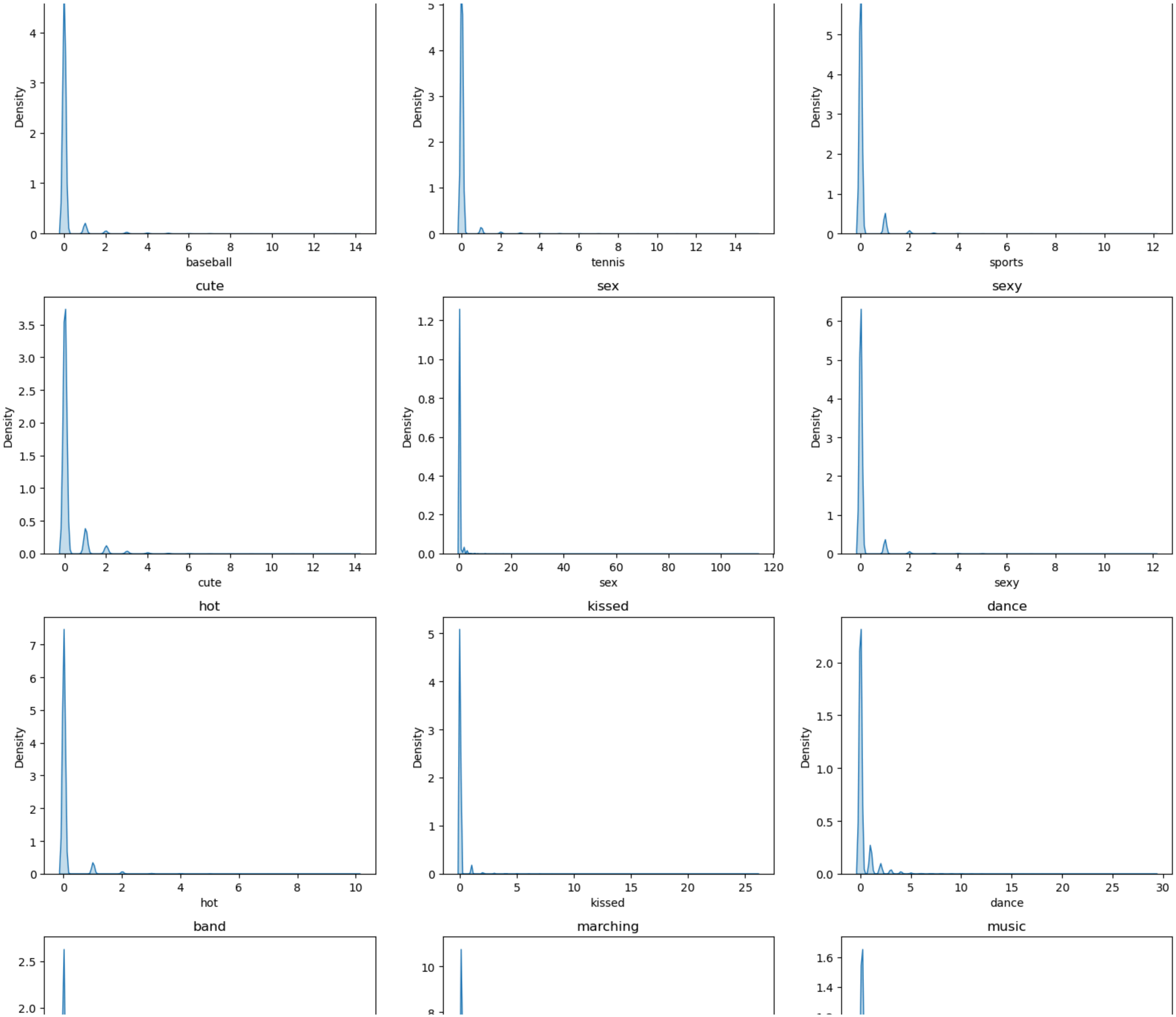
Check Skewness of Feature's data

In [174…

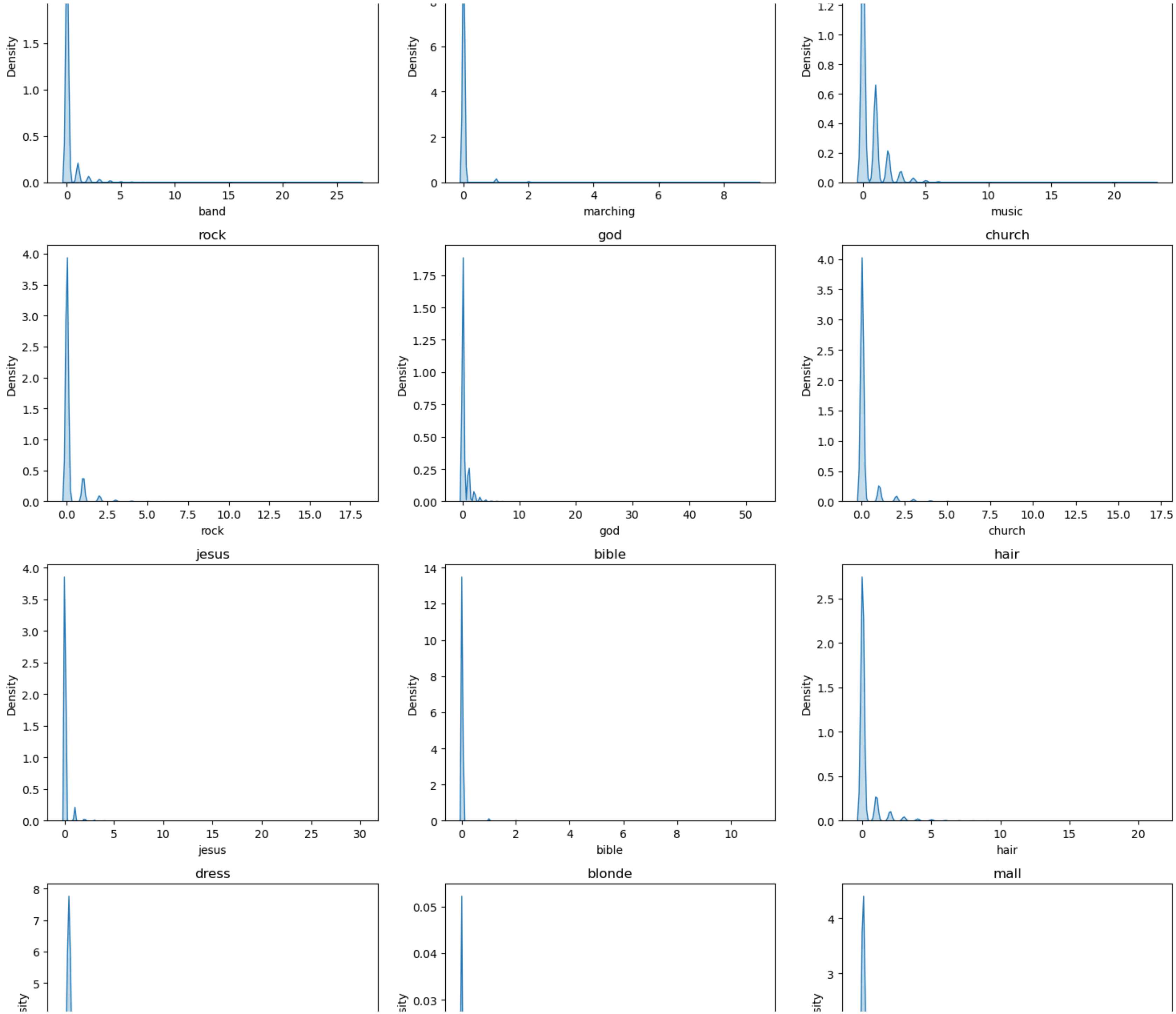
```
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

# Example: Create a sample DataFrame with 13 features
import numpy as np
# Set the number of rows and columns for the grid
n_features = len(df_resampled.columns)
n_cols = 3
n_rows = -(-n_features // n_cols)  # Ceiling division

# Create the figure and axes
fig, axes = plt.subplots(n_rows, n_cols, figsize=(15, n_rows * 4))
axes = axes.flatten()  # Flatten to easily iterate

# Plot each feature's KDE
for i, feature in enumerate(df_resampled.columns):
    sns.kdeplot(data=df_resampled[feature], ax=axes[i], fill=True)
    axes[i].set_title(feature)

# Hide any extra subplots
for j in range(i + 1, len(axes)):
    axes[j].axis('off')
```

```
# Adjust layout
plt.tight_layout()
plt.show()
```

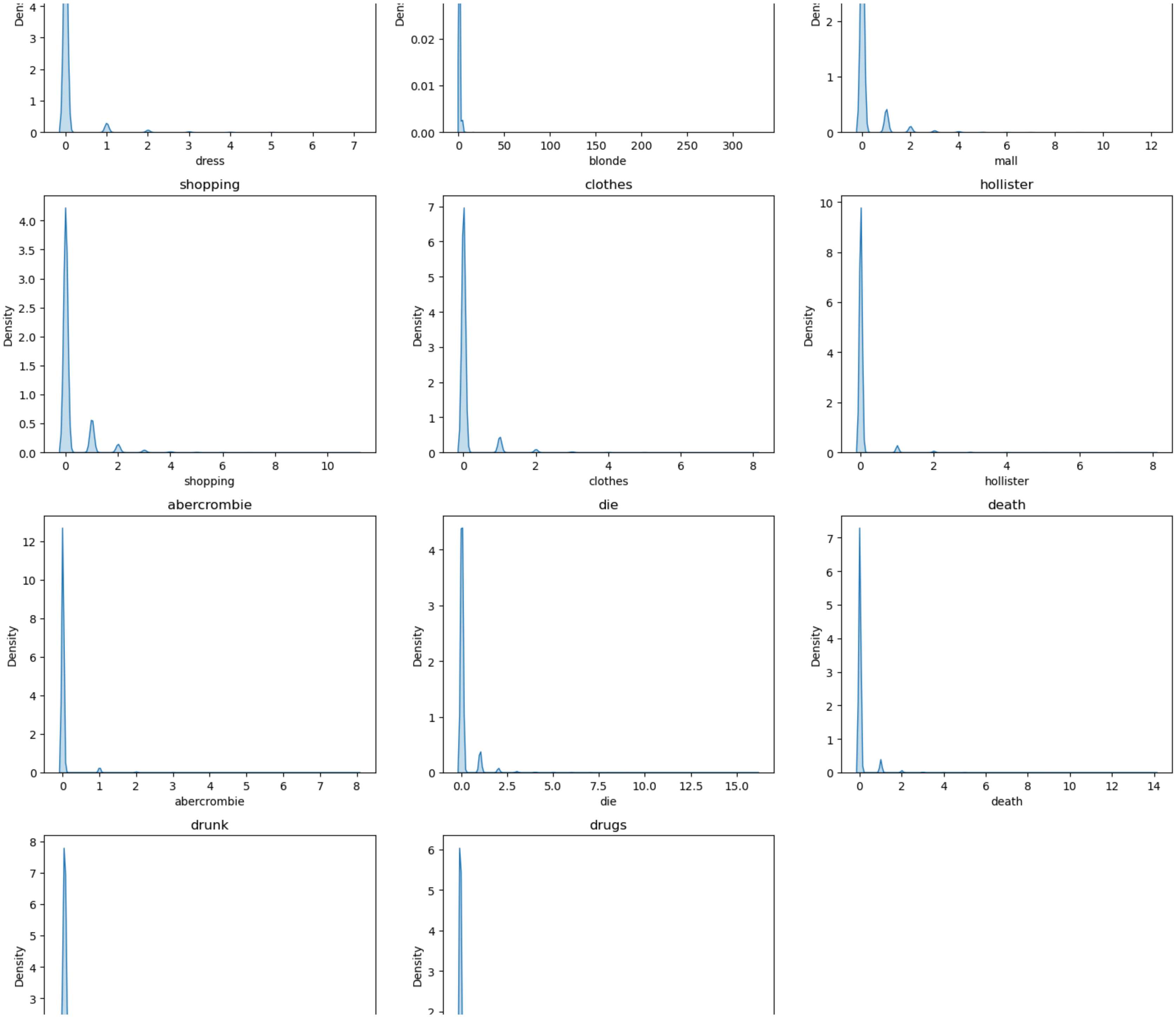

age
Density
age
NumberOffriends
Density
NumberOffriends
basketball
Density
basketball
football
Density
football
soccer
Density
soccer
softball
Density
softball
volleyball
Density
volleyball
swimming
Density
swimming
cheerleading
Density
cheerleading
baseball
tennis
sports

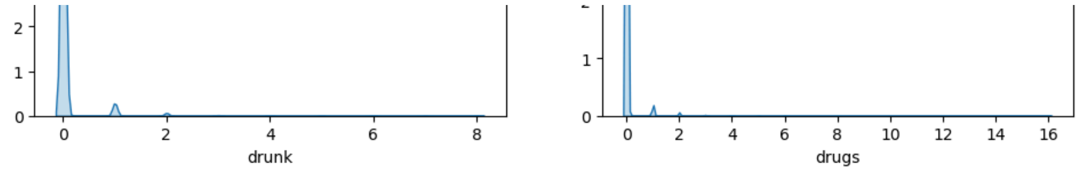


Density
baseball
tennis
sports
cute
sex
sexy
hot
kissed
dance
band
marching
music



Density
0.0 0.5 1.0 1.5
0 5 10 15 20 25
band
Density
0 2 4 6 8
0 2 4 6 8
marching
Density
0.0 0.2 0.4 0.6 0.8 1.0 1.2
0 5 10 15 20
music
rock
Density
0.0 0.5 1.0 1.5 2.0 2.5 3.0 3.5 4.0
0.0 2.5 5.0 7.5 10.0 12.5 15.0 17.5
rock
god
Density
0.00 0.25 0.50 0.75 1.00 1.25 1.50 1.75
0 10 20 30 40 50
god
church
Density
0.0 0.5 1.0 1.5 2.0 2.5 3.0 3.5 4.0
0.0 2.5 5.0 7.5 10.0 12.5 15.0 17.5
church
jesus
Density
0.0 0.5 1.0 1.5 2.0 2.5 3.0 3.5 4.0
0 5 10 15 20 25 30
jesus
bible
Density
0 2 4 6 8 10 12 14
0 2 4 6 8 10
bible
hair
Density
0.0 0.5 1.0 1.5 2.0 2.5
0 5 10 15 20
hair
dress
5 6 7 8
blonde
0.03 0.04 0.05
mall
3 4

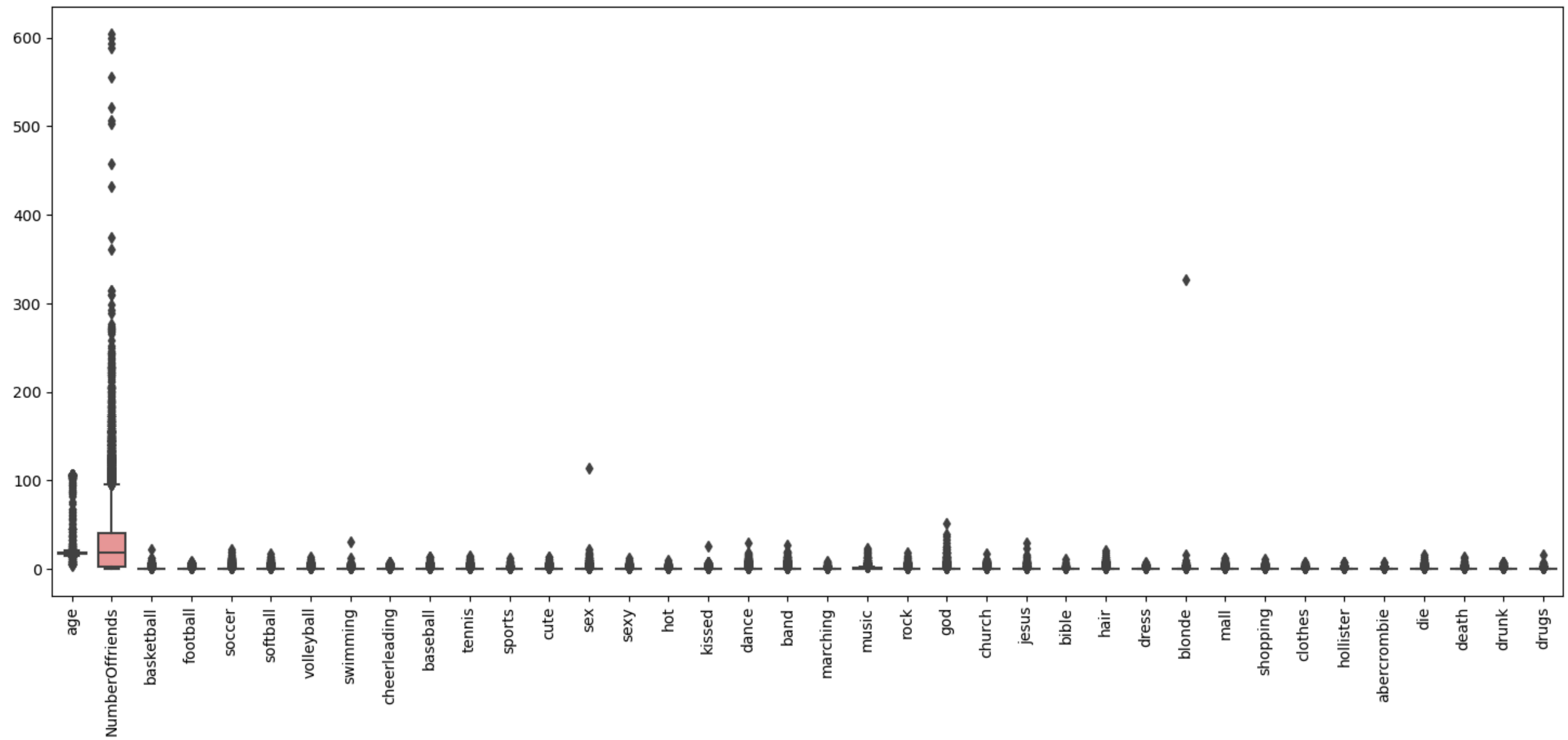

dress
blonde
mall
shopping
clothes
hollister
abercrombie
die
death
drunk
drugs
Density



Check outliers

In [175…

```
import seaborn as sns
import matplotlib.pyplot as plt

# Set figure size (increase width for better label visibility)
plt.figure(figsize=(18, 8))  # Adjust width and height to your preference

# Create boxplot with wider boxes for better spacing
sns.boxplot(data=df_resampled, width=0.7)  # Adjust width for better box size

# Rotate x-axis labels for better readability
plt.xticks(rotation=90, ha='center')  # Ensure label alignment and rotation

# Adjust space between labels and chart for clarity
plt.subplots_adjust(bottom=0.2)

plt.show()
```



In [ ]:

```

```

Skewness table for all features

In [87]:

```
import pandas as pd

# Assuming df is your DataFrame
# Calculate skewness for all columns
skewness = df_resampled.skew().reset_index()
skewness.columns = ['Column', 'Skewness']

# Create a styled DataFrame
styled_skewness = skewness.style.set_table_styles(
    [
        {'selector': 'thead th', 'props': [('background-color', '#3E4149'), ('color', 'white'), ('text-align', 'cente
        {'selector': 'tbody td', 'props': [('border', '1px solid black'), ('text-align', 'center')]}
```

```
    ]
).set_properties(**{'text-align': 'center'})

# Display the styled DataFrame
styled_skewness
```

Out[87]:

	Column	Skewness
0	age	11.307936
1	NumberOffriends	3.369088
2	basketball	5.946086
3	football	4.157755
4	soccer	8.292498
5	softball	10.074827
6	volleyball	9.477595
7	swimming	18.064531
8	cheerleading	8.902783
9	baseball	8.217805
10	tennis	12.160276
11	sports	6.354679
12	cute	5.304289
13	sex	58.935406
14	sexy	8.244574
15	hot	7.194804
16	kissed	15.393511
17	dance	6.736319
18	band	7.715261
19	marching	12.985644
20	music	4.481942
21	rock	6.961204

	Column	Skewness
22	god	12.605701
23	church	6.337984
24	jesus	19.908198
25	bible	24.320872
26	hair	5.644085
27	dress	7.098667
28	blonde	149.728607
29	mall	5.878145
30	shopping	3.884168
31	clothes	5.787493
32	hollister	9.583568
33	abercrombie	10.146850
34	die	8.010582
35	death	9.452787
36	drunk	9.029793
37	drugs	13.730353

Yeo-Johnson transformation

In [177…

```
import pandas as pd
from scipy.stats import yeojohnson
# Apply Yeo-Johnson transformation
transformed_data = {}
lambdas = {}  # Store the lambda values for each column
for column in df_resampled.columns:
    transformed_data[column], lambdas[column] = yeojohnson(df_resampled[column])
```

```
# Create a new DataFrame with the transformed data
transformed_df = pd.DataFrame(transformed_data)
# Shift all values to ensure positivity
shift_value = abs(transformed_df.min().min()) + 1
transformed_df += shift_value
```

Preserve data after Transformation :

In [178…

```
db_transform = transformed_df.copy()
```

Capping Outlier : Using IQR method

In [180…

```
import pandas as pd
import numpy as np
df_resampled = transformed_df.copy()
df_remove_outliers = df_resampled.copy()
columns = df_resampled.columns

# Cap the outliers at 1st and 99th percentile
for column in columns:
    if df_remove_outliers[column].dtype != 'object':  # Ignore categorical columns
        Q1 = df_remove_outliers[column].quantile(0.25)
        Q3 = df_remove_outliers[column].quantile(0.75)
        IQR = Q3 - Q1
        # Calculate the lower and upper bounds using 1.5 * IQR
        lower_bound = Q1 - 1.5 * IQR
        upper_bound = Q3 + 1.5 * IQR
        df_remove_outliers[column] = df_remove_outliers[column].clip(lower=lower_bound, upper=upper_bound)
```

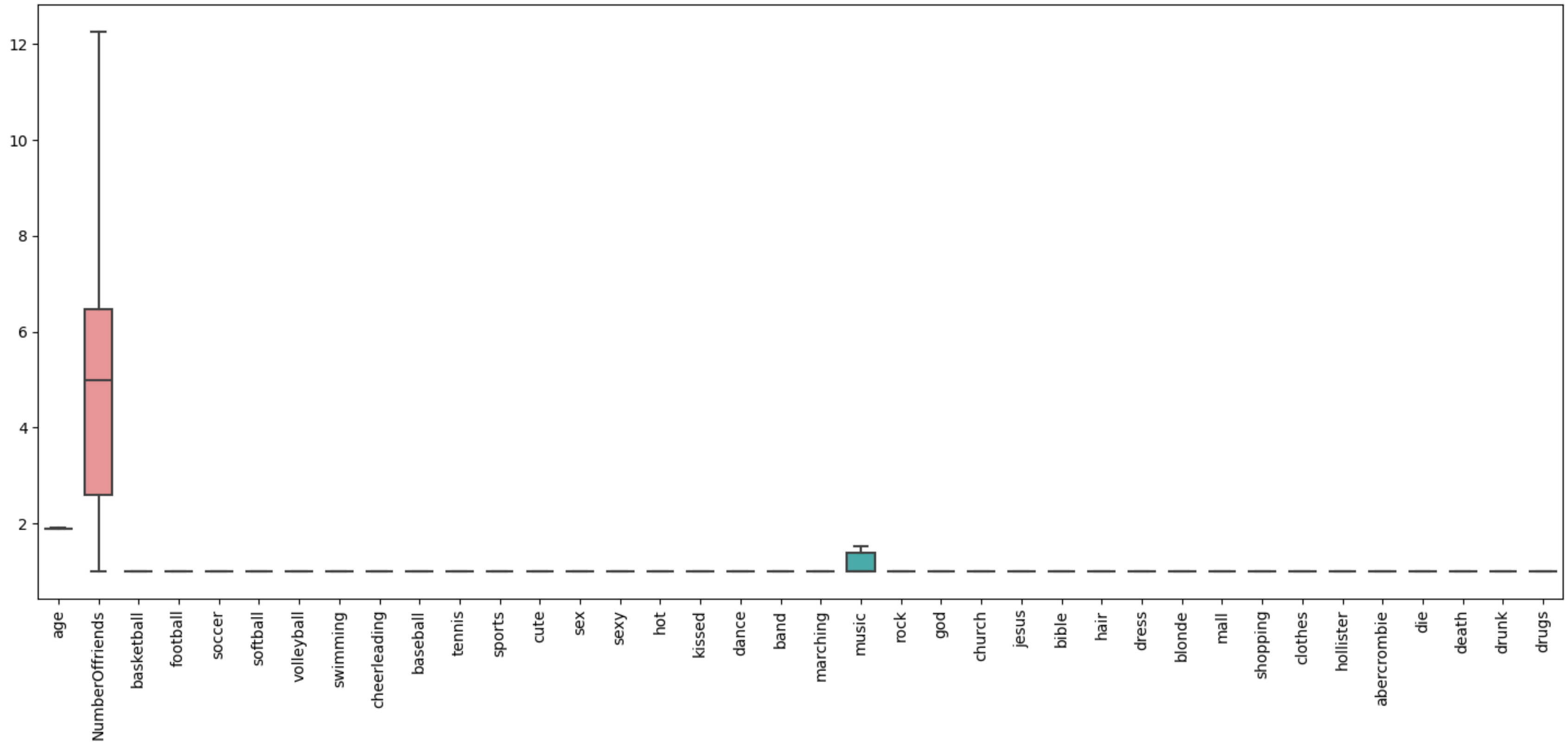
In [ ]:

```
#### AFter Capping Outlier : Box plot
```

In [181…

```
import seaborn as sns
import matplotlib.pyplot as plt

# Set figure size (increase width for better label visibility)
plt.figure(figsize=(18, 8))  # Adjust width and height to your preference

# Create boxplot with wider boxes for better spacing
```

```
sns.boxplot(data=df_remove_outliers, width=0.7)  # Adjust width for better box size

# Rotate x-axis labels for better readability
plt.xticks(rotation=90, ha='center')  # Ensure label alignment and rotation

# Adjust space between labels and chart for clarity
plt.subplots_adjust(bottom=0.2)

plt.show()
```



In [ ]:

====================== Create MODEL : CLUSTERING =========================

K-MEAN

In [184...

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
df_resampled = df_remove_outliers.copy()
# Scaling the data (optional, but helps in clustering)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(df_resampled)
```

In [144...

```
```

K-MEAN custer with range(6,13) cluster

In [185...

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import silhouette_score

# Scaling the data (optional, but helps in clustering)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(df_remove_outliers)
data = df_remove_outliers.copy()
# Apply K-means clustering with k=6-13
for cluster_count in range(4,10):
    kmeans = KMeans(n_clusters=cluster_count, random_state=42)
    data['Cluster'] = kmeans.fit_predict(X_scaled)
    # Calculate silhouette score
    sil_score = silhouette_score(X_scaled, data['Cluster'])
    print(f"Silhouette Score clusters: {cluster_count} : {sil_score}")
```

```
Silhouette Score clusters: 6 : 0.40678031633179224
Silhouette Score clusters: 7 : 0.40127739654973155
Silhouette Score clusters: 8 : 0.3990232844274842
Silhouette Score clusters: 9 : 0.37701373670276866
Silhouette Score clusters: 10 : 0.37865252651608283
Silhouette Score clusters: 11 : 0.37232719723055546
Silhouette Score clusters: 12 : 0.37135694957350956
```

In [ ]:

In [ ]:

DB-SCAN apply for Clustering.

In [186…

```
from sklearn.cluster import DBSCAN
from sklearn.metrics import silhouette_score
from sklearn.decomposition import PCA

# Apply DBSCAN
dbscan = DBSCAN(eps=7.0, min_samples=30)  # Adjust `eps` and `min_samples` as needed
data['Cluster_DBSCAN'] = dbscan.fit_predict(X_scaled)

# Check the number of clusters (excluding noise, labeled as -1)
n_clusters = len(set(data['Cluster_DBSCAN']) - {-1})
print(f"Number of clusters found (excluding noise): {n_clusters}")

# Evaluate Silhouette Score (only if at least 2 clusters are formed)
if n_clusters > 1:
    sil_score = silhouette_score(X_reduced, data['Cluster_DBSCAN'])
    print(f"Silhouette Score for DBSCAN: {sil_score}")
else:
    print("DBSCAN did not find sufficient clusters.")
```

```
Number of clusters found (excluding noise): 1
DBSCAN did not find sufficient clusters.
```

Appy HDBSCAN

In [152…

```
#!pip install hdbscan
```

In [153…

```
import hdbscan
from sklearn.metrics import silhouette_score

# Apply HDBSCAN
hdbscan_clusterer = hdbscan.HDBSCAN(min_samples=200, min_cluster_size=30, metric='euclidean')  # Adjust parameters as
data['Cluster_HDBSCAN'] = hdbscan_clusterer.fit_predict(X_scaled)

# Extract cluster labels
```

```python
labels = data['Cluster_HDBSCAN']

# Check number of clusters (excluding noise, labeled as -1)
n_clusters = len(set(labels) - {-1})
print(f"Number of clusters found (excluding noise): {n_clusters}")

# Calculate Silhouette Score (only if at least 2 clusters are found)
if n_clusters > 1:
    sil_score = silhouette_score(X_scaled, labels)
    print(f"Silhouette Score for HDBSCAN: {sil_score}")
else:
    print("HDBSCAN did not find sufficient clusters.")
```

```
Number of clusters found (excluding noise): 18
Silhouette Score for HDBSCAN: -0.2210357246888267
```

:: Clustering Results ::

The "DBSCAN" have good Silhouette score -52, with two clusters, and K-means have highest Silhouette score with 8-clusters. But HDBCAN have very poor score which is not acceptable.

In [ ]:

Question: c) Use the Anime Recommendations dataset from below Kaggle link and create an end-to-end project on Jupyter/Colab. https://www.kaggle.com/datasets/CooperUnion/anime-recommendations-database/data Code for reference - https://www.kaggle.com/code/benroshan/content-collaborative-anime-recommendation i. Download the dataset from above link and load it into your Python environment. ii. Perform the EDA and do the visualizations. iii. Check the distributions/skewness in the variables and do the transformations if required. iv. Create a content based Recommender system

In [ ]:

Data set link: https://www.kaggle.com/datasets/CooperUnion/anime-recommendations-database/data

Load Anime data from CSV file.

In [93]:
```
import pandas as pd
import numpy as np
# Load dataset
df_ani = pd.read_csv(r'anime.csv')
df_ani.head()
```

Out[93]:

	anime_id	name	genre	type	episodes	rating	members
0	32281	Kimi no Na wa.	Drama, Romance, School, Supernatural	Movie	1	9.37	200630
1	5114	Fullmetal Alchemist: Brotherhood	Action, Adventure, Drama, Fantasy, Magic, Mili...	TV	64	9.26	793665
2	28977	Gintama°	Action, Comedy, Historical, Parody, Samurai, S...	TV	51	9.25	114262
3	9253	Steins;Gate	Sci-Fi, Thriller	TV	24	9.17	673572
4	9969	Gintama'	Action, Comedy, Historical, Parody, Samurai, S...	TV	51	9.16	151266

Filtering text : clean_text, for cleaning text data.

In [94]:
```
import re
import emoji

def clean_text(text):
    # Remove URLs
    text = str(text)
    text = re.sub(r'http\S+|www\S+|https\S+', ' ', text, flags=re.MULTILINE)
    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)
    # Remove emojis
    text = emoji.replace_emoji(text, replace=' ')
    text = re.sub(r'\s+',' ', text)
    return text
```

Load Rating data from csv file.

In [96]:
```
import pandas as pd
import numpy as np
# Load dataset
df_rating = pd.read_csv(r'rating.csv')
df_rating.head()
```

Out[96]:

	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

================== Exploratry Data Analysis ============================

Eliminate inappropriately rating : Some rating are -1 : means user don't provide any rating.

In [97]:
```
# Convert empty strings to NaN
df_rating['rating'].replace('', np.nan, inplace=True)
# Remove rows where rating is -1, NaN, or blank
df_rating_cleaned = df_rating.dropna(subset=['rating'])
df_rating_cleaned = df_rating_cleaned[df_rating_cleaned['rating'] != -1]
```

```
print(f"Rating shape before cleanup: {df_rating.shape}")
print(f"rating after remove rating is -1 or NaN/Null/Empty : {df_rating_cleaned.shape}")
```

```
Rating shape before cleanup: (7813737, 3)
rating after remove rating is -1 or NaN/Null/Empty : (6337241, 3)
```

Clean 'name' feature,

In [98]:

```
df_ani['name'] = df_ani['name'].apply(clean_text)
print(f"Shape of df_ani : {df_ani.shape}")
unique_df_ani = df_ani.drop_duplicates(subset=['name'])
print(f"After remove duplicates, Shape of unique_df_ani : {unique_df_ani.shape}")
```

```
Shape of df_ani : (12294, 7)
After remove duplicates, Shape of unique_df_ani : (12286, 7)
```

In [ ]:

```

```

Merge the datasets on the 'anime_id' column :: Merged anime and rating data set after cleanup.

In [99]:

```
# Merge the datasets on the 'anime_id' column
merged_data = pd.merge(df_rating_cleaned, unique_df_ani, on='anime_id', how='inner')
print(merged_data.describe().map("{:.0f}".format))
merged_data.head()
```

```
       user_id anime_id rating_x rating_y  members
count  6323076  6323076  6323076  6323071  6323076
mean     36749     8909        8        8   184535
std      21014     8889        2        1   191106
min          1        1        1        2       33
25%      18981     1239        7        7    46712
50%      36820     6211        8        8   117090
75%      54877    14131        9        8   256325
max      73516    34475       10        9  1013917
```

Out[99]:

	user_id	anime_id	rating_x	name	genre	type	episodes	rating_y	members
0	1	8074	10	Highschool of the Dead	Action, Ecchi, Horror, Supernatural	TV	12	7.46	535892
1	3	8074	6	Highschool of the Dead	Action, Ecchi, Horror, Supernatural	TV	12	7.46	535892
2	5	8074	2	Highschool of the Dead	Action, Ecchi, Horror, Supernatural	TV	12	7.46	535892
3	12	8074	6	Highschool of the Dead	Action, Ecchi, Horror, Supernatural	TV	12	7.46	535892
4	14	8074	6	Highschool of the Dead	Action, Ecchi, Horror, Supernatural	TV	12	7.46	535892

In [100…

```
# Convert columns to integers
merged_data['anime_id'] = merged_data['anime_id'].astype(int)
merged_data['user_id'] = merged_data['user_id'].astype(int)
# Verify the data types
print(merged_data.dtypes)
```

```
user_id       int32
anime_id      int32
rating_x      int64
name         object
genre        object
type         object
episodes     object
rating_y    float64
members       int64
dtype: object
```

In [101…

```
merged_data[merged_data.duplicated()].shape
anime_all = merged_data.drop_duplicates()
print(f"Dupllicated: {anime_all[anime_all.duplicated()].shape}")
```

```
Dupllicated: (0, 9)
```

Rename column for better understading : rating_x to user_rating and rating_y to avg_rating.

In [ ]:

```
anime_all.rename(columns={'rating_x': 'user_rating', 'rating_y': 'avg_rating'}, inplace=True)
```

In [104…

```
anime_all.head(15)
```

Out[104…

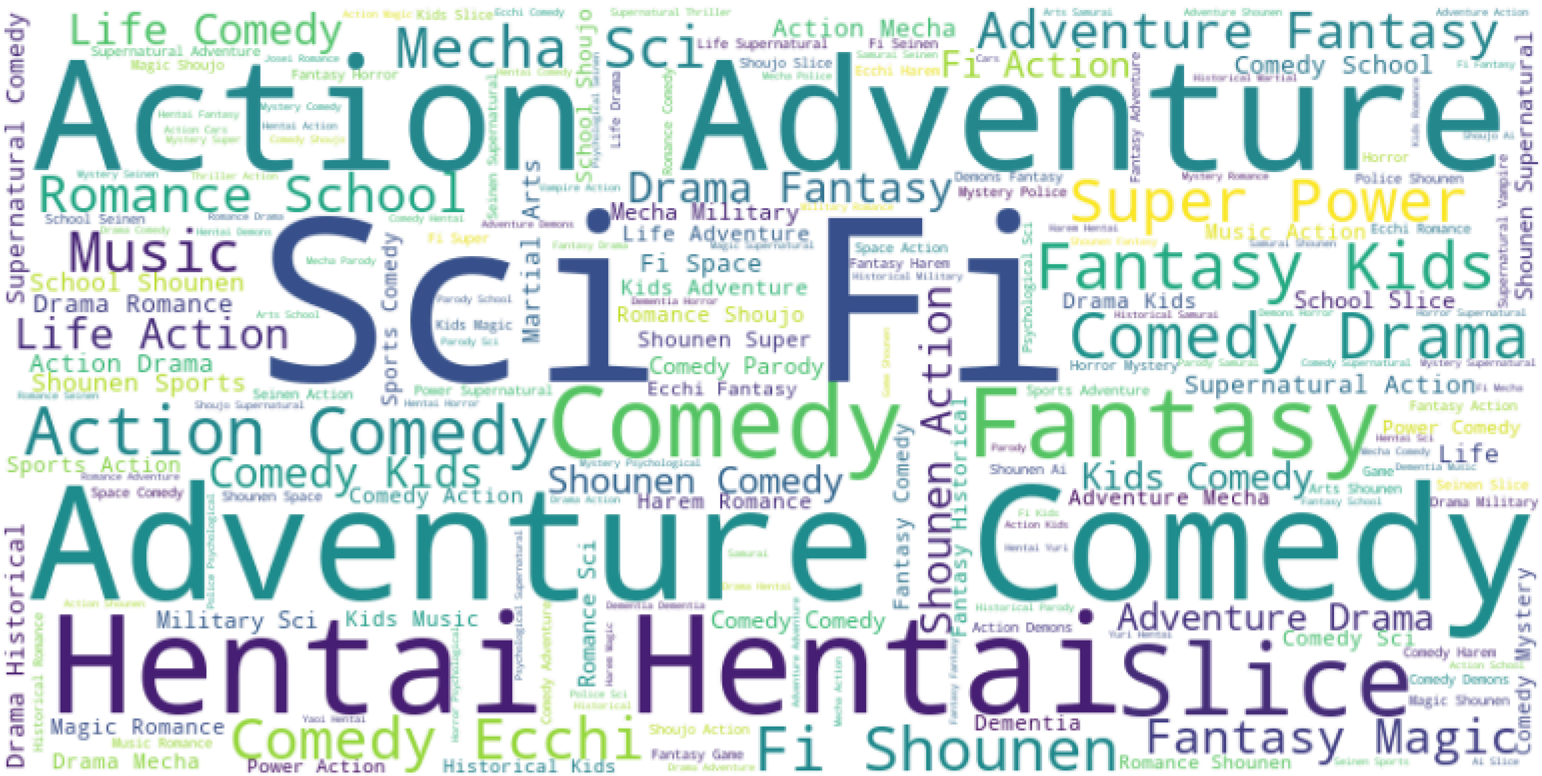
	user_id	anime_id	user_rating	name	genre	type	episodes	avg_rating	members
0	1	8074	10	Highschool of the Dead	Action, Ecchi, Horror, Supernatural	TV	12	7.46	535892
1	3	8074	6	Highschool of the Dead	Action, Ecchi, Horror, Supernatural	TV	12	7.46	535892
2	5	8074	2	Highschool of the Dead	Action, Ecchi, Horror, Supernatural	TV	12	7.46	535892
3	12	8074	6	Highschool of the Dead	Action, Ecchi, Horror, Supernatural	TV	12	7.46	535892
4	14	8074	6	Highschool of the Dead	Action, Ecchi, Horror, Supernatural	TV	12	7.46	535892
5	17	8074	7	Highschool of the Dead	Action, Ecchi, Horror, Supernatural	TV	12	7.46	535892
6	24	8074	7	Highschool of the Dead	Action, Ecchi, Horror, Supernatural	TV	12	7.46	535892
7	27	8074	9	Highschool of the Dead	Action, Ecchi, Horror, Supernatural	TV	12	7.46	535892
8	29	8074	2	Highschool of the Dead	Action, Ecchi, Horror, Supernatural	TV	12	7.46	535892
9	30	8074	8	Highschool of the Dead	Action, Ecchi, Horror, Supernatural	TV	12	7.46	535892
10	37	8074	4	Highschool of the Dead	Action, Ecchi, Horror, Supernatural	TV	12	7.46	535892
11	38	8074	5	Highschool of the Dead	Action, Ecchi, Horror, Supernatural	TV	12	7.46	535892
12	40	8074	9	Highschool of the Dead	Action, Ecchi, Horror, Supernatural	TV	12	7.46	535892
13	41	8074	10	Highschool of the Dead	Action, Ecchi, Horror, Supernatural	TV	12	7.46	535892

	user_id	anime_id	user_rating	name	genre	type	episodes	avg_rating	members
14	46	8074	5	Highschool of the Dead	Action, Ecchi, Horror, Supernatural	TV	12	7.46	535892

Word cloud for genre.

In [172…

```
import pandas as pd
from wordcloud import WordCloud
import matplotlib.pyplot as plt

# Combine all genres into a single string
genres = ' '.join(df_ani['genre'].fillna(''))
# Create the word cloud
wordcloud = WordCloud(width=800, height=400, background_color='white').generate(genres)
# Display the word cloud
plt.figure(figsize=(10, 5))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.show()
```



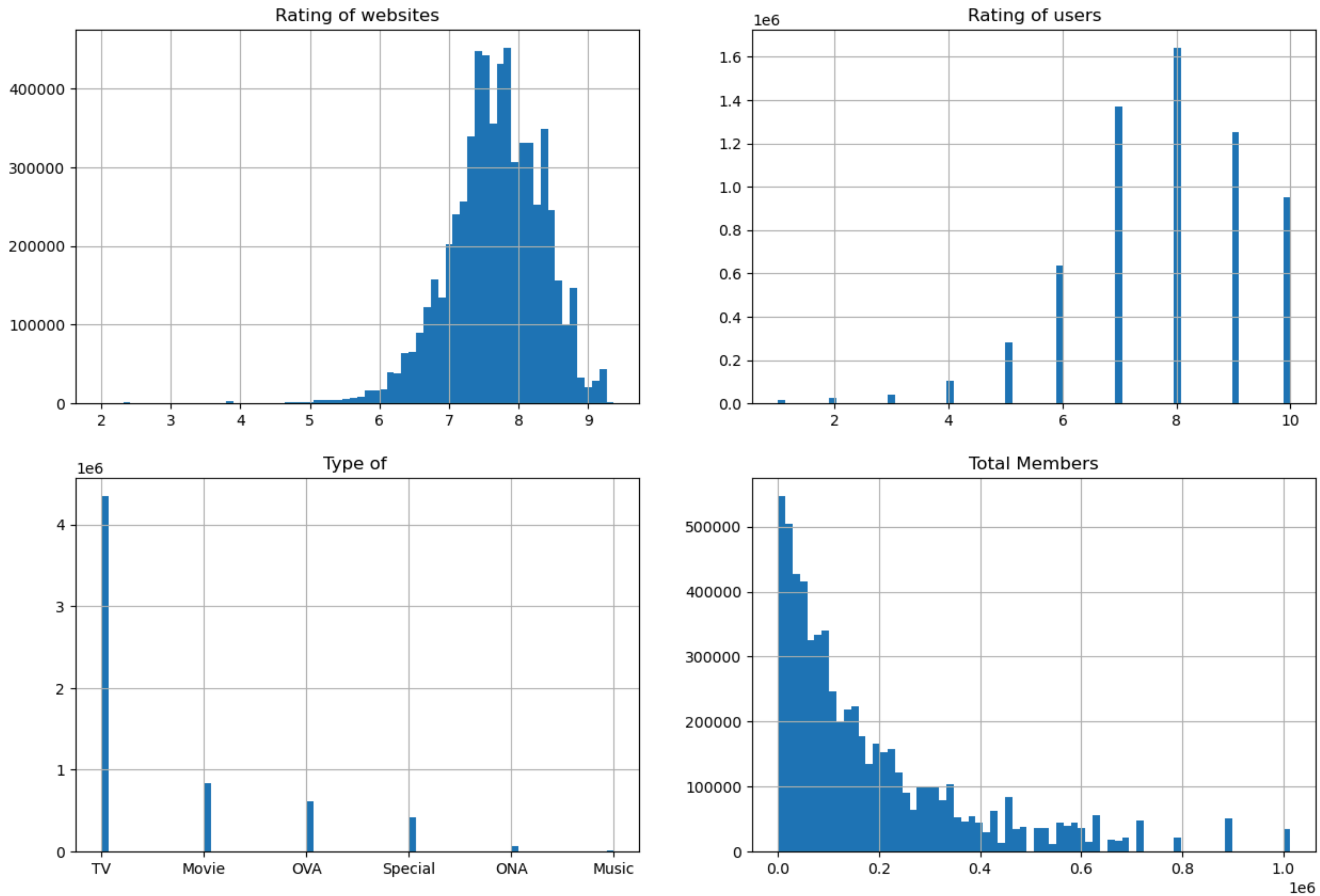
Display distribution by : Rating Avg, Rating users, Type and Total Members.

In [158…

```
plt.figure(figsize = (15, 10))
plt.subplot(2,2,1)
anime_all['avg_rating'].hist(bins=70)
plt.title("Rating of websites")

plt.subplot(2,2,2)
anime_all['user_rating'].hist(bins=70)
plt.title("Rating of users")

plt.subplot(2,2,3)
anime_all['type'].hist(bins=70)
plt.title("Type of ")

plt.subplot(2,2,4)
```

```
anime_all['members'].hist(bins=70)
plt.title("Total Members")
```

Out[158… Text(0.5, 1.0, 'Total Members')



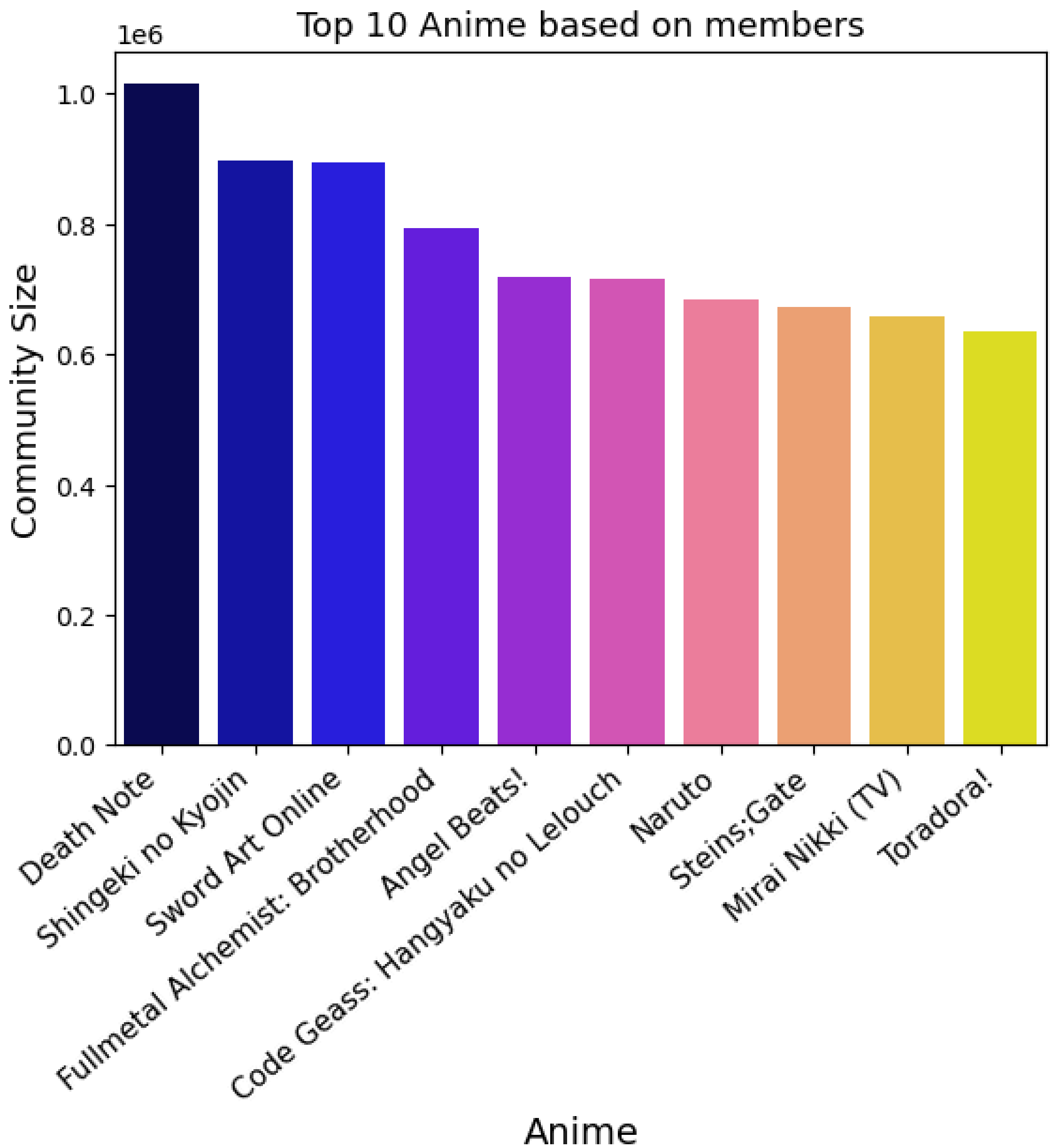
Top 10 Anime based on members.

In [170…

```
import seaborn as sns
anime_uniques=anime_all.copy()
anime_uniques.drop_duplicates(subset ="name",
                     keep = 'first', inplace = True)
top10_members=anime_uniques[['name', 'members']].sort_values(by = 'members',ascending = False).head(10)
ax=sns.barplot(x="name", y="members", data=top10_members, palette="gnuplot2")
ax.set_xticklabels(ax.get_xticklabels(), fontsize=11, rotation=40, ha="right")
ax.set_title('Top 10 Anime based on members',fontsize = 13)
ax.set_xlabel('Anime',fontsize = 14)
ax.set_ylabel('Community Size', fontsize = 13)
```

Out[170…

```
Text(0, 0.5, 'Community Size')
```




```
In [ ]:
```

Filter based on user_id : User which participate in rating more and equall to 200 times.

In [105…

```
anime_GE200 = anime_all.copy()
counts = anime_all['user_id'].value_counts()
anime_GE200 = anime_all[anime_all['user_id'].isin(counts[counts >= 200].index)]
anime_L200 = anime_all[anime_all['user_id'].isin(counts[counts < 200].index)]
print(f"total {len(anime_all)} <==> {len(anime_L200)} + {len(anime_GE200)} <==> {len(anime_L200) + len(anime_GE200)}'
```

```
total 6323075 <==> 3160067 + 3163008 <==> 6323075
```

For creating sparse matrix and finding cosine similarity : Create Pivote table.

In [106…

```
anime_pivot=anime_GE200.pivot_table(index='name',columns='user_id',values='user_rating').fillna(0)
anime_pivot.head()
```

Out[106…

user_id	5	7	17	38	43	46	123	129	139	160	...	73406	73417	73422	73457	73460	73476	73499	73502	7
name																				
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	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	0.0	
009 Re:Cyborg	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	6.0	0.0	0.0	0.0	0.0	
009-1	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	0.0	
009-1: RandB	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	0.0	

5 rows × 8669 columns

Use k-nearest neighbors (k-NN) algorithm with cosine matrix.

In [108…

```
from scipy.sparse import csr_matrix
from sklearn.neighbors import NearestNeighbors

### pivot table into a sparse matrix
anime_matrix = csr_matrix(anime_pivot.values)
### using a k-nearest neighbors (k-NN) algorithm to create a model
```

```
### Using cosine Matrix for finding similarity items features.
model_knn = NearestNeighbors(metric = 'cosine', algorithm = 'brute')
### Fitting the Model
model_knn.fit(anime_matrix)
```

Out[108…

```
NearestNeighbors
NearestNeighbors(algorithm='brute', metric='cosine')
```

Select a random index for 'name' feature, from the anime_pivot

In [114…

```
name_index = np.random.choice(anime_pivot.shape[0])
print(f"Index of name :{name_index}")
### returning the Cosine distances and indices of 5 neighbours through KNN
cosine_distances, indices = model_knn.kneighbors(anime_pivot.iloc[name_index,:].values.reshape(1, -1), n_neighbors =
```

```
Index of name :4737
```

cosine_distances values represent the distances between the query point and each of the 6 nearest neighbors found.

''' Cosine Distance vs. Cosine Similarity:

Cosine similarity measures the cosine of the angle between two vectors, with values ranging from -1 (exactly opposite) to 1 (exactly the same).

Cosine distance is calculated as 1 - cosine similarity. Therefore, a lower cosine distance indicates higher similarity. '''

Display top 5 recommended names.

In [115…

```
for i in range(0, len(cosine_distances.flatten())):
    if i == 0:
        print('Recommendations for {0}:\n'.format(anime_pivot.index[name_index]))
    else:
        print('{0}: {1}, with distance of {2}:'.format(i, anime_pivot.index[indices.flatten()[i]], cosine_distances.
```

```
Recommendations for Kuruneko: Nyaalock Holmes no Bouken:

1: Ashinaga Ojisan, with distance of 0.2928932188134524:
2: Fushigiboshi no☆Futagohime Gyu! Recap, with distance of 0.2928932188134524:
3: Kuruneko: Kurunekobin, with distance of 0.2928932188134524:
4: Gunparade Orchestra OVA, with distance of 0.2928932188134524:
5: Ai to Ken no Camelot: Mangaka Marina Time Slip Jiken, with distance of 0.41165159458544787:
```

In [ ]:

```
```

TF-IDF apply for 'genere' recommendation.

TF-IDF is based on the Bag of Words (BoW) model, which counts the occurrence of words in a document. However, TF-IDF adjusts for the fact that some words appear more frequently than others. It does this by: Term frequency (TF): The number of times a term appears in a document Inverse document frequency (IDF): Weighs down frequent terms and increases the weight of rare terms Calculating the TF-IDF score: Multiplying the TF and IDF values together.

The higher the TF-IDF score, the more relevant the word is in the document.

Create TFidf Matrix for 'genre' feature.

In [122…

```
from sklearn.feature_extraction.text import TfidfVectorizer
tfv = TfidfVectorizer(min_df=3,  max_features=None,
            strip_accents='unicode', analyzer='word',token_pattern=r'\w{1,}',
            ngram_range=(1, 3),
            stop_words = 'english')

# Filling NaNs with empty string
unique_df_ani.loc[:, 'genre'] = unique_df_ani['genre'].fillna('')
genres_str = unique_df_ani['genre'].str.split(',').astype(str)
tfv_matrix = tfv.fit_transform(genres_str)
```

Calculate sigmoid kerenal :

The sigmoid kernel is used to compute the similarity between two data points using the hyperbolic tangent function.

In [124…

```
from sklearn.metrics.pairwise import sigmoid_kernel
sigmoid_matrix = sigmoid_kernel(tfv_matrix, tfv_matrix)
```

Collect the name indices :

In [126…

```
name_indices = pd.Series(unique_df_ani.index, index=unique_df_ani['name']).drop_duplicates()
```

Recommendation function: Using sigmoid kernel.

In [129…

```
def recommended_name(title, name_indices = name_indices, sigmoid_matrix=sigmoid_matrix):
    # Get the index corresponding to original_title
    idx = name_indices[title]
    # Get the pairwsie similarity scores
    sig_scores = list(enumerate(sigmoid_matrix[idx]))
    # Sort the movies
    sig_scores = sorted(sig_scores, key=lambda x: x[1], reverse=True)
    # Scores of the 10 most similar movies
    sig_scores = sig_scores[1:6]
    # Movie indices
    anime_indices = [i[0] for i in sig_scores]
    # Top 10 most similar movies
    return pd.DataFrame({'Anime name': unique_df_ani['name'].iloc[anime_indices].values,
                                 'Rating': unique_df_ani['rating'].iloc[anime_indices].values})
```

Recommendation for 5 samples.

Get random 5 name and applyy the recommended function

Recommended name ascending user rating.

In [143…

```
# ANSI escape codes for bold text
BOLD = '\033[1m'
END = '\033[0m'
np.random.seed()
random_names = unique_df_ani['name'].sample(5).values
for anime_name in random_names:
    print(f"{BOLD} Recommendation for: {anime_name} {END}")
    result = recommended_name(anime_name)
```

```
    print(f"{result}")
    print(f"{BOLD}{'=' * 60}{END}")
```

Recommendation for: Peeping Life: Gekijou Original-ban

	Anime name	Rating
0	Gochuumon wa Usagi Desu ka??	8.01
1	Working!!	7.98
2	Jungle wa Itsumo Hare nochi Guu Deluxe	7.95
3	Chis Sweet Home	7.83
4	New Game!	7.81

==

Recommendation for: Pugyuru

	Anime name	Rating
0	Isshuukan Friends. Specials	6.81
1	Mai-Otome Special: Otome no Inori	6.60
2	Girlfriend (Kari)	6.03
3	Turnover	5.92
4	Kanojo ga Kanji wo Suki na Riyuu.	5.71

==

Recommendation for: Yozakura Quartet: Tsuki ni Naku

	Anime name	Rating
0	Zero no Tsukaima	7.62
1	Zero no Tsukaima: Princesses no Rondo	7.60
2	Negima!?	7.21
3	Campione!: Matsurowanu Kamigami to Kamigoroshi...	7.36
4	Trinity Seven Movie: Eternity Library to Alche...	NaN

==

Recommendation for: Tiger Mask

	Anime name	Rating
0	Zone of the Enders: Dolores, I	7.04
1	Chou Mashin Eiyuuden Wataru	7.16
2	Game Tengoku OVA	4.96
3	Chikyuu Bouei Kigyou Dai-Guard	7.21
4	Mashin Eiyuuden Wataru 2	7.05

==

Recommendation for: Kindan no Mokushiroku: Crystal Triangle

	Anime name	Rating
0	Kyoukai no Kanata Movie: Ill Be Here - Kako-he...	7.92
1	Snow Halation	7.61
2	Perfect Day	7.59
3	Higashi no Eden: Falling Down	7.56
4	Macross F Music Clip Shuu: Nyankuri	7.47

==

In [ ]:

```

```

In [ ]:

```

```