We initially received a dataset with the label 'diagnosis' as (M = malignant, B = benign). However, we suspect that in addition to being classified as malignant or benign, malignant cancer can actually be further categorized into four stages, which are Stage I (1) to IV (4).

In [43]:
```
df1 = df.copy()
df1['diagnosis'].value_counts()
```

Out[43]:
```
B    357
M    212
Name: diagnosis, dtype: int64
```

Based on our analysis target, we first remove unnecessary columns such as features related to standard error and worst measurements. We also exclude ID and diagnosis features. We will only keep the columns that end with '_mean' for the further analysis.

After cleaning up, we have 212 instances and 10 features.

In [44]:
```
df_M = df1[df1['diagnosis'] == 'M']
df_M = df_M.filter(regex='^(id|diagnosis|.*_mean$)')
df1 = df_M.drop(['id','diagnosis'], axis=1)
df1.head(5)
df1.shape
```

Out[44]:
```
(212, 10)
```

To investigate the possibility of classifying malignant cancer into four stages, we will perform further analysis using clustering methods. There are various clustering algorithms available, such as K-means clustering, spectral clustering,hierarchical clustering, DBSCAN, Gaussian Mixture Model, and K-medoids, each with different assumptions and characteristics.

K-means clustering assumes that clusters have a linear boundary and a circular shape, making it suitable for data that can be separated into well-defined, compact clusters. Spectral clustering is a kernelized version of K-means that can discover non-linear boundaries between clusters. Hierarchical clustering is a bottom-up approach that groups clusters into a binary tree structure. All clustering method above, they need to predefine the number of clusters.

Later on, we also have DBSCAN and K-medoids, which do not need to specify k. DBSCAN is particularly effective for data with clusters of irregular shapes and largely depending on density seperation. K-medoids, different from k-means, choose actual point as centers and able to handle non-spherical clusters.

Of these methods, K-means clustering is often the first choice due to its simplicity, speed, and guaranteed convergence. Let's discover the characteristics of our data with K-means first, then decide which clustering method is most suitable.

K-means clustering is an unsupervised learning method that does not require labeled data. Since our filtered dataset is already small, further splitting it into training and testing sets may result in an insufficient number of data points, which could lead to overfitting due to the increased flexibility of the model. Therefore, we can perform K-means clustering on the entire dataset without splitting it into training and testing sets.

In [45]:
```
from sklearn.model_selection import train_test_split
X_train=df1
```

K-means clustering is sensitive to the scale of the data because it uses Euclidean distance to measure the similarity between data points. When the scale of the features is different, features with a larger scale will have a greater impact on the distance calculations, potentially leading to biased cluster assignments.

To avoid this issue, we standardize the training data.

In [46]:
```
from sklearn.preprocessing import StandardScaler
X_train = StandardScaler().fit_transform(X_train)
```

In [47]:
```
#clustering into four stages
from sklearn.cluster import KMeans
k = 4
kmeans = KMeans(n_clusters=k,n_init=10)
y_pred = kmeans.fit_predict(X_train)
y_pred
```

Out[47]:
```
array([3, 0, 0, 1, 0, 1, 0, 1, 1, 1, 2, 2, 3, 2, 1, 1, 2, 1, 0, 1, 0, 1,
       1, 1, 0, 1, 2, 0, 1, 1, 0, 1, 2, 2, 2, 2, 2, 1, 0, 1, 2, 0, 1, 0,
       2, 0, 1, 1, 1, 1, 0, 0, 2, 2, 1, 3, 3, 0, 0, 2, 0, 2, 1, 0, 2, 2,
       1, 3, 1, 1, 2, 0, 3, 2, 2, 0, 2, 2, 0, 2, 1, 2, 1, 0, 0, 0, 0, 2,
       0, 2, 1, 0, 3, 3, 2, 2, 2, 1, 1, 1, 1, 2, 0, 2, 2, 3, 1, 2, 2, 0,
       3, 0, 2, 1, 0, 0, 2, 1, 1, 0, 0, 0, 0, 0, 0, 0, 2, 0, 2, 0, 1, 1,
       1, 0, 2, 2, 2, 2, 0, 0, 2, 2, 0, 0, 1, 2, 0, 3, 0, 0, 3, 1, 1, 2,
       0, 0, 0, 0, 1, 3, 2, 0, 0, 0, 3, 1, 0, 0, 1, 2, 0, 1, 3, 3, 0, 2,
       1, 1, 0, 0, 2, 2, 2, 0, 0, 0, 2, 3, 0, 1, 0, 2, 0, 0, 0, 1, 0, 1,
       1, 2, 0, 0, 3, 0, 0, 2, 1, 3, 0, 0, 2, 3])
```

In [48]:
```
y_pred is kmeans.labels_
```

Out[48]:
```
True
```

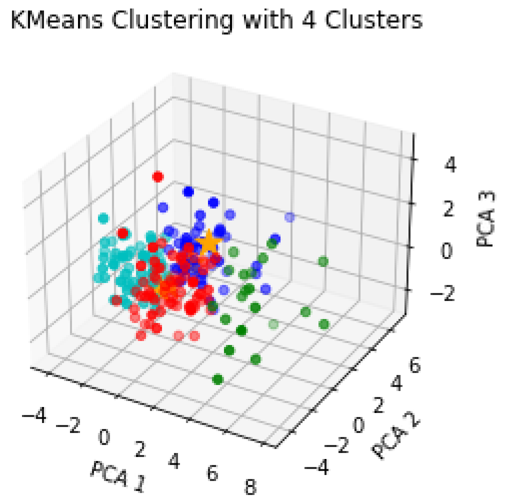
In [49]:
```
kmeans.cluster_centers_
```

```
Out[49]: array([[ 0.66524741,  0.15286286,  0.626824  ,  0.61241303, -0.29347298,
                 -0.16890106,  0.06743798,  0.25793377, -0.33803206, -0.48434731],
                [-0.90692545, -0.1465055 , -0.83844856, -0.85551602,  0.90017028,
                  0.59770822,  0.21732774, -0.03388368,  0.72075227,  1.06468975],
                [-0.56266946, -0.136057  , -0.62698595, -0.56368367, -0.81279105,
                 -0.92902435, -0.98965565, -1.01683323, -0.59736453, -0.61509858],
                [ 1.57666779,  0.20861553,  1.72130163,  1.64171668,  0.98116189,
                  1.67930172,  1.9984756 ,  2.03774895,  1.03257579,  0.69160909]])
```

We can see that our clusters are centered in 10 dimensions, as visualizing them in this high-dimensional space may not be easily interpretable. Therefore, reducing the dimensionality of the data and visualizing it in 2 dimensions using PCA may be a better approach. We are using PCA because our dataset is small and not too complexed. Compare to t-Distributed Stochastic Neighbor Embedding (t-SNE) with a nonlinear boundary and complexity, PCA is better to fit with our needs.

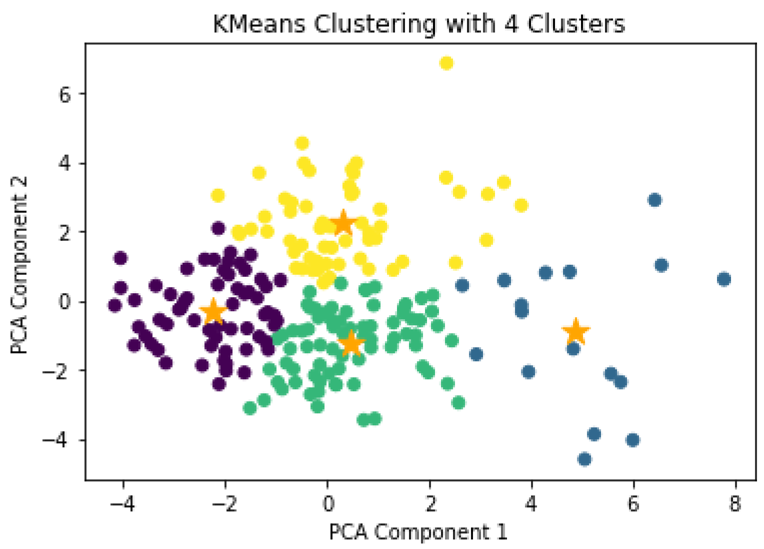
PCA assumes that the data is centered around the origin. Since we have already applied the StandardScaler in the beginning of our analysis, the data has been centered and scaled, satisfying the centering assumption of PCA.

```
In [50]: from sklearn.cluster import KMeans
         from sklearn.decomposition import PCA
         import matplotlib.pyplot as plt
         import numpy as np

         kmeans = KMeans(n_clusters=4, random_state=42)
         kmeans.fit(X_train)

         pca = PCA(n_components=3, random_state=42)
         X_train_pca = pca.fit_transform(X_train)

         # Visualize the clusters and cluster centers in 3D space
         fig = plt.figure()
         ax = fig.add_subplot(111, projection='3d')
         colors = ['r', 'g', 'b', 'c']

         for i in range(4):
             cluster_points = X_train_pca[kmeans.labels_ == i]
             ax.scatter(cluster_points[:, 0], cluster_points[:, 1], cluster_points[:, 2], c=colors[i])
             ax.scatter(kmeans.cluster_centers_[i][0], kmeans.cluster_centers_[i][1], kmeans.cluster_centers_[i][2], s=200, c='o

         ax.set_xlabel('PCA 1')
         ax.set_ylabel('PCA 2')
         ax.set_zlabel('PCA 3')
         ax.set_title('KMeans Clustering with 4 Clusters')
         plt.show()
```



From the 3D plot, we can confirm that we do not have obvious rolling and folding appearances. Thus, we further try the 2D plot.

In [51]:
```
#Instead of three dimensions, two dimensions are enough.
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt

#pca
pca = PCA(n_components=2)
X_train_pca = pca.fit_transform(X_train)


kmeans = KMeans(n_clusters=4, random_state=42)
kmeans.fit(X_train_pca)
labels = kmeans.labels_
centers = kmeans.cluster_centers_

# Visualize
plt.scatter(X_train_pca[:, 0], X_train_pca[:, 1], c=labels)
plt.scatter(centers[:, 0], centers[:, 1], marker='*', s=200, c='orange')
plt.title('KMeans Clustering with 4 Clusters')
plt.xlabel('PCA Component 1')
plt.ylabel('PCA Component 2')
plt.show()
```



We can observe four clusters from the plot above. However, these boundaries may not be optimal, as clusters are very close to each other, which may indicate improving space.

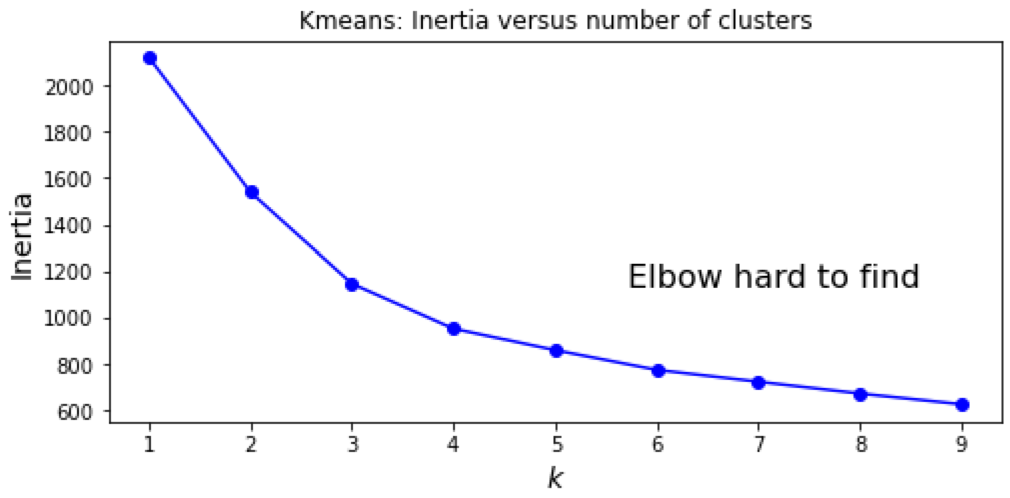
To improve the clustering, we can explore different methods to determine the optimal number of clusters. There are three common approaches:

- Assume a predetermined number of clusters based on prior knowledge or assumptions. In our case, we guessed that there may be four clusters, corresponding to stage I to stage IV.
- Use the inertia metric to measure the distance between each instance and its centroid. We can plot inertia as a function of k, the number of clusters, and analyze the resulting curve, paying attention to the elbow.
- Use silhouette score and silhouette coefficient. Silhoutte score is the average of silhouette coefficient over all instance.

In [52]:
```
kmeans_per_k = [KMeans(n_clusters=k, random_state=3224, n_init=10).fit(X_train)
                for k in range(1, 10)]
inertias = [model.inertia_ for model in kmeans_per_k]
inertias
```

```
Out[52]: [2120.0,
          1538.6715980594377,
          1145.3845133159693,
          951.6716535973347,
          858.719166726742,
          773.5739969765708,
          722.6585158538046,
          671.7025318026689,
          625.361081526653]
```

```
In [53]: plt.figure(figsize=(8, 3.5))
         plt.plot(range(1, 10), inertias, "bo-")
         plt.xlabel("$k$", fontsize=14)
         plt.ylabel("Inertia", fontsize=14)
         plt.annotate('Elbow hard to find',
                      xy=(2, inertias[1]),
                      xytext=(0.55, 0.44),
                      textcoords='figure fraction',
                      fontsize=16
                     #  arrowprops=dict(facecolor='black', shrink=0.1)
                     )
         plt.title('Kmeans: Inertia versus number of clusters')
         plt.show()
```

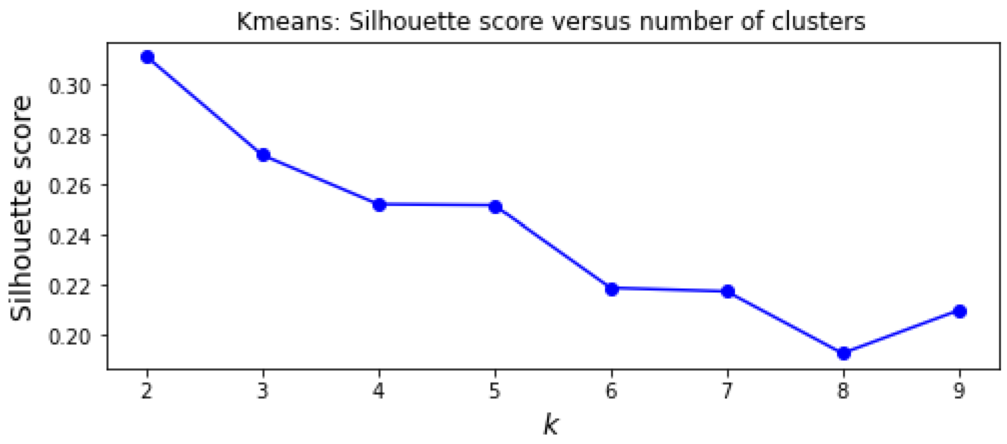


Inertia plot versus number of cluster is almost a linear line. Although k=2 and k=3 look like elbow. It can be difficult for us to determine the optimal number of clusters using this method. Let's try third approaches below.

In [54]:
```python
from sklearn.metrics import silhouette_score
```

In [55]:
```python
silhouette_scores = [silhouette_score(X_train, model.labels_)
                     for model in kmeans_per_k[1:]]
```

In [56]:
```python
plt.figure(figsize=(8, 3))
plt.plot(range(2, 10), silhouette_scores, "bo-")
plt.xlabel("$k$", fontsize=14)
plt.ylabel("Silhouette score", fontsize=14)
plt.title('Kmeans: Silhouette score versus number of clusters')
plt.show()
```



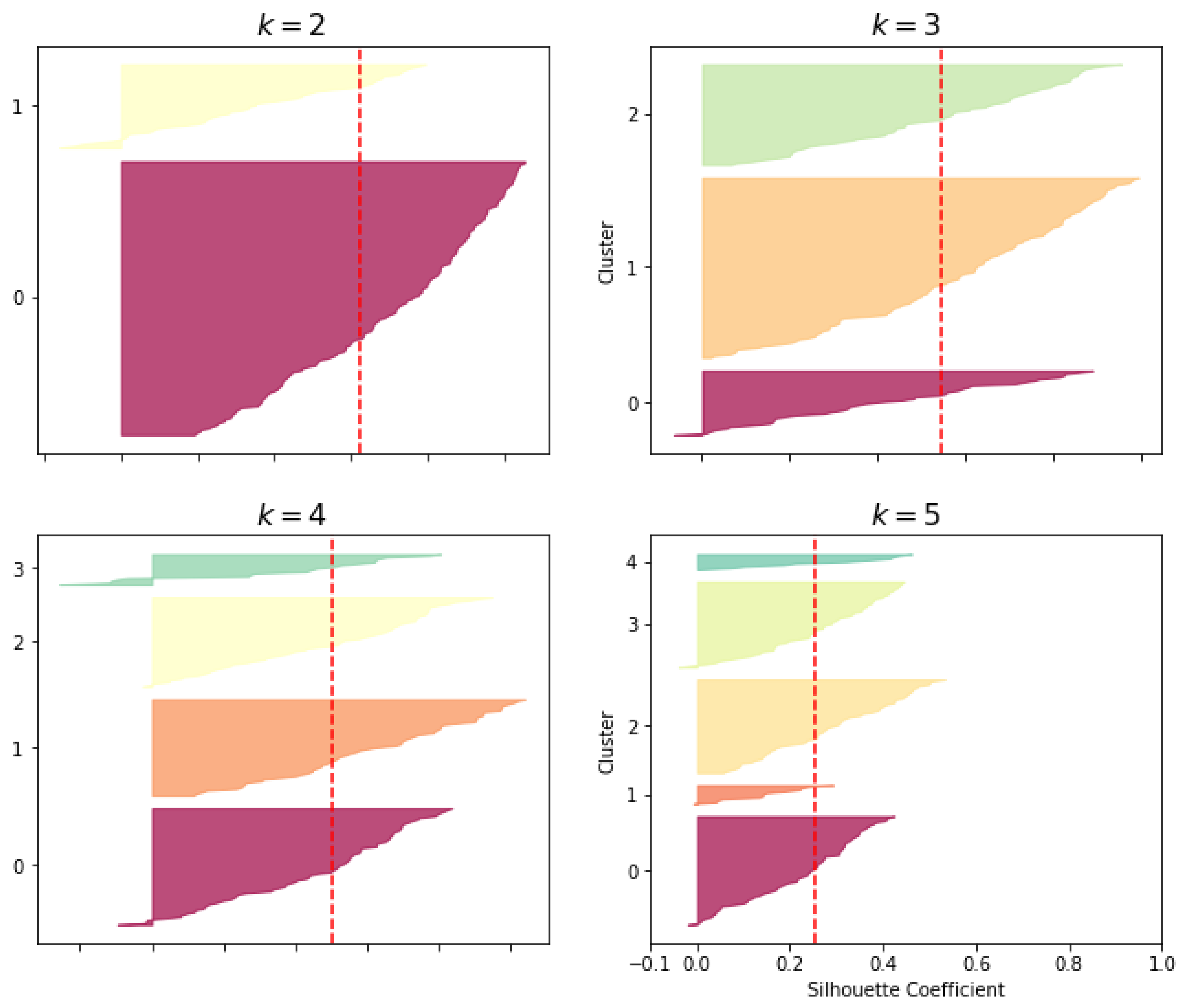
The silhouette coefficient ranges between -1 and 1, with a higher value indicating a better clustering solution. When evaluating different cluster numbers, a silhouette score between 0.25 and 0.5 suggests moderate separation between the clusters, while a score above 0.5 indicates strong separation.

In our case, for k=2 and k=4, the silhouette scores are around 0.32 and 0.25, respectively. However, these scores do not show a clear difference because they are still in the moderate range, making it difficult to determine the optimal number of clusters.

To gain more insights, we can use a silhouette diagram, which provides a visual representation of the silhouette scores for each data point in each cluster. It can help use to visualize and decide the reasonable balance between separation and interpretability.

In [57]:
```python
from sklearn.metrics import silhouette_samples
from matplotlib.ticker import FixedLocator, FixedFormatter

plt.figure(figsize=(11, 9))
```

```python
for k in (2, 3, 4, 5):
    plt.subplot(2, 2, k - 1)

    y_pred = kmeans_per_k[k - 1].labels_
    silhouette_coefficients = silhouette_samples(X_train, y_pred)

    padding = len(X_train) // 30
    pos = padding
    ticks = []
    for i in range(k):
        coeffs = silhouette_coefficients[y_pred == i]
        coeffs.sort()

        color = mpl.cm.Spectral(i / k)
        plt.fill_betweenx(np.arange(pos, pos + len(coeffs)), 0, coeffs,
                          facecolor=color, edgecolor=color, alpha=0.7)
        ticks.append(pos + len(coeffs) // 2)
        pos += len(coeffs) + padding

    plt.gca().yaxis.set_major_locator(FixedLocator(ticks))
    plt.gca().yaxis.set_major_formatter(FixedFormatter(range(k)))
    if k in (3, 5):
        plt.ylabel("Cluster")

    if k in (5, 6):
        plt.gca().set_xticks([-0.1, 0, 0.2, 0.4, 0.6, 0.8, 1])
        plt.xlabel("Silhouette Coefficient")
    else:
        plt.tick_params(labelbottom=False)

    plt.axvline(x=silhouette_scores[k - 2], color="red", linestyle="--")
    plt.title("$k={}$".format(k), fontsize=16)

plt.show()
```



The dashed line in the silhouette diagram represents the silhouette score for different values of k that we are considering. Based on the diagram, choosing k=2 is not ideal because there is a significant difference in the sizes of the clusters. On the other hand, for k=3, the silhouette coefficients are on average maximized compared to other values of k. The resulting clusters are not perfectly balanced in terms of size, with one cluster being slightly smaller than the others. In comparison, k=4 produces more balanced clusters in terms of size, but the silhouette coefficient is negative, which is a concern.

Given the negative silhouette coefficient for k=4, we believe that k=3 would be a better choice for us. This value of k produces well-defined clusters with mostly positive and uniform silhouette widths, and the resulting clusters are reasonably balanced in terms of size.

We had list clustering methods: K-means clustering, spectral clustering,hierarchical clustering, DBSCAN, Gaussian Mixture Model(GMM), and K-medoids in the beginning.

Among those method, we choose DBSCAN as alternative method to try.

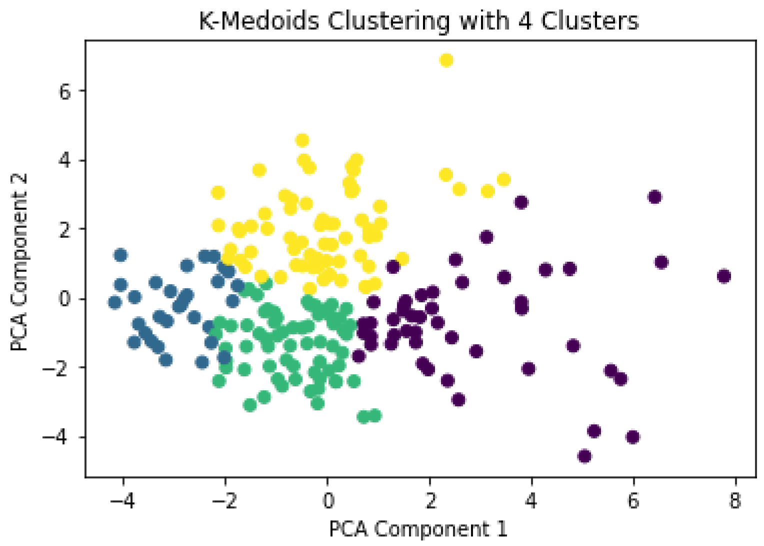
- DBSCAN assume that clusters are dense region seperated by regions of low density. Notice that we may not satisfy this assumption.
- DBSCAN has the advantage of not needing to specify the number of clusters in advance, and it can handle clusters with different shapes and sizes.

We attempted to use DBSCAN for clustering, but it failed due to the violation of the density assumption. DBSCAN requires high-density clusters separated by low-density regions, and our data violates this assumption, which is detrimental.

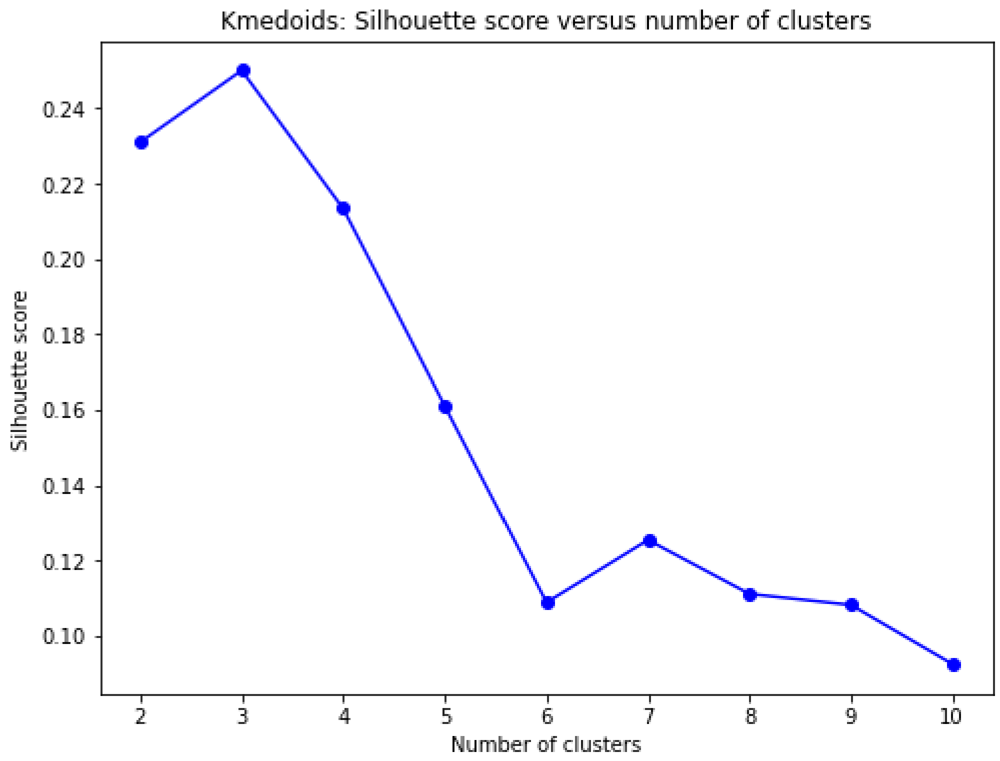
Because the density seperation assumption of DBSCAN is not satisfied, we consider to use K-mediods, because it does not rely on the density separation.

- K-mediod different from k-means by choosing actual points as centers and able to handle non-spherical clusters.
- K-mediod is slower, but our data points are not very large. It is worth a try.
- K-mediod need us to specify the number of clusters.

In [58]:
```python
from sklearn_extra.cluster import KMedoids #can run in local, but need to install some packages for colab
from sklearn.metrics import silhouette_score
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt

kmedoids = KMedoids(n_clusters=4, random_state=42)
kmedoids.fit(X_train)

pca = PCA(n_components=2, random_state=42)
X_pca = pca.fit_transform(X_train)

cluster_labels = kmedoids.labels_
silhouette_avg = silhouette_score(X_train, cluster_labels)

plt.scatter(X_pca[:,0], X_pca[:,1], c=cluster_labels)
plt.title('K-Medoids Clustering with 4 Clusters')
plt.xlabel('PCA Component 1')
plt.ylabel('PCA Component 2')
plt.show()
```



In [59]:
```
k_range = range(2, 11)

silhouette_scores = np.zeros(len(k_range))
cluster_labels = np.zeros((len(k_range), X_train.shape[0]))

for i, k in enumerate(k_range):
    km = KMedoids(n_clusters=k).fit(X_train)
    cluster_labels[i] = km.labels_
    silhouette_scores[i] = silhouette_score(X_train, km.labels_)

best_k_index = np.argmax(silhouette_scores)

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

plt.figure(figsize=(8, 6))
plt.plot(k_range, silhouette_scores, 'bo-')
plt.title('Kmedoids: Silhouette score versus number of clusters')
plt.xlabel('Number of clusters')
plt.ylabel('Silhouette score')
plt.show()
```



Conclusion and Discussion: Clustering

Conclusion

Based on the silhouette diagram for the K-Medoids clustering method, it appears that k=3 is the best choice as it has a high silhouette score of 0.25. However, when we compare this to the results of K-Means clustering with k=3, we can achieve a higher silhouette score of 0.27. Therefore, we conclude that K-Means is a better clustering method for our breast cancer data.

Since we have four cancer stages and the scatter plot clearly shows distinct clusters for each stage, we are confident in our decision to choose k=4 as the optimal number of clusters. Therefore, based on our breast cancer data, we have the possiblity to classify them into four stages.

Discussion

To be critical, our clustering analysis can have several potential errors, and clustering method have some nature weaknesses.

At first, although we tried three different methods to determine the optimal number of clusters, four clusters may not be the best. The plot of inertia and the silhouette diagram helped us decide, but they were not strong enough to indicate well-separated and distinct clusters. Part of this moderate distinction may come from the nature of our dataset. Since all four stages were initially classified as malignant, it's reasonable to view the four clusters as one large cluster with moderate separations in between.

Second, during the clustering process, we considered seven different clustering methods, but there may be other algorithms more suitable for clustering this breast cancer dataset. Inherently, K-means is a linear clustering method. Although we used DBSCAN as our representative of nonlinear clustering methods, there may be other non-linear clustering methods worth trying.

Third, when we use PCA dimension reduction and visualized the graph in 3D and 2D, it also has the risk of distorting the shape of our data and losing important information. Therefore, we need to pay attention to this and possibly explore other dimension reduction methods that may be more suitable.

Fourth, clustering methods are sensitive to small variations in instances, such as outliers, but we did not examine any possible outliers in our data.

Toward the discussion about the weakness of clustering, one is that clustering method cannot be used to make predictions with new data because they are only used to group similar data points based on their existing features. Therefore, the four clusters found in our analysis can only apply to the instances in our breast cancer dataset. To make predictions, we should try supervised methods instead.

Another weakness of clustering methods is that clustering methods become more computationally expensive and less efficient as the size of the dataset grows larger. At our scale of 200 instances, it is still efficient to use clustering methods.

Bibliography

Ujvari, B., Roche, B., & Thomas, F. (2017). Ecology and Evolution of Cancer. Academic Press.

Aurélien Géron. (2019). Hands-on machine learning with Scikit-Learn and TensorFlow concepts, tools, and techniques to build intelligent systems. O'Reilly Media, Inc.

James, G., Witten, D., Hastie, T., & Tibshirani, R. (2022). An introduction to statistical learning: With applications in R. Springer.

Appendix

Contribution: In this group project, everyone participated in selecting the dataset and discussed possible methods that we could use. Tianqi and Wenjie contributed to the lab and introduction parts, while Mingcong and Zai contributed to the classification analysis and its conclusion and discussion. Bihan contributed to clustering and its conclusion and discussion. Everyone actively attended the project and shared their thoughts.

Dataset Source: UCI Machine Learning Repository: Breast Cancer Wisconsin (Diagnostic) Data Set

Software: We used jupyter notebook for classification and clustering and R for lab part.