Exploring Cosine Similarity VS Euclidean Distance Metrics in K-means

Anagha Manikantan Nair Sindhu

Student ID: 23035680

In this tutorial, we'll look at K-Means clustering, a popular unsupervised learning approach, as well as the different distance metrics that are utilized. We will utilize these strategies to categorize customers based on their demographics and purchasing patterns. This tutorial will walk you through the procedures of preprocessing the data, applying the K-Means algorithm with two distinct distance metrics (cosine similarity and Euclidean distance), and visualizing the results with PCA.

What is clustering?

Clustering is an unsupervised machine learning technique that organizes data points into clusters based on their similarities. Unlike supervised learning, clustering does not require labeled data. KMeans is a typical clustering algorithm that minimizes variance within each cluster.

Table of Contents:

1. **Introduction**
2. **Dataset Overview**
3. **Data Preprocessing**
4. **Evaluating the Model: Elbow Method**
5. **Euclidean Distance**
6. **Cosine Similarity**
7. **Dimensionality Reduction with PCA**
8. **Results Visualization**
9. **Conclusion**
10. **Code & GitHub Repository**

1. Introduction

In this tutorial, we will show how to utilize K-Means clustering on a customer dataset with two alternative distance metrics, as well as how to reduce the dataset's dimensionality using PCA for simpler visualization.

K-Means clustering separates data into k groups based on similarities; the distance metric is critical in detecting similar data points and constructing appropriate clusters. In this tutorial, we'll look at how the choice of distance metric Cosine Similarity versus Euclidean Distance influences the clustering process in K-means. After clustering, PCA will allow us to show the high-dimensional data in a 2D scatter plot, making the clustering results easier to grasp.

2. Dataset Overview

The dataset includes 200 records and five features:

CustomerID: A unique identification for each customer (not relevant for clustering).

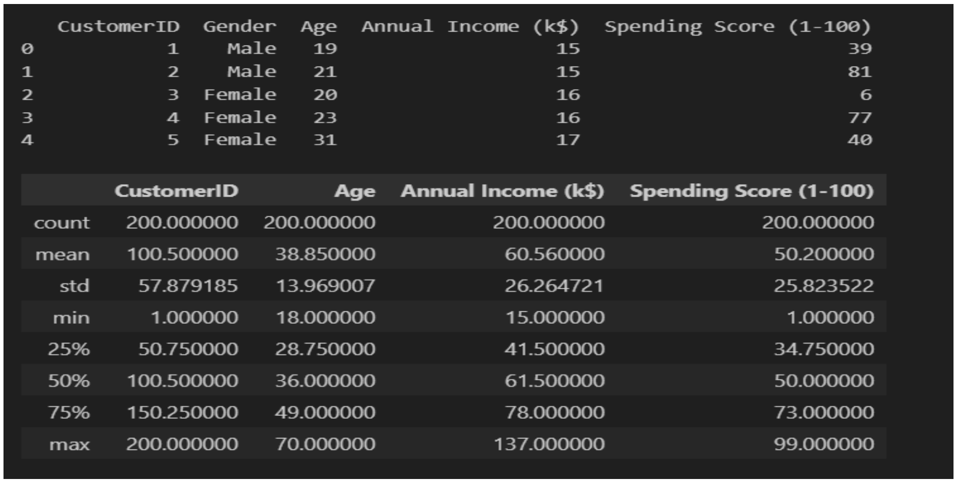
Gender: Categorical data (Male/Female) will be encoded numerically.

Age: The customer's age ranges from 18 to 70 years.

Annual Income (k$): Customer earnings range from $15k to $137k.

Spending score (1-100): A behavioral score that ranges from 1 to 99 and indicates client spending behaviors.

The range of characteristics, such as Annual Income and Spending Score, make this dataset appropriate for customer segmentation using clustering techniques.



```
   CustomerID  Gender  Age  Annual Income (k$)  Spending Score (1-100)
0           1    Male   19                  15                      39
1           2    Male   21                  15                      81
2           3  Female   20                  16                       6
3           4  Female   23                  16                      77
4           5  Female   31                  17                      40
```

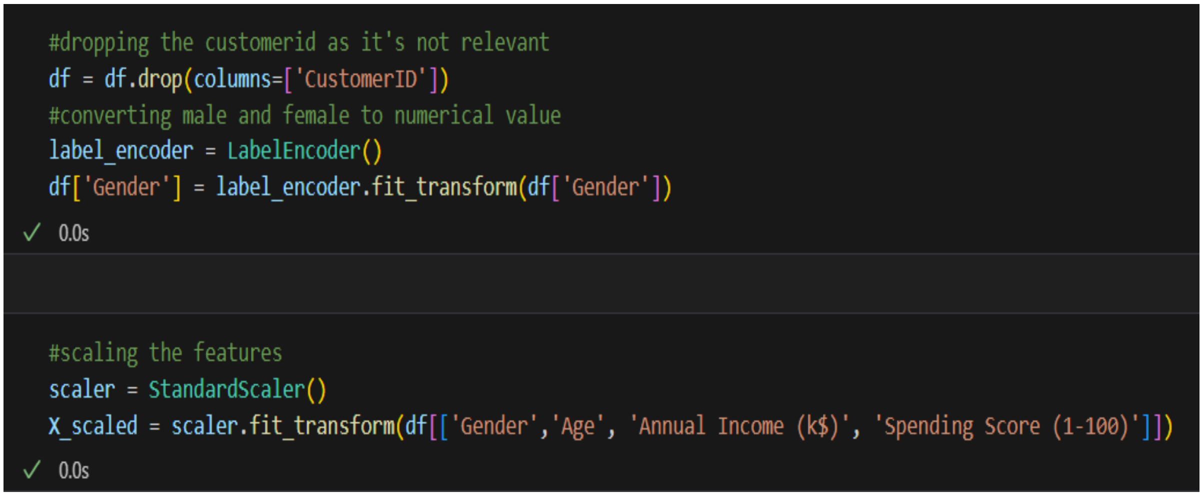
	CustomerID	Age	Annual Income (k$)	Spending Score (1-100)
count	200.000000	200.000000	200.000000	200.000000
mean	100.500000	38.850000	60.560000	50.200000
std	57.879185	13.969007	26.264721	25.823522
min	1.000000	18.000000	15.000000	1.000000
25%	50.750000	28.750000	41.500000	34.750000
50%	100.500000	36.000000	61.500000	50.000000
75%	150.250000	49.000000	78.000000	73.000000
max	200.000000	70.000000	137.000000	99.000000

3. Data Preprocessing

Removing Irrelevant Columns: Remove the CustomerID column, which is a unique identification for each client but contains no important information about their behavior or characteristics. Including such a column in clustering may result in incorrect findings since Kmeans would try to assign clusters based on arbitrary ID numbers rather than relevant patterns in the data.

Encoding Categorical Data: Categorical data, such as gender (male or female), cannot be utilized directly for distance calculations. Encoding Gender as numeric values (e.g., Male = 0, Female = 1) allows us to mathematically describe this attribute.

Scaling Features: Standardize numerical features to ensure that all features have the same distribution characteristics and are comparable in terms of their impact on clustering findings.



```
#dropping the customerid as it's not relevant
df = df.drop(columns=['CustomerID'])
#converting male and female to numerical value
label_encoder = LabelEncoder()
df['Gender'] = label_encoder.fit_transform(df['Gender'])
```

✓ 0.0s

```
#scaling the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(df[['Gender','Age', 'Annual Income (k$)', 'Spending Score (1-100)']])
```

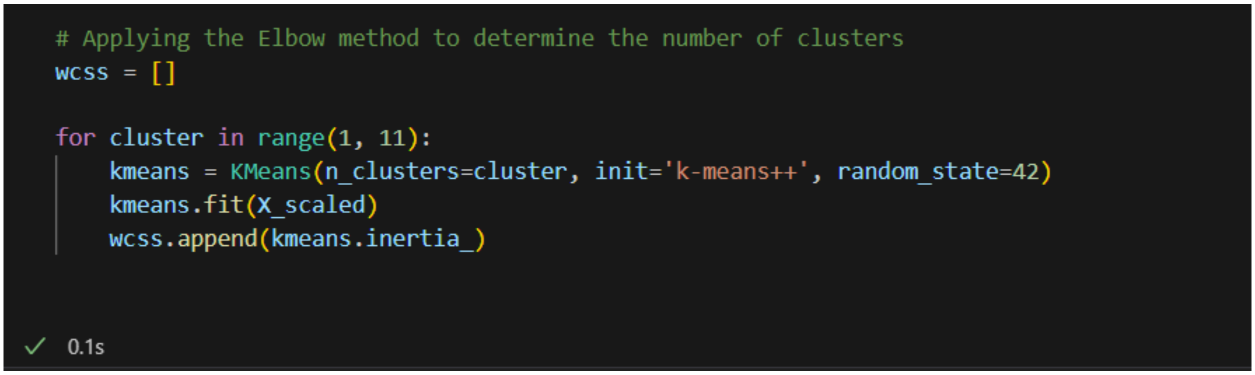
✓ 0.0s

4. **Evaluating the Model: Elbow Method**

When using K-means to group data, you must first determine how many clusters (groups) to create. The Elbow Method lets you determine the optimal number of clusters by examining how "tight" the clusters are.

WCSS (Within-Cluster Sum of Squares) is a measure that indicates how near each cluster's points are to its center. A lower WCSS indicates that the points are closer to the center, thus the cluster is "tighter."

The Elbow Method assumes that as the number of clusters increases, the WCSS decreases. However, adding more clusters does not improve matters. This is where you find the "elbow" in the graph which tells you the optimal number of clusters.

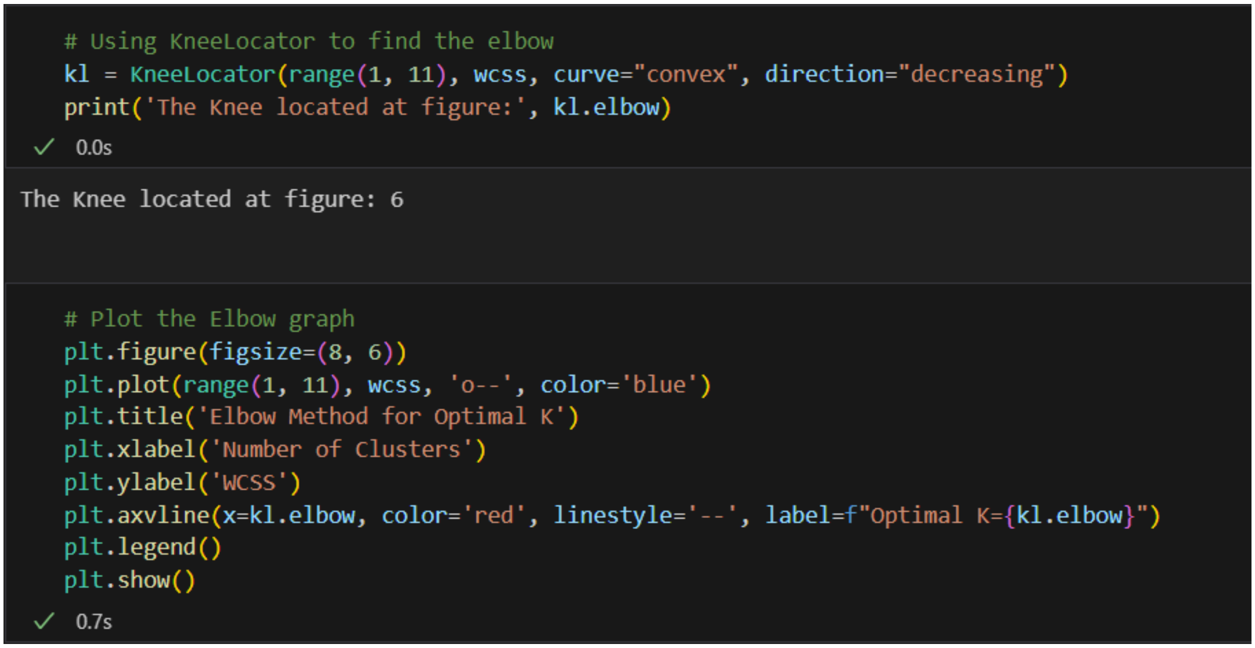


```
# Applying the Elbow method to determine the number of clusters
wcss = []

for cluster in range(1, 11):
    kmeans = KMeans(n_clusters=cluster, init='k-means++', random_state=42)
    kmeans.fit(X_scaled)
    wcss.append(kmeans.inertia_)
```

✓ 0.1s

- We loop through the cluster values from 1 to 10 (the top limit can be adjusted as desired).
- For each K value, we generate a KMeans object and fit it to the scaled data (X_scaled).
- The inertia_ attribute of the KMeans object denotes the WCSS for that K.



```
# Using KneeLocator to find the elbow
kl = KneeLocator(range(1, 11), wcss, curve="convex", direction="decreasing")
print('The Knee located at figure:', kl.elbow)
```

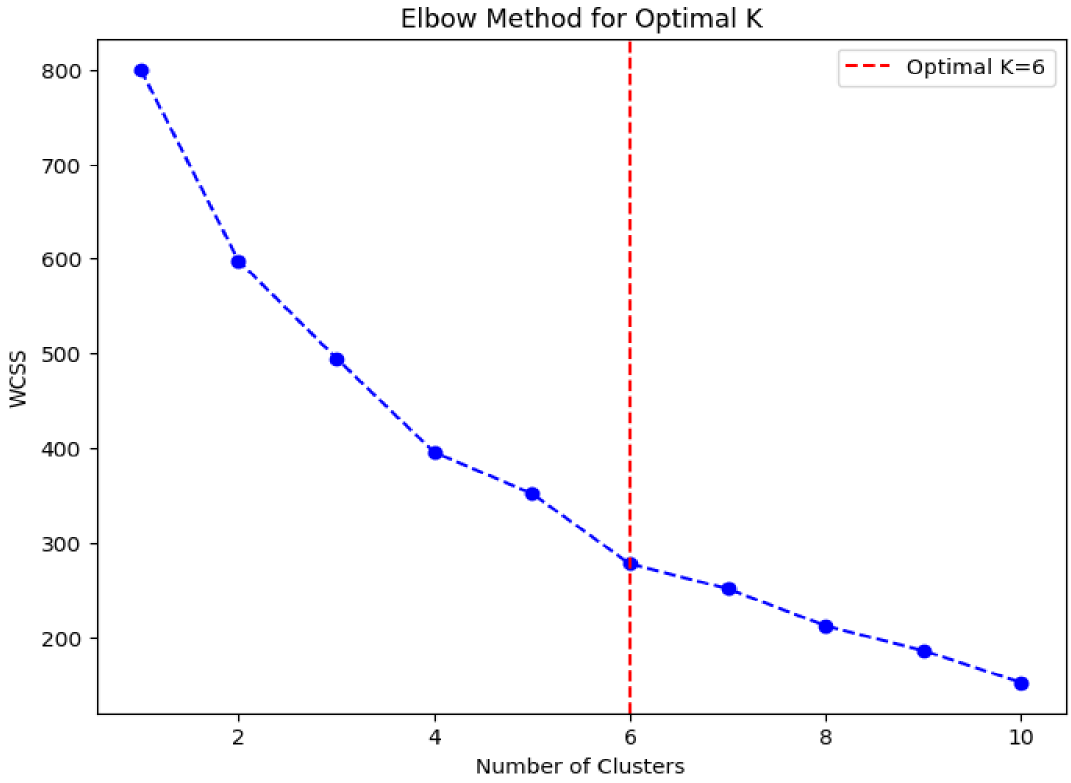
✓ 0.0s

```
The Knee located at figure: 6
```

```
# Plot the Elbow graph
plt.figure(figsize=(8, 6))
plt.plot(range(1, 11), wcss, 'o--', color='blue')
plt.title('Elbow Method for Optimal K')
plt.xlabel('Number of Clusters')
plt.ylabel('WCSS')
plt.axvline(x=kl.elbow, color='red', linestyle='--', label=f"Optimal K={kl.elbow}")
plt.legend()
plt.show()
```

✓ 0.7s

- KneeLocator searches for the "knee" or elbow point using the range of K (1 to 10), as well as the list of WCSS values (wcss).
- The output kl.elbow indicates the optimal number of clusters, which in this instance is 6.



X-axis: Represents the number of clusters (K).

Y-Axis: Represents the WCSS (Within-Cluster Sum of Squares), also known as inertia, is the total squared distance between data points and the nearest cluster center.

Blue dashed line: Illustrates how WCSS diminishes as the number of clusters grows. Initially, WCSS drops rapidly as more clusters match the data better.The pace of reduction reduces after a certain point.

Red vertical line: Indicates the elbow point, where K=6. This is the point at which adding more clusters does not significantly lower WCSS.

5. Euclidean Distance

Euclidean Distance is the straight-line (or geometric) distance between two points in a multidimensional space. It is the most popular distance metric in K-Means.

Assume you're standing at one spot on a map and want to know how far you are from another point. Euclidean Distance essentially addresses that question by determining the shortest way between two points, such as a straight line, rather than using any twisting roads or trails.

Formula:

Euclidean distance between two points $x = (x1, x2, \ldots, xn)$ and $y = (y1, y2, \ldots, yn)$ in an n-dimensional space is:

$$d(x, y) = \sqrt{\sum_{i=1}^{n} (xi - yi)^2}$$

It depends on the data size and is thus sensitive to the scale of the characteristics. K-Means minimizes the sum of squared Euclidean distances between points and cluster centroids during the clustering process.

Silhouette Score with Euclidean Distance: 0.2657.

This score indicates that the grouping is slightly weak. While there is some separation between clusters, there is significant overlap between them, or the clusters are not well-separated in feature space.

6. Cosine Similarity

Cosine similarity calculates the cosine of the angle between two vectors. Instead of focusing on magnitude, it compares the direction of the two vectors.

Imagine you're attempting to compare two book recommendations based on their content. Cosine Similarity measures how similar these recommendations are, not by their size or length, but by comparing the direction of the material. It's like two books with differing lengths but similar themes or topics. Cosine Similarity would tell us how closely those concepts aligned, regardless of the book's word count or content.

Formula:

The cosine similarity between two vectors, x and y is:

$$similarity(x,y) = \frac{x \cdot y}{\parallel x \parallel \parallel y \parallel}$$

where:

$x \cdot y = \sum_{i=1}^{n} xiyi$ is the dot product

$\parallel x \parallel = \sqrt{\sum_{i=1}^{n} xi^2}$ is the magnitude (or norm) of vector.

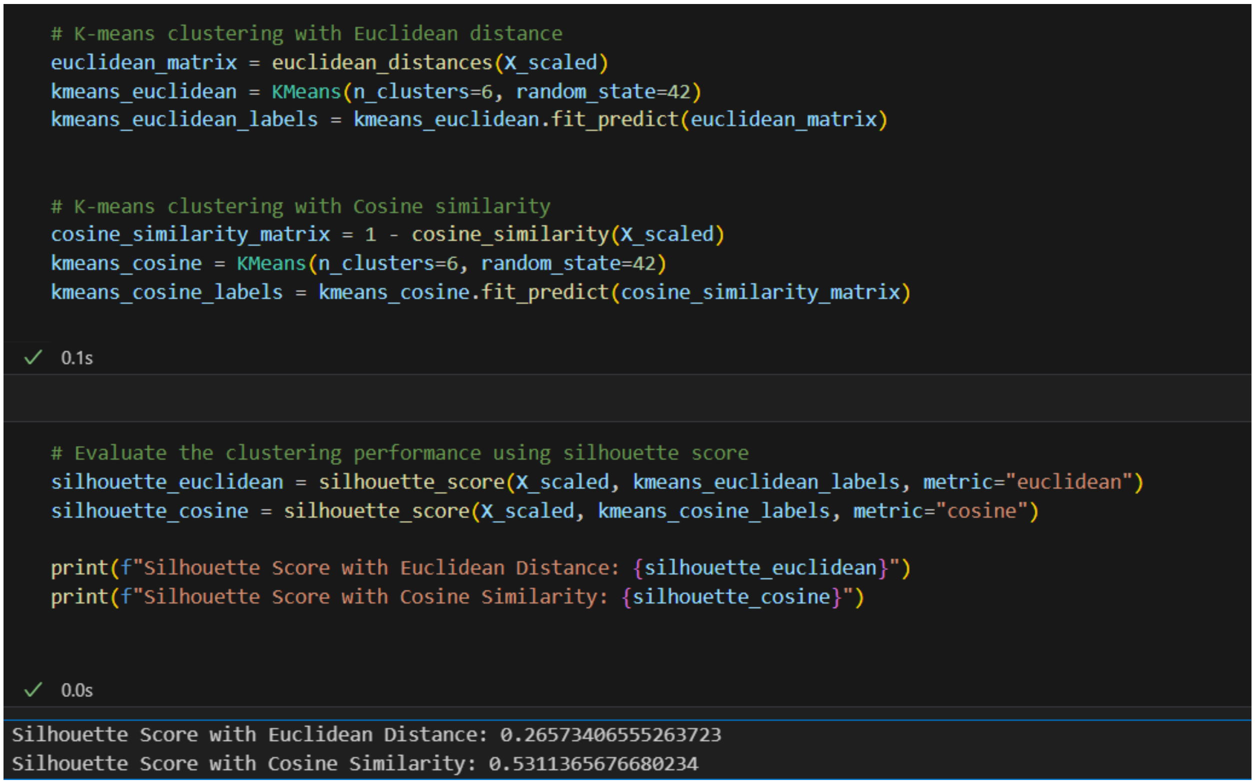
Outputs values in the range $[-1,1]$, where

- 1 represents perfectly aligned vectors.
- 0 represents orthogonality (no resemblance).
- -1 indicates entirely opposite directions.

To employ Cosine Similarity in K-Means, you often change the algorithm to maximize similarity (or reduce 1 - similarity) rather than minimizing Euclidean distance.

The silhouette score with cosine similarity is 0.5311.

A significantly higher score implies that cosine similarity is a superior metric for this dataset. It implies that the clusters are more distinct when similarity in direction (angle) is considered instead of absolute distances.



```python
# K-means clustering with Euclidean distance
euclidean_matrix = euclidean_distances(X_scaled)
kmeans_euclidean = KMeans(n_clusters=6, random_state=42)
kmeans_euclidean_labels = kmeans_euclidean.fit_predict(euclidean_matrix)


# K-means clustering with Cosine similarity
cosine_similarity_matrix = 1 - cosine_similarity(X_scaled)
kmeans_cosine = KMeans(n_clusters=6, random_state=42)
kmeans_cosine_labels = kmeans_cosine.fit_predict(cosine_similarity_matrix)
```

✓ 0.1s

```python
# Evaluate the clustering performance using silhouette score
silhouette_euclidean = silhouette_score(X_scaled, kmeans_euclidean_labels, metric="euclidean")
silhouette_cosine = silhouette_score(X_scaled, kmeans_cosine_labels, metric="cosine")

print(f"Silhouette Score with Euclidean Distance: {silhouette_euclidean}")
print(f"Silhouette Score with Cosine Similarity: {silhouette_cosine}")
```

✓ 0.0s

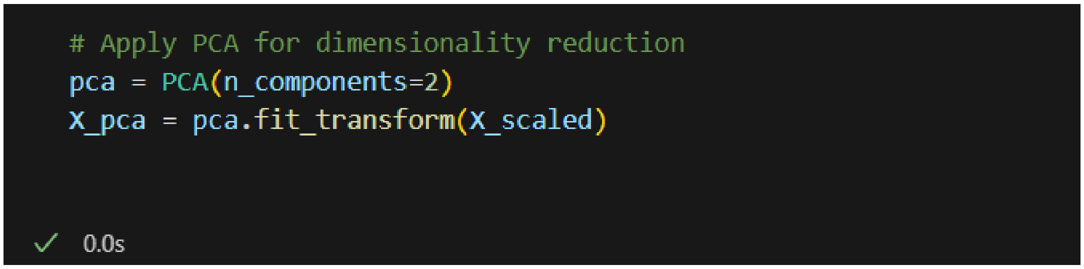
```
Silhouette Score with Euclidean Distance: 0.26573406555263723
Silhouette Score with Cosine Similarity: 0.5311365676680234
```

7.Dimensionality Reduction with PCA

PCA is a dimensionality-reduction approach. It converts a dataset into a new set of features (principal components) that represent the most essential information in the data.

In this case, n_components =2 will calculate the top two principal components: the directions in the feature space that represent the most essential information in the data.

PCA is employed here to transform the data into a 2D format to make it easier to visualize.



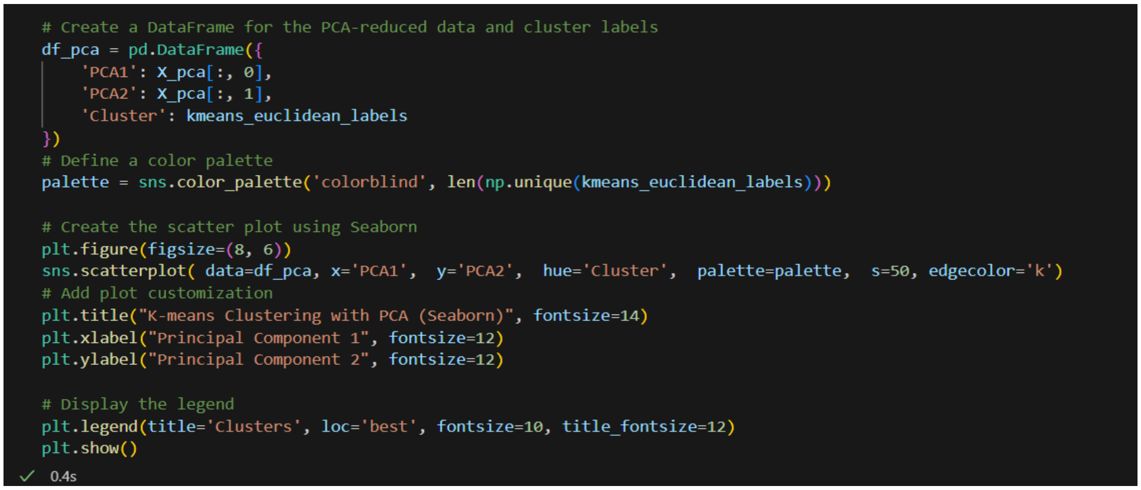
```
# Apply PCA for dimensionality reduction
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)
```

✓ 0.0s

8. Results Visualization

A scatter plot is used to show how K-Means clustering grouped the data after lowering its dimensionality via PCA. The clusters should be clearly separated in 2D space to indicate successful clustering.

One can change the color palette, marker size, and axis names to suit that dataset's demands.

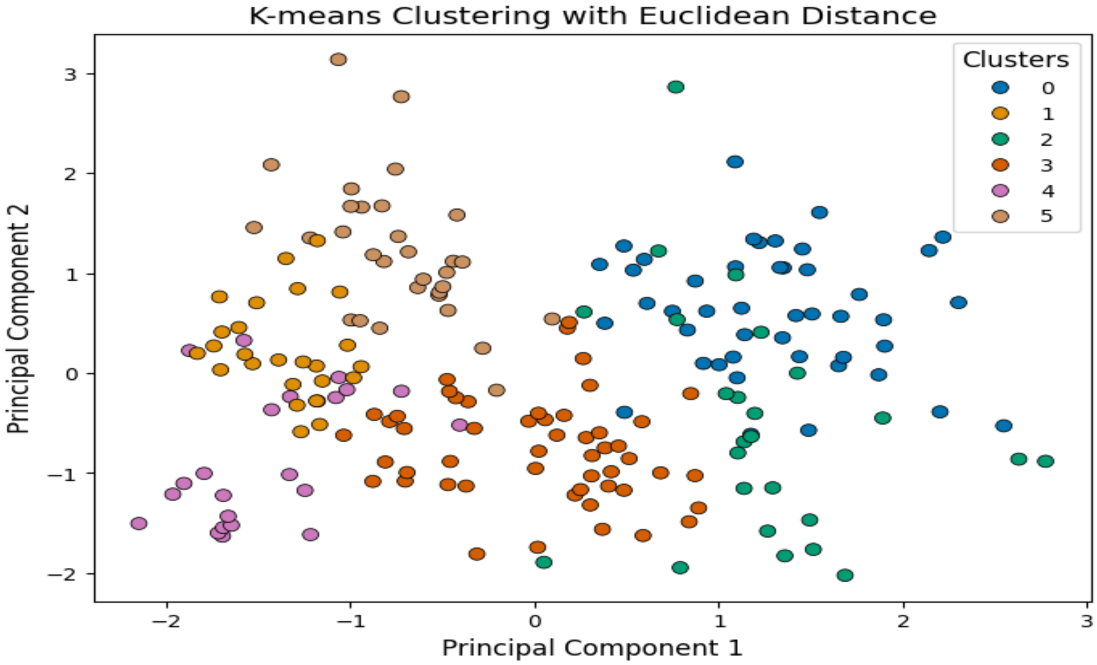


```
# Create a DataFrame for the PCA-reduced data and cluster labels
df_pca = pd.DataFrame({
    'PCA1': X_pca[:, 0],
    'PCA2': X_pca[:, 1],
    'Cluster': kmeans_euclidean_labels
})
# Define a color palette
palette = sns.color_palette('colorblind', len(np.unique(kmeans_euclidean_labels)))

# Create the scatter plot using Seaborn
plt.figure(figsize=(8, 6))
sns.scatterplot( data=df_pca, x='PCA1',  y='PCA2',  hue='Cluster',  palette=palette,  s=50, edgecolor='k')
# Add plot customization
plt.title("K-means Clustering with PCA (Seaborn)", fontsize=14)
plt.xlabel("Principal Component 1", fontsize=12)
plt.ylabel("Principal Component 2", fontsize=12)

# Display the legend
plt.legend(title='Clusters', loc='best', fontsize=10, title_fontsize=12)
plt.show()
```

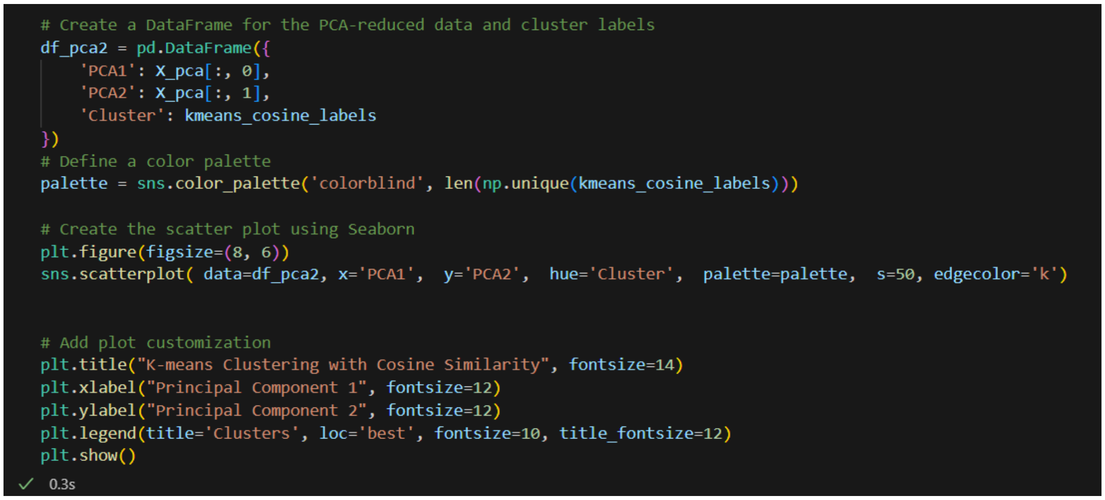
✓ 0.4s



The clusters appear to be somewhat mixed, with considerable overlap. This is demonstrated by the fact that data points are spread throughout regions where multiple clusters intersect. In some regions, clusters are not clearly distinguished, resulting in unclear categorization.

Some sites appear to be near cluster boundaries, perhaps leading to inaccurate cluster assignments.

The Euclidean distance is useful when the data is compact and relatively near in terms of absolute distances. However, this plot indicates that the data may not be well-suited to Euclidean distance.

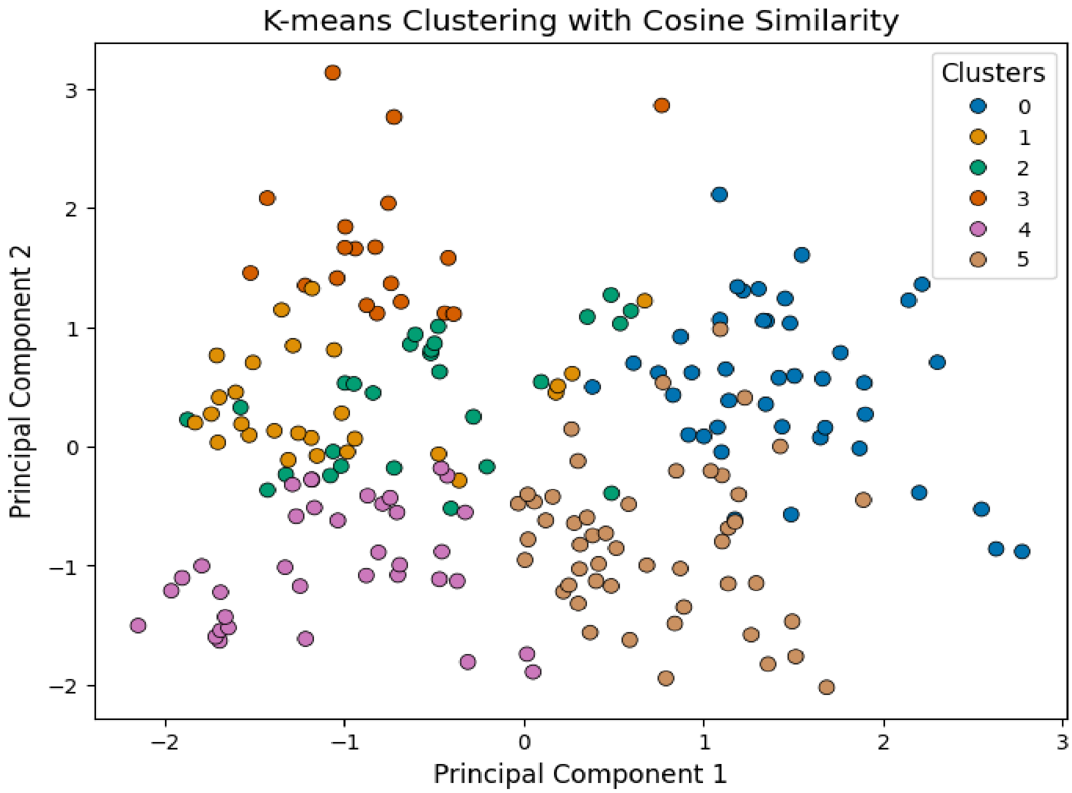


```
# Create a DataFrame for the PCA-reduced data and cluster labels
df_pca2 = pd.DataFrame({
    'PCA1': X_pca[:, 0],
    'PCA2': X_pca[:, 1],
    'Cluster': kmeans_cosine_labels
})
# Define a color palette
palette = sns.color_palette('colorblind', len(np.unique(kmeans_cosine_labels)))

# Create the scatter plot using Seaborn
plt.figure(figsize=(8, 6))
sns.scatterplot( data=df_pca2, x='PCA1',  y='PCA2',  hue='Cluster',  palette=palette,  s=50, edgecolor='k')


# Add plot customization
plt.title("K-means Clustering with Cosine Similarity", fontsize=14)
plt.xlabel("Principal Component 1", fontsize=12)
plt.ylabel("Principal Component 2", fontsize=12)
plt.legend(title='Clusters', loc='best', fontsize=10, title_fontsize=12)
plt.show()
```

✓ 0.3s



The clusters are more clearly distinguished here than in the Euclidean distance diagram. The points inside each cluster appear to be closer together, while the boundaries between clusters are more clearly defined.

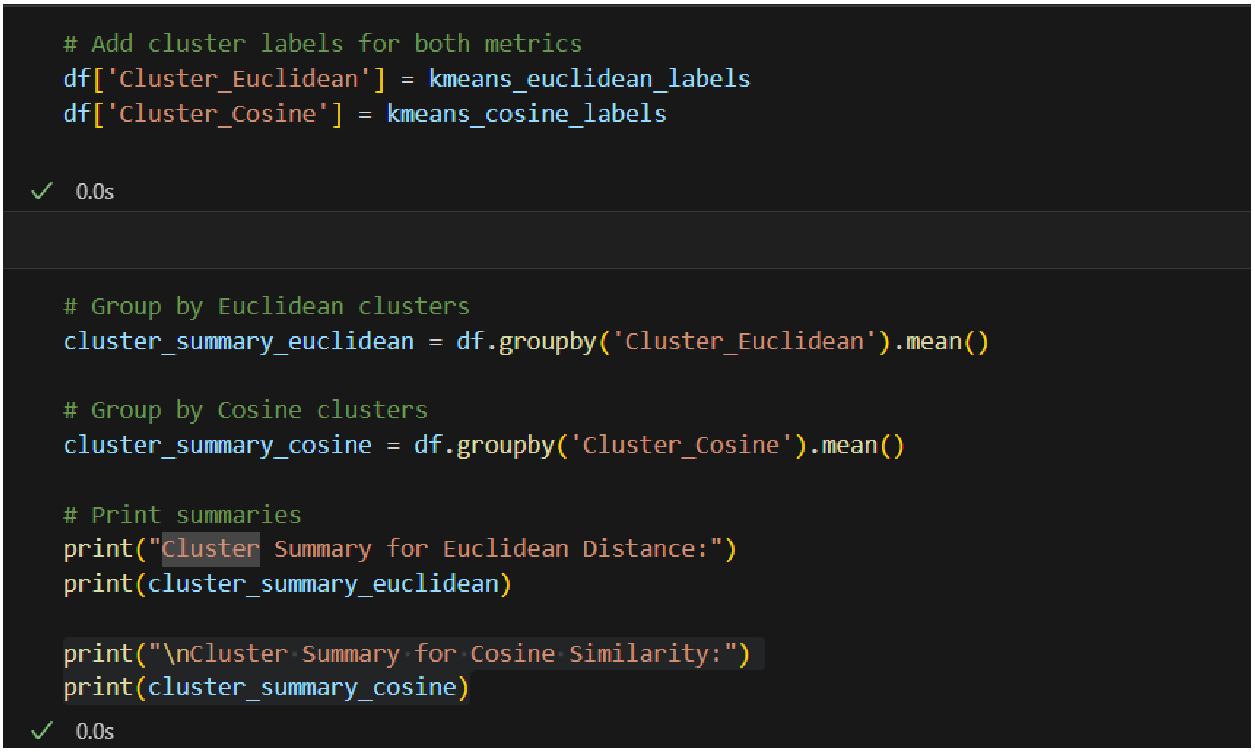
There is less overlap between the clusters, indicating that Cosine similarity captures the data's structure more efficiently. Cosine similarity works better for data with directional relationships.

After running K-means clustering with Euclidean Distance and Cosine Similarity and visualizing the results, the next step is to examine and compare the clusters created by each approach. This is accomplished by determining the mean values for each feature in the dataset within each cluster.

Cluster Summary allows us to understand the average behavior of customers in each cluster. We may make informed consumer segmentation decisions by looking at the mean values of the characteristics (such as age, income, spending score, and gender).

The groupby() function groups the original data (df) into clusters based on Euclidean Distance (Cluster_Euclidean) and Cosine Similarity.

The.mean() function is then used to calculate the mean value of each attribute (such as gender, age, annual income, and spending score) for each cluster.



```
# Add cluster labels for both metrics
df['Cluster_Euclidean'] = kmeans_euclidean_labels
df['Cluster_Cosine'] = kmeans_cosine_labels
```

✓ 0.0s

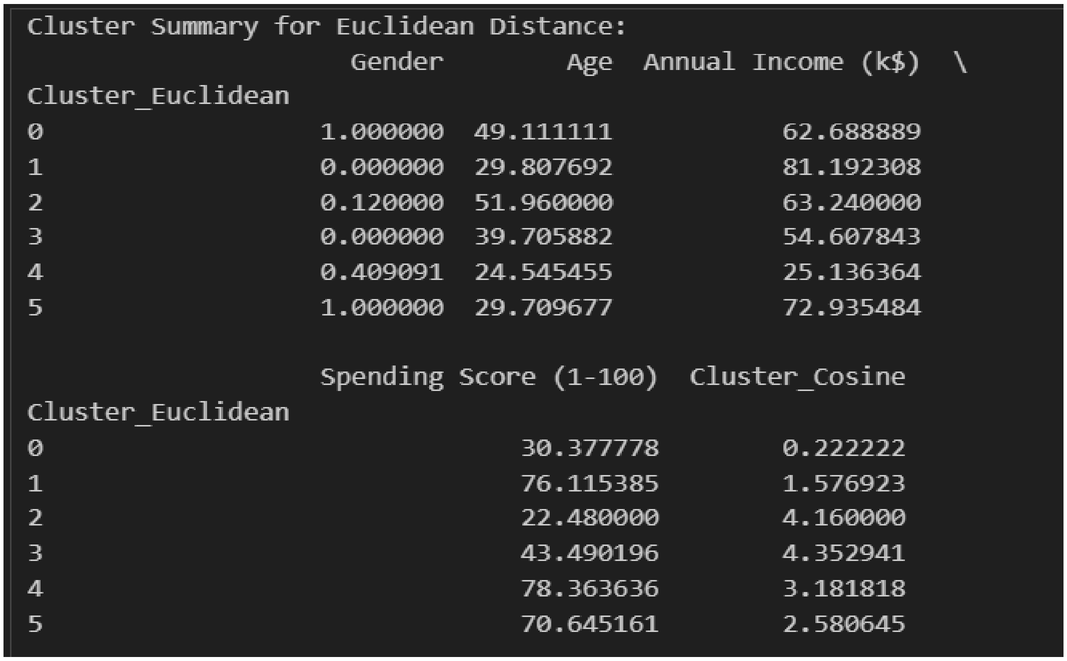
```
# Group by Euclidean clusters
cluster_summary_euclidean = df.groupby('Cluster_Euclidean').mean()

# Group by Cosine clusters
cluster_summary_cosine = df.groupby('Cluster_Cosine').mean()

# Print summaries
print("Cluster Summary for Euclidean Distance:")
print(cluster_summary_euclidean)

print("\nCluster Summary for Cosine Similarity:")
print(cluster_summary_cosine)
```

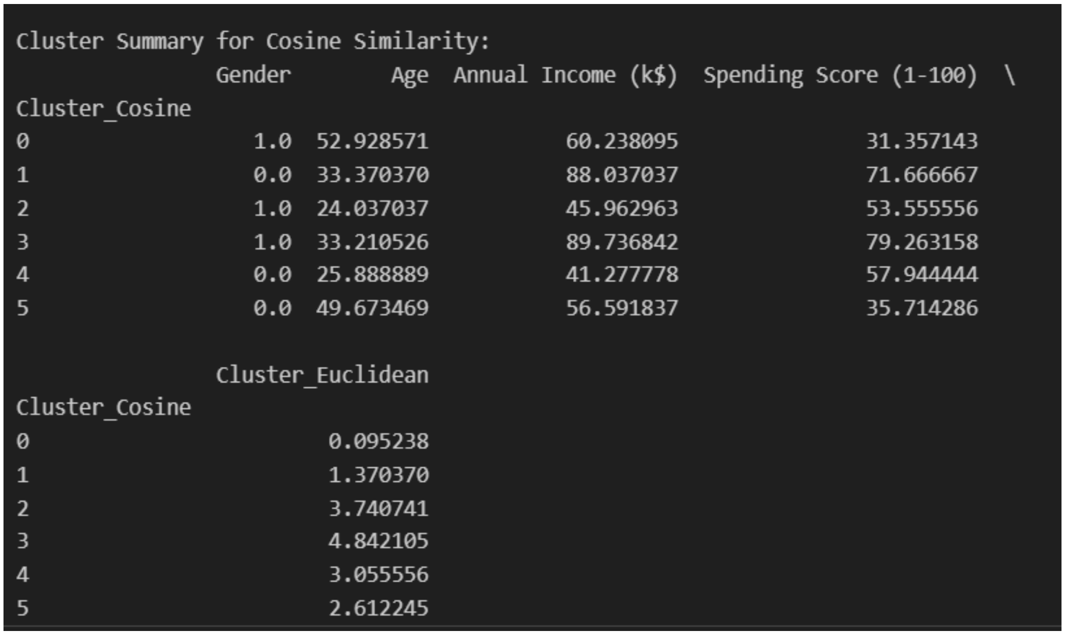
✓ 0.0s



```
Cluster Summary for Euclidean Distance:
                     Gender        Age  Annual Income (k$)  \
Cluster_Euclidean
0                  1.000000  49.111111           62.688889
1                  0.000000  29.807692           81.192308
2                  0.120000  51.960000           63.240000
3                  0.000000  39.705882           54.607843
4                  0.409091  24.545455           25.136364
5                  1.000000  29.709677           72.935484

                   Spending Score (1-100)  Cluster_Cosine
Cluster_Euclidean
0                               30.377778        0.222222
1                               76.115385        1.576923
2                               22.480000        4.160000
3                               43.490196        4.352941
4                               78.363636        3.181818
5                               70.645161        2.580645
```

Euclidean Distance creates more varied clusters, with gender mixing in various clusters and less consistent income and spending patterns across clusters.



```
Cluster Summary for Cosine Similarity:
                Gender        Age  Annual Income (k$)  Spending Score (1-100)  \
Cluster_Cosine
0                  1.0  52.928571           60.238095               31.357143
1                  0.0  33.370370           88.037037               71.666667
2                  1.0  24.037037           45.962963               53.555556
3                  1.0  33.210526           89.736842               79.263158
4                  0.0  25.888889           41.277778               57.944444
5                  0.0  49.673469           56.591837               35.714286

                Cluster_Euclidean
Cluster_Cosine
0                        0.095238
1                        1.370370
2                        3.740741
3                        4.842105
4                        3.055556
5                        2.612245
```

Cosine Similarity leads to more homogeneous clusters in terms of gender, wealth, and spending habits. Gender distribution is clearer, with fewer mixed-gender groups. Income and spending habits are more regularly organized.

Conclusion

In this lesson, we investigated the effect of various distance metrics on K-Means clustering by contrasting Euclidean Distance and Cosine Similarity for customer segmentation. We applied both measures to a customer dataset containing variables such as age, income, and expenditure score, and then visualized the findings using PCA to reduce dimensionality.

Clustering using Euclidean Distance:

Euclidean Distance, which is sensitive to the magnitude of the features, created clusters with diverse characteristics. The clusters overlapped significantly, and the silhouette score of 0.2657 indicated a weak separation between the groups. Gender and wealth varied among clusters, and expenditure score patterns were less constant.

Clustering using cosine similarity:

Cosine Similarity, on the other hand, resulted in more homogeneous clusters, with a clearer gender distribution and more similar income and expenditure patterns across clusters. The silhouette score of 0.5311 was much greater than Euclidean Distance, indicating more defined and identifiable groups. This shows that Cosine Similarity is a better alternative for grouping customers based on similar behaviors than absolute distances.

Final Takeaway:

Given that this is a consumer mall dataset, Cosine Similarity is a far superior method for clustering customers based on behavior. It better captures the relationships in data that are important for customer segmentation. A higher silhouette score of 0.531 suggests more significant, well-defined clusters of consumers who behave similarly, making this strategy more valuable for tasks such as tailored marketing, customer targeting, and loyalty program building.

On the other hand, Euclidean Distance may struggle to construct meaningful clusters since its assumptions regarding distances between consumer behaviors do not always correspond to the nature of customer preferences, resulting in poor separation and lower quality clusters (silhouette score of 0.2657). As a result, Cosine Similarity is the most appropriate metric for clustering customer data in this scenario.

10. Code & GitHub Repository

https://github.com/Anagha235/Machine-Learning-and-Neural-Network.git

References

- **GeeksforGeeks.** (2021). "K-Means Clustering - Introduction." Retrieved from GeeksforGeeks.
- **Analytics Vidhya.** (2021). "K-Mean: Getting the Optimal Number of Clusters." Retrieved from Analytics Vidhya.
- **DataStax.** (2021). "What is Cosine Similarity?" Retrieved from DataStax.
- **Machine Learning Mastery.** (2020). "Distance Measures for Machine Learning." Retrieved from Machine Learning Mastery.
- **Medium.** (2021). "Worked Examples for Distance/Similarity Measures in Clustering." Retrieved from Medium.