Your Name Here: Yuyang Zhang

A4 Unsupervised Learning algorithms (Total 75 points)

1. Feature Preprocessing (Total 5 points)

1.1 Import numpy, matplotlib, pandas and seaborn (1pt)

```
# TODO
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
```

1.2 Load the dataset from given *.csv file using *pandas* (1pt)

```
# uncomment the following line if you are running this code in google colab and have uploaded the dataset to your drive
from google.colab import drive
drive.mount('/content/drive')

# TODO
df = pd.read_csv("/content/drive/MyDrive/Wine_Quality_Data.csv")
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

1.3 We want to use only the numeric features. So drop the 'color' column from the dataframe. (1pt)

```
# TODO
df = df.drop('color', axis=1)
```

1.4 Scale the features using Scikitlearn's StandardScaler library (2pt)

```
# TODO
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaled_df = scaler.fit_transform(df)
scaled_df = pd.DataFrame(scaled_df, columns=df.columns)
```

2. KMeans Clustering with PCA selection (Total 25 points)

2.1 Import PCA from scikitlearn's decomposition library. (3pt)

Use fit_transform method on the scaled data to get the PCA transformed data.

```
# TODO
from sklearn.decomposition import PCA
pca = PCA()
pca_df = pca.fit_transform(scaled_df)
```

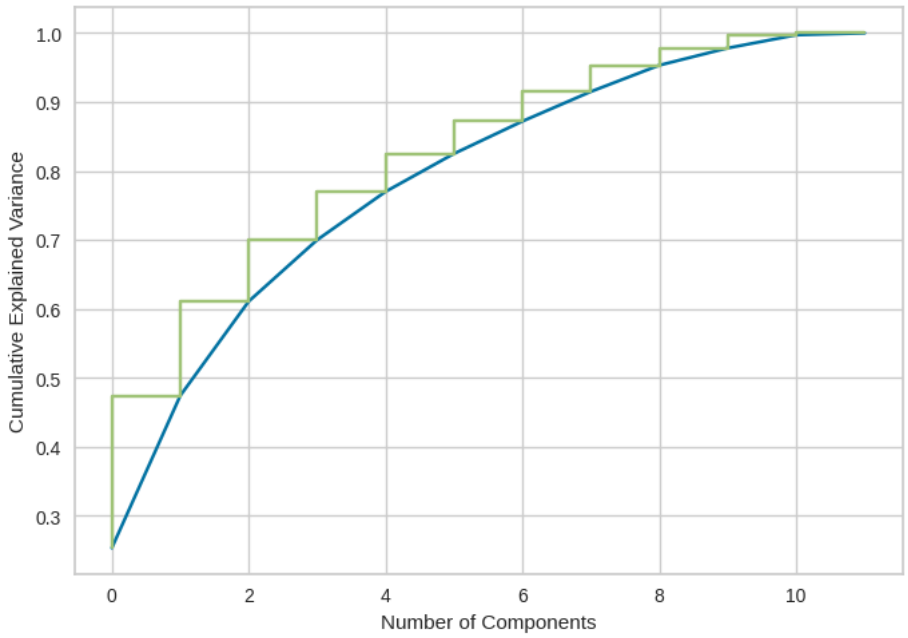
2.2 Plot the cumulative sum of explained_variance_ratio_. You can retrieve this quantity from the PCA-transformed data in the previous step. (4pt)

This plot should show the set of features along X axis and proportion of variance explained by each of the features along Y axis. You can use matplotlib's plot and step function for this.

```
# TODO
explained_var = pca_df.var(axis=0) / np.sum(pca_df.var(axis=0))
```

```
cumulative_var = np.cumsum(explained_var)

plt.plot(cumulative_var)
plt.step(range(len(cumulative_var)), cumulative_var)
plt.xlabel('Number of Components')
plt.ylabel('Cumulative Explained Variance')
plt.show()
```



2.3 How many features are required to capture at least 90% of the variance? Drop the other features. (3pt)

You can get this number visually from the cumulative sum of explained_variance_ratio_ plot in the previous step. Alternatively, you can use numpy's *argwhere* function to find the index of the first element in the cumulative sum array that is greater than 0.9.

```
# TODO
n_components = np.argwhere(cumulative_var > 0.9)[0, 0]
print(n_components)
pca = PCA(n_components=n_components).fit(pca_df)
pca_df = pca.transform(pca_df)
```

```
7
```

2.4 Import KElbowVisualizer from yellowbrick.cluster library and fit the data to it. Visualize the elbow curve and find the optimal number of clusters. (6pt)

KElbowVisualizer is a useful visualization tool for using Elbow method with K-Means clustering algorithm. The official documentation and example can be found here: https://www.scikit-yb.org/en/latest/api/cluster/elbow.html

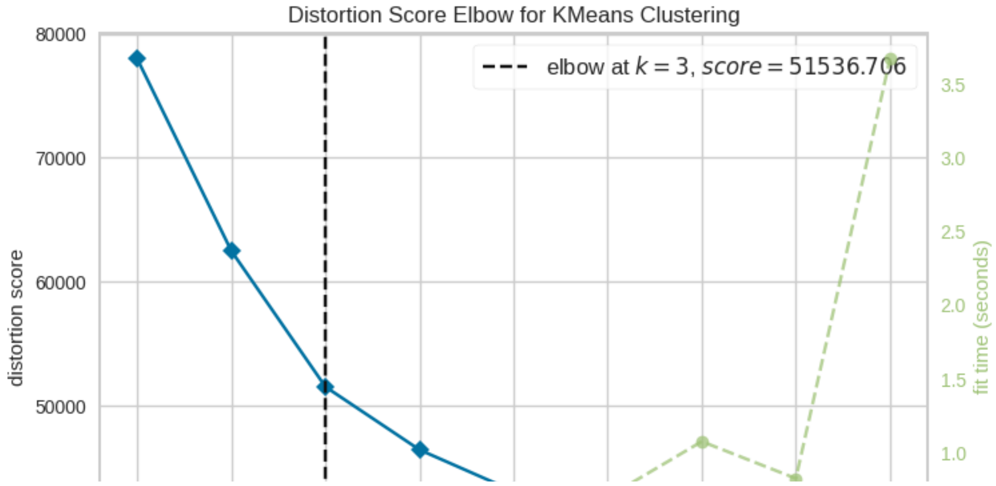
```
# TODO
from yellowbrick.cluster import KElbowVisualizer
from sklearn.cluster import KMeans

model = KMeans(random_state=42)
visualizer = KElbowVisualizer(model, k=(1, 10))

# visualizer.fit(pca_df)
visualizer.fit(scaled_df)

visualizer.show()
```

```
/usr/local/lib/python3.9/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10 to 'auto
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10 to 'auto
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10 to 'auto
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10 to 'auto
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10 to 'auto
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10 to 'auto
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10 to 'auto
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10 to 'auto
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10 to 'auto
  warnings.warn(
```



2.5 Instantiate a KMeans object with the optimal number of clusters (from previous step) and fit the data to it. (2pt)

```
# TODO
from sklearn.cluster import KMeans

kmeans = KMeans(n_clusters=3)

# kmeans.fit(pca_df)
kmeans.fit(scaled_df)
```

```
/usr/local/lib/python3.9/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10 to 'auto
  warnings.warn(
▾ KMeans
KMeans(n_clusters=3)
```

2.6 Get the labels from the fitted KMeans object (*labels_* method). Import *silhouette_score* from scikitlearn's metrics library. Print (1) the value counts of unique class labels in percentage, and (2) print the corresponding silhouette score. (1+1+2+3 =7pts)

```
# TODO
from sklearn.metrics import silhouette_score

labels = kmeans.labels_

vc = pd.Series(labels).value_counts(normalize=True) * 100
print("Value counts of unique class labels in percentage:")
print(vc)

# score = silhouette_score(pca_df, labels)
score = silhouette_score(scaled_df, labels)
print("Silhouette score:", score)
```

```
Value counts of unique class labels in percentage:
1    44.789903
0    30.290903
2    24.919193
```

```
dtype: float64
Silhouette score: 0.21662522573894066
```

3. KMeans Clustering without PCA selection (Total 15 points)

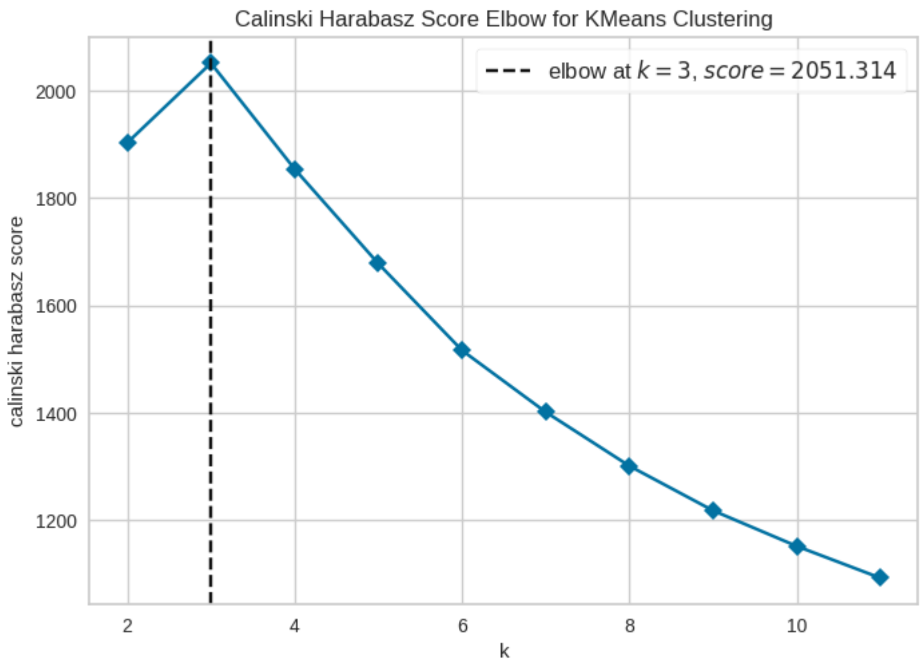
3.1 Use KElbowVisualizer to find the optimal number of clusters for the data without PCA selection (ie the data with all the features). Also plot the score (similar to step 2.4) (3pt)

```
# TODO
model = KMeans()
visualizer = KElbowVisualizer(model, k=(2,12), metric='calinski_harabasz', timings=False)

# visualizer.fit(scaled_df)
visualizer.fit(pca_df)

visualizer.show()
```

```
/usr/local/lib/python3.9/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10 to 'auto
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10 to 'auto
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10 to 'auto
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10 to 'auto
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10 to 'auto
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10 to 'auto
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10 to 'auto
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10 to 'auto
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10 to 'auto
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10 to 'auto
  warnings.warn(
```




```
<Axes: title={'center': 'Calinski Harabasz Score Elbow for KMeans Clustering'}, xlabel='k', ylabel='calinski harabasz score'>
```

3.2 Instantiate a KMeans object with the optimal number of clusters (from previous step) and fit the data (without PCA selection) to it. (2pt)

```
# TODO
kmeans2 = KMeans(3)

# kmeans2.fit(scaled_df)
kmeans2.fit(pca_df)
```

```
/usr/local/lib/python3.9/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10 to 'auto
  warnings.warn(
```

KMeans(n_clusters=3)

3.3 Retrieve the class labels from this KMeans object (from model's *labels_* method). Print (1) the value counts of unique class labels in percentage, and (2) print the corresponding silhouette score for the fitted data (without PCA selection) (2+2+3 =7pt)

```
# TODO
labels2 = kmeans2.labels_

counts = pd.Series(labels2).value_counts(normalize=True) * 100
print(counts)

# score2 = silhouette_score(scaled_df, labels2)
score2 = silhouette_score(pca_df, labels2)
print("Silhouette Score:", score2)
```

```
1    44.435893
2    30.552563
0    25.011544
dtype: float64
Silhouette Score: 0.2516982820572182
```

3.4 Discussion: Did PCA selection help in clustering (did it yield higher silhouette score)? Why or why not? (2-3 sentences) (3pt)

~ # TODO ~

According to the silhouette scores above, the clustering with PCA has about 0.25 and the clustering without PCA has about 0.21. This shows that using PCA for feature selection slightly improves the clustering performance. This is due to the fact that PCA can help clustering by reducing the dimensionality of the data while preserving most of the variance, making clustering algorithms work better in lower dimensional spaces.

4. Hierarchical (agglomerative) clustering (Total 10 points)

4.1 Import dendogram and linkage from scipy.cluster.hierarchy library. Use the *linkage* function to fit the data and plot the dendogram. (5pt)

An example is provided in the lecture slide.

Official Documentation: https://docs.scipy.org/doc/scipy/reference/cluster.hierarchy.html

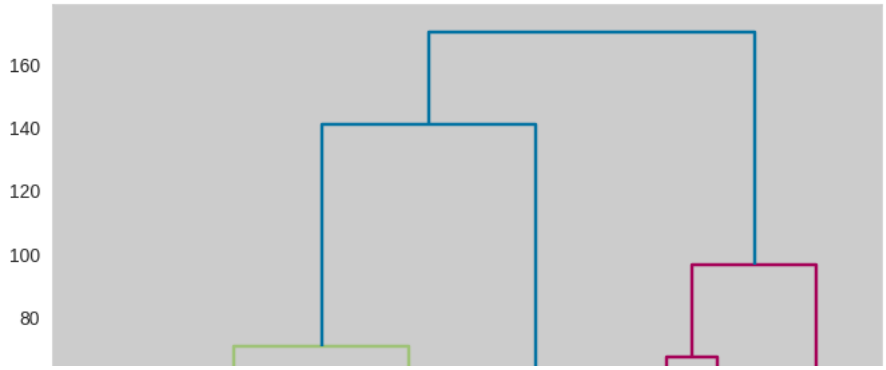
With the *linkage* function, you can use different linkage methods. For this assignment you can use 'ward' method.

(https://docs.scipy.org/doc/scipy/reference/generated/scipy.cluster.hierarchy.linkage.html#scipy.cluster.hierarchy.linkage:~:text=method%3D%E2%80%99ward%E2%80%99%20uses,the%20incremental%20algorithm.)

(This can take a while to complete running)

```
# TODO
from scipy.cluster.hierarchy import dendrogram, linkage

# linked = linkage(scaled_df, 'ward')
linked = linkage(pca_df, 'ward')
dendrogram(linked, orientation='top', distance_sort='descending', show_leaf_counts=True)
plt.show()
```



4.2 Import AgglomerativeClustering from sklearn.cluster library. Use the fit_predict method to fit the data and print the corresponding silhouette score. (5pt)

(Use the above dendogram to find a suitable value for the number of clusters (*k*) in AgglomerativeClustering, OR run the algorithm multiple times with different values of *k* and print the *k* with the highest silhouette score)


```
# TODO
from sklearn.cluster import AgglomerativeClustering

# n_clusters=2 has largest score
agg = AgglomerativeClustering(n_clusters=2)

# labels3 = agg.fit_predict(scaled_df)
labels3 = agg.fit_predict(pca_df)

print(pd.Series(labels3).value_counts(normalize=True) * 100)

# score3 = silhouette_score(scaled_df, labels3)
score3 = silhouette_score(pca_df, labels3)
print("Silhouette Score: ", score3)
```

```
0    72.294905
1    27.705095
dtype: float64
Silhouette Score: 0.2698275783475057
```

5. DBSCAN Clustering (Total 20 points)

5.1 Import *NearestNeighbors* from sklearn.neighbors library. Use the *fit* method to fit the data with n_neighbors=4. Then use *kneighbors* method to find the distance of the 4th nearest neighbor for each point. Sort and plot the distances. (6pt)

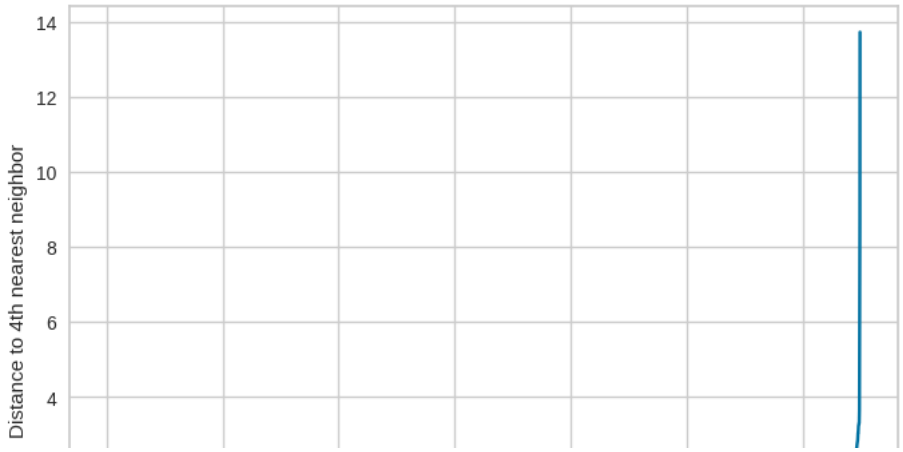
This plots the elbow curve (distance of the 4th nearest neighbor for each point). This is useful for finding the optimal value of epsilon for DBSCAN clustering.

```
# TODO
from sklearn.neighbors import NearestNeighbors

neigh = NearestNeighbors(n_neighbors=4)
# neigh.fit(scaled_df)
neigh.fit(pca_df)

# distances, indices = neigh.kneighbors(scaled_df)
distances, indices = neigh.kneighbors(pca_df)
distances = np.sort(distances[:,3])

plt.plot(distances)
plt.xlabel('Points')
plt.ylabel('Distance to 4th nearest neighbor')
plt.show()
```



5.2 From the distance plot above, estimate the 'elbow' point. The distance at the elbow point will be used as epsilon in the DBSCAN model. (2pt)

Simply write the distance value in the cell below.

Points

~ # TODO ~

2

5.3 Import DBSCAN from sklearn.cluster library. Instantiate a DBSCAN model (with eps set to the elbow point suggested by above plot, and min_samples=10). (2pt)

(You can experiment with several values of eps and min_samples to find the best combination)

```
# TODO
from sklearn.cluster import DBSCAN
dbscan = DBSCAN(eps=2, min_samples=10)
```

5.4 Use the fit method to fit your model to the data. Print (1) the value counts of unique class labels (from model's *labels_* method) in percentage, and (2) print the corresponding silhouette score. (2+2+2=6pt)

```
# TODO
# Get the labels from the model
# dbscan.fit(scaled_df)
dbscan.fit(pca_df)

labels4 = dbscan.labels_

print(pd.Series(labels4).value_counts(normalize=True) * 100)

# score4 = silhouette_score(scaled_df, labels4)
score4 = silhouette_score(pca_df, labels4)

print("Silhouette Score: ", score4)
```

```
 0    98.399261
-1     1.431430
 1     0.169309
dtype: float64
Silhouette Score: 0.4401166439002839
```

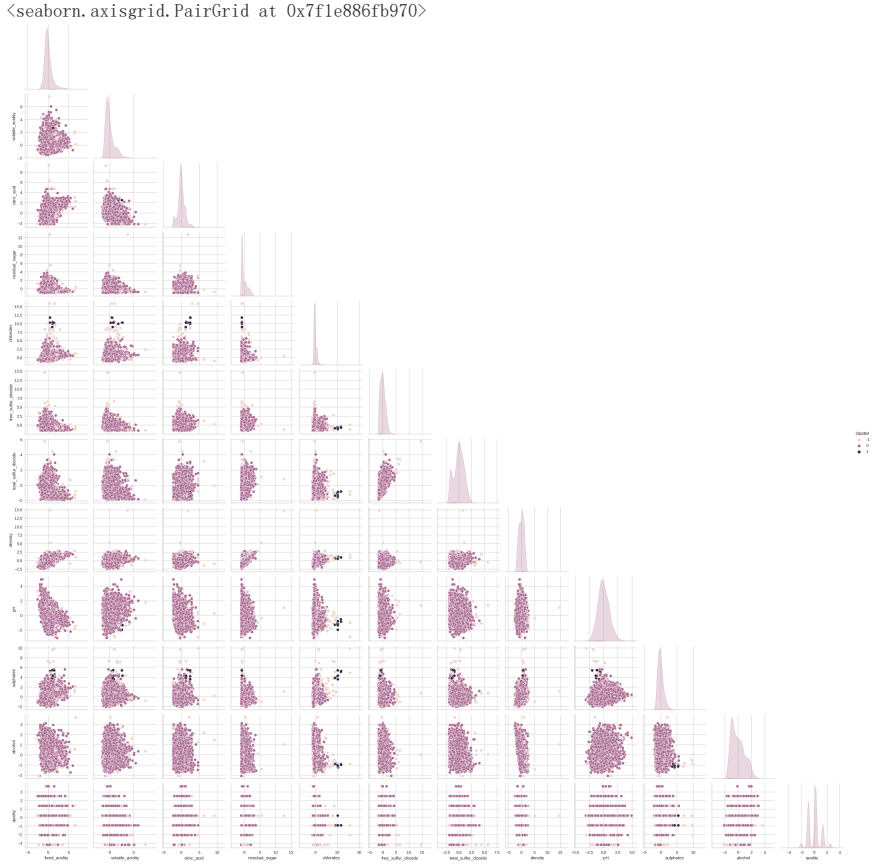
5.5 Use seaborn's pairplot to plot the data with *hue* as the cluster labels obtained from DBSCAN. (2pt)

Set *corner* to *True* for a better visualization.

```
# TODO
scaled_df['cluster'] = labels4
# pca_df['cluster'] = labels4

sns.pairplot(scaled_df, hue='cluster', corner=True)
```

```
<seaborn.axisgrid.PairGrid at 0x7f1e886fb970>
```



6. Discussion: which clustering algorithm performed the best? Why? (2pt)

(2-3 sentences)

~ # TODO ~

The DBSCAN performs the best since it has 0.44 on silhoutte score. This is because DBSCAN is a density-based clustering algorithm that is particularly effective at identifying clusters with irregular shapes and varying densities. This makes it well-suited for datasets that do not have a clear separation between clusters, or where clusters have varying densities, In con k-means and hierarchical clustering algorithms assume that clusters are spherical and have similar densities, which can limit their effectiveness on certain types of datasets.