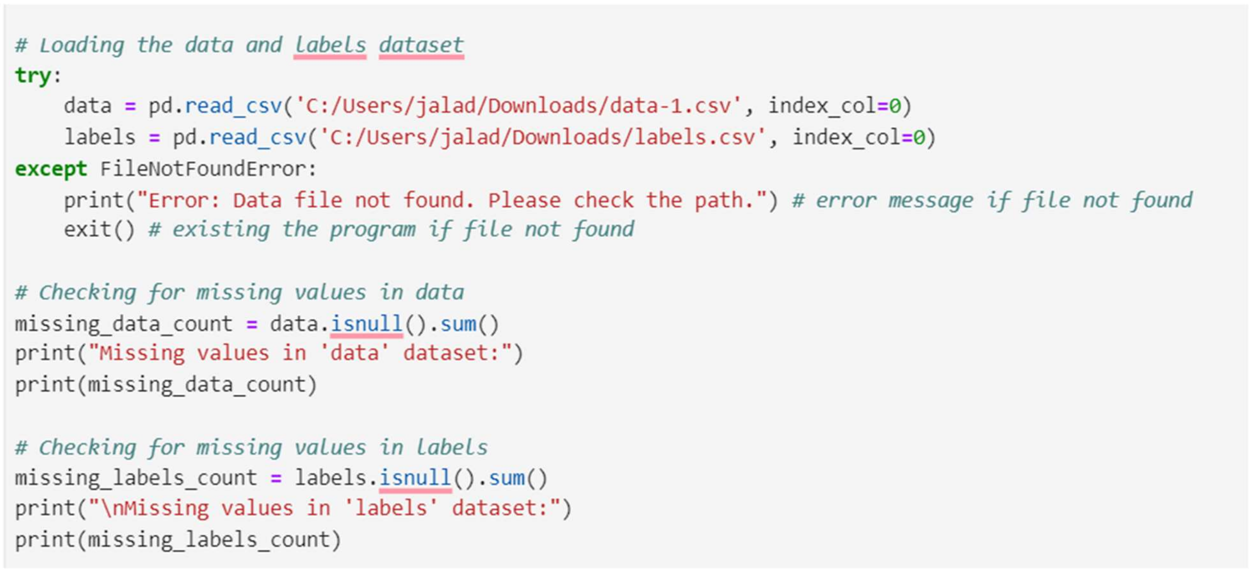
Unsupervised Learning (Clustering)
gene expression cancer RNA-Seq

Abstract and Introduction:

This study looks at how genes behave in different types of cancer. Used RNA-Seq (HiSeq) PANCAN data set, which has information about 801 patients and 20,531 different gene behaviors. I used a method called K-means clustering to put these patients into groups based on their gene behavior. Then, to find the best way to group them using something called silhouette analysis. This helps us understand cancer better by seeing how genes act in different types of tumors like BRCA, KIRC, COAD, LUAD, and PRAD. Subsequently , evaluated the clustering performance, visualized results through PCA, and assessed accuracy without relying on label information. Additionally, calculated the Adjusted Rand Index (ARI) to see how well our groups match up with the actual types of cancer.

Methodology:

Loading the two datasets data-1 and labels. After loading the datasets preprocessing steps is performed . Checking the missing values in the dataset and found that there are no missing values in the dataset.



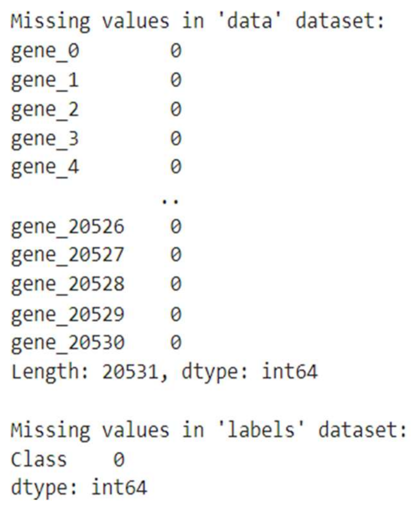
```
# Loading the data and labels dataset
try:
    data = pd.read_csv('C:/Users/jalad/Downloads/data-1.csv', index_col=0)
    labels = pd.read_csv('C:/Users/jalad/Downloads/labels.csv', index_col=0)
except FileNotFoundError:
    print("Error: Data file not found. Please check the path.") # error message if file not found
    exit() # existing the program if file not found

# Checking for missing values in data
missing_data_count = data.isnull().sum()
print("Missing values in 'data' dataset:")
print(missing_data_count)

# Checking for missing values in labels
missing_labels_count = labels.isnull().sum()
print("\nMissing values in 'labels' dataset:")
print(missing_labels_count)
```

Fig: 1 Code for loading dataset and checking missing values.

Result:



```
Missing values in 'data' dataset:
gene_0        0
gene_1        0
gene_2        0
gene_3        0
gene_4        0
             ..
gene_20526    0
gene_20527    0
gene_20528    0
gene_20529    0
gene_20530    0
Length: 20531, dtype: int64

Missing values in 'labels' dataset:
Class    0
dtype: int64
```

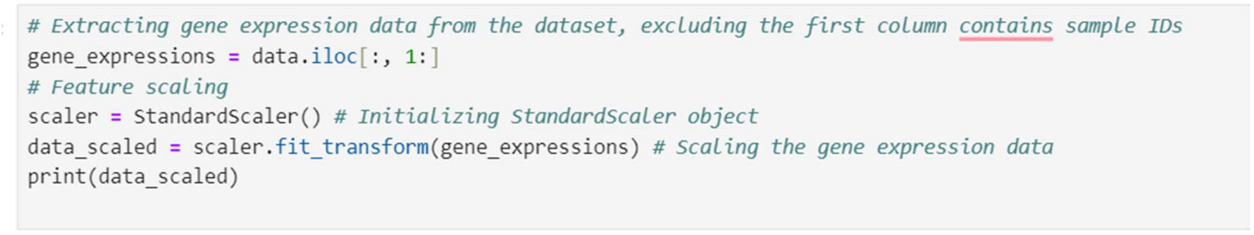
Fig :2 result of missing values in the data set

There are no missing values in the dataset.

Gene Expression Data Extraction:

The gene expression data is extracted from the dataset data, excluding the first column which typically contains sample IDs.

Feature Scaling: After extracting the gene expression data , it is standardized using the StandardScaler object. StandardScaler changes the data so that its average value becomes 0 and its spread becomes 1. The reason to perform scaling is to standardize the range of features or variables in our dataset.



```
# Extracting gene expression data from the dataset, excluding the first column contains sample IDs
gene_expressions = data.iloc[:, 1:]
# Feature scaling
scaler = StandardScaler() # Initializing StandardScaler object
data_scaled = scaler.fit_transform(gene_expressions) # Scaling the gene expression data
print(data_scaled)
```

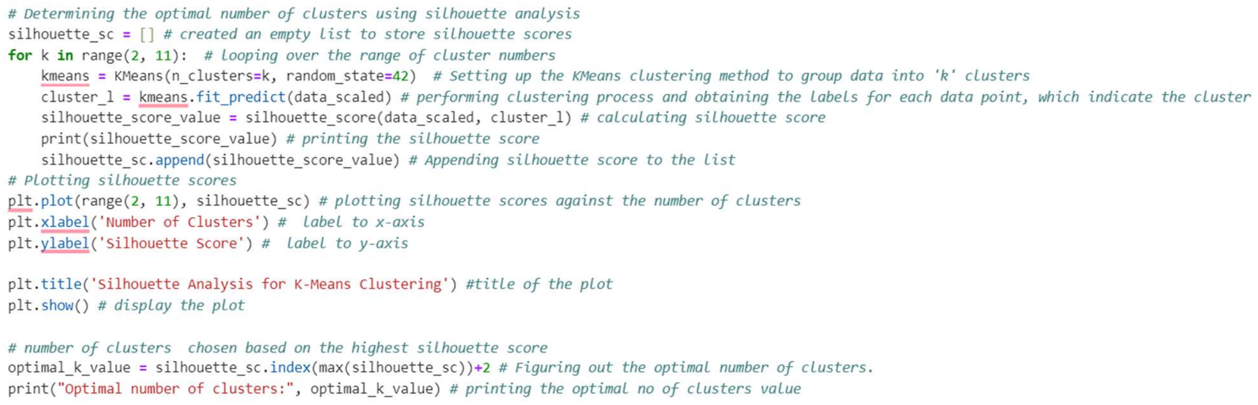
Fig 3: Code for the Feature Scaling

Silhouette analysis:

The method used in the code is silhouette analysis, this is used for determining the optimal number of clusters in K-means clustering. The silhouette score tells us how well each data point fits into its cluster compared to other clusters. The score ranges from -1 to 1, if the grade is high close to 1 means that the instance is well matched with to its own cluster and poorly matched to neighboring clusters.

Kmeans:

Kmeans refers to an instance of the KMeans clustering algorithm . This algorithm is used to partition the data into a specified number of clusters ranging from 2 to 10.



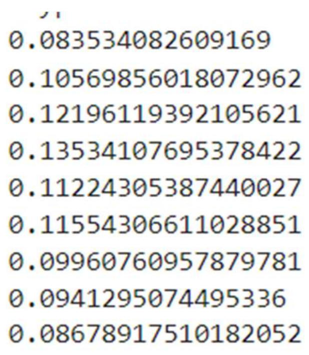
```
# Determining the optimal number of clusters using silhouette analysis
silhouette_sc = [] # created an empty list to store silhouette scores
for k in range(2, 11):  # looping over the range of cluster numbers
    kmeans = KMeans(n_clusters=k, random_state=42)  # Setting up the KMeans clustering method to group data into 'k' clusters
    cluster_l = kmeans.fit_predict(data_scaled) # performing clustering process and obtaining the labels for each data point, which indicate the cluster
    silhouette_score_value = silhouette_score(data_scaled, cluster_l) # calculating silhouette score
    print(silhouette_score_value) # printing the silhouette score
    silhouette_sc.append(silhouette_score_value) # Appending silhouette score to the list
# Plotting silhouette scores
plt.plot(range(2, 11), silhouette_sc) # plotting silhouette scores against the number of clusters
plt.xlabel('Number of Clusters') #  label to x-axis
plt.ylabel('Silhouette Score') #  label to y-axis

plt.title('Silhouette Analysis for K-Means Clustering') #title of the plot
plt.show() # display the plot

# number of clusters  chosen based on the highest silhouette score
optimal_k_value = silhouette_sc.index(max(silhouette_sc))+2 # Figuring out the optimal number of clusters.
print("Optimal number of clusters:", optimal_k_value) # printing the optimal no of clusters value
```

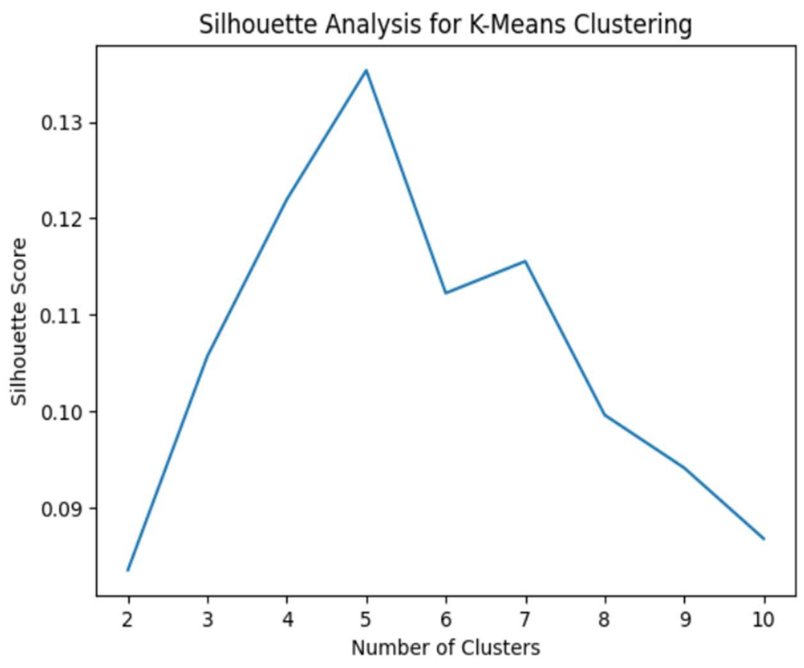
Fig 4: Code for determining the optimal value.

In the above code optimal_k_value = silhouette_sc.index(max(silhouette_sc))+2 this line of code finds the index of the maximum silhouette scorein the silhouette_sc , which contain silhouette scores for different number of clusters. The max() function returns the maximum value form the list and index returns the index of the value. Since range of clusters starts from 2 .So , I added 2 to the index to get the optimal number of clusters.



```
0.083534082609169
0.10569856018072962
0.12196119392105621
0.13534107695378422
0.11224305387440027
0.11554306611028851
0.09960760957879781
0.0941295074495336
0.08678917510182052
```

Fig 5: Silhouette scores

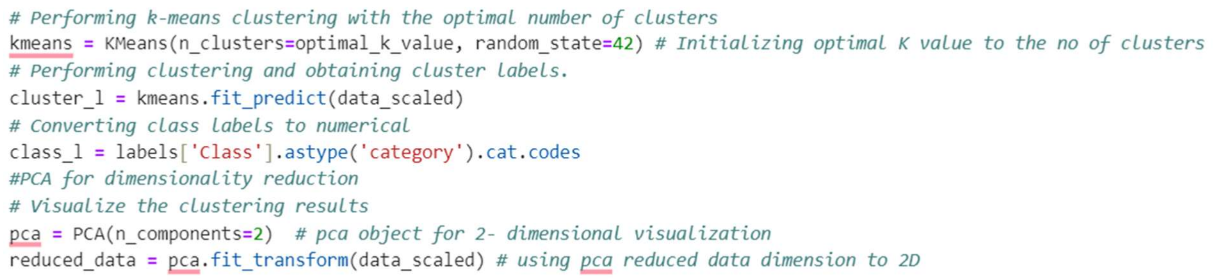


```
Optimal number of clusters: 5
```

Fig 6: Optimal number of clusters

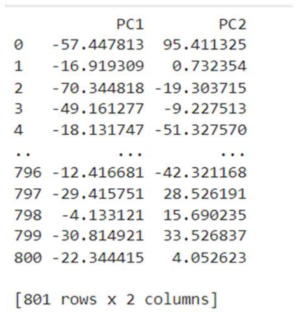
KMeans Clustering: KMeans algorithm is used for clustering the data into a specified number of clusters. By minimizing the within cluster sum of squares it partitions the data into clusters.

Principal Component Analysis (PCA): PCA is a dimensionality reduction technique which is used to visualize high-dimensional data in a lower-dimensional space. Here in the code below used 2 components to reduce dimensionality of the scaled gene expression data.



```
# Performing k-means clustering with the optimal number of clusters
kmeans = KMeans(n_clusters=optimal_k_value, random_state=42) # Initializing optimal K value to the no of clusters
# Performing clustering and obtaining cluster labels.
cluster_l = kmeans.fit_predict(data_scaled)
# Converting class labels to numerical
class_l = labels['Class'].astype('category').cat.codes
#PCA for dimensionality reduction
# Visualize the clustering results
pca = PCA(n_components=2)  # pca object for 2- dimensional visualization
reduced_data = pca.fit_transform(data_scaled) # using pca reduced data dimension to 2D
```

Fig 7 : Code for PCA and KMeans

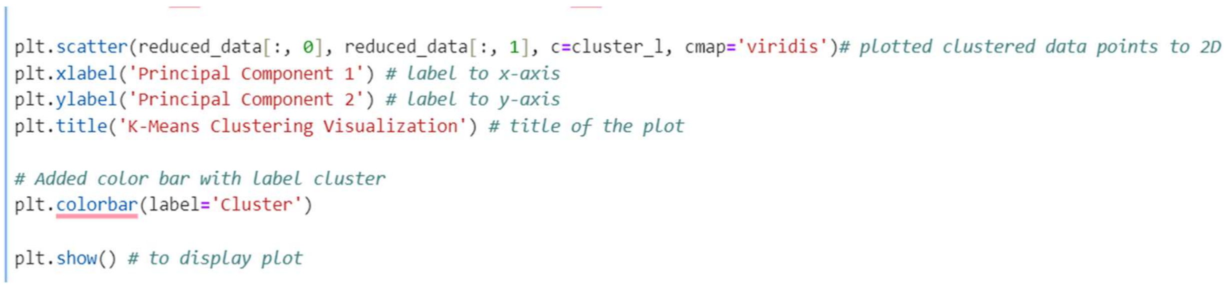


```
          PC1        PC2
0   -57.447813  95.411325
1   -16.919309   0.732354
2   -70.344818 -19.303715
3   -49.161277  -9.227513
4   -18.131747 -51.327570
..         ...        ...
796 -12.416681 -42.321168
797 -29.415751  28.526191
798  -4.133121  15.690235
799 -30.814921  33.526837
800 -22.344415   4.052623

[801 rows x 2 columns]
```

Fig 8: reduced the dimensionality of your data to two dimensions

Scatterplot is used to visualize :



```
plt.scatter(reduced_data[:, 0], reduced_data[:, 1], c=cluster_l, cmap='viridis')# plotted clustered data points to 2D
plt.xlabel('Principal Component 1') # label to x-axis
plt.ylabel('Principal Component 2') # label to y-axis
plt.title('K-Means Clustering Visualization') # title of the plot

# Added color bar with label cluster
plt.colorbar(label='Cluster')

plt.show() # to display plot
```

Fig 9: Code for scatterplot

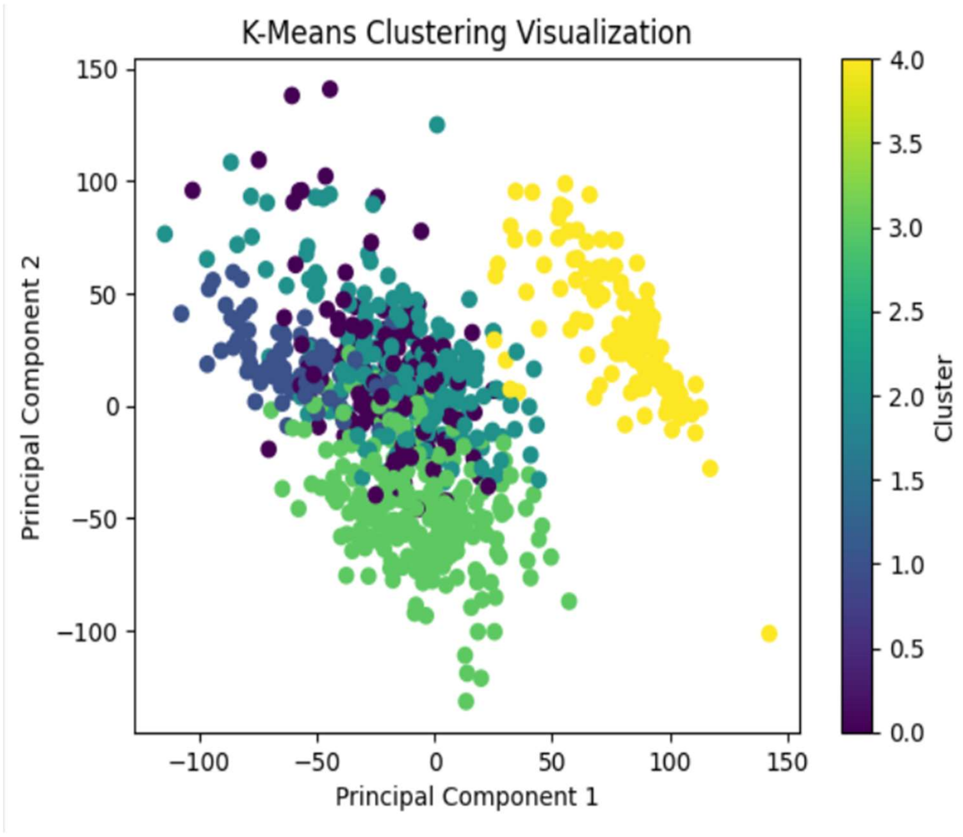
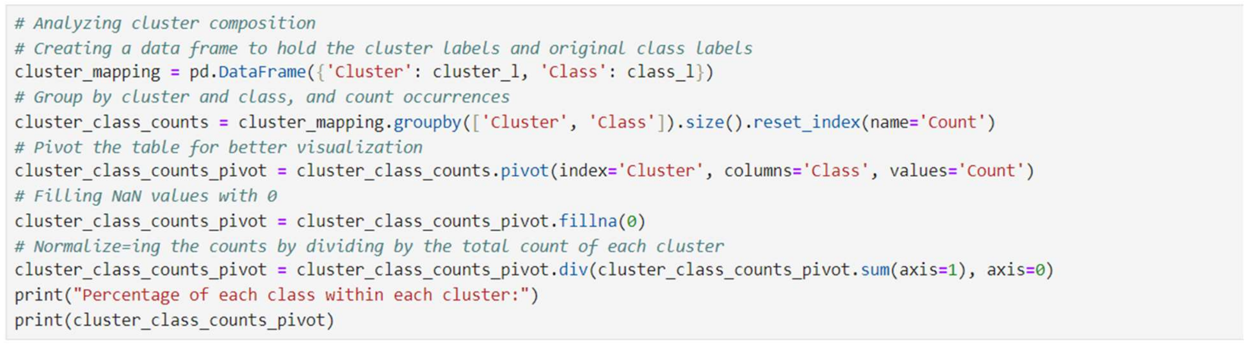


Fig 10 :KMeans clustering visualization

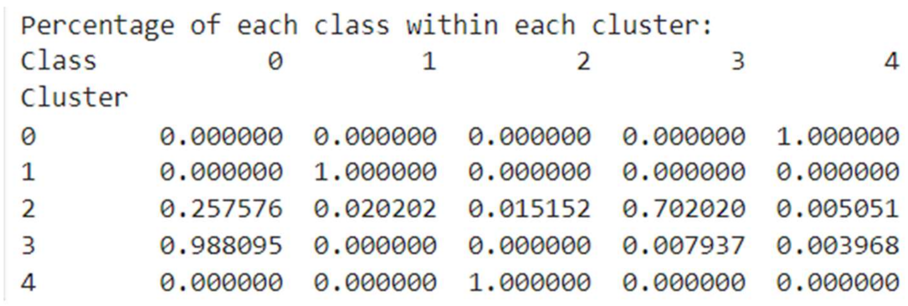
From the above graph there are five types of tumors .



```
# Analyzing cluster composition
# Creating a data frame to hold the cluster labels and original class labels
cluster_mapping = pd.DataFrame({'Cluster': cluster_l, 'Class': class_l})
# Group by cluster and class, and count occurrences
cluster_class_counts = cluster_mapping.groupby(['Cluster', 'Class']).size().reset_index(name='Count')
# Pivot the table for better visualization
cluster_class_counts_pivot = cluster_class_counts.pivot(index='Cluster', columns='Class', values='Count')
# Filling NaN values with 0
cluster_class_counts_pivot = cluster_class_counts_pivot.fillna(0)
# Normalize=ing the counts by dividing by the total count of each cluster
cluster_class_counts_pivot = cluster_class_counts_pivot.div(cluster_class_counts_pivot.sum(axis=1), axis=0)
print("Percentage of each class within each cluster:")
print(cluster_class_counts_pivot)
```

Fig :11 cluster composition

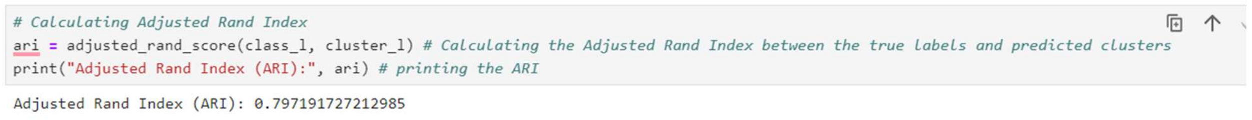
In the above code calculated percentage of each class within each cluster.



```
Percentage of each class within each cluster:
Class            0         1         2         3         4
Cluster
0         0.000000  0.000000  0.000000  0.000000  1.000000
1         0.000000  1.000000  0.000000  0.000000  0.000000
2         0.257576  0.020202  0.015152  0.702020  0.005051
3         0.988095  0.000000  0.000000  0.007937  0.003968
4         0.000000  0.000000  1.000000  0.000000  0.000000
```

Accuracy:

Adjusted Rand Index (ARI): This is used to find out how similar two different groupings of data are. The score between -1 to 1. Where 1 means grouping the exactly the same, 0 means they are about to good as random and -1 means they are completely different.



```
# Calculating Adjusted Rand Index
ari = adjusted_rand_score(class_l, cluster_l) # Calculating the Adjusted Rand Index between the true labels and predicted clusters
print("Adjusted Rand Index (ARI):", ari) # printing the ARI
```

```
Adjusted Rand Index (ARI): 0.797191727212985
```

Fig 12: Code and accuracy

The accuracy is 79%.

Conclusion:

This shows that using KMeans clustering on gene expression data. I found that having 5 clusters. In the clusters on the graph using PCA can see different groups of tumors . Achieved 79% showing this method is good at capturing the main patterns in the gene data.