Q. Can we identify users who have similar viewing habits and recommend anime based on these similarities?

To answer this I am trying to see if I can create cluster of users based on viewing attributes: Completed(Number of animes completed), Rewatched(Number of animes rewatched), and Watching(Number of animes a user is currently watching). Based on this clustering we can suggest anime to a user based on the animes being seen by the other users in the same cluster.

The algorithm being used now is HDBSCAN, it is a robust algorithm and works well when we do not know the number of clusters.

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import MultiLabelBinarizer

data = pd.read_csv('../../joined_datasets/joined_rating_dataset.csv')
cleaned_dataset =
pd.read_csv("../../cleaned_datasets/users_details_dataset_cleaned.csv"
)
data.dropna()
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5207686 entries, 0 to 5207685
Data columns (total 21 columns):
 #   Column            Dtype
---  ------            -----
 0   Unnamed: 0        int64
 1   user_id           int64
 2   anime_id          int64
 3   rating            int64
 4   Gender            object
 5   Location          object
 6   Birthday_Date     object
 7   Joined_Date       object
 8   Age_Join          float64
 9   Episodes Watched  float64
 10  Age               float64
 11  Name              object
 12  Genres            object
 13  Type              object
 14  Start Date        object
 15  End Date          object
 16  Studios           object
 17  Source            object
 18  Rank              object
 19  Episodes          float64
 20  Episodes_Norm     float64
```

```
dtypes: float64(5), int64(4), object(12)
memory usage: 834.4+ MB
```

```
cleaned_dataset.info()
cleaned_dataset = cleaned_dataset.dropna()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 41486 entries, 0 to 41485
Data columns (total 20 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   Unnamed: 0        41486 non-null  int64  
 1   Mal ID            41486 non-null  int64  
 2   Username          41486 non-null  object 
 3   Gender            40316 non-null  object 
 4   Birthday          41486 non-null  object 
 5   Location          41486 non-null  object 
 6   Joined            41486 non-null  object 
 7   Days Watched      41485 non-null  float64
 8   Mean Score        41485 non-null  float64
 9   Watching          41485 non-null  float64
 10  Completed         41485 non-null  float64
 11  On Hold           41485 non-null  float64
 12  Dropped           41485 non-null  float64
 13  Plan to Watch     41485 non-null  float64
 14  Total Entries     41485 non-null  float64
 15  Rewatched         41485 non-null  float64
 16  Episodes Watched  41485 non-null  float64
 17  Birthday_Date     41486 non-null  object 
 18  Joined_Date       41486 non-null  object 
 19  Age_Join          41486 non-null  float64
dtypes: float64(11), int64(2), object(7)
memory usage: 6.3+ MB
```

```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.decomposition import PCA
```

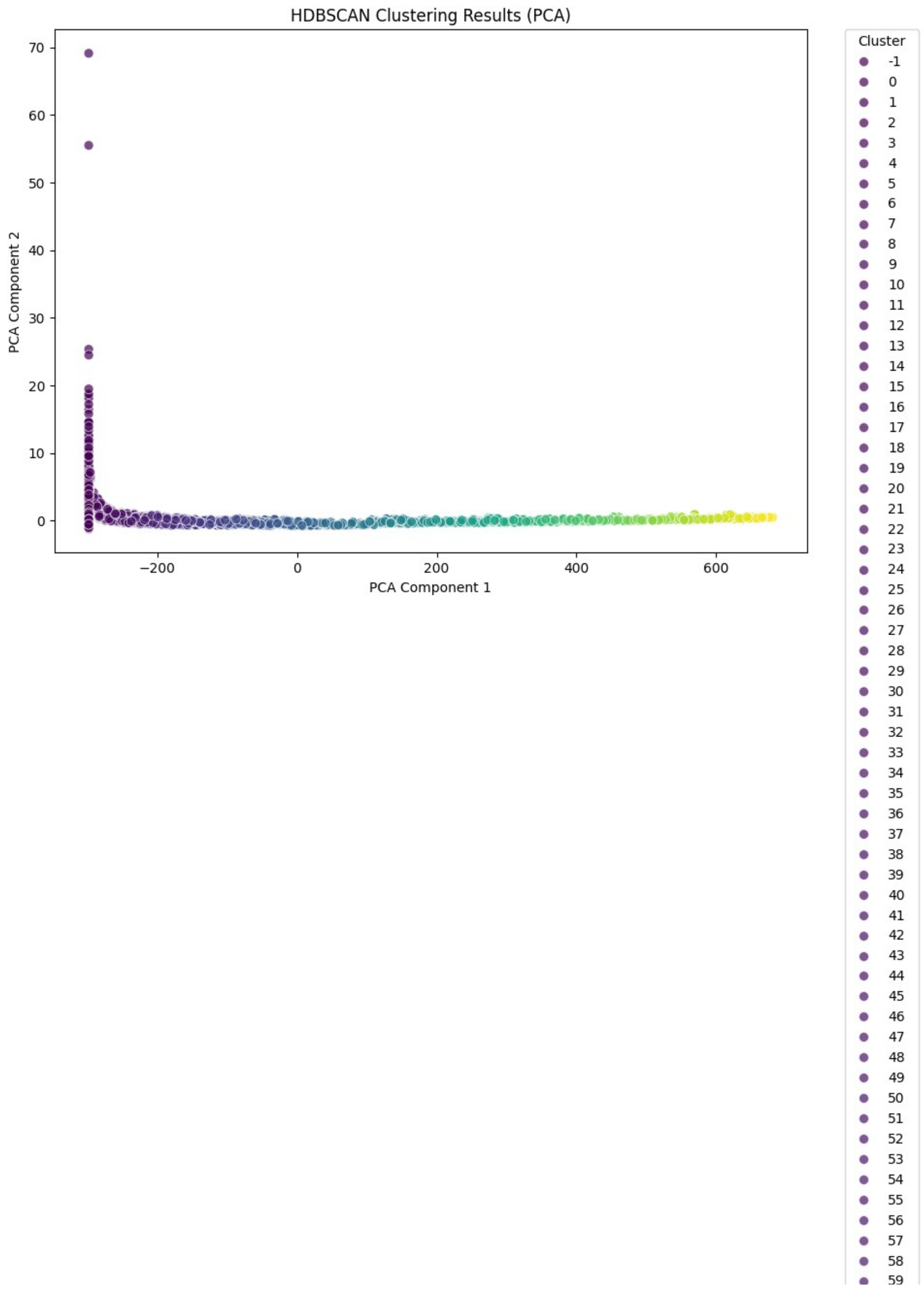
Standardized all data in data set

```
data_filtered = cleaned_dataset[['Mal ID', 'Completed', 'Rewatched', 
'Watching']]
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
data_filtered_scaled = 
scaler.fit_transform(data_filtered[['Completed', 'Rewatched', 
'Watching']])
```

```
# Adding the user_id back after scaling (to keep track of users)
data_filtered_scaled = pd.DataFrame(data_filtered_scaled,
columns=['Completed', 'Rewatched', 'Watching'])
```

First pass with cluster size and samples. We can see that there are many data with cluster as -1. This means that our data was not clustered correctly. We will need to update our inputs min_samples and min_cluster_size to reduce this.

```
import hdbscan

# Apply HDBSCAN clustering
hdbscan_model = hdbscan.HDBSCAN(min_samples=5, min_cluster_size=10,
metric='euclidean')
data_filtered_scaled['Cluster'] =
hdbscan_model.fit_predict(data_filtered_scaled)

# Add the cluster info back to the original data
cleaned_dataset['Cluster'] = data_filtered_scaled['Cluster']
cluster_counts = cleaned_dataset['Cluster'].value_counts()
print(cluster_counts)

Cluster
-1.0      16214
 774.0      374
 350.0      351
 204.0      277
 209.0      211
          ...
 414.0        9
 92.0         9
 272.0        9
 52.0         9
 761.0        9
Name: count, Length: 980, dtype: int64

pca = PCA(n_components=2)
pca_result = pca.fit_transform(data_filtered_scaled)

# Plot the clusters
plt.figure(figsize=(10, 7))
sns.scatterplot(x=pca_result[:, 0], y=pca_result[:, 1],
hue=data_filtered_scaled['Cluster'], palette="viridis", s=50,
alpha=0.7, legend='full')
plt.title("HDBSCAN Clustering Results (PCA)")
plt.xlabel("PCA Component 1")
plt.ylabel("PCA Component 2")
plt.legend(title='Cluster', loc='best', bbox_to_anchor=(1.05, 1),
borderaxespad=0.)
plt.show()
```

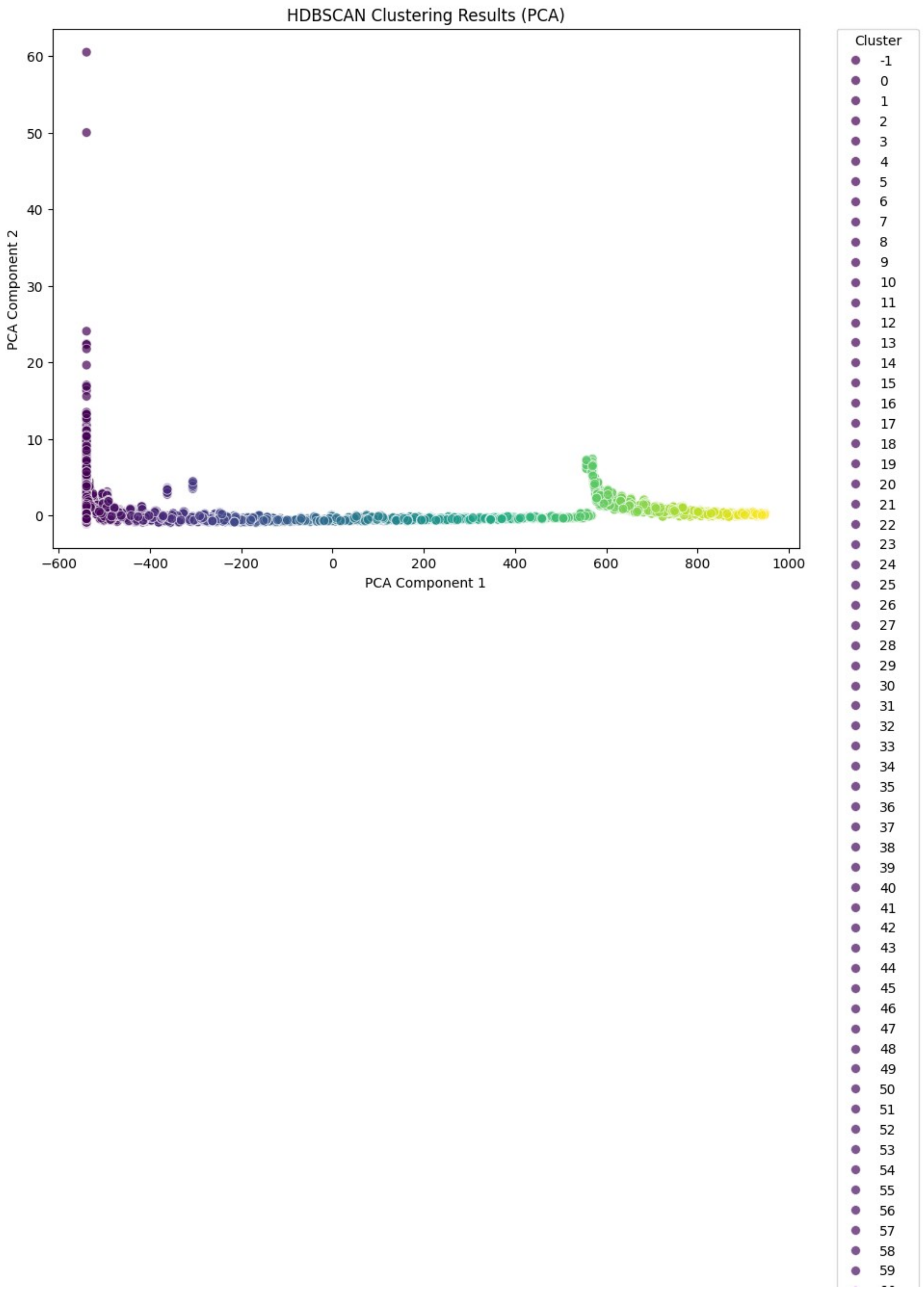

HDBSCAN Clustering Results (PCA)
PCA Component 1
PCA Component 2
−200
0
200
400
600
0
10
20
30
40
50
60
70
Cluster
-1
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59

Second pass with updated values of min_samples=3 and min_cluster_size=8 and for distance metric using manhattan instead of euclidean The negative values have reduced quite a lot after this. ~40%

```python
# Using updated values 3, 10, manhattan


hdbscan_model = hdbscan.HDBSCAN(min_samples=3, min_cluster_size=8,
metric='manhattan')
data_filtered_scaled['Cluster'] =
hdbscan_model.fit_predict(data_filtered_scaled)

# Add the cluster info back to the original data
cleaned_dataset['Cluster'] = data_filtered_scaled['Cluster']

cluster_counts = cleaned_dataset['Cluster'].value_counts()
print(cluster_counts)
```

```
Cluster
-1.0      10182
 998.0      374
 765.0      351
 999.0      182
 1278.0     175
          ...
 651.0        7
 1403.0       6
 1450.0       6
 1080.0       6
 1459.0       6
Name: count, Length: 1489, dtype: int64
```

```python
pca = PCA(n_components=2)
pca_result = pca.fit_transform(data_filtered_scaled)

# Plot the clusters
plt.figure(figsize=(10, 7))
sns.scatterplot(x=pca_result[:, 0], y=pca_result[:, 1],
hue=data_filtered_scaled['Cluster'], palette="viridis", s=50,
alpha=0.7, legend='full')
plt.title("HDBSCAN Clustering Results (PCA)")
plt.xlabel("PCA Component 1")
plt.ylabel("PCA Component 2")
plt.legend(title='Cluster', loc='best', bbox_to_anchor=(1.05, 1),
borderaxespad=0.)
plt.show()
```

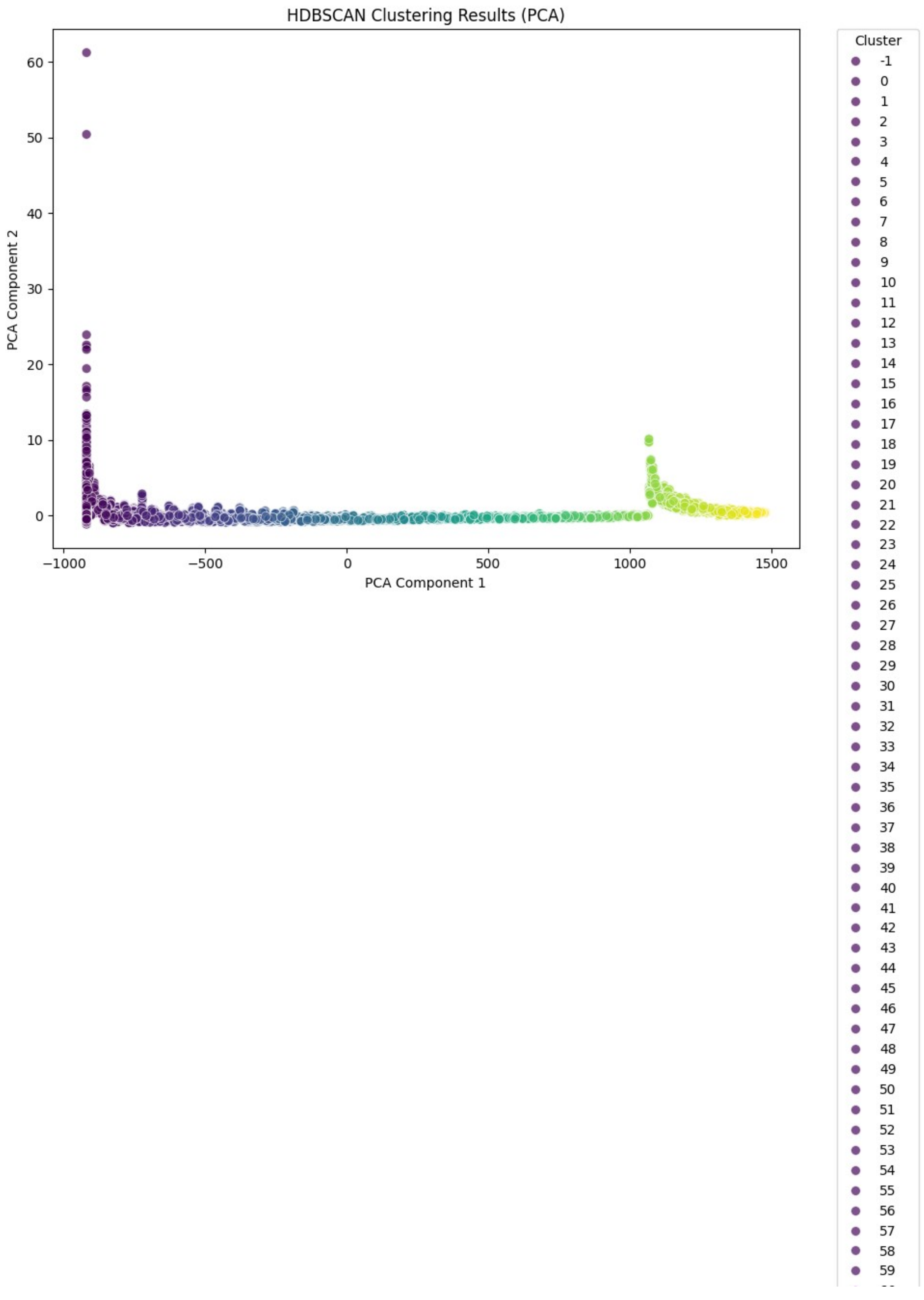

HDBSCAN Clustering Results (PCA)
PCA Component 1
PCA Component 2
−600
−400
−200
0
200
400
600
800
1000
0
10
20
30
40
50
60
Cluster
-1
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59

Second pass with updated values of min_samples=3 and min_cluster_size=5 and for distance metric using manhattan instead of euclidean The negative values have reduced quite a lot after this. ~30%

```
hdbscan_model = hdbscan.HDBSCAN(min_samples=3, min_cluster_size=5,
metric='manhattan')
data_filtered_scaled['Cluster'] =
hdbscan_model.fit_predict(data_filtered_scaled)

# Add the cluster info back to the original data
cleaned_dataset['Cluster'] = data_filtered_scaled['Cluster']

cluster_counts = cleaned_dataset['Cluster'].value_counts()
print(cluster_counts)
```

```
Cluster
-1.0      6858
 1856.0    374
 1779.0    351
 1857.0    182
 345.0     175
          ...
 565.0       4
 1054.0      4
 2394.0      4
 2014.0      4
 1841.0      3
Name: count, Length: 2398, dtype: int64
```

```
pca = PCA(n_components=2)
pca_result = pca.fit_transform(data_filtered_scaled)

# Plot the clusters
plt.figure(figsize=(10, 7))
sns.scatterplot(x=pca_result[:, 0], y=pca_result[:, 1],
hue=data_filtered_scaled['Cluster'], palette="viridis", s=50,
alpha=0.7, legend='full')
plt.title("HDBSCAN Clustering Results (PCA)")
plt.xlabel("PCA Component 1")
plt.ylabel("PCA Component 2")
plt.legend(title='Cluster', loc='best', bbox_to_anchor=(1.05, 1),
borderaxespad=0.)
plt.show()
```


HDBSCAN Clustering Results (PCA)
PCA Component 1
PCA Component 2
−1000
−500
0
500
1000
1500
0
10
20
30
40
50
60
Cluster
-1
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59

I cannot reduce the size of the cluster or minimum samples anymore, so as last resort we'll add all the data with negative values to their respective closest clusters

```
import numpy as np
from sklearn.metrics.pairwise import manhattan_distances
data_filtered_scaled['Mal ID'] = cleaned_dataset['Mal ID']
cluster_centroids =
data_filtered_scaled.groupby('Cluster').mean().values


distances = manhattan_distances(data_filtered_scaled.drop('Mal ID',
axis=1), cluster_centroids)


outliers = data_filtered_scaled[data_filtered_scaled['Cluster'] == -
1].index
for outlier in outliers:

    nearest_cluster = np.argmin(distances[outlier])
    data_filtered_scaled.at[outlier, 'Cluster'] = nearest_cluster


cleaned_dataset['Cluster'] = data_filtered_scaled['Cluster']

cluster_counts = cleaned_dataset['Cluster'].value_counts()
print(cluster_counts)
```

```
Cluster
2193.0    6864
1856.0     374
1779.0     351
1857.0     182
345.0      175
          ...
1723.0       4
1197.0       4
1902.0       4
1812.0       4
1841.0       3
Name: count, Length: 2397, dtype: int64
```

```
pca = PCA(n_components=2)
pca_result = pca.fit_transform(data_filtered_scaled.drop('Mal ID',
axis=1))

# Plot the clusters
plt.figure(figsize=(10, 7))
sns.scatterplot(x=pca_result[:, 0], y=pca_result[:, 1],
hue=data_filtered_scaled['Cluster'], palette="viridis", s=50,
alpha=0.7, legend='full')
```

```
plt.title("HDBSCAN Clustering Results (PCA)")
plt.xlabel("PCA Component 1")
plt.ylabel("PCA Component 2")
plt.legend(title='Cluster', loc='best', bbox_to_anchor=(1.05, 1),
borderaxespad=0.)
plt.show()
```

```
----------------------------------------------------------------------
-----
ValueError                                Traceback (most recent call
last)
Cell In[113], line 2
      1 pca = PCA(n_components=2)
----> 2 pca_result = pca.fit_transform(data_filtered_scaled)
      4 # Plot the clusters
      5 plt.figure(figsize=(10, 7))

File
~/Repos/MS/sem1/.venv/lib/python3.12/site-packages/sklearn/utils/_set_
output.py:316, in _wrap_method_output.<locals>.wrapped(self, X, *args,
**kwargs)
    314 @wraps(f)
    315 def wrapped(self, X, *args, **kwargs):
--> 316     data_to_wrap = f(self, X, *args, **kwargs)
    317     if isinstance(data_to_wrap, tuple):
    318         # only wrap the first output for cross decomposition
    319         return_tuple = (
    320             _wrap_data_with_container(method, data_to_wrap[0],
X, self),
    321             *data_to_wrap[1:],
    322         )

File
~/Repos/MS/sem1/.venv/lib/python3.12/site-packages/sklearn/base.py:147
3, in _fit_context.<locals>.decorator.<locals>.wrapper(estimator,
*args, **kwargs)
   1466     estimator._validate_params()
   1468 with config_context(
   1469     skip_parameter_validation=(
   1470         prefer_skip_nested_validation or
global_skip_validation
   1471     )
   1472 ):
-> 1473     return fit_method(estimator, *args, **kwargs)

File
~/Repos/MS/sem1/.venv/lib/python3.12/site-packages/sklearn/decompositi
on/_pca.py:474, in PCA.fit_transform(self, X, y)
    451 @_fit_context(prefer_skip_nested_validation=True)
    452 def fit_transform(self, X, y=None):
```

```
    453     """Fit the model with X and apply the dimensionality
reduction on X.
    454 
    455     Parameters
   (...)
    472     C-ordered array, use 'np.ascontiguousarray'.
    473     """
--> 474     U, S, _, X, x_is_centered, xp = self._fit(X)
    475     if U is not None:
    476         U = U[:, : self.n_components_]

File
~/Repos/MS/sem1/.venv/lib/python3.12/site-packages/sklearn/decompositi
on/_pca.py:511, in PCA._fit(self, X)
    501     raise ValueError(
    502         "PCA with svd_solver='arpack' is not supported for
Array API inputs."
    503     )
    505 # Validate the data, without ever forcing a copy as any solver
that
    506 # supports sparse input data and the `covariance_eigh` solver
are
    507 # written in a way to avoid the need for any inplace
modification of
    508 # the input data contrary to the other solvers.
    509 # The copy will happen
    510 # later, only if needed, once the solver negotiation below is
done.
--> 511 X = self._validate_data(
    512     X,
    513     dtype=[xp.float64, xp.float32],
    514     force_writeable=True,
    515     accept_sparse=("csr", "csc"),
    516     ensure_2d=True,
    517     copy=False,
    518 )
    519 self._fit_svd_solver = self.svd_solver
    520 if self._fit_svd_solver == "auto" and issparse(X):

File
~/Repos/MS/sem1/.venv/lib/python3.12/site-packages/sklearn/base.py:633
, in BaseEstimator._validate_data(self, X, y, reset,
validate_separately, cast_to_ndarray, **check_params)
    631         out = X, y
    632 elif not no_val_X and no_val_y:
--> 633     out = check_array(X, input_name="X", **check_params)
    634 elif no_val_X and not no_val_y:
    635     out = _check_y(y, **check_params)

File
```

```
~/Repos/MS/sem1/.venv/lib/python3.12/site-packages/sklearn/utils/valid
ation.py:1064, in check_array(array, accept_sparse,
accept_large_sparse, dtype, order, copy, force_writeable,
force_all_finite, ensure_2d, allow_nd, ensure_min_samples,
ensure_min_features, estimator, input_name)
   1058     raise ValueError(
   1059         "Found array with dim %d. %s expected <= 2."
   1060         % (array.ndim, estimator_name)
   1061     )
   1063 if force_all_finite:
-> 1064     _assert_all_finite(
   1065         array,
   1066         input_name=input_name,
   1067         estimator_name=estimator_name,
   1068         allow_nan=force_all_finite == "allow-nan",
   1069     )
   1071 if copy:
   1072     if _is_numpy_namespace(xp):
   1073         # only make a copy if `array` and `array_orig` may
share memory`

File
~/Repos/MS/sem1/.venv/lib/python3.12/site-packages/sklearn/utils/valid
ation.py:123, in _assert_all_finite(X, allow_nan, msg_dtype,
estimator_name, input_name)
    120 if first_pass_isfinite:
    121     return
--> 123 _assert_all_finite_element_wise(
    124     X,
    125     xp=xp,
    126     allow_nan=allow_nan,
    127     msg_dtype=msg_dtype,
    128     estimator_name=estimator_name,
    129     input_name=input_name,
    130 )

File
~/Repos/MS/sem1/.venv/lib/python3.12/site-packages/sklearn/utils/valid
ation.py:172, in _assert_all_finite_element_wise(X, xp, allow_nan,
msg_dtype, estimator_name, input_name)
    155 if estimator_name and input_name == "X" and has_nan_error:
    156     # Improve the error message on how to handle missing
values in
    157     # scikit-learn.
    158     msg_err += (
    159         f"\n{estimator_name} does not accept missing values"
    160         " encoded as NaN natively. For supervised learning,
you might want"
   (...)
```

```
    170         "#estimators-that-handle-nan-values"
    171     )
--> 172 raise ValueError(msg_err)

ValueError: Input X contains NaN.
PCA does not accept missing values encoded as NaN natively. For
supervised learning, you might want to consider
sklearn.ensemble.HistGradientBoostingClassifier and Regressor which
accept missing values encoded as NaNs natively. Alternatively, it is
possible to preprocess the data, for instance by using an imputer
transformer in a pipeline or drop samples with missing values. See
https://scikit-learn.org/stable/modules/impute.html You can find a
list of all estimators that handle NaN values at the following page:
https://scikit-learn.org/stable/modules/impute.html#estimators-that-
handle-nan-values
```

With this we can get similar users and then recommend the top rated animes from these similar users

```

```