Project 3
PCA_SVD_Clusters

Sumant Anantha

November 2023

Introduction:

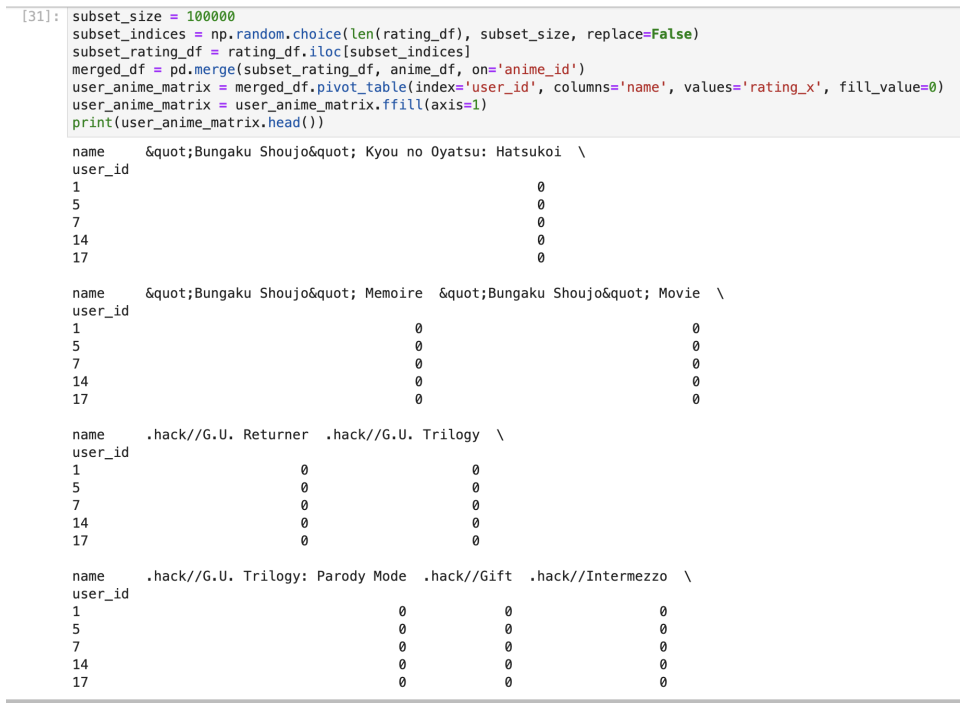
Using a large dataset of user ratings for a wide range of anime titles, this research explores the field of anime recommendation. By using advanced analytics techniques like singular value decomposition (SVD), dimensionality reduction, and clustering, we hope to identify trends in the user-item interaction matrix. The goal of this investigation is to improve our understanding of collaborative filtering in anime recommendation systems by investigating the interplay between user preferences and content popularity.

Summary:

This research starts by highlighting highly rated anime titles and devoted viewers, offering insight into popular content and user involvement as it navigates the vast terrain of anime tastes. Clustering algorithms are the next step in the journey, exposing different user segments according to their rating behaviors. Principal Component Analysis (PCA) dimensionality reduction provides visual insights into the correlations between anime titles. Singular Value Decomposition (SVD), which reveals the underlying structure of the user-item matrix, is the culmination of the investigation. The research critically analyzes collaborative filtering techniques during various analytical stages, illuminating how well they capture and interpret user preferences in the anime industry. The results open the door to more complex understandings of collaborative filtering dynamics and user-item interactions in this dynamic entertainment space, as well as better anime recommendation systems.

1 Transforming Data

To create the matrix, the first step is to create a subset of n rows from the large dataset. This will allow the notebook to run the code a lot faster. A random function will be used to select a 100,000 random rows with anime ratings with user ratings from the dataset, which will account for any bias in sampling. Both the anime and rating CSV files are merged in the anime_id column. The function used to create the matrix is the pivot. A pivot table is created from the merged DataFrame, where rows represent users, columns represent anime titles ('name'), and values represent ratings. Missing values (NaNs) are filled using forward fill (method='ffill').



```
[31]: subset_size = 100000
      subset_indices = np.random.choice(len(rating_df), subset_size, replace=False)
      subset_rating_df = rating_df.iloc[subset_indices]
      merged_df = pd.merge(subset_rating_df, anime_df, on='anime_id')
      user_anime_matrix = merged_df.pivot_table(index='user_id', columns='name', values='rating_x', fill_value=0)
      user_anime_matrix = user_anime_matrix.ffill(axis=1)
      print(user_anime_matrix.head())
```

```
name     &quot;Bungaku Shoujo&quot; Kyou no Oyatsu: Hatsukoi  \
user_id
1                                                        0
5                                                        0
7                                                        0
14                                                       0
17                                                       0

name     &quot;Bungaku Shoujo&quot; Memoire  &quot;Bungaku Shoujo&quot; Movie  \
user_id
1                                         0                                  0
5                                         0                                  0
7                                         0                                  0
14                                        0                                  0
17                                        0                                  0

name     .hack//G.U. Returner  .hack//G.U. Trilogy  \
user_id
1                           0                    0
5                           0                    0
7                           0                    0
14                          0                    0
17                          0                    0

name     .hack//G.U. Trilogy: Parody Mode  .hack//Gift  .hack//Intermezzo  \
user_id
1                                       0            0                  0
5                                       0            0                  0
7                                       0            0                  0
14                                      0            0                  0
17                                      0            0                  0
```

This is the User Anime Matrix which will be the matrix used for the rest of the problems below. A user who hasn't rated the matching anime is indicated with a value of 0 in the matrix. Sparse matrices with plenty of zero entries are frequent in collaborative filtering recommendation systems since users usually interact with a tiny subset of the things available. The values do exists as the output does not fully display the matrix. The matrix will then be used to find the top 3 anime with the highest number of ratings and the top 3 users with the most ratings.

```
[5 rows x 5789 columns]
```

```
32]: # (a) Top 3 anime by the number of users who rated them
top_anime_by_users = user_anime_matrix.sum().sort_values(ascending=False).head(3)
anime_df_reset = anime_df.reset_index()
anime_indices = anime_df_reset[anime_df_reset['name'].isin(top_anime_by_users.index)].index.tolist()
top_anime_titles = anime_df_reset.loc[anime_indices, 'name']
print("Top 3 Anime by Number of User ratings:")
for title, count in zip(top_anime_titles, top_anime_by_users):
    print(f"{title}: {count} user ratings")
```

```
Top 3 Anime by Number of User ratings:
Code Geass: Hangyaku no Lelouch: 3880 user ratings
Death Note: 2873 user ratings
Shingeki no Kyojin: 2765 user ratings
```

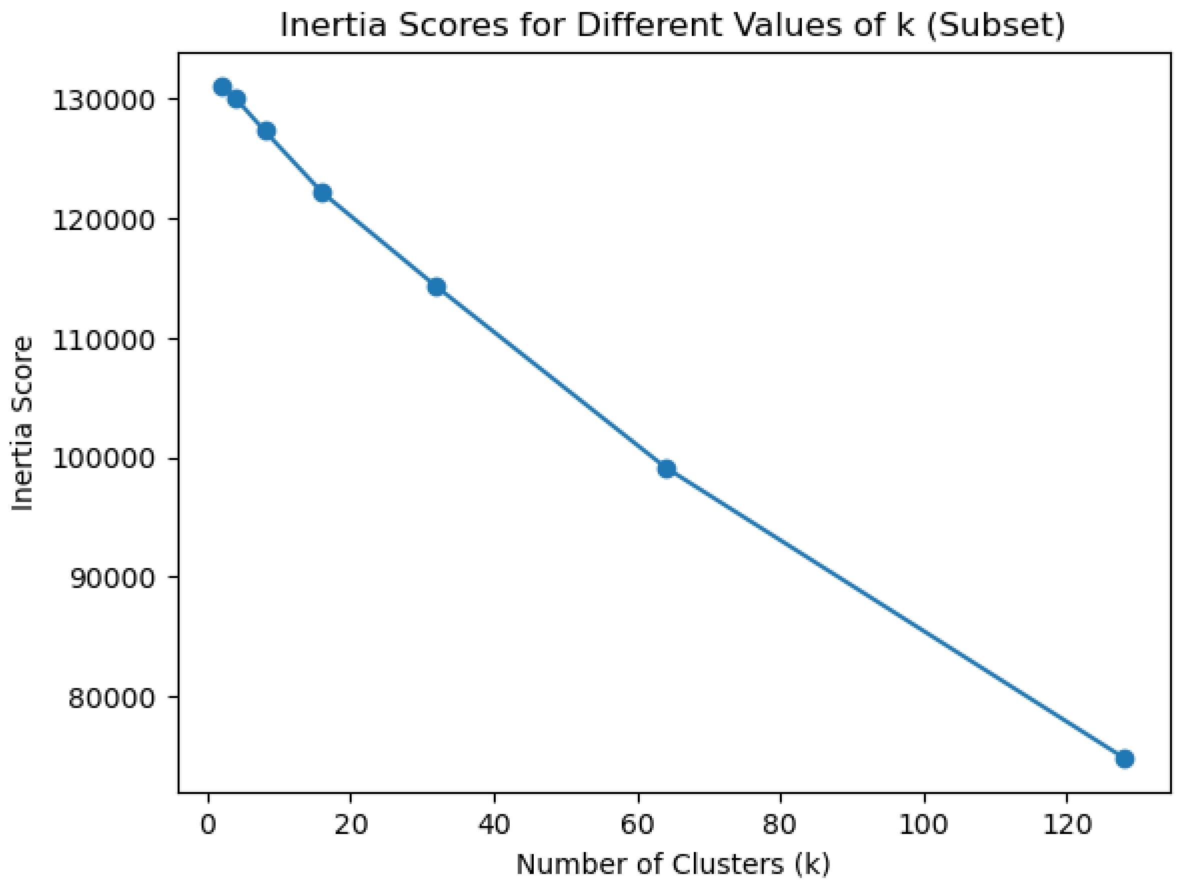
```
35]: # (b) Top 3 users by the number of animes they have rated
top_users_by_anime = user_anime_matrix.sum(axis=1).sort_values(ascending=False).head(3)
print("\nTop 3 Users by Number of Anime Rated:")
for user_id, count in zip(top_users_by_anime.index, top_users_by_anime):
    print(f"User ID {user_id}: {count} anime")
```

```

Top 3 Users by Number of Anime Rated:
User ID 42635: 330 anime
User ID 51270: 292 anime
User ID 57620: 270 anime
```

From our matrix, we can see that the top 3 animes are: Code Geass (rated 3880 times), Death Note (rated 2873 times), and Attack on Titan (Shingeki no Kyojin rated 2765 times). Then for the top 3 users, from largest to smallest number of ratings: ID 42635 rated 330 anime, ID 51270 rated 292 anime, and ID 57620 rated 270 anime. The process of finding the top 3 anime involved indexing the matrix because, when indexing the matrix, the values can be obtained from the index of the matrix and then used with the .isin() function to find the anime with the highest rating number count. The function zip is used to pair the anime with their respective counts of ratings. To find the top 3 users with the highest number of ratings, the same code is used, but since the top anime list is created in Part A, the values and ID just need to be extracted.

2 Clustering

The methodology for creating the graph comes from creating a subset of 1000 indices without replacement from the user anime matrix and storing in in a variable to use in the k-means clustering. Matplotlib is used to create the graph, while k-means is used to calculate the inertia, which will be the dependent variables. The independent variables will be the k-values.This graph here shows the inertia values for every k-value for values of k = [2, 4, 8, 16, 32, 64, 128] when applying k-means.



From observing the graph here, the inertia values become smaller as the k-value increases. To find the most appropriate value of k, the elbow point must be found. The elbow point is the k-value where the graph gradually starts to slow down; in this case, the value is k = 16. From the graph, after k = 16, the graph starts to show a slower change in inertia for every increase in k-value so 16 is the right cluster value. Using the chosen k-value (16), K-means clustering with the chosen k clusters is applied to the user_anime_matrix. The cluster assignments are stored in a new column named 'cluster' in user_anime_matrix. This will allow easy calculation when finding the means for each cluster anime value. To ensure indexing is easier, the anime cluster matrix will be transposed, which will display the top 3 anime results for each cluster.

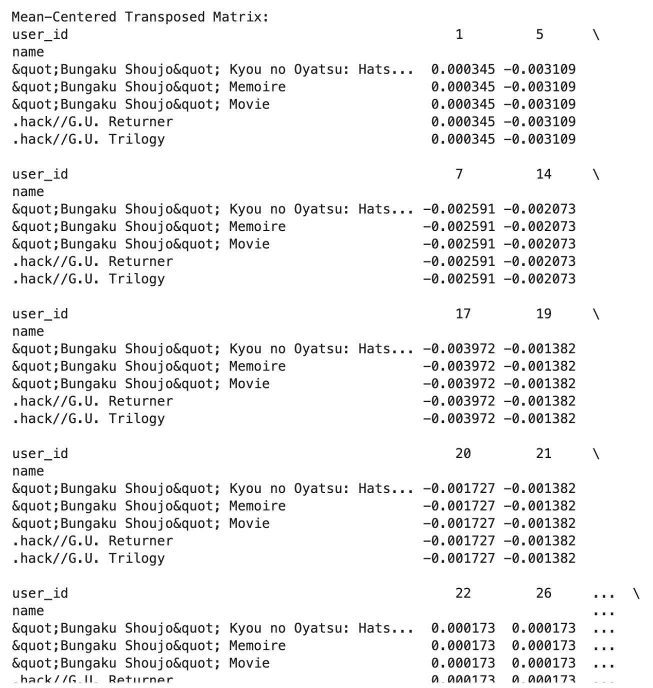
```
Cluster 1:
Rokka no Yuusha Picture Drama: Average Rating 0.08
Servant x Service: Average Rating 0.07
Tenchi Muyou!: Average Rating 0.07

Cluster 2:
Aria The Animation: Average Rating 9.00
Lupin III: Part III: Average Rating 9.00
Little Lulu to Chicchai Nakama: Average Rating 9.00

Cluster 3:
Neo Angelique Abyss: Second Age: Average Rating 8.65
Peace Maker Kurogane Special: Average Rating 0.21
Shin Koihime†Musou: Otome Tairan OVA: Average Rating 0.21

Cluster 4:
Gun x Sword: Average Rating 7.86
Ganbare Genki: Average Rating 0.17
Koi Kaze: Average Rating 0.16

Cluster 5:
Bronze: Kouji Nanjo Cathexis: Average Rating 1.82
Gekkan Shoujo Nozaki-kun: Average Rating 1.31
Utawarerumono OVA Picture Drama: Average Rating 1.26

Cluster 6:
Crayon Shin-chan Movie 08: Arashi wo Yobu Jungle: Average Rating 4.36
Kuroko no Basket 2nd Season NG-shuu: Average Rating 3.99
Code:Breaker OVA: Average Rating 0.17

Cluster 7:
Sasuga no Sarutobi: Average Rating 8.46
Saki Achiga-hen: Episode of Side-A: Average Rating 0.27
Tokyo Ravens: Average Rating 0.23

Cluster 8:
Aria The Natural: Sono Futatabi Deaeru Kiseki ni...: Average Rating 8.65
Doraemon Movie 36: Shin Nobita no Nippon Tanjou: Average Rating 0.22
Carnival Phantasm EX Season: Average Rating 0.19
```

```
Cluster 9:
Saint Seiya Omega: Average Rating 8.46
One Piece: Straw Hat Theater: Average Rating 0.12
Sakura Taisen: Le Nouveau Paris: Average Rating 0.11

Cluster 10:
Mainichi ga Nichiyoubi: Average Rating 9.00
Choujin Locke: Average Rating 8.00
Ginga Eiyuu Densetsu Gaiden: Rasen Meikyuu: Average Rating 8.00

Cluster 11:
Teekyuu 5: Average Rating 10.00
Touhou Niji Sousaku Doujin Anime: Musou Kakyou: Average Rating 10.00
Solty Rei: Average Rating 9.00

Cluster 12:
Natsume Yuujinchou: Nyanko-sensei to Hajimete no Otsukai: Average Rating 8.80
Rokka no Yuusha Picture Drama: Average Rating 0.11
The Disappearance of Conan Edogawa: The Worst Two Days in History: Average Rating 0.09

Cluster 13:
Seikimatsu Occult Gakuin: Average Rating 8.22
Dragon Ball Z Movie 12: Fukkatsu no Fusion!! Gokuu to Vegeta: Average Rating 0.18
JoJo no Kimyou na Bouken: Stardust Crusaders: Average Rating 0.13

Cluster 14:
Summer Wars: Average Rating 8.48
Hunter x Hunter: Greed Island Final: Average Rating 0.12
JoJo no Kimyou na Bouken: Stardust Crusaders: Average Rating 0.11

Cluster 15:
Dragon Nest: Warriors&#039; Dawn: Average Rating 7.96
Servant x Service: Average Rating 0.33
Tokyo Ravens: Average Rating 0.25

Cluster 16:
Shin Taketori Monogatari: 1000-nen Joou: Average Rating 8.56
Bronze: Kouji Nanjo Cathexis: Average Rating 0.53
Tenjou Tenge: The Ultimate Fight: Average Rating 0.50
```

These are the top 3 anime in each of the 16 clusters. The results look somewhat reasonable for each cluster, as the anime varies for each cluster, but some ratings are either too big or too small, which is interesting. Based on average ratings, the clusters seem to be fairly aligned with user preferences. Notable findings include limited ratings indicating possible outliers or less well-liked titles, mixed ratings reflecting a range of tastes, and consistently high rating clusters. All things considered, the clustering results appear to be in line with predictions, offering insights into various user engagement habits with anime.

3 Principal Component Analysis

Here is the result of the transposed matrix for the user anime matrix:



```
Mean-Centered Transposed Matrix:
user_id                                              1         5      \
name
&quot;Bungaku Shoujo&quot; Kyou no Oyatsu: Hats...  0.000345 -0.003109
&quot;Bungaku Shoujo&quot; Memoire                 0.000345 -0.003109
&quot;Bungaku Shoujo&quot; Movie                   0.000345 -0.003109
.hack//G.U. Returner                              0.000345 -0.003109
.hack//G.U. Trilogy                               0.000345 -0.003109

user_id                                              7         14     \
name
&quot;Bungaku Shoujo&quot; Kyou no Oyatsu: Hats... -0.002591 -0.002073
&quot;Bungaku Shoujo&quot; Memoire                -0.002591 -0.002073
&quot;Bungaku Shoujo&quot; Movie                  -0.002591 -0.002073
.hack//G.U. Returner                             -0.002591 -0.002073
.hack//G.U. Trilogy                              -0.002591 -0.002073

user_id                                              17        19     \
name
&quot;Bungaku Shoujo&quot; Kyou no Oyatsu: Hats... -0.003972 -0.001382
&quot;Bungaku Shoujo&quot; Memoire                -0.003972 -0.001382
&quot;Bungaku Shoujo&quot; Movie                  -0.003972 -0.001382
.hack//G.U. Returner                             -0.003972 -0.001382
.hack//G.U. Trilogy                              -0.003972 -0.001382

user_id                                              20        21     \
name
&quot;Bungaku Shoujo&quot; Kyou no Oyatsu: Hats... -0.001727 -0.001382
&quot;Bungaku Shoujo&quot; Memoire                -0.001727 -0.001382
&quot;Bungaku Shoujo&quot; Movie                  -0.001727 -0.001382
.hack//G.U. Returner                             -0.001727 -0.001382
.hack//G.U. Trilogy                              -0.001727 -0.001382

user_id                                              22        26     ...  \
name                                                                  ...
&quot;Bungaku Shoujo&quot; Kyou no Oyatsu: Hats...  0.000173  0.000173  ...
&quot;Bungaku Shoujo&quot; Memoire                 0.000173  0.000173  ...
&quot;Bungaku Shoujo&quot; Movie                   0.000173  0.000173  ...
.hack//G.U. Returner                              0.000173  0.000173  ...
```

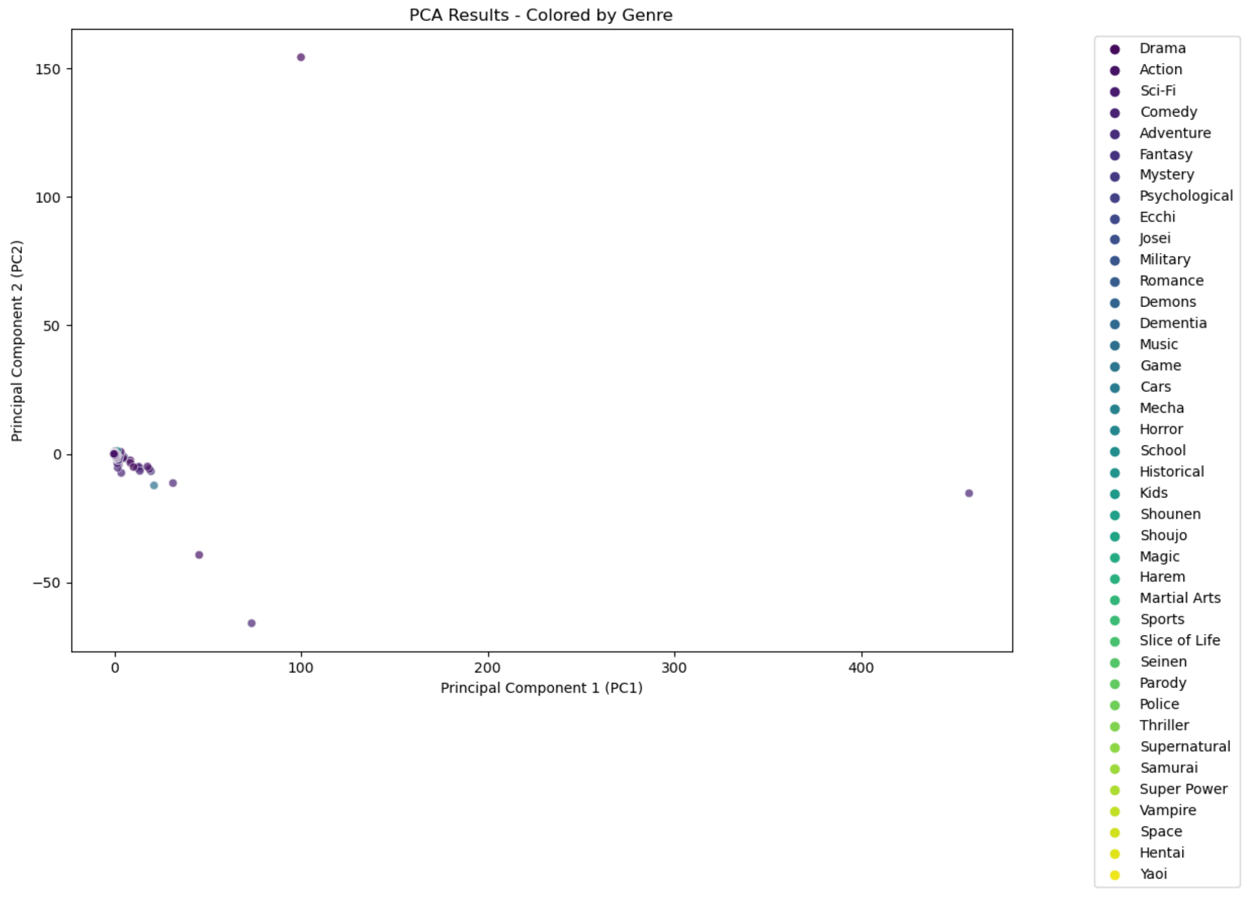
The mean centered values were found by subtracting the transpose matrix values from their respective means; hence, we have some negative results.

Applying PCA with value of k = 2 to the user anime matrix gives the following result below:

```
]: k_components = 2
   pca = PCA(n_components=k_components)
   pca_result = pca.fit_transform(mean_centered_matrix)
   pca_df = pd.DataFrame(data=pca_result, columns=['PC1', 'PC2'])
   print("\nPCA Result:")
   print(pca_df.head())
```

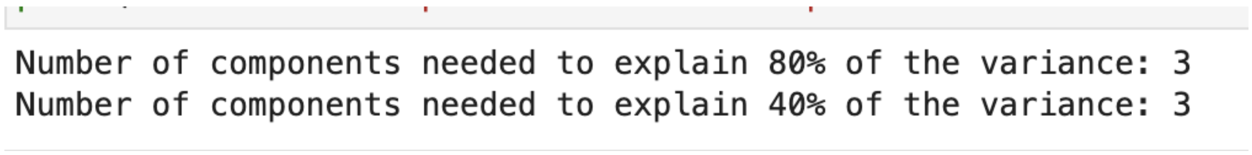
```
PCA Result:
        PC1       PC2
0 -0.302751  0.061533
1 -0.297577  0.017829
2 -0.301649  0.040960
3 -0.171778 -0.020446
4 -0.229802  0.044387
```

This reduces the matrix dimension to the k-value dimension.



This graph uses the PCA matrix from the previous part. Each point is colored based on the anime genre; additionally, not every point is on this graph since the matrix from Part 1 uses a subset of 100,000 rows. Looking at the graph, most of the values are clustered between 0 and 50 for both PC1 and PC2, which is reasonable because the mean-centered values in the transposed matrix are very small. There appear to be no significant patterns, despite the cluster on the far left containing the majority of the values in the transposed matrix. Overall, the graph looks small but reasonable based on the values received from the previous parts.

To find the number of components needed for 80% and 40% of the explained variance, the cumsum() function is used to return the cumulative sum for the explained variance ratio. This will be used to find the number of components required by the explained variance percentages.



```
Number of components needed to explain 80% of the variance: 3
Number of components needed to explain 40% of the variance: 3
```

This shows the number of principal components needed to explain 80% and 40% of the variance. In this case, it reports that three principal components are needed for both 80% and 40% of the variance. Given that 80% and 40% of the variation, respectively, cannot be explained by just two principal components, this indicates that the data's internal dimensionality is higher than that of the 2D depiction made with k = 2. It's possible that the visualization created with just two principal components (k = 2) exaggerates the underlying structure by failing to capture enough variance in the data. A more accurate depiction of the variance in the data can be achieved by increasing the number of primary components, although doing so may result in a higher-dimensional space that is difficult to visualize. The compromise between dimensionality reduction and variance preservation should be taken into account. Hence, the graph would show a stronger variance if the component was 3 in this case.

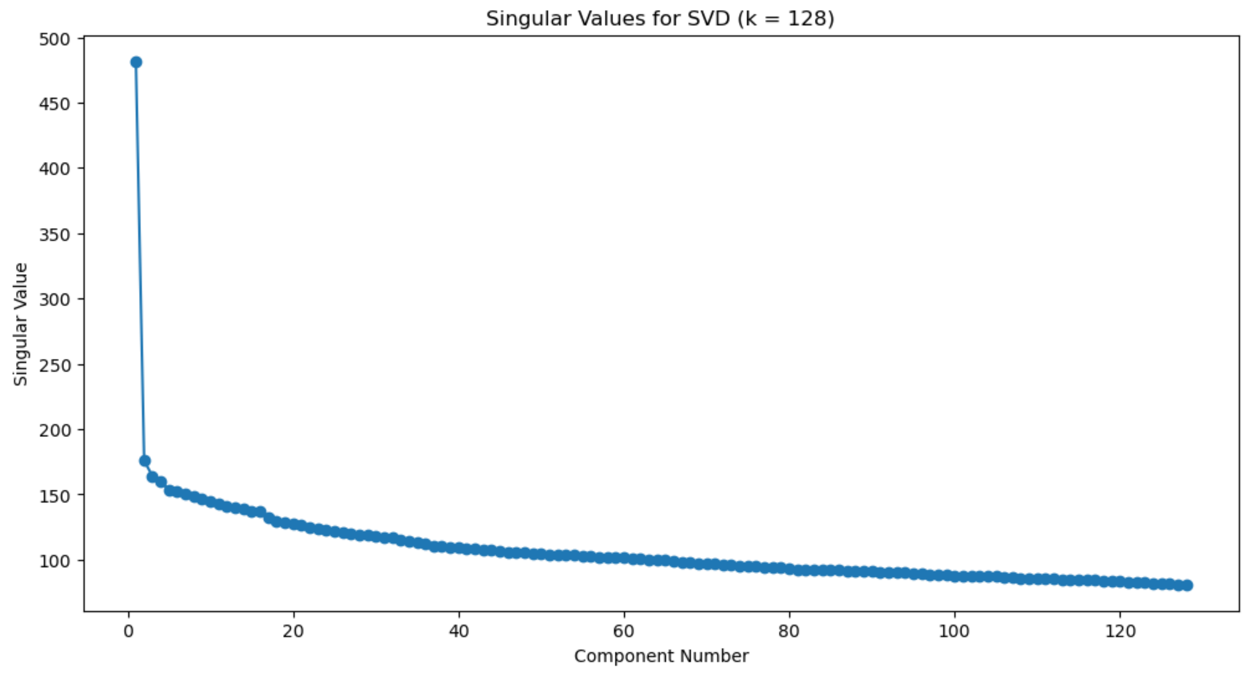
4 Singular Value Decomposition

Using SVD with k-value = 128, the graph is displayed below:

k_svd = 128 - > k-value

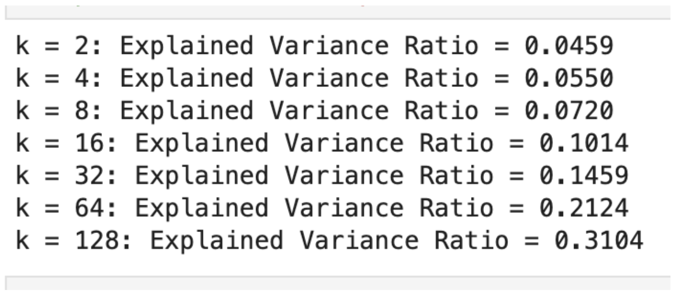
svd = TruncatedSVD(n_components=k_svd, random_state=42) # create the svd

svd.fit(user_anime_matrix) # fit the user anime matrix inside the svd



Observing the graph, only the first component has a high singular value, while the remaining component numbers have smaller values. After component one, the singular values gradually begin to decrease, and when k = 128, the value is below 50.

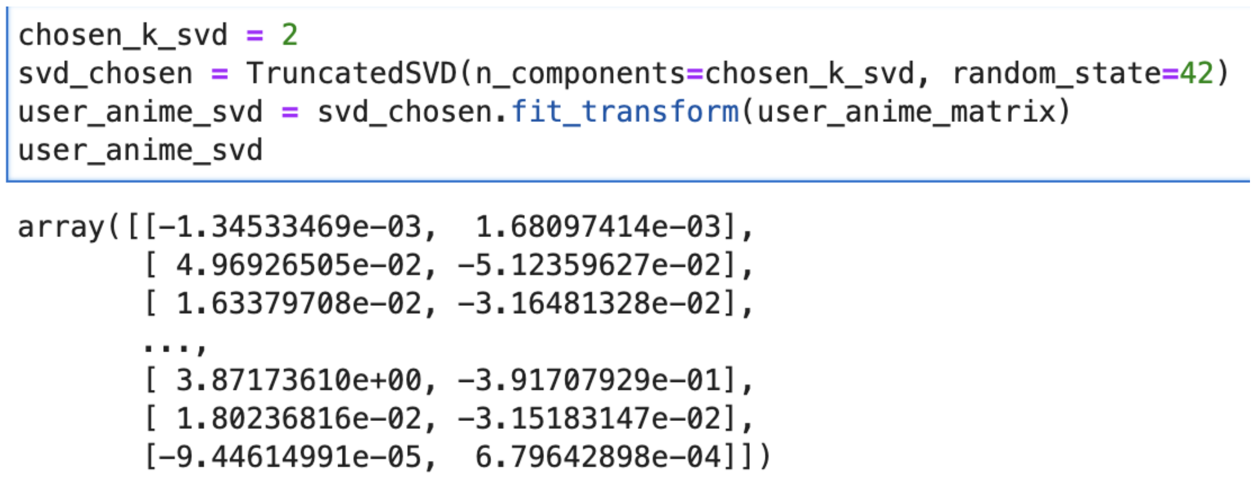
Now moving on to the next part, which is finding the explained variance ratio sum for each value of k. Create a new list for the new explained variance ratio values. Next, follow the same steps for SVD as in the previous part, but instead of creating a graph, append the explained variance ratio sum for each k-value to the new list. Finally, print the list out, which will display the explained variance ratio sum for each k-value.



```
k = 2: Explained Variance Ratio = 0.0459
k = 4: Explained Variance Ratio = 0.0550
k = 8: Explained Variance Ratio = 0.0720
k = 16: Explained Variance Ratio = 0.1014
k = 32: Explained Variance Ratio = 0.1459
k = 64: Explained Variance Ratio = 0.2124
k = 128: Explained Variance Ratio = 0.3104
```

There is an inverse connection between explained variance and inertia. Lower k-means inertia values indicate tighter, more distinct clusters, whereas higher explained variance suggests a better representation of the data. An ideal k would strike a compromise between preserving compact clusters and capturing enough variance. How well the reduced dimensions capture the total variance should be the basis for choosing k in SVD if dimensionality reduction and visualization are the main objectives. The selection of k in K-Means should produce compact and well-separated clusters if the main objective is clustering. For a given k, a higher explained variance ratio in SVD indicates that a greater proportion of the total variance is captured by the representation in that reduced-dimensional space. This may validate the selection of k, suggesting that the chosen dimensionality offers a more illuminating depiction of the data. In conclusion, even if there is a connection between the explained variance ratio in SVD and the inertia values in K-Means, the particular goals of the analysis, such as clustering or dimensionality reduction, should lead to the choice of k. The quality of the representation or clustering structure is shown through the complimentary insights offered by the two measures.

Applying SVD with k = 2 and transforming the data below

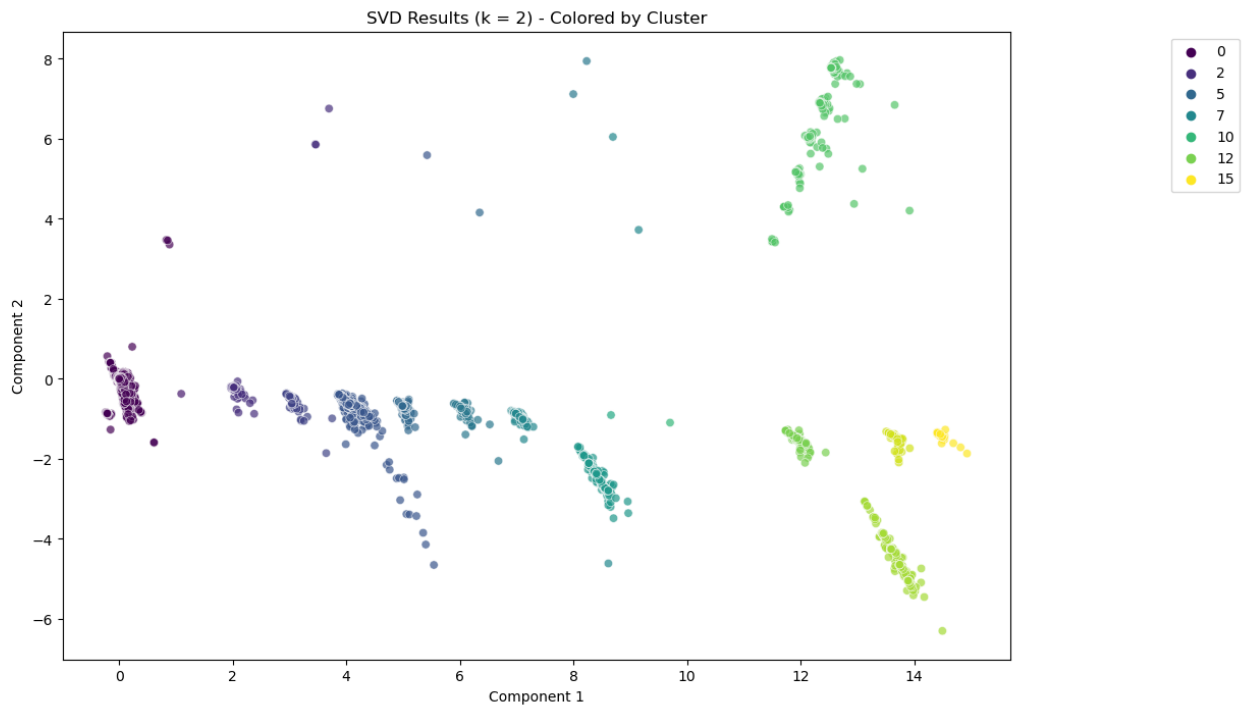


```
chosen_k_svd = 2
svd_chosen = TruncatedSVD(n_components=chosen_k_svd, random_state=42)
user_anime_svd = svd_chosen.fit_transform(user_anime_matrix)
user_anime_svd
```

```
array([[-1.34533469e-03,  1.68097414e-03],
       [ 4.96926505e-02, -5.12359627e-02],
       [ 1.63379708e-02, -3.16481328e-02],
       ...,
       [ 3.87173610e+00, -3.91707929e-01],
       [ 1.80236816e-02, -3.15183147e-02],
       [-9.44614991e-05,  6.79642898e-04]])
```

The result above is using SVD with k = 2 to transform the matrix, showing an array with 2 columns worth of singular values.

Here are the plotted results below, colored by each of the users with cluster memberships from the PCA results graph above.



Based on the SVD transformation, the scatter plot provides a visual representation of the users in a reduced-dimensional space (k = 2). Based on the cluster membership determined using k-means clustering, each point is colored differently. The plot shows that the users in the k-means determine whether the status clusters are close to each other in the SVD-transformed space. In this case, the clusters are very close, which implies that SVD and k-means work well together to identify and classify user patterns. There appears to be tons of overlap, which shows the graph is accurate. Comparing this graph to the PCA graph, there is a huge difference, as applying SVD presents more clusters and spreads based on the user.