

LAB-Experiment-8 (Unsupervised Learning)

K-means and K-medoid Clustering**About Dataset:**

Measurements of geometrical properties of kernels belonging to three different varieties of wheat.

A soft X-ray technique and GRAINS package were used to construct all seven, real-valued attributes.

Dataset Information:

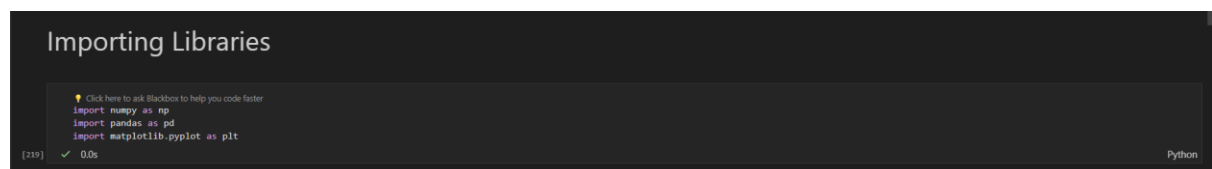
The examined group comprised kernels belonging to three different varieties of wheat: Kama, Rosa and Canadian, 70 elements each, randomly selected for the experiment. High quality visualization of the internal kernel structure was detected using a soft X-ray technique. It is non-destructive and considerably cheaper than other more sophisticated imaging techniques like scanning microscopy or laser technology. The images were recorded on 13x18 cm X-ray KODAK plates. Studies were conducted using combine harvested wheat grain originating from experimental fields, explored at the Institute of Agrophysics of the Polish Academy of Sciences in Lublin.

The data set can be used for the tasks of classification and cluster analysis.

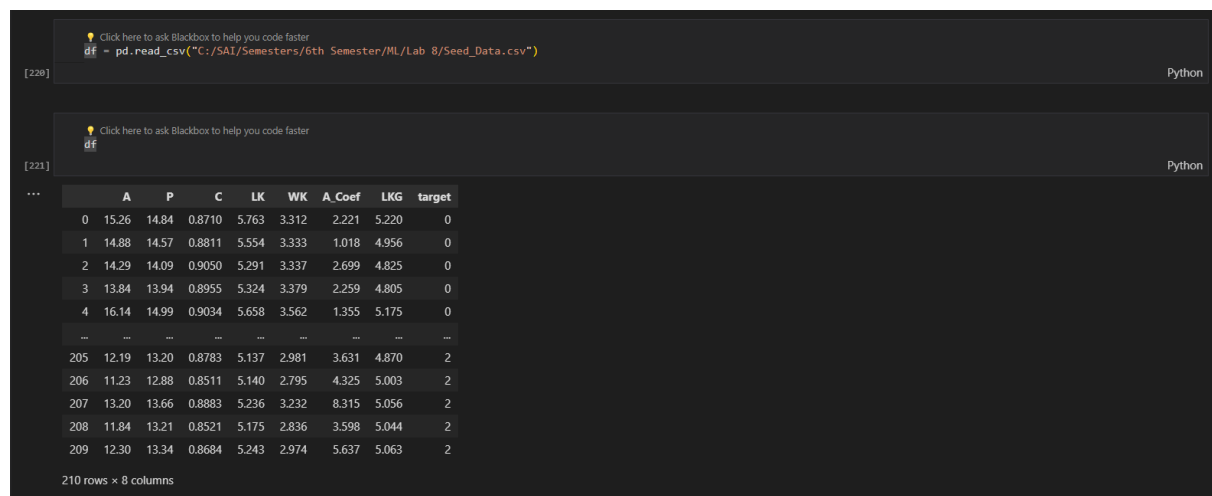
Code Analysis:

Step1: Import the libraries

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```



Step2: Loading the Dataset



Step 3: preprocessing the dataset

```
df.head()
```

The above command gives the top 5 rows of the dataset

```
Click here to ask Blackbox to help you code faster
df.head()
```

```
[222]
```

```
Python
```

```
...
   A    P    C    LK    WK    A_Coef    LKG    target
0  15.26  14.84  0.8710  5.763  3.312  2.221  5.220  0
1  14.88  14.57  0.8811  5.554  3.333  1.018  4.956  0
2  14.29  14.09  0.9050  5.291  3.337  2.699  4.825  0
3  13.84  13.94  0.8955  5.324  3.379  2.259  4.805  0
4  16.14  14.99  0.9034  5.658  3.562  1.355  5.175  0
```

```
df.info()
```

df.info() – it includes the data types of columns, non-null counts, and memory usage in one line.

```
Click here to ask Blackbox to help you code faster
df.info()
```

```
[223]
```

```
Python
```

```
...
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 210 entries, 0 to 209
Data columns (total 8 columns):
#   Column  Non-Null Count  Dtype
---  -
0    A      210 non-null     float64
1    P      210 non-null     float64
2    C      210 non-null     float64
3    LK      210 non-null     float64
4    WK      210 non-null     float64
5    A_Coef  210 non-null     float64
6    LKG      210 non-null     float64
7    target  210 non-null     int64
dtypes: float64(7), int64(1)
memory usage: 13.2 KB
```

```
df.isnull().sum()
```

df.isnull().sum() provides the count of null (missing) values for each column in the DataFrame 'df'.

```
Click here to ask Blackbox to help you code faster
df.isnull().sum()
```

```
[224]
```

```
Python
```

```
...
A      0
P      0
C      0
LK      0
WK      0
A_Coef  0
LKG      0
target  0
dtype: int64
```

```
df.duplicated().sum()
```

df.duplicated().sum() provides the count of all duplicate values for each column in the DataFrame 'df'.

```
Click here to ask Blackbox to help you code faster
df.duplicated().sum()
```

```
[225]
```

```
Python
```

```
...
0
```

```
df.describe()
```

we use describe method to Provide summary statistics for key features of the data contains count, mean, std, min, 1st quartile, 2nd quartile, 3rd quartile and the maximum in each column

Click here to ask Blackbox to help you code faster

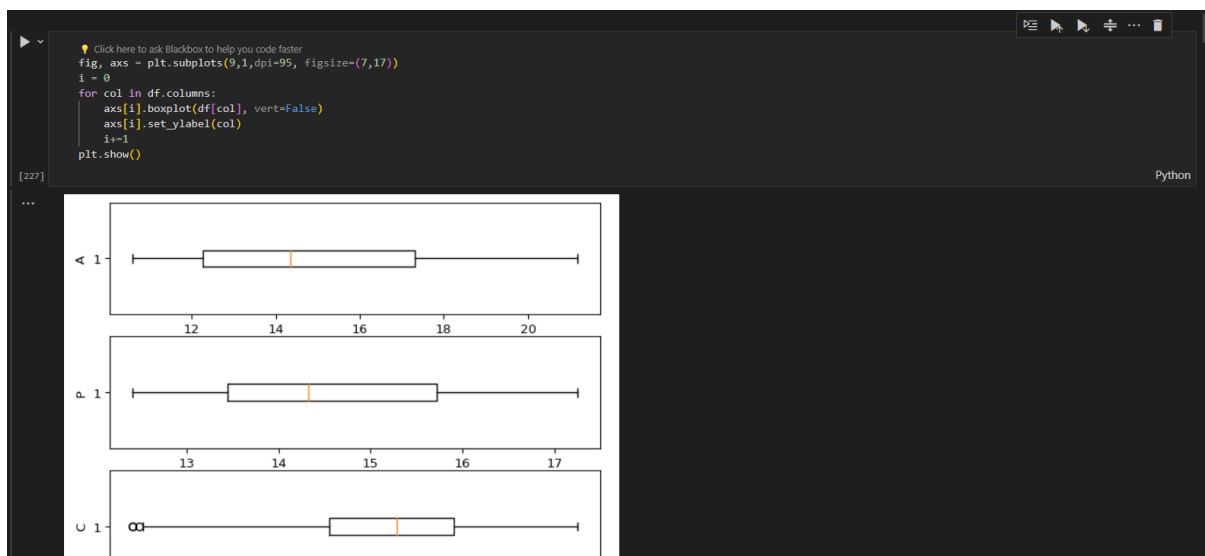
```
df.describe()
```

[226] Python

	A	P	C	LK	WK	A_Coef	LKG	target
count	210.000000	210.000000	210.000000	210.000000	210.000000	210.000000	210.000000	210.000000
mean	14.847524	14.559286	0.870999	5.628533	3.258605	3.700201	5.408071	1.000000
std	2.909699	1.305959	0.023629	0.443063	0.377714	1.503557	0.491480	0.818448
min	10.590000	12.410000	0.808100	4.899000	2.630000	0.765100	4.519000	0.000000
25%	12.270000	13.450000	0.856900	5.262250	2.944000	2.561500	5.045000	0.000000
50%	14.355000	14.320000	0.873450	5.523500	3.237000	3.599000	5.223000	1.000000
75%	17.305000	15.715000	0.887775	5.979750	3.581750	4.768750	5.877000	2.000000
max	21.180000	17.250000	0.918300	6.675000	4.033000	8.456000	6.550000	2.000000

The above creates subplots for each column in your DataFrame df, displaying boxplots of the data. However, it seems you want to create 9 subplots vertically arranged, each representing a column in the DataFrame.

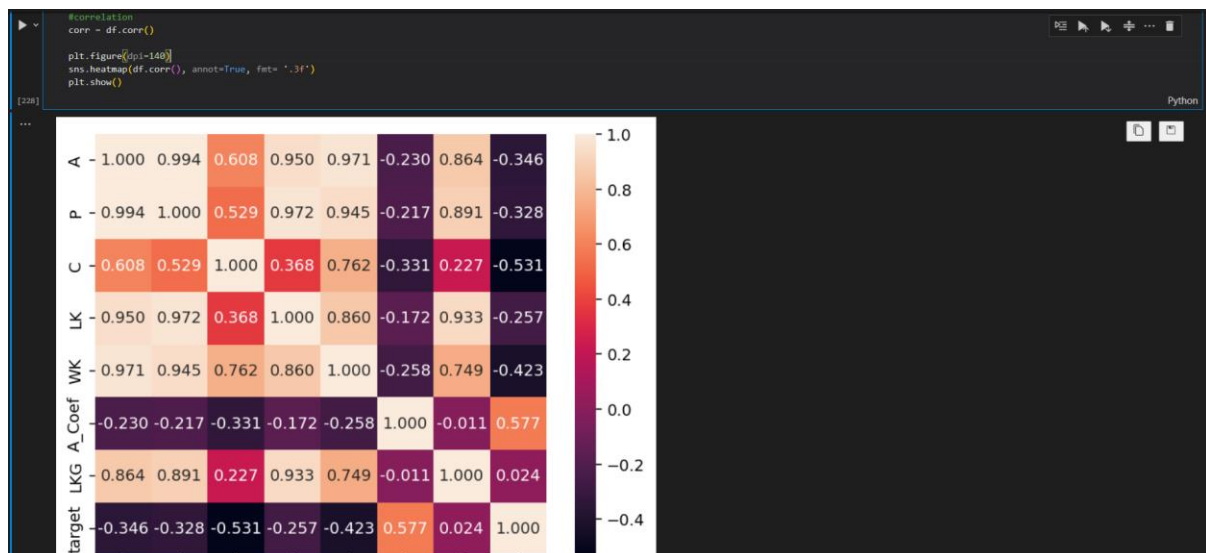
```
fig, axs = plt.subplots(9,1,dpi=95, figsize=(7,17))
i = 0
for col in df.columns:
    axs[i].boxplot(df[col], vert=False)
    axs[i].set_ylabel(col)
    i+=1
plt.show()
```



```
import seaborn as sns
#correlation
corr = df.corr()

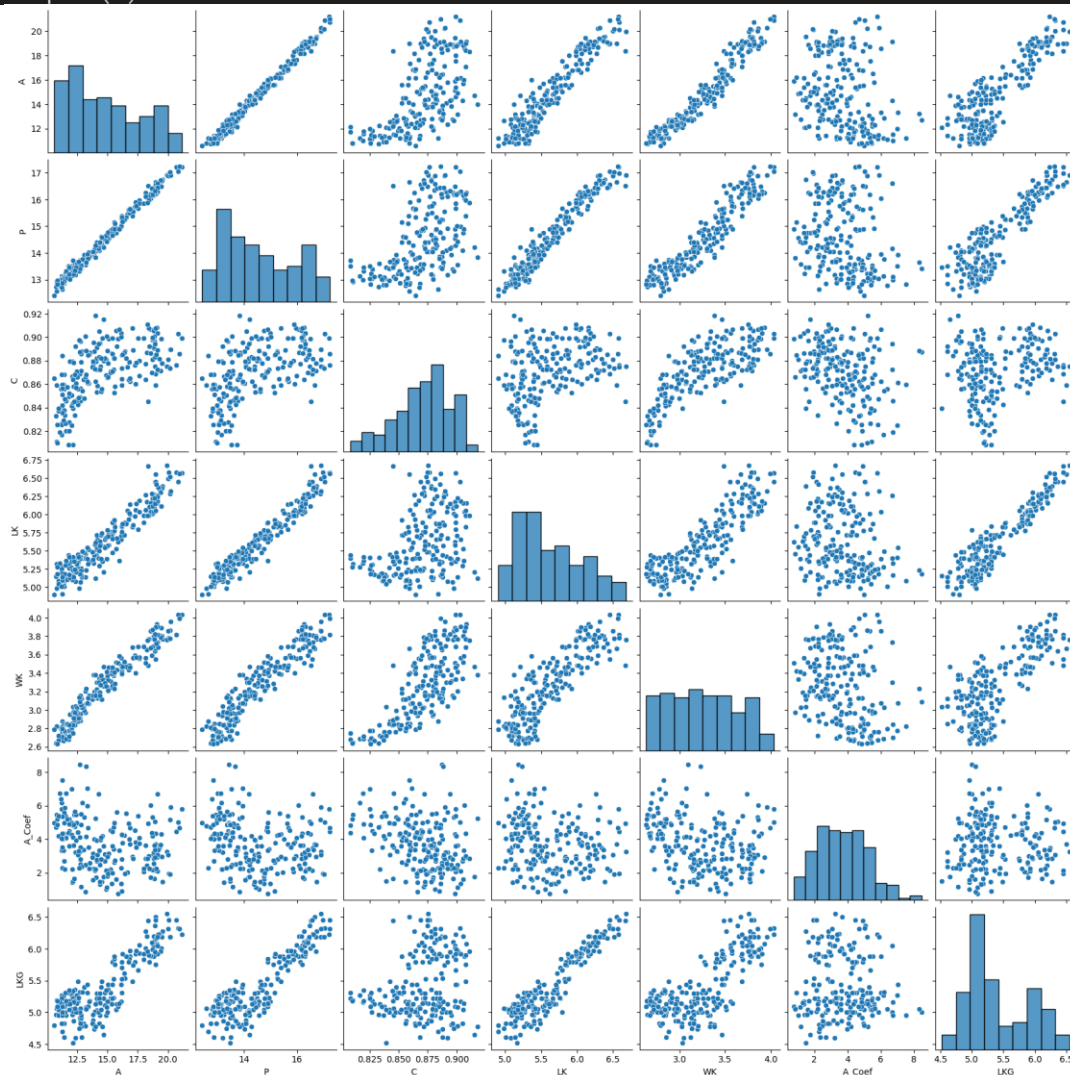
plt.figure(dpi=140)
sns.heatmap(df.corr(), annot=True, fmt= '.3f')
plt.show()
```

The above code utilizes Seaborn and Matplotlib to generate a heatmap displaying the correlation matrix of a DataFrame df. Each cell in the heatmap represents the correlation coefficient between two variables, with annotations showing the correlation values. Positive correlations are depicted in brighter colors, negative correlations in darker colors, and zero correlations in neutral colors. This visualization aids in identifying relationships and patterns among variables, facilitating data exploration and feature selection processes.



sns.pairplot(df): Generates a grid of scatterplots showing pairwise relationships between numeric columns in the DataFrame iris_df. Diagonal elements display univariate distributions, while the off-diagonal elements show bivariate relationships.

sns.pairplot(X)



Splitting dataset into independent variables and dependent variables

```
X = df.iloc[:, :-1]
```

```
[231] X = df.iloc[:, :-1] Python
```

```
[232] X Python
```

```
...
   A    P    C    LK    WK    A_Coef    LKG
0  15.26  14.84  0.8710  5.763  3.312  2.221  5.220
1  14.88  14.57  0.8811  5.554  3.333  1.018  4.956
2  14.29  14.09  0.9050  5.291  3.337  2.699  4.825
3  13.84  13.94  0.8955  5.324  3.379  2.259  4.805
4  16.14  14.99  0.9034  5.658  3.562  1.355  5.175
...
205 12.19  12.20  0.8783  5.137  2.981  3.631  4.870
206 11.23  12.88  0.8511  5.140  2.795  4.325  5.003
207 13.20  13.66  0.8883  5.236  3.232  8.315  5.056
208 11.84  13.21  0.8521  5.175  2.836  3.598  5.044
209 12.30  13.34  0.8684  5.243  2.974  5.637  5.063
210 rows x 7 columns
```

```
y = df.iloc[:, -1]
```

```
[233] y = df.iloc[:, -1] Python
```

```
[234] y Python
```

```
...
0    0
1    0
2    0
3    0
4    0
...
205  2
206  2
207  2
208  2
209  2
Name: target, Length: 210, dtype: int64
```

Implementing KMeans for the dataset

```
from sklearn.cluster import KMeans
wcss = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters = i, init = 'k-means++', max_iter = 300, n_init = 10,
random_state = 0)
    kmeans.fit(X)
    wcss.append(kmeans.inertia_)
```

The above code gives the KMeans clustering algorithm from scikit-learn to compute the within-cluster sum of squares (WCSS) for different numbers of clusters ranging from 1 to 10. For each number of clusters, KMeans is fitted to the data X, and the corresponding WCSS value is appended to a list. The resulting list contains the WCSS values for each number of clusters, which can be used to determine the optimal number of clusters for the dataset.

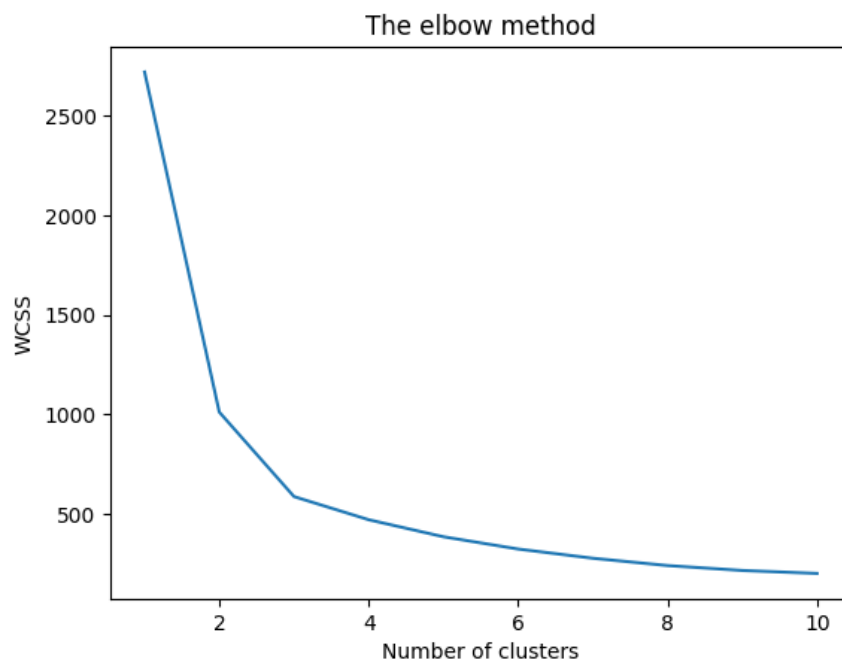
```
[237] from sklearn.cluster import KMeans
wcss = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters = i, init = 'k-means++', max_iter = 300, n_init = 10, random_state = 0)
    kmeans.fit(X)
    wcss.append(kmeans.inertia_) Python
```

```
[238] wcss Python
```

```
...
[2719.8524101779526,
1011.7123453151188,
587.3186115940429,
471.0033955251924,
385.5072923490406,
323.870247593612,
277.5611792670318,
240.8203228638082,
216.19696041808385,
201.3908710705668]
```

```
plt.plot(range(1, 11), wcss)
plt.title('The elbow method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS') #within cluster sum of squares
plt.show()
```

The code plots the number of clusters against the within-cluster sum of squares (WCSS) values calculated previously. It visualizes the "elbow method," helping to identify the optimal number of clusters. The plot's x-axis represents the number of clusters, while the y-axis represents the corresponding WCSS values. By inspecting the plot, the point where the decrease in WCSS slows down (the "elbow") indicates the optimal number of clusters. This helps in making an informed decision about the appropriate number of clusters for the dataset. Here in our data the no of clusters is 3 for better accuracy.



The selection of the number of clusters depends on the complexity and structure of the data, as well as the desired level of granularity in the clustering results. It's essential to consider various factors and validation methods to choose the most suitable number of clusters for a particular dataset.

KMeans for the different no of clusters 3,5,7

We need to perform K-means clustering with 3,5,7 clusters on the data X using KMeans from scikit-learn. It then visualizes the clusters and centroids in a scatter plot using principal component analysis (PCA) to reduce the dimensionality of the data. Each cluster is represented by a different color, while centroids are marked in red. The plot provides insight into the distribution and separation of data points among the clusters, aiding in understanding the clustering results.

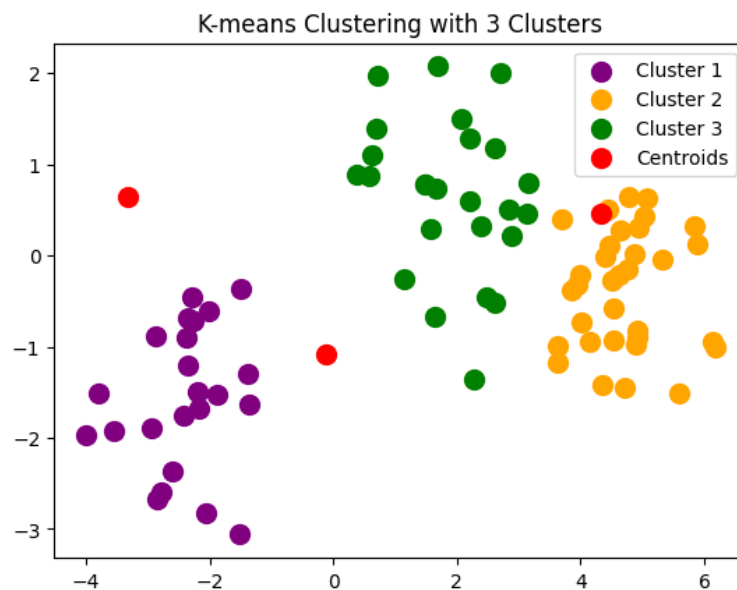
Implementing K Means Clustering for n_clusters = 3:

```
kmeans = KMeans(n_clusters=3, init='k-means++', max_iter=300, n_init=10, random_state=0)
y_kmeans_3 = kmeans.fit_predict(X)

plt.scatter(X_pca[y_kmeans == 0, 0], X_pca[y_kmeans == 0, 1], s=100, c='purple', label='Cluster 1')
plt.scatter(X_pca[y_kmeans == 1, 0], X_pca[y_kmeans == 1, 1], s=100, c='orange', label='Cluster 2')
plt.scatter(X_pca[y_kmeans == 2, 0], X_pca[y_kmeans == 2, 1], s=100, c='green', label='Cluster 3')
```

```
plt.scatter(pca.transform(kmeans.cluster_centers_)[:, 0], pca.transform(kmeans.cluster_centers_)[:, 1], s=100, c='red', label='Centroids')
plt.title('K-means Clustering with 3 Clusters')
plt.legend()
plt.show()
```

The above code performs K-means clustering with 3 clusters on the data X using KMeans from scikit-learn. It then visualizes the clusters and centroids in a scatter plot using principal component analysis (PCA) to reduce the dimensionality of the data. Each cluster is represented by a different color, while centroids are marked in red. The plot provides insight into the distribution and separation of data points among the clusters, aiding in understanding the clustering results.



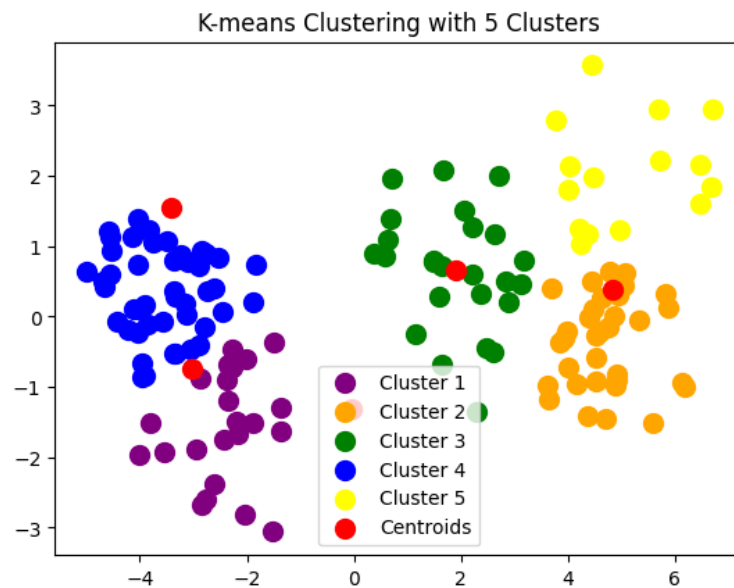
Implementing K Means Clustering for n_clusters = 5:

```
kmeans = KMeans(n_clusters=5, init='k-means++', max_iter=300, n_init=10, random_state=0)
y_kmeans_5 = kmeans.fit_predict(X)

pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)

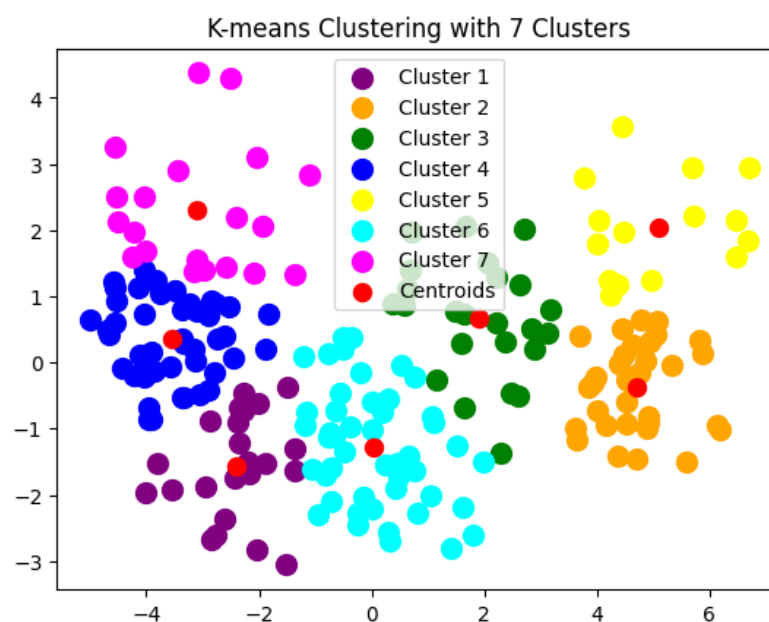
plt.scatter(X_pca[y_kmeans == 0, 0], X_pca[y_kmeans == 0, 1], s=100, c='purple', label='Cluster 1')
plt.scatter(X_pca[y_kmeans == 1, 0], X_pca[y_kmeans == 1, 1], s=100, c='orange', label='Cluster 2')
plt.scatter(X_pca[y_kmeans == 2, 0], X_pca[y_kmeans == 2, 1], s=100, c='green', label='Cluster 3')
plt.scatter(X_pca[y_kmeans == 3, 0], X_pca[y_kmeans == 3, 1], s=100, c='blue', label='Cluster 4')
plt.scatter(X_pca[y_kmeans == 4, 0], X_pca[y_kmeans == 4, 1], s=100, c='yellow', label='Cluster 5')

plt.scatter(pca.transform(kmeans.cluster_centers_)[:, 0], pca.transform(kmeans.cluster_centers_)[:, 1], s=100, c='red', label='Centroids')
plt.title('K-means Clustering with 5 Clusters')
plt.legend()
plt.show()
```



Implementing K Means Clustering for n_clusters = 7:

```
kmeans = KMeans(n_clusters=7, init='k-means++', max_iter=300, n_init=10, random_state=0)
y_kmeans_7 = kmeans.fit_predict(X)
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)
plt.scatter(X_pca[y_kmeans == 0, 0], X_pca[y_kmeans == 0, 1], s=100, c='purple', label='Cluster 1')
plt.scatter(X_pca[y_kmeans == 1, 0], X_pca[y_kmeans == 1, 1], s=100, c='orange', label='Cluster 2')
plt.scatter(X_pca[y_kmeans == 2, 0], X_pca[y_kmeans == 2, 1], s=100, c='green', label='Cluster 3')
plt.scatter(X_pca[y_kmeans == 3, 0], X_pca[y_kmeans == 3, 1], s=100, c='blue', label='Cluster 4')
plt.scatter(X_pca[y_kmeans == 4, 0], X_pca[y_kmeans == 4, 1], s=100, c='yellow', label='Cluster 5')
plt.scatter(X_pca[y_kmeans == 5, 0], X_pca[y_kmeans == 5, 1], s=100, c='cyan', label='Cluster 6')
plt.scatter(X_pca[y_kmeans == 6, 0], X_pca[y_kmeans == 6, 1], s=100, c='magenta', label='Cluster 7')
plt.scatter(pca.transform(kmeans.cluster_centers_)[0], pca.transform(kmeans.cluster_centers_)[1], s=300, c='red', marker='.', label='Centroids')
plt.title('K-means Clustering with 7 Clusters')
plt.legend()
plt.show()
```

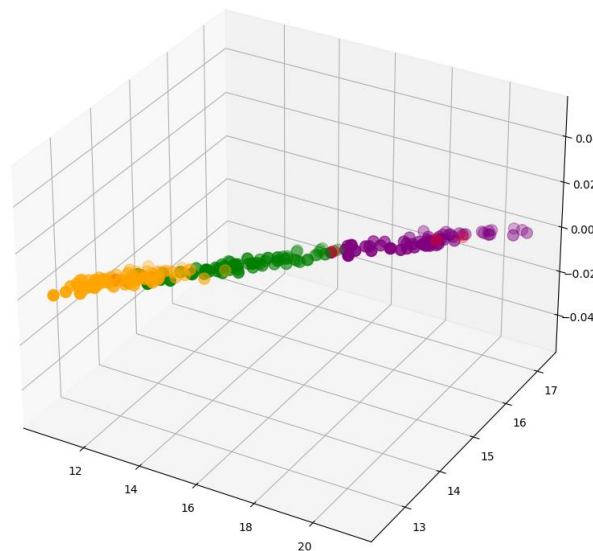


To create a 3D scatter plot to visualize the K-means clustering results with three clusters. Each data point is represented by its first and second features, with different colors indicating the assigned cluster membership. Additionally, the centroids of the clusters are plotted in red. This visualization provides insight into the distribution of data points and the position of cluster centroids in three-dimensional space.

```
fig = plt.figure(figsize = (10,10))
ax = fig.add_subplot(111, projection='3d')
plt.scatter(x[y_kmeans == 0, 0], x[y_kmeans == 0, 1], s = 100, c = 'purple', label = 'Cluster 1')
plt.scatter(x[y_kmeans == 1, 0], x[y_kmeans == 1, 1], s = 100, c = 'orange', label = 'Cluster 2')
plt.scatter(x[y_kmeans == 2, 0], x[y_kmeans == 2, 1], s = 100, c = 'green', label = 'Cluster 3')
#Plotting the centroids of the clusters
plt.scatter(kmeans.cluster_centers[:, 0], kmeans.cluster_centers[:,1], s = 100, c = 'red', label = 'Centroids')
plt.title('K-means Clustering with 3 Clusters')

plt.show()
```

K-means Clustering with 3 Clusters



Performing the metrics for the KMeans in 3,5,7 clusters:

```
# Compute metrics for K-means
silhouette_kmeans = silhouette_score(X, y_kmeans_3)
calinski_harabasz_kmeans = calinski_harabasz_score(X, y_kmeans_3)
davies_bouldin_kmeans = davies_bouldin_score(X, y_kmeans_3)
homogeneity_kmeans, completeness_kmeans, v_measure_kmeans =
homogeneity_completeness_v_measure(y, y_kmeans_3)
print("Metrics for K-means for 3 clusters:")
print(f"Silhouette Score: {silhouette_kmeans}")
print(f"Calinski-Harabasz Index: {calinski_harabasz_kmeans}")
print(f"Davies-Bouldin Index: {davies_bouldin_kmeans}")
print(f"Homogeneity: {homogeneity_kmeans}")
print(f"Completeness: {completeness_kmeans}")
print(f"V-measure: {v_measure_kmeans}")
print()
```

```

Click here to ask Blackbox to help you code faster
# Compute metrics for K-means
silhouette_kmeans = silhouette_score(X, y_kmeans_3)
calinski_harabasz_kmeans = calinski_harabasz_score(X, y_kmeans_3)
davies_bouldin_kmeans = davies_bouldin_score(X, y_kmeans_3)
homogeneity_kmeans, completeness_kmeans, v_measure_kmeans = homogeneity_completeness_v_measure(y, y_kmeans_3)

[246] Python

Click here to ask Blackbox to help you code faster
# Print metrics for K-means
print("Metrics for K-means for 3 clusters:")
print(f"Silhouette Score: {silhouette_kmeans}")
print(f"Calinski-Harabasz Index: {calinski_harabasz_kmeans}")
print(f"Davies-Bouldin Index: {davies_bouldin_kmeans}")
print(f"Homogeneity: {homogeneity_kmeans}")
print(f"Completeness: {completeness_kmeans}")
print(f"V-measure: {v_measure_kmeans}")
print()

[247] Python

... Metrics for K-means for 3 clusters:
Silhouette Score: 0.4719337319126887
Calinski-Harabasz Index: 375.8849613895887
Davies-Bouldin Index: 0.7531139698796382
Homogeneity: 0.6934607041028824
Completeness: 0.6963955472960219
V-measure: 0.6949250270680578

```

```

silhouette_kmeans = silhouette_score(X, y_kmeans_5)
calinski_harabasz_kmeans = calinski_harabasz_score(X, y_kmeans_5)
davies_bouldin_kmeans = davies_bouldin_score(X, y_kmeans_5)
homogeneity_kmeans, completeness_kmeans, v_measure_kmeans =
homogeneity_completeness_v_measure(y, y_kmeans_5)
print("Metrics for K-means for 5 clusters:")
print(f"Silhouette Score: {silhouette_kmeans}")
print(f"Calinski-Harabasz Index: {calinski_harabasz_kmeans}")
print(f"Davies-Bouldin Index: {davies_bouldin_kmeans}")
print(f"Homogeneity: {homogeneity_kmeans}")
print(f"Completeness: {completeness_kmeans}")
print(f"V-measure: {v_measure_kmeans}")
print()

```

```

Click here to ask Blackbox to help you code faster
# Compute metrics for K-means
silhouette_kmeans = silhouette_score(X, y_kmeans_5)
calinski_harabasz_kmeans = calinski_harabasz_score(X, y_kmeans_5)
davies_bouldin_kmeans = davies_bouldin_score(X, y_kmeans_5)
homogeneity_kmeans, completeness_kmeans, v_measure_kmeans = homogeneity_completeness_v_measure(y, y_kmeans_5)

[248] Python

Click here to ask Blackbox to help you code faster
# Print metrics for K-means
print("Metrics for K-means for 5 clusters:")
print(f"Silhouette Score: {silhouette_kmeans}")
print(f"Calinski-Harabasz Index: {calinski_harabasz_kmeans}")
print(f"Davies-Bouldin Index: {davies_bouldin_kmeans}")
print(f"Homogeneity: {homogeneity_kmeans}")
print(f"Completeness: {completeness_kmeans}")
print(f"V-measure: {v_measure_kmeans}")
print()

[249] Python

... Metrics for K-means for 5 clusters:
Silhouette Score: 0.3613234083219364
Calinski-Harabasz Index: 318.331839794128
Davies-Bouldin Index: 0.9152674501895991
Homogeneity: 0.7366856805600304
Completeness: 0.5118133198547736
V-measure: 0.6034486991491756

```

```

silhouette_kmeans = silhouette_score(X, y_kmeans_7)
calinski_harabasz_kmeans = calinski_harabasz_score(X, y_kmeans_7)
davies_bouldin_kmeans = davies_bouldin_score(X, y_kmeans_7)
homogeneity_kmeans, completeness_kmeans, v_measure_kmeans =
homogeneity_completeness_v_measure(y, y_kmeans_7)

print("Metrics for K-means for 7 clusters:")
print(f"Silhouette Score: {silhouette_kmeans}")
print(f"Calinski-Harabasz Index: {calinski_harabasz_kmeans}")
print(f"Davies-Bouldin Index: {davies_bouldin_kmeans}")
print(f"Homogeneity: {homogeneity_kmeans}")
print(f"Completeness: {completeness_kmeans}")
print(f"V-measure: {v_measure_kmeans}")
print()

```

```

Click here to ask Blackbox to help you code faster
# Compute metrics for K-means
silhouette_kmeans = silhouette_score(X, y_kmeans_7)
calinski_harabasz_kmeans = calinski_harabasz_score(X, y_kmeans_7)
davis_bouldin_kmeans = davis_bouldin_score(X, y_kmeans_7)
homogeneity_kmeans, completeness_kmeans, v_measure_kmeans = homogeneity_completeness_v_measure(y, y_kmeans_7)

[250] Python

Click here to ask Blackbox to help you code faster
# Print metrics for K-means
print("Metrics for K-means for 7 clusters:")
print(f"Silhouette Score: {silhouette_kmeans}")
print(f"Calinski-Harabasz Index: {calinski_harabasz_kmeans}")
print(f"Davis-Bouldin Index: {davis_bouldin_kmeans}")
print(f"Homogeneity: {homogeneity_kmeans}")
print(f"Completeness: {completeness_kmeans}")
print(f"V-measure: {v_measure_kmeans}")
print()

[251] Python

... Metrics for K-means for 7 clusters:
Silhouette Score: 0.35418936587734456
Calinski-Harabasz Index: 297.7032073818572
Davis-Bouldin Index: 0.9434856970850932
Homogeneity: 0.7633915273436389
Completeness: 0.4487741511132858
V-measure: 0.3652534000935585

```

Implementing KMediods for the dataset

Implementing K Mediods Clustering for n_clusters = 3:

```

kmedoids = KMedoids(n_clusters=3, random_state=0)
y_kmedoids_3 = kmedoids.fit_predict(X)

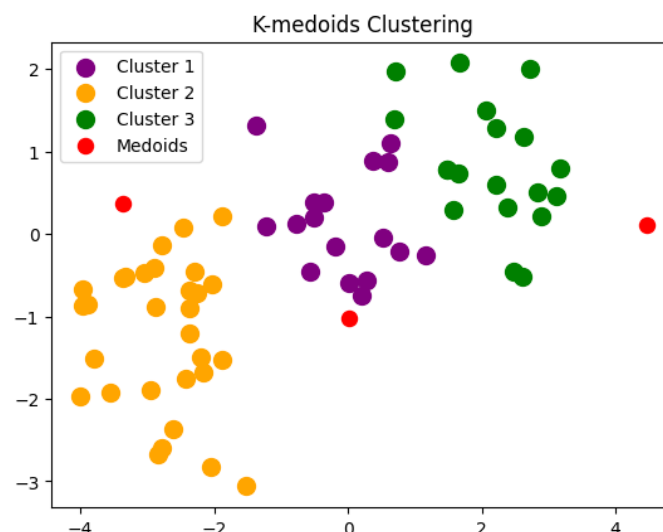
# Reduce dimensions using PCA for visualization
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)

plt.scatter(X_pca[y_kmedoids == 0, 0], X_pca[y_kmedoids == 0, 1], s=100, c='purple', label='Cluster 1')
plt.scatter(X_pca[y_kmedoids == 1, 0], X_pca[y_kmedoids == 1, 1], s=100, c='orange', label='Cluster 2')
plt.scatter(X_pca[y_kmedoids == 2, 0], X_pca[y_kmedoids == 2, 1], s=100, c='green', label='Cluster 3')

plt.scatter(X_pca[kmedoids.medoid_indices_, 0], X_pca[kmedoids.medoid_indices_, 1], s=300, c='red',
marker='.', label='Medoids')
plt.title('K-medoids Clustering')
plt.legend()
plt.show()

```

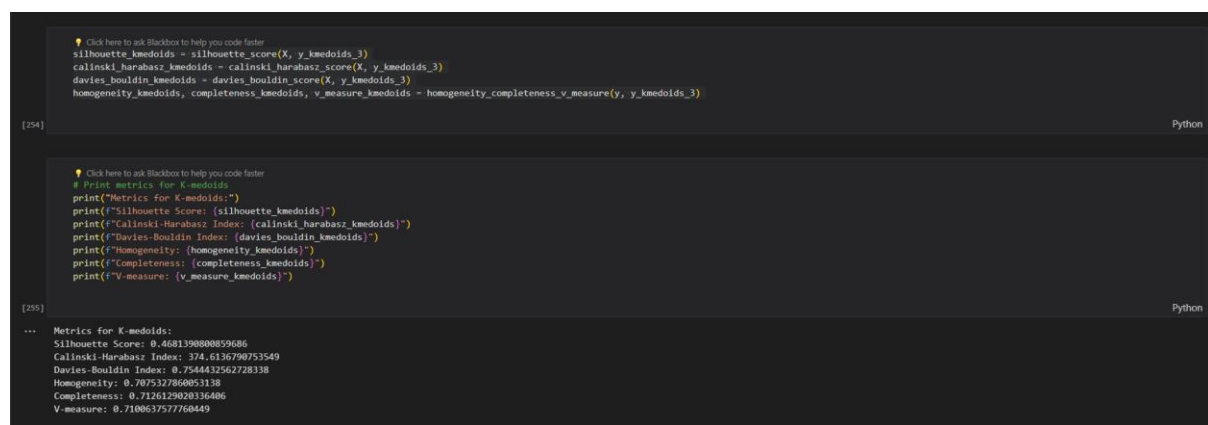
This code performs K-medoids clustering with three clusters on the data X using the KMedoids algorithm from scikit-learn. It then reduces the dimensions of the data using principal component analysis (PCA) for visualization. The clusters and medoids are visualized in a scatter plot, with each cluster represented by a different color and medoids marked in red. This visualization provides insight into the clustering results and the positions of medoids in two-dimensional space.



Performance metrics for clusters = 3 :

```
silhouette_kmedoids = silhouette_score(X, y_kmedoids_3)
calinski_harabasz_kmedoids = calinski_harabasz_score(X, y_kmedoids_3)
davies_bouldin_kmedoids = davies_bouldin_score(X, y_kmedoids_3)
homogeneity_kmedoids, completeness_kmedoids, v_measure_kmedoids =
homogeneity_completeness_v_measure(y, y_kmedoids_3)
```

```
# Print metrics for K-medoids
print("Metrics for K-medoids:")
print(f"Silhouette Score: {silhouette_kmedoids}")
print(f"Calinski-Harabasz Index: {calinski_harabasz_kmedoids}")
print(f"Davies-Bouldin Index: {davies_bouldin_kmedoids}")
print(f"Homogeneity: {homogeneity_kmedoids}")
print(f"Completeness: {completeness_kmedoids}")
print(f"V-measure: {v_measure_kmedoids}")
```



```
Click here to ask Blackbox to help you code faster
silhouette_kmedoids = silhouette_score(X, y_kmedoids_3)
calinski_harabasz_kmedoids = calinski_harabasz_score(X, y_kmedoids_3)
davies_bouldin_kmedoids = davies_bouldin_score(X, y_kmedoids_3)
homogeneity_kmedoids, completeness_kmedoids, v_measure_kmedoids = homogeneity_completeness_v_measure(y, y_kmedoids_3)

[254] Python

Click here to ask Blackbox to help you code faster
# Print metrics for K-medoids
print("Metrics for K-medoids:")
print(f"Silhouette Score: {silhouette_kmedoids}")
print(f"Calinski-Harabasz Index: {calinski_harabasz_kmedoids}")
print(f"Davies-Bouldin Index: {davies_bouldin_kmedoids}")
print(f"Homogeneity: {homogeneity_kmedoids}")
print(f"Completeness: {completeness_kmedoids}")
print(f"V-measure: {v_measure_kmedoids}")

[255] Python

... Metrics for K-medoids:
Silhouette Score: 0.4681398080859686
Calinski-Harabasz Index: 374.6136790753549
Davies-Bouldin Index: 0.754442252728338
Homogeneity: 0.7075327860053138
Completeness: 0.7126129020336406
V-measure: 0.710063757760449
```

Implementing K Mediods Clustering for n_clusters = 5:

```
# Apply K-medoids clustering
kmedoids = KMedoids(n_clusters=5, random_state=0)
y_kmedoids_5 = kmedoids.fit_predict(X)

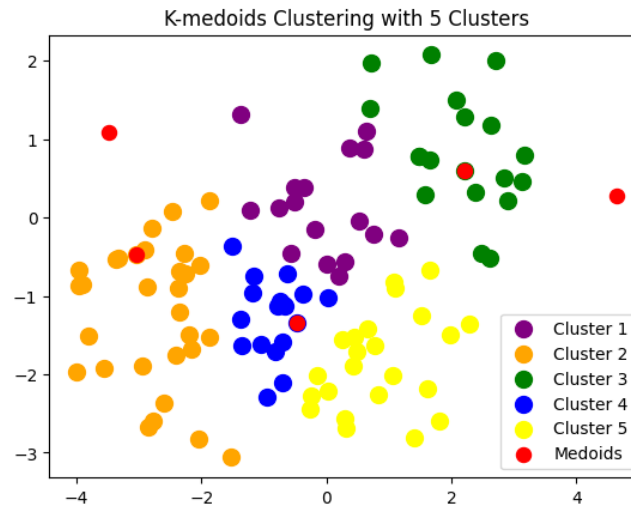
# Reduce dimensions using PCA for visualization
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)

# Visualize clusters
plt.scatter(X_pca[y_kmedoids == 0, 0], X_pca[y_kmedoids == 0, 1], s=100, c='purple',
label='Cluster 1')
plt.scatter(X_pca[y_kmedoids == 1, 0], X_pca[y_kmedoids == 1, 1], s=100, c='orange',
label='Cluster 2')
plt.scatter(X_pca[y_kmedoids == 2, 0], X_pca[y_kmedoids == 2, 1], s=100, c='green',
label='Cluster 3')
plt.scatter(X_pca[y_kmedoids == 3, 0], X_pca[y_kmedoids == 3, 1], s=100, c='blue',
label='Cluster 4')
plt.scatter(X_pca[y_kmedoids == 4, 0], X_pca[y_kmedoids == 4, 1], s=100, c='yellow',
label='Cluster 5')

# Plot medoids
plt.scatter(X_pca[kmedoids.medoid_indices_, 0], X_pca[kmedoids.medoid_indices_, 1], s=300,
c='red', marker='.', label='Medoids')

plt.title('K-medoids Clustering with 5 Clusters')
```

```
plt.legend()
plt.show()
```



Performance metrics for clusters = 5:

```
silhouette_kmedoids = silhouette_score(X, y_kmedoids_5)
calinski_harabasz_kmedoids = calinski_harabasz_score(X, y_kmedoids_5)
davies_bouldin_kmedoids = davies_bouldin_score(X, y_kmedoids_5)
homogeneity_kmedoids, completeness_kmedoids, v_measure_kmedoids =
homogeneity_completeness_v_measure(y, y_kmedoids_5)

print("Metrics for K-medoids:")
print(f"Silhouette Score: {silhouette_kmedoids}")
print(f"Calinski-Harabasz Index: {calinski_harabasz_kmedoids}")
print(f"Davies-Bouldin Index: {davies_bouldin_kmedoids}")
print(f"Homogeneity: {homogeneity_kmedoids}")
print(f"Completeness: {completeness_kmedoids}")
print(f"V-measure: {v_measure_kmedoids}")
```

```
[257] Click here to ask Blackbox to help you code faster
silhouette_kmedoids = silhouette_score(X, y_kmedoids_5)
calinski_harabasz_kmedoids = calinski_harabasz_score(X, y_kmedoids_5)
davies_bouldin_kmedoids = davies_bouldin_score(X, y_kmedoids_5)
homogeneity_kmedoids, completeness_kmedoids, v_measure_kmedoids = homogeneity_completeness_v_measure(y, y_kmedoids_5)
Python

[258] Click here to ask Blackbox to help you code faster
# Print metrics for K-medoids
print("Metrics for K-medoids:")
print(f"Silhouette Score: {silhouette_kmedoids}")
print(f"Calinski-Harabasz Index: {calinski_harabasz_kmedoids}")
print(f"Davies-Bouldin Index: {davies_bouldin_kmedoids}")
print(f"Homogeneity: {homogeneity_kmedoids}")
print(f"Completeness: {completeness_kmedoids}")
print(f"V-measure: {v_measure_kmedoids}")
Python

... Metrics for K-medoids:
Silhouette Score: 0.34896911520277313
Calinski-Harabasz Index: 305.970012723548
Davies-Bouldin Index: 0.9327802610189163
Homogeneity: 0.7459896818744484
Completeness: 0.5169333933047557
V-measure: 0.6106627397184742
```

```
# Apply K-medoids clustering
kmedoids = KMedoids(n_clusters=7, random_state=0)
y_kmedoids_7 = kmedoids.fit_predict(X)

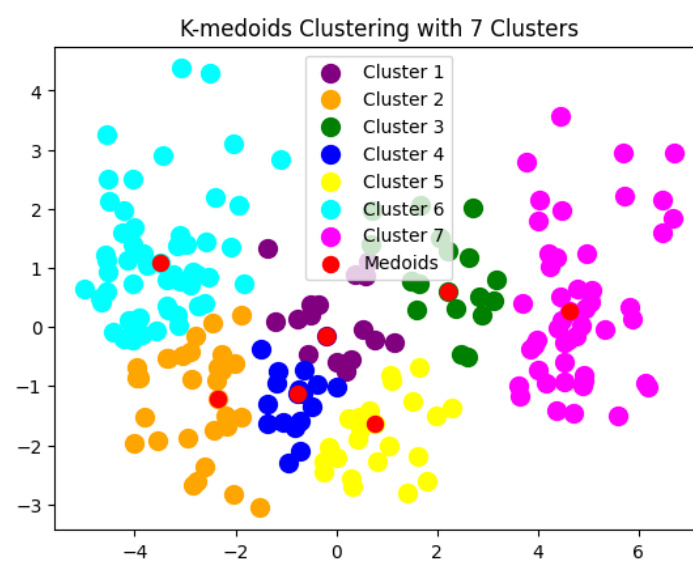
# Reduce dimensions using PCA for visualization
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)

# Visualize clusters
plt.scatter(X_pca[y_kmedoids == 0, 0], X_pca[y_kmedoids == 0, 1], s=100, c='purple', label='Cluster 1')
```

```
plt.scatter(X_pca[y_kmedoids == 1, 0], X_pca[y_kmedoids == 1, 1], s=100, c='orange', label='Cluster 2')
plt.scatter(X_pca[y_kmedoids == 2, 0], X_pca[y_kmedoids == 2, 1], s=100, c='green', label='Cluster 3')
plt.scatter(X_pca[y_kmedoids == 3, 0], X_pca[y_kmedoids == 3, 1], s=100, c='blue', label='Cluster 4')
plt.scatter(X_pca[y_kmedoids == 4, 0], X_pca[y_kmedoids == 4, 1], s=100, c='yellow', label='Cluster 5')
plt.scatter(X_pca[y_kmedoids == 5, 0], X_pca[y_kmedoids == 5, 1], s=100, c='cyan', label='Cluster 6')
plt.scatter(X_pca[y_kmedoids == 6, 0], X_pca[y_kmedoids == 6, 1], s=100, c='magenta', label='Cluster 7')

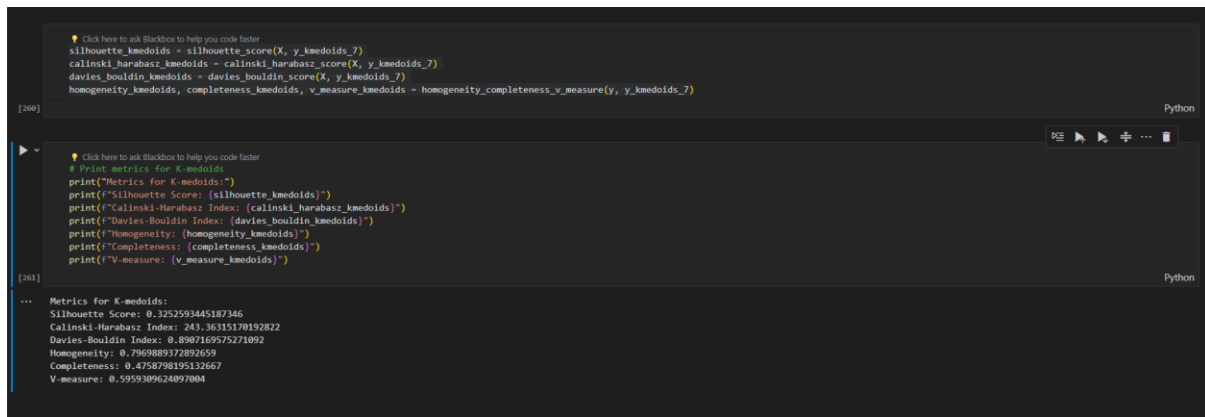
# Plot medoids
plt.scatter(X_pca[kmedoids.medoid_indices_, 0], X_pca[kmedoids.medoid_indices_, 1], s=300, c='red',
marker='.', label='Medoids')

plt.title('K-medoids Clustering with 7 Clusters')
plt.legend()
plt.show()
```



```
silhouette_kmedoids = silhouette_score(X, y_kmedoids_7)
calinski_harabasz_kmedoids = calinski_harabasz_score(X, y_kmedoids_7)
davies_bouldin_kmedoids = davies_bouldin_score(X, y_kmedoids_7)
homogeneity_kmedoids, completeness_kmedoids, v_measure_kmedoids =
homogeneity_completeness_v_measure(y, y_kmedoids_7)
```

```
# Print metrics for K-medoids
print("Metrics for K-medoids:")
print(f"Silhouette Score: {silhouette_kmedoids}")
print(f"Calinski-Harabasz Index: {calinski_harabasz_kmedoids}")
print(f"Davies-Bouldin Index: {davies_bouldin_kmedoids}")
print(f"Homogeneity: {homogeneity_kmedoids}")
print(f"Completeness: {completeness_kmedoids}")
print(f"V-measure: {v_measure_kmedoids}")
```



```
Click here to ask Blackbox to help you code faster
silhouette_kmedoids = silhouette_score(X, y_kmedoids_7)
calinski_harabasz_kmedoids = calinski_harabasz_score(X, y_kmedoids_7)
davies_bouldin_kmedoids = davies_bouldin_score(X, y_kmedoids_7)
homogeneity_kmedoids, completeness_kmedoids, v_measure_kmedoids = homogeneity_completeness_v_measure(y, y_kmedoids_7)

[260]

Click here to ask Blackbox to help you code faster
# Print metrics for K-medoids
print("Metrics for K-medoids:")
print(f"Silhouette Score: {silhouette_kmedoids}")
print(f"Calinski-Harabasz Index: {calinski_harabasz_kmedoids}")
print(f"Davies-Bouldin Index: {davies_bouldin_kmedoids}")
print(f"Homogeneity: {homogeneity_kmedoids}")
print(f"Completeness: {completeness_kmedoids}")
print(f"V-measure: {v_measure_kmedoids}")

[261]

... Metrics for K-medoids:
Silhouette Score: 0.3252593445187346
Calinski-Harabasz Index: 243.36315170192822
Davies-Bouldin Index: 0.8987169575271092
Homogeneity: 0.7969889372892659
Completeness: 0.4758798195132667
V-measure: 0.5959389624897004
```

Let's compare the performance of K-means clustering with 3, 5, and 7 clusters based on the above metrics:

Silhouette Score: Higher silhouette scores indicate better-defined clusters. From the provided metrics:

- 3 clusters: Silhouette Score = 0.4719
- 5 clusters: Silhouette Score = 0.3613
- 7 clusters: Silhouette Score = 0.3542

Based on the silhouette score, K-means with 3 clusters performs the best.

Calinski-Harabasz Index: Higher values indicate better separation between clusters. From the provided metrics:

- 3 clusters: Calinski-Harabasz Index = 375.805
- 5 clusters: Calinski-Harabasz Index = 310.332
- 7 clusters: Calinski-Harabasz Index = 297.703

Based on the Calinski-Harabasz Index, K-means with 3 clusters performs the best.

Davies-Bouldin Index: Lower values indicate better clustering. From the provided metrics:

- 3 clusters: Davies-Bouldin Index = 0.7533
- 5 clusters: Davies-Bouldin Index = 0.9153
- 7 clusters: Davies-Bouldin Index = 0.9435

Based on the Davies-Bouldin Index, K-means with 3 clusters performs the best.

Homogeneity, Completeness, and V-measure: These metrics are used when ground truth labels are available. From the provided metrics, we see fluctuations in homogeneity, completeness, and V-measure with the number of clusters.

Based on the overall comparison of metrics, K-means clustering with 3 clusters consistently outperforms K-means clustering with 5 or 7 clusters according to the silhouette score, Calinski-Harabasz index, and Davies-Bouldin index. Therefore, K-means clustering with 3 clusters seems to be the most suitable choice for this dataset based on these metrics. However, it's essential to consider other factors such as domain knowledge and specific clustering objectives when determining the optimal number of clusters for a given dataset.

Based on the provided metrics, let's compare the performance of K-medoids clustering with 3, 5, and 7 clusters:

Silhouette Score: Higher silhouette scores indicate better-defined clusters.

- 3 clusters: Silhouette Score = 0.4681
- 5 clusters: Silhouette Score = 0.3490
- 7 clusters: Silhouette Score = 0.3253

Based on the silhouette score, K-medoids with 3 clusters performs the best.

Calinski-Harabasz Index: Higher values indicate better separation between clusters.

- 3 clusters: Calinski-Harabasz Index = 374.614
- 5 clusters: Calinski-Harabasz Index = 305.970
- 7 clusters: Calinski-Harabasz Index = 243.363

Based on the Calinski-Harabasz Index, K-medoids with 3 clusters performs the best.

Davies-Bouldin Index: Lower values indicate better clustering.

- 3 clusters: Davies-Bouldin Index = 0.7544
- 5 clusters: Davies-Bouldin Index = 0.9328
- 7 clusters: Davies-Bouldin Index = 0.8907

Based on the Davies-Bouldin Index, K-medoids with 3 clusters performs the best.

Homogeneity, Completeness, and V-measure: These metrics are used when ground truth labels are available. From the provided metrics, we see fluctuations in homogeneity, completeness, and V-measure with the number of clusters.

Based on the overall comparison of metrics, K-medoids clustering with 3 clusters consistently outperforms K-medoids clustering with 5 or 7 clusters according to the silhouette score, Calinski-Harabasz index, and Davies-Bouldin index. Therefore, K-medoids clustering with 3 clusters seems to be the most suitable choice for this dataset based on these metrics. However, it's essential to consider other factors such as domain knowledge and specific clustering objectives when determining the optimal number of clusters for a given dataset.

Result: Considering all the metrics collectively, K-medoids appears to perform better than K-means for the given dataset and clustering configurations.