Project 1

Jacob Hofer, Flint Morgan, Joseph Winjum, Keith Filler

Part 1: Thinking About the Data

1.1 Interest in the Data Set

As Computer Science, Electrical, and Computer Engineering graduate students, this data set studying transistor scaling in CPUs and GPUs directly relates to each of our academic backgrounds. We all possess a good foundation of knowledge to be able to obtain relevant information from this study.

1.2 Number of Attributes

We have a total of ten attributes in this data set. If we consider “release date” to be a numerical attribute of the data set, then we have 6 numerical attributes. We then have four categorical attributes.

1.3 Missing Values

Our data set does have missing values. Our approach is to simply remove all data instances with missing values. Our initial data set contains over 4,800 entries. After removing the rows with missing data, we are left with 3,422 data instances. This is still a substantial data set to complete our project and obtain meaningful data.

1.4 Attributes that Describe the Data

For this data set we expect the numerical attributes related to the physical structure of the CPU, GPU and underlying transistors to be the most descriptive. The knowledge to be gained from studying this data lies in the functionality of the various CPUs and GPUs. Some key findings from the original study of this data set include:

- Moore’s law still holds as of the time of the initial study in July 2020
- Dennard scaling is still valid.
- CPUs have higher frequencies, but GPUs are catching up.

These findings represent a few from the initial study performed on this data set by Sun, Yifan et al [1]. These findings, and others, were determined from studying the numerical attributes of the data set.

1.5 Expecting Clusters

We expect clusters to be present in the data because we are studying features of physical components that have similar traits. The data set is looking at slight changes in manufacture and performance for electronic components. While the distinction between clusters may not be entirely clear initially, there will be some distinct, numerical, dividing lines among these differences.

1.6 Importance of Clusters

Finding clusters may be helpful in studying these differences in the physical design of components and seeing the effect these features have on the performance of the CPUs and GPUs. While it may be unclear to see a recognizable pattern from simply looking at plotted data values, clustering may offer some further insight into the connection between physical attributes nd performance.

1.7 Expected Number of Clusters

Initially we would expect to see at least two clusters, if the data was clustered into characteristics for CPU and GPU. Perhaps we would see up to 4 clusters if these two distinctions were separated further, but the most likely expectation is for two clusters.

1.8 Expected Size of Clusters

The initial data set has approximately the same amount of data entries for CPU and GPU, so we anticipate similar size clusters. Depending how many null values we remove from each attribute this result may vary.

Loading data

Below is the code to load our data from the CSV into a Pandas DataFrame

We are dropping the columns including semantic data, such as the product name, since we do not want to attempt to cluster based on those columns.

We also drop the GFLOPS columns, as those are specific to GPUs, and we want to include both GPUs and CPUs.

Lastly, we drop any rows that have missing data.

We can't fill the empty values with the mean for the column because as the die size, transistors and frequency will increase quickly with year, and if we choose to put the mean it could miss represent the data.

In [ ]:

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

df = pd.read_csv("chip_dataset.csv")

Type = df['Type']
# Drop semantic data
df.pop("Unnamed: 0");df.pop("Type");df.pop("Foundry");df.pop("Vendor");df.pop("Product");df.po
# GFLOPS are specific to GPUs, so we exclude them here so we can also look at CPUs
df.pop("FP16 GFLOPS");df.pop("FP32 GFLOPS");df.pop("FP64 GFLOPS")
df = df.dropna()
display(df)
```

	Process Size (nm)	TDP (W)	Die Size (mm^2)	Transistors (million)	Freq (MHz)
0	65.0	45.0	77.0	122.0	2200.0
1	14.0	35.0	192.0	4800.0	3200.0
3	22.0	80.0	160.0	1400.0	1800.0
4	45.0	125.0	258.0	758.0	3700.0
5	22.0	95.0	160.0	1400.0	2400.0
...	...	...	...	...	...
4844	40.0	150.0	334.0	2154.0	700.0
4845	40.0	20.0	80.0	10.0	416.0
4846	28.0	21.0	68.0	302.0	550.0
4849	40.0	75.0	332.0	1950.0	450.0
4851	40.0	23.0	100.0	486.0	500.0

3422 rows × 5 columns

Part 2

2.1 What is the multivariate mean of the numerical data matrix (where categorical data have been converted to numerical values)?

The mean of the data per variable is shown below. There are no categorical values kept and so they did not need to be converted.

In [ ]:

```
multivariate_mean = np.mean(df, axis=0)
print("Mean:");print(multivariate_mean)
```

```
Mean:
Process Size (nm)          53.048510
TDP (W)                    83.740795
Die Size (mm^2)           200.003799
Transistors (million)    2163.295441
Freq (MHz)               1507.964641
dtype: float64
```

2.2 What is the covariance matrix of the numerical data matrix (where categorical data have been converted to numerical values)?

The covariance matrix is as follows:

In [ ]:

```
np.set_printoptions(4) # makes it so that the numpy arrays only print 4 decimals
print(np.cov(df.T))
```

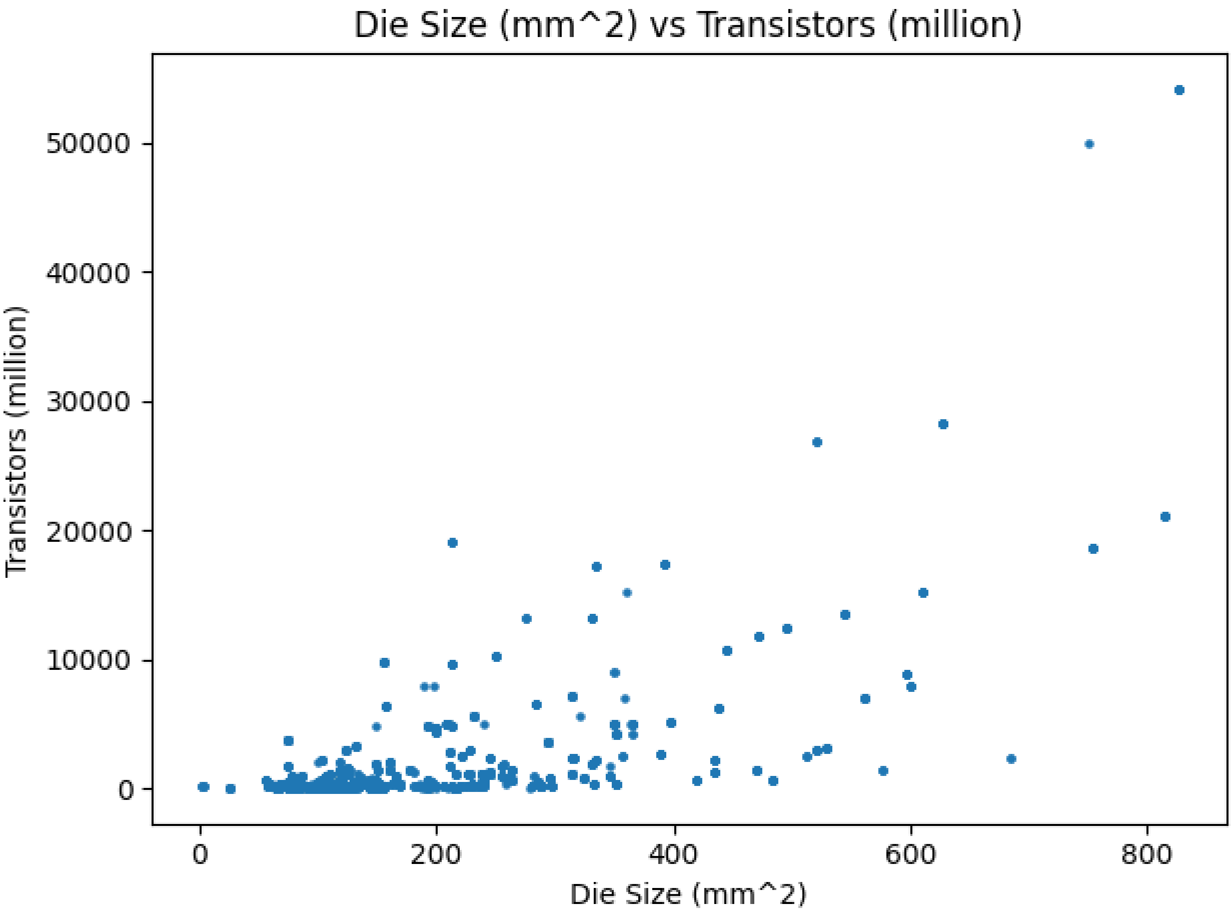
```
[[ 1.6930e+03 -6.3019e+02 -1.2010e+03 -7.0768e+04 -4.1532e+03]
 [-6.3019e+02  6.1910e+03  6.9883e+03  1.5981e+05  5.0407e+03]
 [-1.2010e+03  6.9883e+03  1.7108e+04  3.6212e+05 -7.3312e+03]
 [-7.0768e+04  1.5981e+05  3.6212e+05  1.8698e+07 -1.9320e+05]
 [-4.1532e+03  5.0407e+03 -7.3312e+03 -1.9320e+05  1.0257e+06]]
```

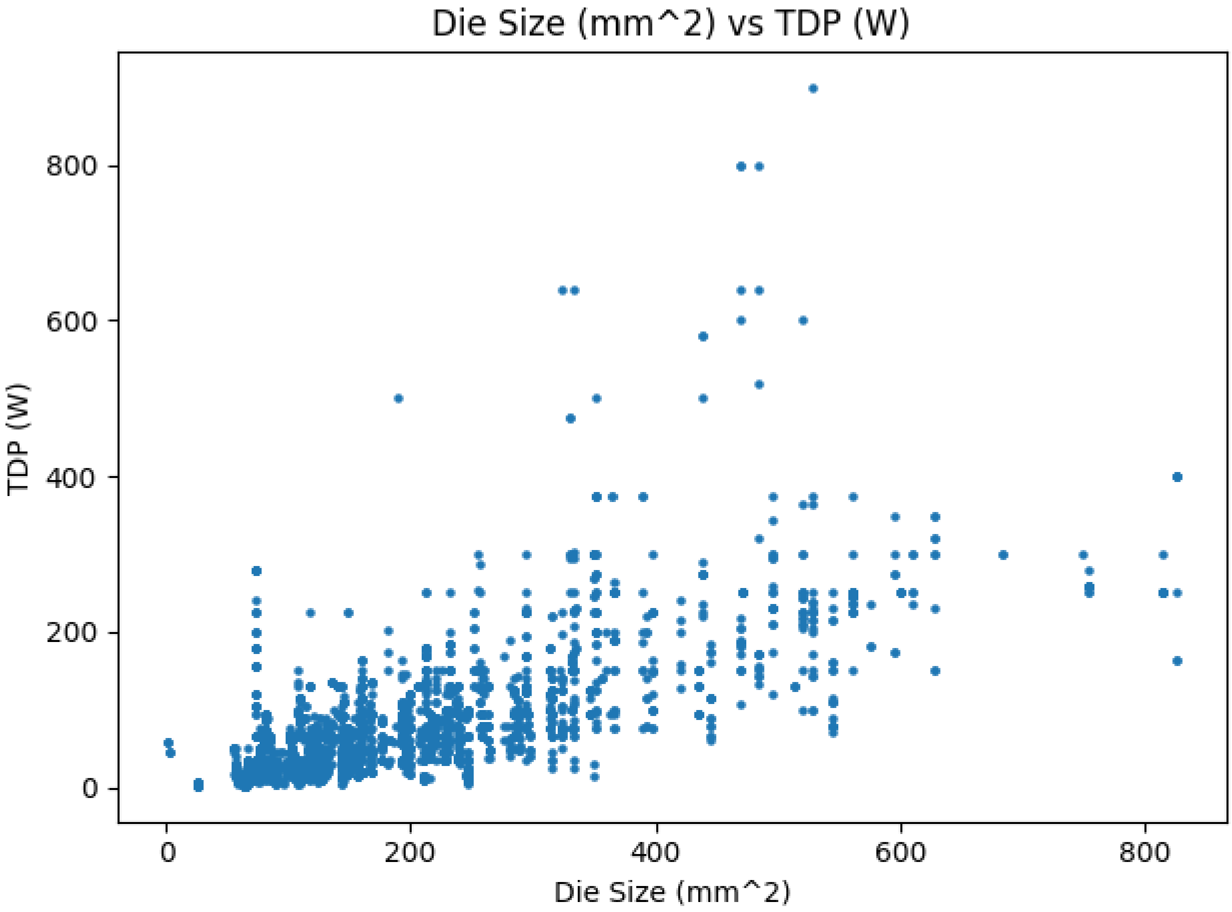
2.3 Choose 2 pairs of attributes that you think could be related. Create scatter plots of all 2 pairs and include these in your report, along with a description and analysis that summarizes why these pairs of attributes might be related, and how the scatter plots do or do not support this intuition.

We believe that die size and transistors will be related because as the die size increases the total number of transistors could increase given the same size of transistor. The scatter plot seems to show a positive corrolation, although it does not look that strong. It appears that die size and TDP (Thermal Design Power) have a stronger positive correlation. This makes sense because as the die size increases the total power is also likely to rise.

In [ ]:

```python
np_df = df.to_numpy() #conmverts pandas dataframe to numpy array
titles = df.columns #keeps titles from column names
i=2;j=3 #this code is re purposed from when I did not understand the question and ploted every
#plots
Title = titles[i]+" vs "+titles[j]
plt.figure()
plt.scatter(np_df[:,i],np_df[:,j],s=5)
plt.title(Title);plt.xlabel(titles[i]);plt.ylabel(titles[j])
plt.tight_layout()
plt.show()
plt.close()

i=2;j=1
Title = titles[i]+" vs "+titles[j]
plt.figure()
plt.scatter(np_df[:,i],np_df[:,j],s=5)
plt.title(Title);plt.xlabel(titles[i]);plt.ylabel(titles[j])
plt.tight_layout()
plt.show()
plt.close()
```


Die Size (mm^2) vs Transistors (million)
Transistors (million)
50000
40000
30000
20000
10000
0
0
200
400
600
800
Die Size (mm^2)

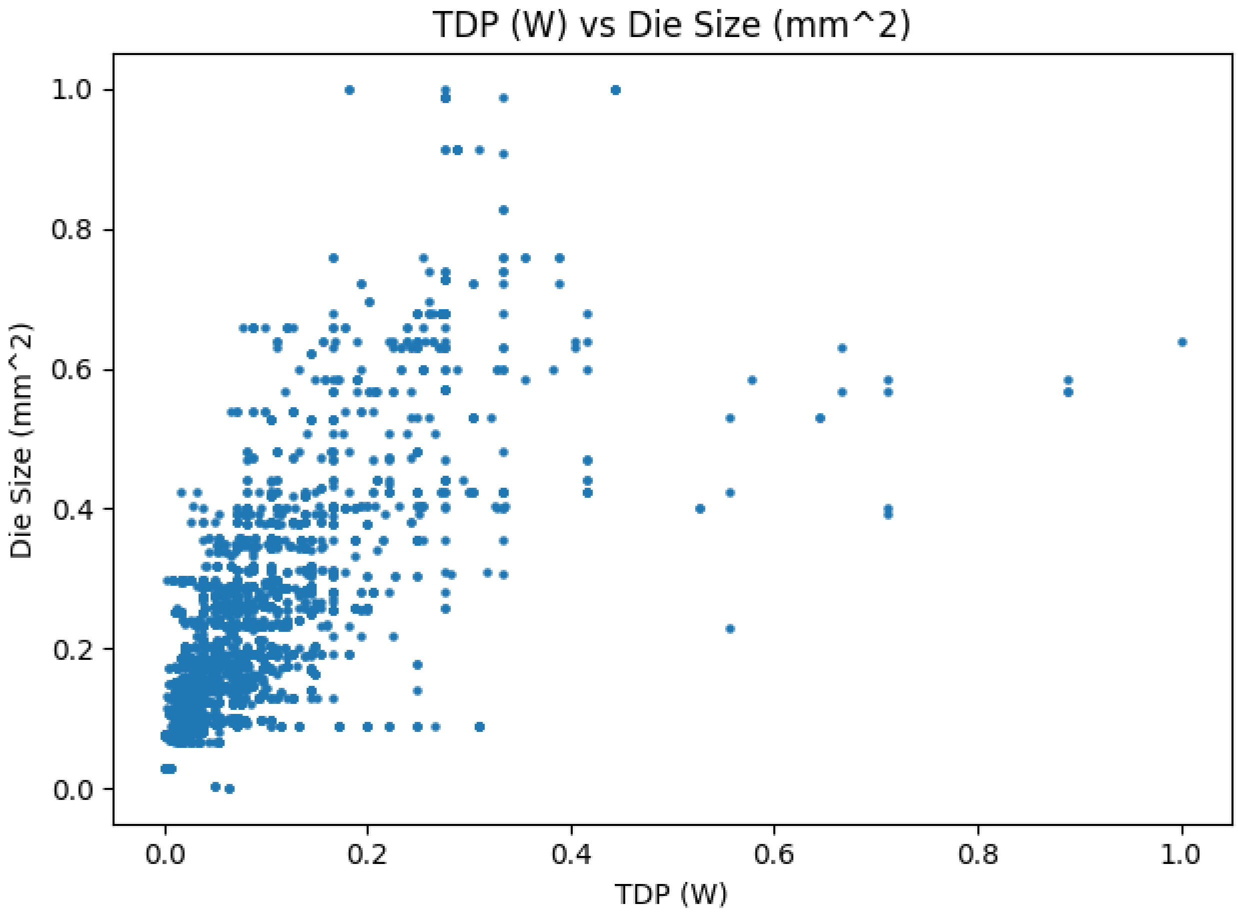

Die Size (mm^2) vs TDP (W)
TDP (W)
800
600
400
200
0
0
200
400
600
800
Die Size (mm^2)

2.4 Which range-normalized numerical attributes have the greatest sample covariance? What is their sample covariance? Create a scatter plot of these range-normalized attributes.

The max sample covariance is listed below.

```
In [ ]: from sklearn.preprocessing import MinMaxScaler
        import copy
        normalized_df = MinMaxScaler().fit_transform(np_df); # creates range-normalized atributes as wo
        cov_matrix = np.cov(normalized_df.T) # calculates covariance matrix
        cov_only = copy.copy(cov_matrix); np.fill_diagonal(cov_only,0) #makes a copy of the covariance
        max_cov = cov_matrix.flat[np.abs(cov_only).argmax()] #finds the max

        x,y = np.where(abs(cov_matrix) == max_cov)[0] #gets the index of the max
        #prints
        print("Covariance Matrix:")
        print(cov_matrix,"\n\n Max sample covariance:",max_cov, titles[x]+" and " +titles[y]+"\n")
        Title = titles[x]+" vs "+titles[y]
        plt.figure()
        plt.scatter(normalized_df[:,x],normalized_df[:,y],s=5)
        plt.title(Title);plt.xlabel(titles[x]);plt.ylabel(titles[y])
        plt.tight_layout()
        plt.show()
        plt.close()
```

```
Covariance Matrix:
[[ 0.0287 -0.0029 -0.006  -0.0054 -0.0037]
 [-0.0029  0.0077  0.0094  0.0033  0.0012]
 [-0.006   0.0094  0.0251  0.0081 -0.0019]
 [-0.0054  0.0033  0.0081  0.0064 -0.0008]
 [-0.0037  0.0012 -0.0019 -0.0008  0.0485]]

 Max sample covariance: 0.009422291959770054 TDP (W) and Die Size (mm^2)
```

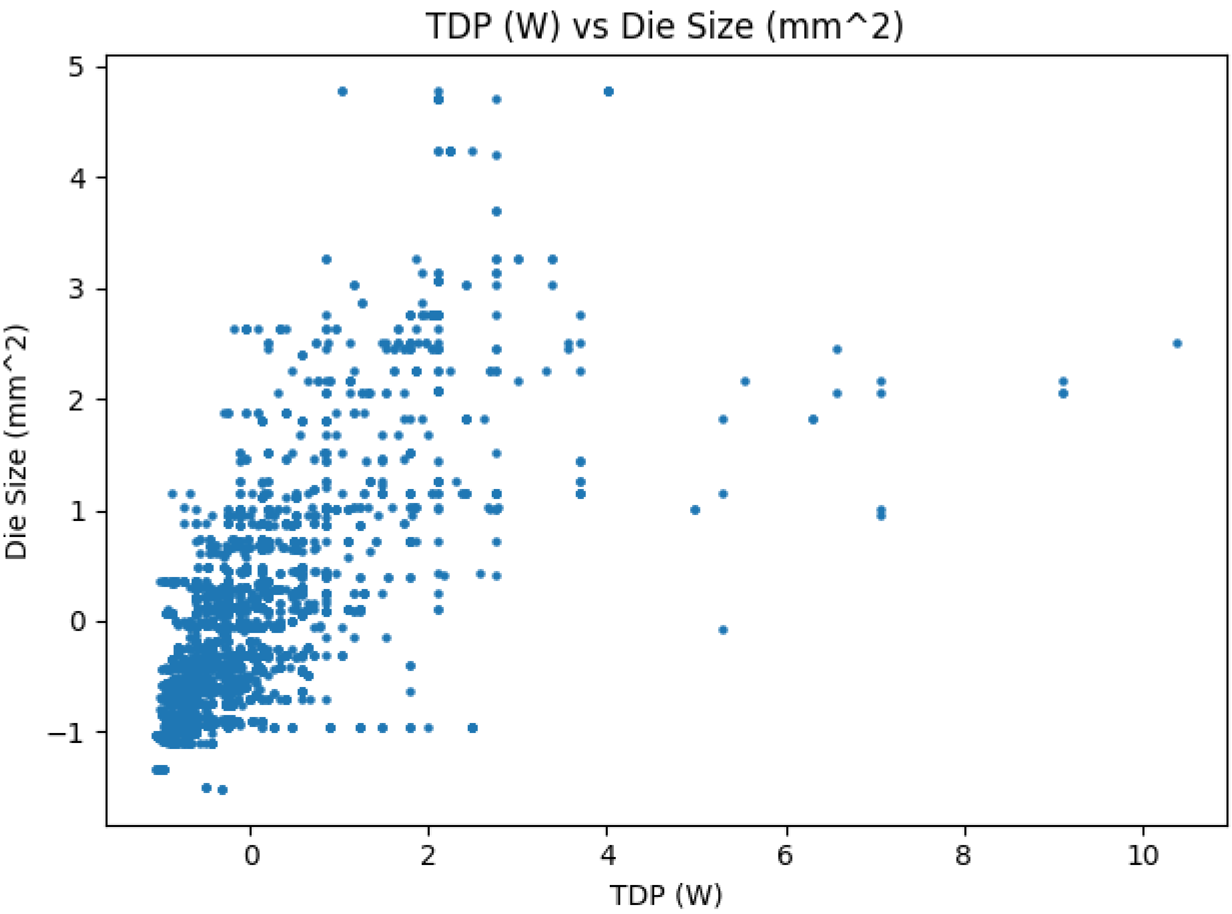


2.5 Which Z-score-normalized numerical attributes have the greatest correlation? What is their correlation? Create a scatter plot of these Z-score-normalized attributes.

The greatest correlation is listed below.

```
In [ ]:
from scipy.stats import zscore
z_df = (df-df.mean())/df.std() #calculated Z-score normalized atributes
z_df = z_df.to_numpy()
corr = np.corrcoef(z_df.T) #calculates correlation matrix
print("Correlation Matrix\n"+str(corr))
np.fill_diagonal(corr,0) #sets diagonal to zero so it is not considered for greatest correlatio
max_corr = corr.flat[np.abs(corr).argmax()] #finds greatest correlation
#prints
print("\n Max sample correlation:",max_corr, titles[x]+" and " +titles[y]+"\n")
x,y = np.where(abs(corr) == max_corr)[0]

Title = titles[x]+" vs "+titles[y]
plt.figure()
plt.scatter(z_df[:,x],z_df[:,y],s=5)
plt.title(Title);plt.xlabel(titles[x]);plt.ylabel(titles[y])
plt.tight_layout()
plt.show()
plt.close()
```

```
Correlation Matrix
[[ 1.     -0.1947 -0.2232 -0.3977 -0.0997]
 [-0.1947  1.      0.679   0.4697  0.0633]
 [-0.2232  0.679   1.      0.6403 -0.0553]
 [-0.3977  0.4697  0.6403  1.     -0.0441]
 [-0.0997  0.0633 -0.0553 -0.0441  1.    ]]

 Max sample correlation: 0.679039414508252 TDP (W) and Die Size (mm^2)
```



2.6 How many pairs of features have correlation greater than or equal to 0.5?

```
In [ ]: print(int(len(corr[corr>=0.5])/2),"pairs of features have greater than or equal to 0.5 correlat
```

```
2 pairs of features have greater than or equal to 0.5 correlation
```

2.7 How many pairs of features have negative sample covariance?

```
In [ ]: print(int(len(cov_only[cov_only<0])/2),"pairs of features have negative sample covariance")
```

```
6 pairs of features have negative sample covariance
```

2.8 What is the total variance of the data?

```
In [ ]: print("The total variance of the data is", np.trace(cov_matrix))
```

```
The total variance of the data is 0.11630722871961785
```

2.9 What is the total variance of the data, restricted to the five features that have the greatest sample variance?

Since our data has exactly 5 numerical features considered, this is going to be equal to the total variance.

In [ ]:

```python
variances = np.diagonal(cov_matrix)

print("The total varience of the five features that have the greatest sample variance:", sum(s
```

```
The total varience of the five features that have the greatest sample variance: 0.1163072287196
1784
```

Part 3: Custom Functions for Clustering

3.1 K-Means Clustering Algorithm (`kmeans.py`)

Computing the Clusters

The code below computes clusters in the data using our custom K-Means Clustering algorithm.

We found that a good number of clusters is 3, as any higher K value tended to produce clusters containing a very low number of points.

Our ϵ value was also chosen to be 0.001, which dictates the threshold of centroid change that denotes the algorithm should terminate.

The code that performs K-Means Clustering is available in `kmeans.py` .

In [ ]:

```python
import kmeans

k = 3
eps = 0.001
# the kMeans function takes in the data frame, the number of clusters, and a convergence epsilo
(centroids, assignments) = kmeans.kMeans(df, k, eps);
for i, centroid in enumerate(centroids):
    print("Cluster", i, "Centroid:", centroid)
```

```
Cluster 0 Centroid: [  15.725   148.3154  295.2615 6779.2077 1531.8442]
Cluster 1 Centroid: [9.6557e+00 2.4639e+02 6.3667e+02 2.5439e+04 1.4016e+03]
Cluster 2 Centroid: [  60.8117   68.4291  173.1925  818.6586 1505.8779]
```

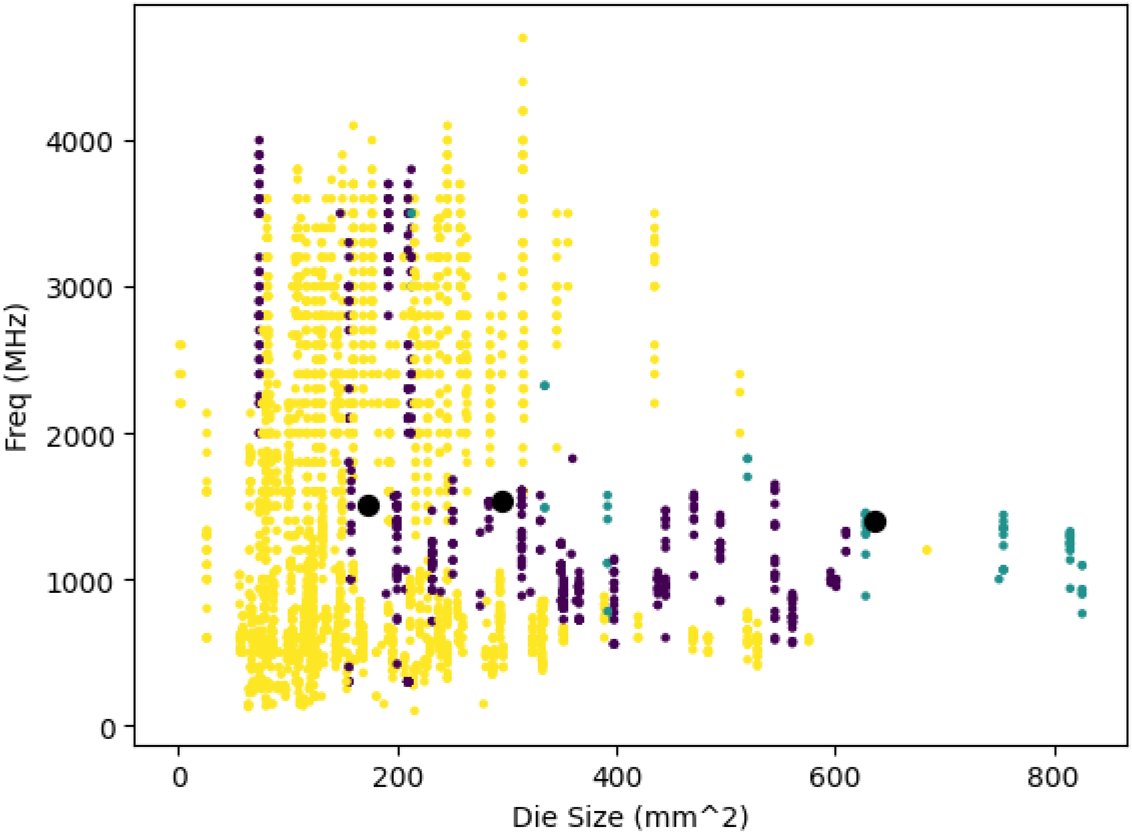
Plotting the Clusters

Below is an example plot of our clustering. The colors of each point correspond to their assigned cluster.

Cluster centroids are denoted by the larger black points.

Changing the `xAxis` and `yAxis` values to any integer between 0 and 4 will change which two features are compared.

```
In [ ]: import matplotlib.pyplot as plt

        xAxis = 2
        yAxis = 4

        plt.scatter(df.values[:,xAxis], df.values[:,yAxis], c=assignments, s=5)
        plt.scatter(centroids[:,xAxis], centroids[:,yAxis], c='black', s=50)
        plt.xlabel(df.columns[xAxis])
        plt.ylabel(df.columns[yAxis]);
```



3.2 DBSCAN Clustering Algorithm (`dbscan.py`)

Computing the Clusters

The code below computes clusters in the data using our custom DBSCAN Clustering algorithm.

Computing the neighborhoods is an $O(n^2)$ operation, where n is the number of observations in the data set. Therefore, our DBSCAN implementation may take some time to compute depending on the speed of the computer.

We determined a good value for `minPts` was 40 with an ϵ value of 0.65.

The code that performs DBSCAN Clustering is available in `dbscan.py` .

In [ ]:

```
import dbscan

dbScanAlg = dbscan.DBScan(40, 0.65)
# dbscan works a bit better with normalized data
normalizedDF = (df-df.mean())/df.std()
(assignments, corePts, borderPts, noisePts) = dbScanAlg.runAlgorithm(normalizedDF);
```

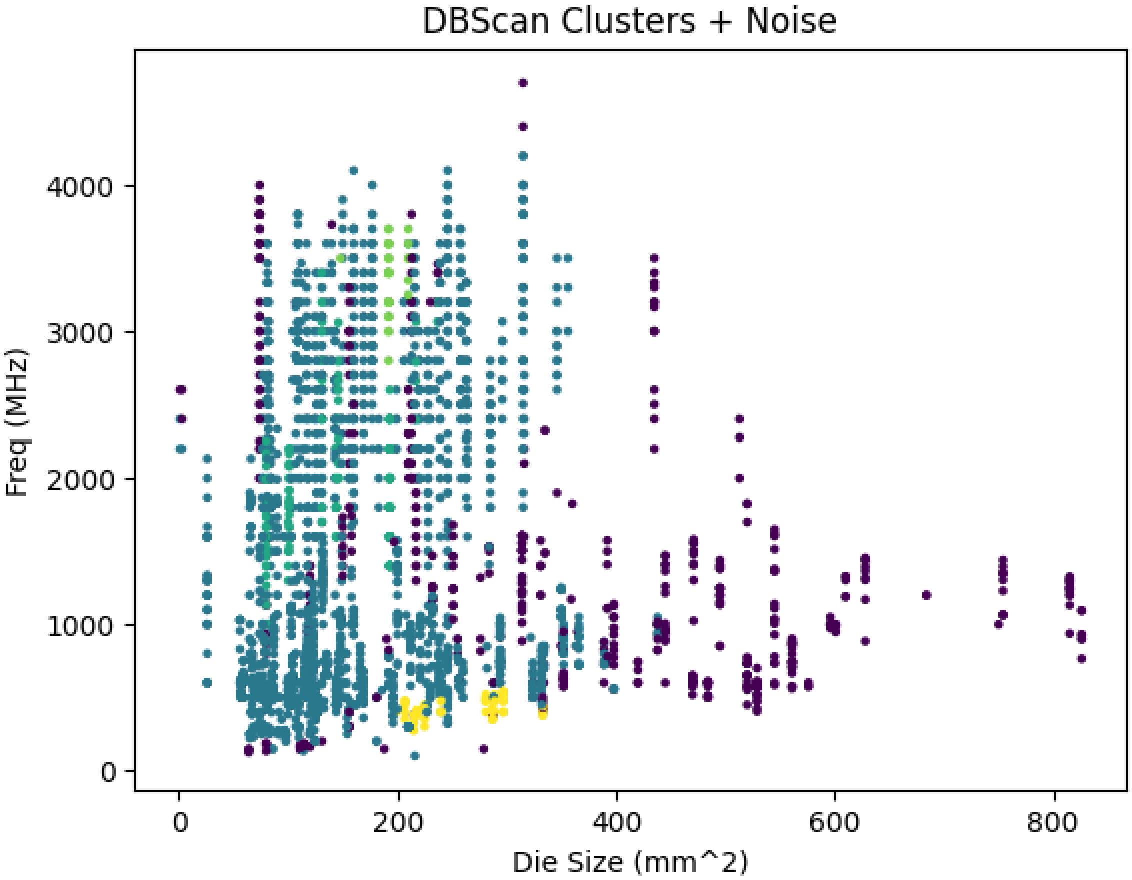
Plotting the Clusters

Below is a plot of our DBSCAN-based clustering. The colors of each point correspond to their assigned cluster, with dark purple points denoting noise points.

Again, changing the `xAxis` and `yAxis` values to any integer between 0 and 4 will change which two features are compared.

In [ ]:

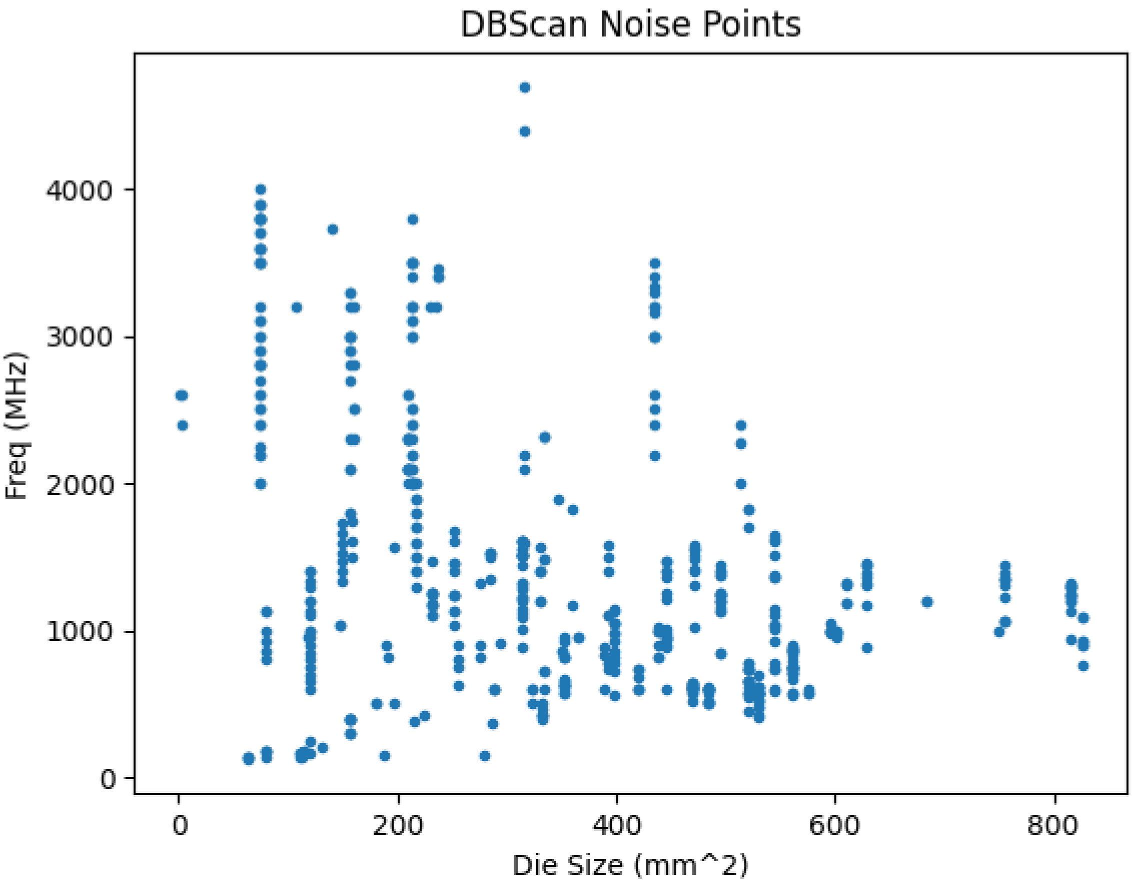
```
xAxis = 2
yAxis = 4

plt.scatter(df.values[:,xAxis], df.values[:,yAxis], c=assignments, s=5)
plt.title("DBScan Clusters + Noise")
plt.xlabel(df.columns[xAxis])
plt.ylabel(df.columns[yAxis]);
```



Plotting Noise Points

In order to better visualize noise points, the below plot contains only the points our DBSCAN implementation determined to be noise.

```
In [ ]: for i in noisePts:
            plt.plot(df.values[i,xAxis], df.values[i,yAxis], marker="o", markerfacecolor='#1f77b4', mar

        plt.title("DBScan Noise Points")
        plt.xlabel(df.columns[xAxis])
        plt.ylabel(df.columns[yAxis]);
```



Part 4: Analyzing the Data

4.1 PCA Graph in 2 Dimensions

```
In [ ]: from sklearn.decomposition import PCA
        from sklearn.preprocessing import StandardScaler
        from sklearn.preprocessing import MinMaxScaler

        scaler = StandardScaler(with_std=True,
                                with_mean=True)
        df_scaled = scaler.fit_transform(df)

        pcadf = PCA() #unspecified number of components
        pcadf2 = PCA(n_components=2) #2 components
        pcadf.fit(df_scaled)
```

```
pcadf2.fit(df_scaled)
pcadf2.mean_
```

Out[ ]:

```
array([ 3.3222e-17, -7.4750e-17,  8.3056e-17,  0.0000e+00,  0.0000e+00])
```

In [ ]:

```
scores = pcadf.transform(df_scaled)
scores2 = pcadf2.transform(df_scaled)
pcadf2.components_
```

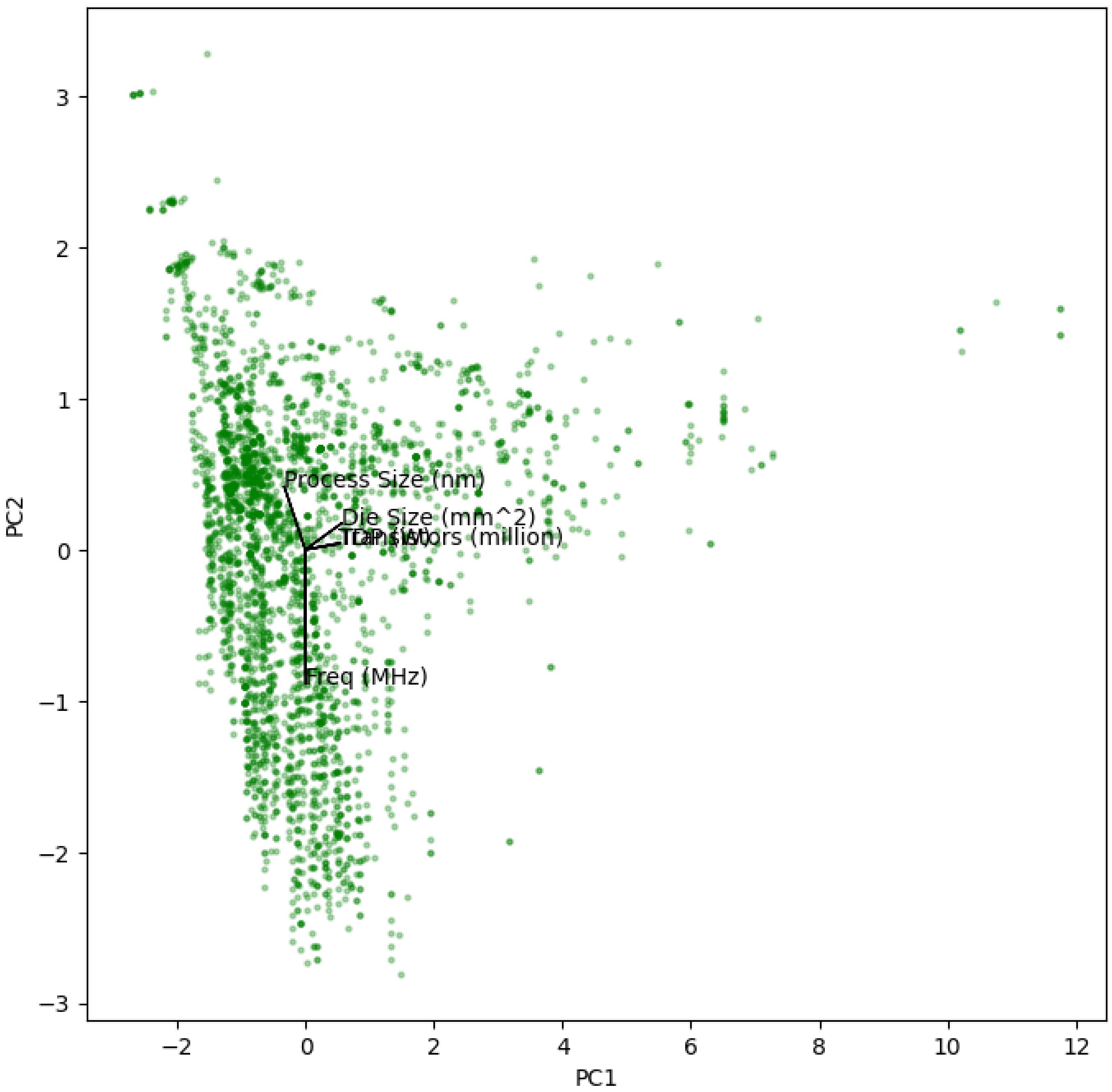
Out[ ]:

```
array([[-0.3281,  0.5205,  0.5702,  0.5444,  0.0074],
       [ 0.4199,  0.0384,  0.1755,  0.0446, -0.8885]])
```

It looks like there are about 2 clusters present after implementing sklearn's PCA implementation to linearly transform the data to two dimensions.

In [ ]:

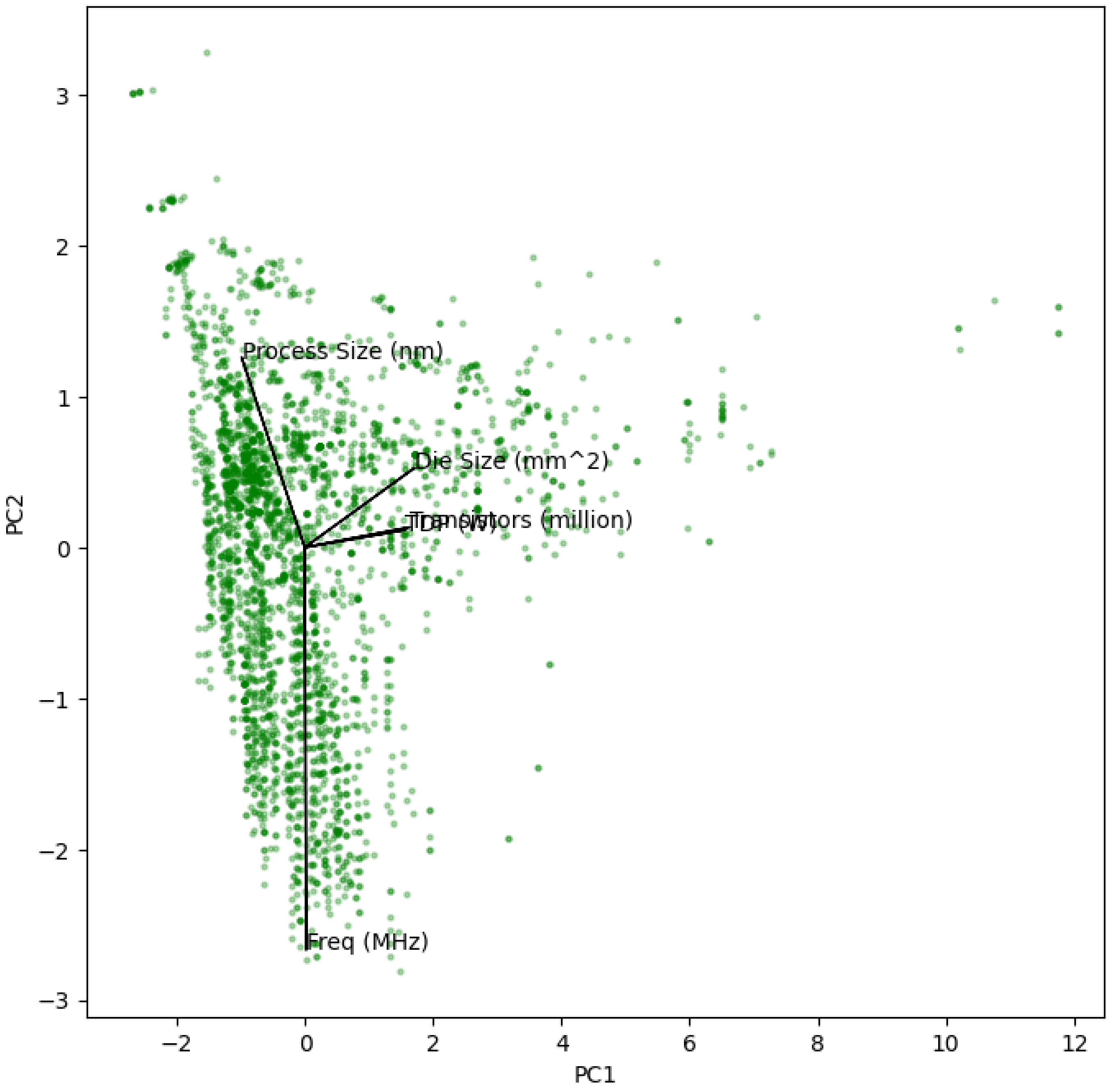
```
i, j = 0, 1 # which components
fig, ax = plt.subplots(1, 1, figsize=(8, 8))
ax.scatter(scores2[:,0], scores2[:,1], s=5, color='green', alpha=0.3)
ax.set_xlabel('PC%d' % (i+1))
ax.set_ylabel('PC%d' % (j+1))
for k in range(pcadf2.components_.shape[1]):
    ax.arrow(0, 0, pcadf2.components_[i,k], pcadf2.components_[j,k],)
    ax.text(pcadf2.components_[i,k],
    pcadf2.components_[j,k],
    df.columns[k],)
```



Same graph with scaled arrows for readability below.

In [ ]:

```
scale_arrow = s_ = 3
scores[:,1] *= -1
pcadf.components_[1] *= -1 # flip the y-axis
fig, ax = plt.subplots(1, 1, figsize=(8, 8))
ax.scatter(scores2[:,0], scores2[:,1], s=5, color='green', alpha=0.3)
ax.set_xlabel('PC%d' % (i+1))
ax.set_ylabel('PC%d' % (j+1))
for k in range(pcadf2.components_.shape[1]):
    ax.arrow(0, 0, s_*pcadf2.components_[i,k], s_*pcadf2.components_[
        j,k])
    ax.text(s_*pcadf2.components_[i,k],
            s_*pcadf2.components_[j,k],
            df.columns[k])
```



In [ ]:
```
scores2.std(0, ddof=1)
```

Out[ ]:
```
array([1.5361, 1.0285])
```

In [ ]:
```
pcadf2.explained_variance_
```

Out[ ]:
```
array([2.3595, 1.0578])
```

In [ ]:
```
pcadf2.explained_variance_ratio_
```

Out[ ]:
```
array([0.4718, 0.2115])
```

4.2 PCA With Unspecified Number of Components

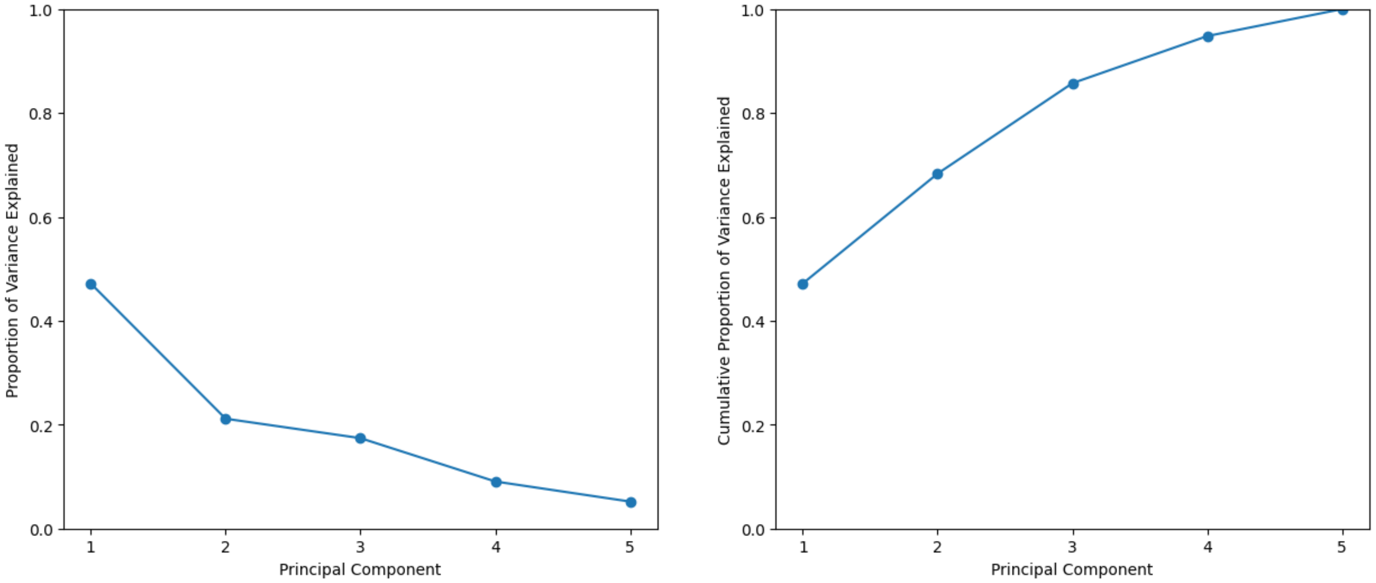
Based on the below graphs we would choose to use 4 principal components, which would capture about 95% of the total variance.

In [ ]:

```
fig, axes = plt.subplots(1, 2, figsize=(15, 6))
ticks = np.arange(pcadf.n_components_)+1
ax = axes[0]
ax.plot(ticks,
    pcadf.explained_variance_ratio_,
    marker='o')
ax.set_xlabel('Principal Component');
ax.set_ylabel('Proportion of Variance Explained')
ax.set_ylim([0,1])
ax.set_xticks(ticks)
ax = axes[1]
ax.plot(ticks,
    pcadf.explained_variance_ratio_.cumsum(),
    marker='o')
ax.set_xlabel('Principal Component')
ax.set_ylabel('Cumulative Proportion of Variance Explained')
ax.set_ylim([0, 1])
ax.set_xticks(ticks)
```

Out[ ]:

```
[<matplotlib.axis.XTick at 0x7f54ecdeedd0>,
 <matplotlib.axis.XTick at 0x7f54ecdecee0>,
 <matplotlib.axis.XTick at 0x7f54ecdefd90>,
 <matplotlib.axis.XTick at 0x7f54f1f6e620>,
 <matplotlib.axis.XTick at 0x7f54f1f6eb30>]
```



In [ ]:

```
pcadf.explained_variance_
```

Out[ ]:

```
array([2.3595, 1.0578, 0.8707, 0.4534, 0.26  ])
```

In [ ]:

```
pcadf.explained_variance_ratio_
```

Out[ ]:

```
array([0.4718, 0.2115, 0.1741, 0.0907, 0.052 ])
```

4.3 K-Means Analysis

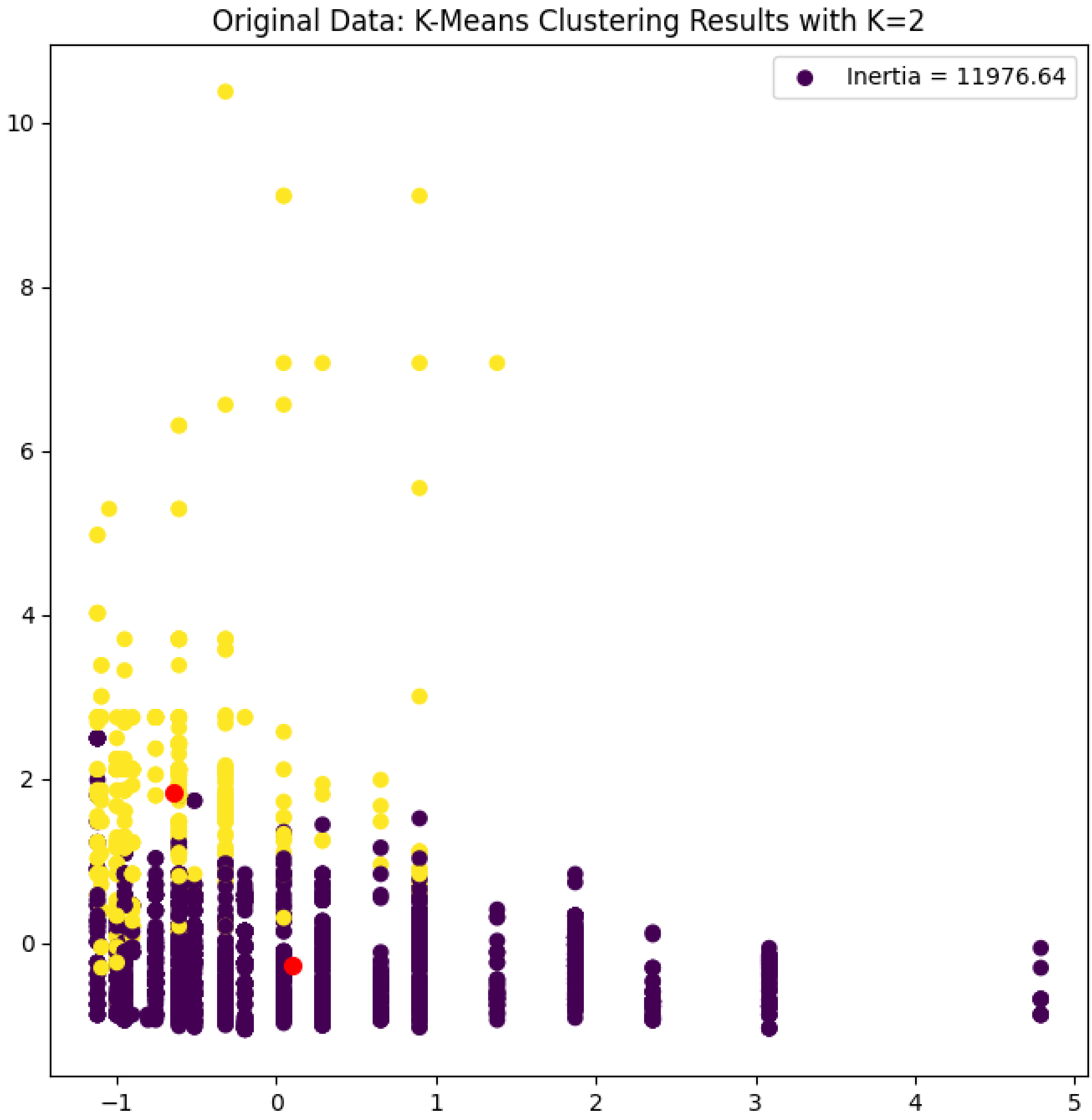
K-Means analysis: The first 5 graphs are done on the original data with 5 different choices of clusters for base-level comparison. Then, the next two sets of graphs are on the original PCA data and the dimension-reduced PCA data, in that order.

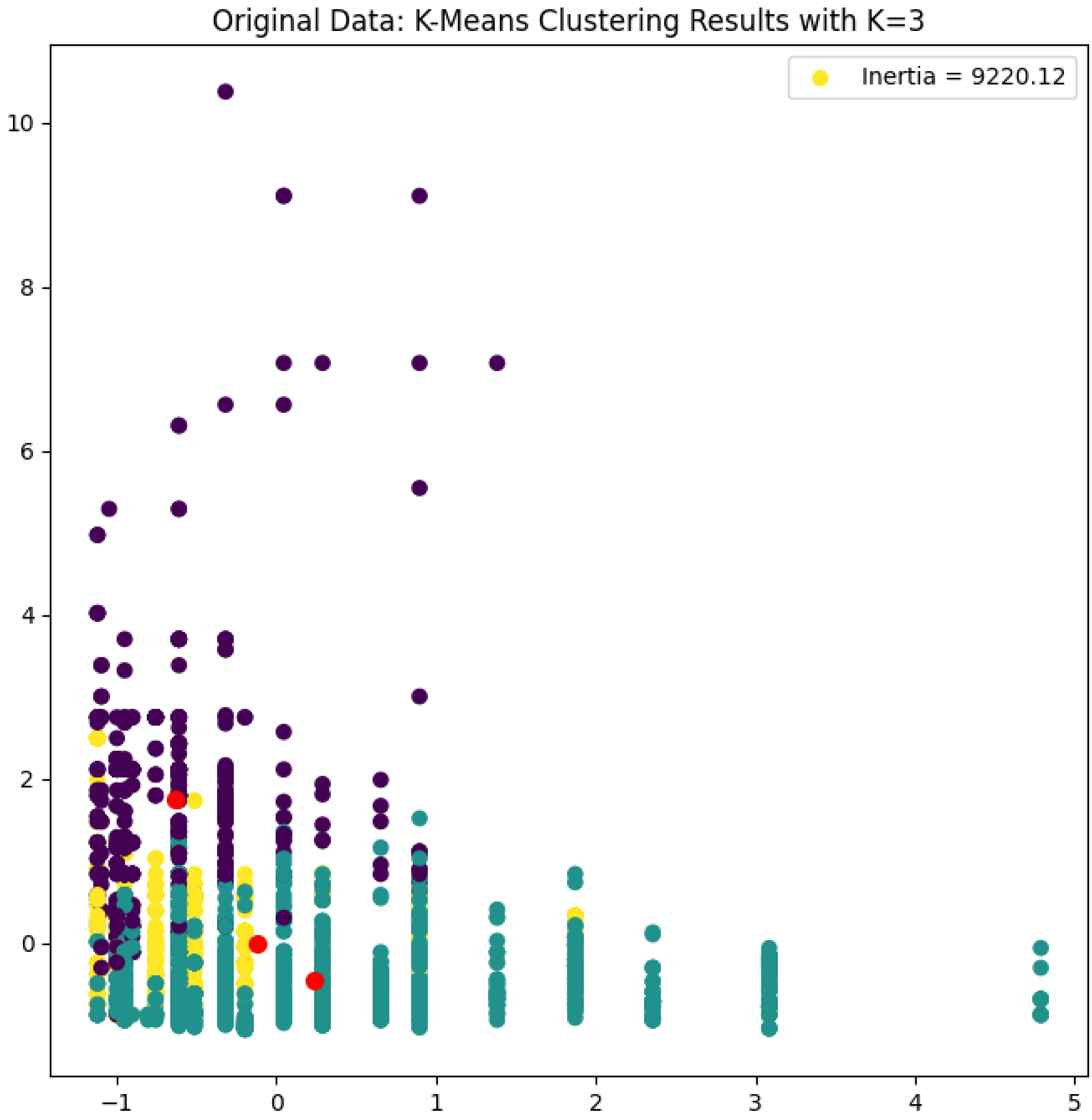
In [ ]:

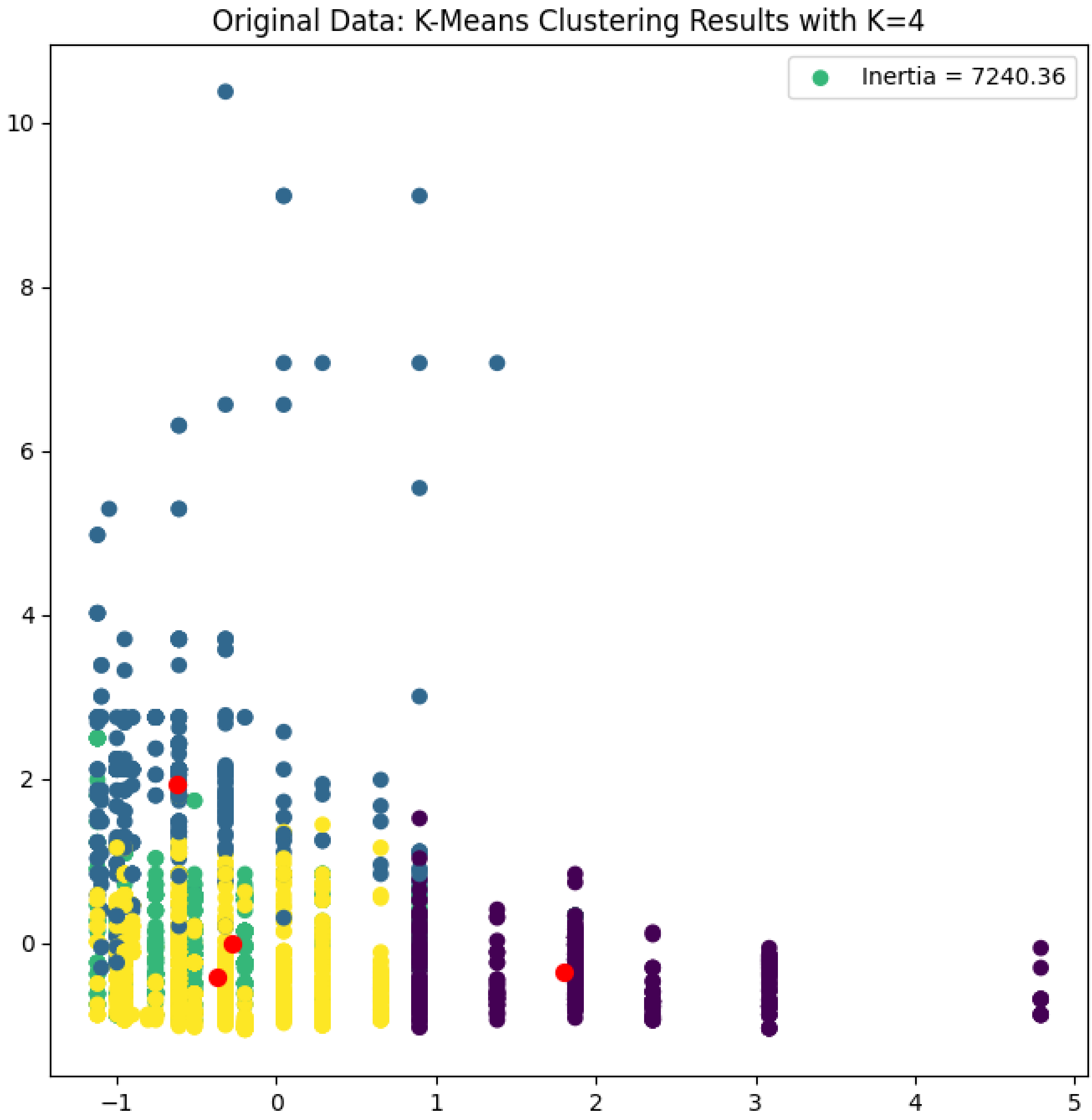
```
from sklearn.cluster import KMeans
from sklearn.cluster import DBSCAN
np.random.seed(126)
```

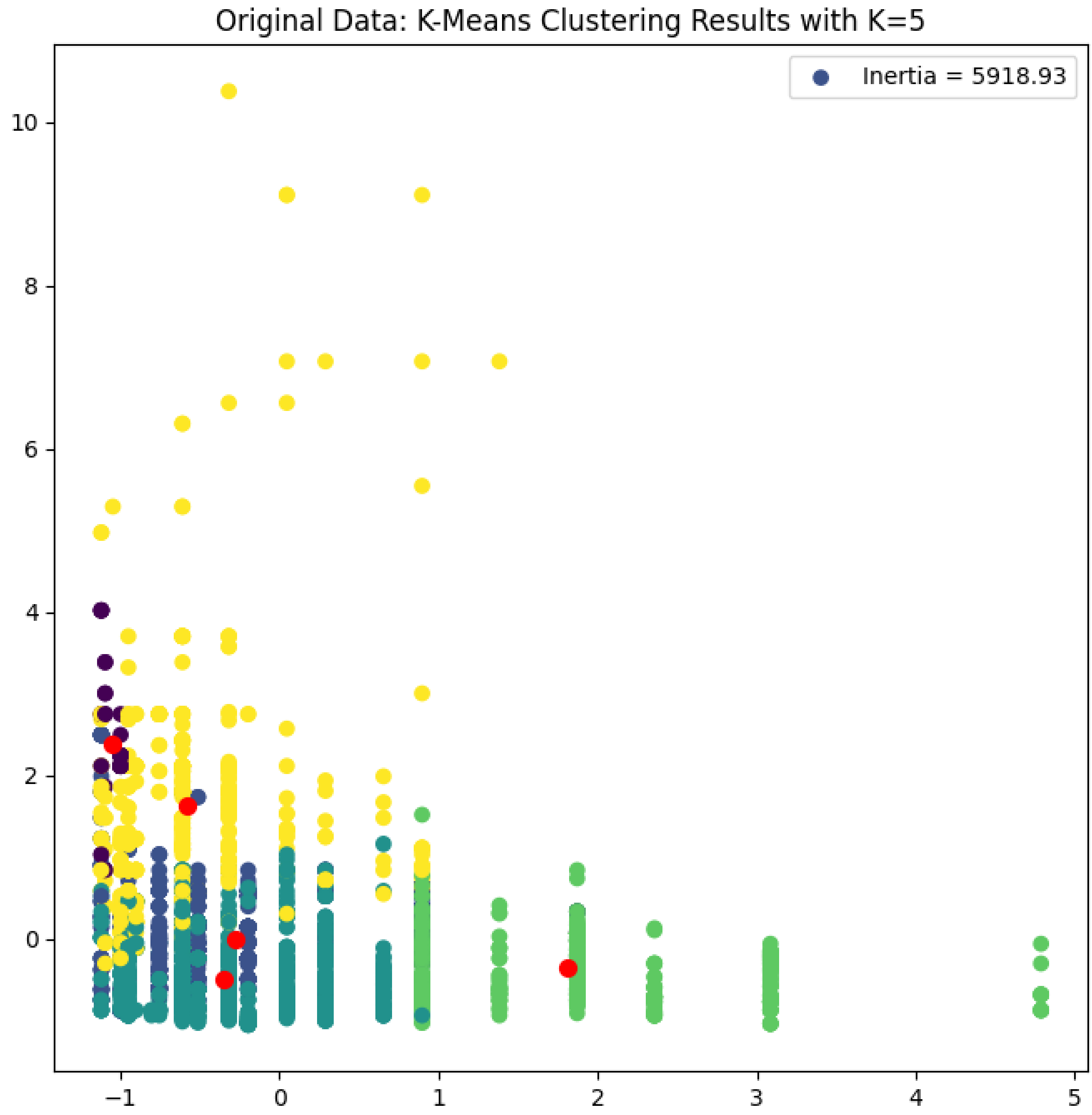
In [ ]:

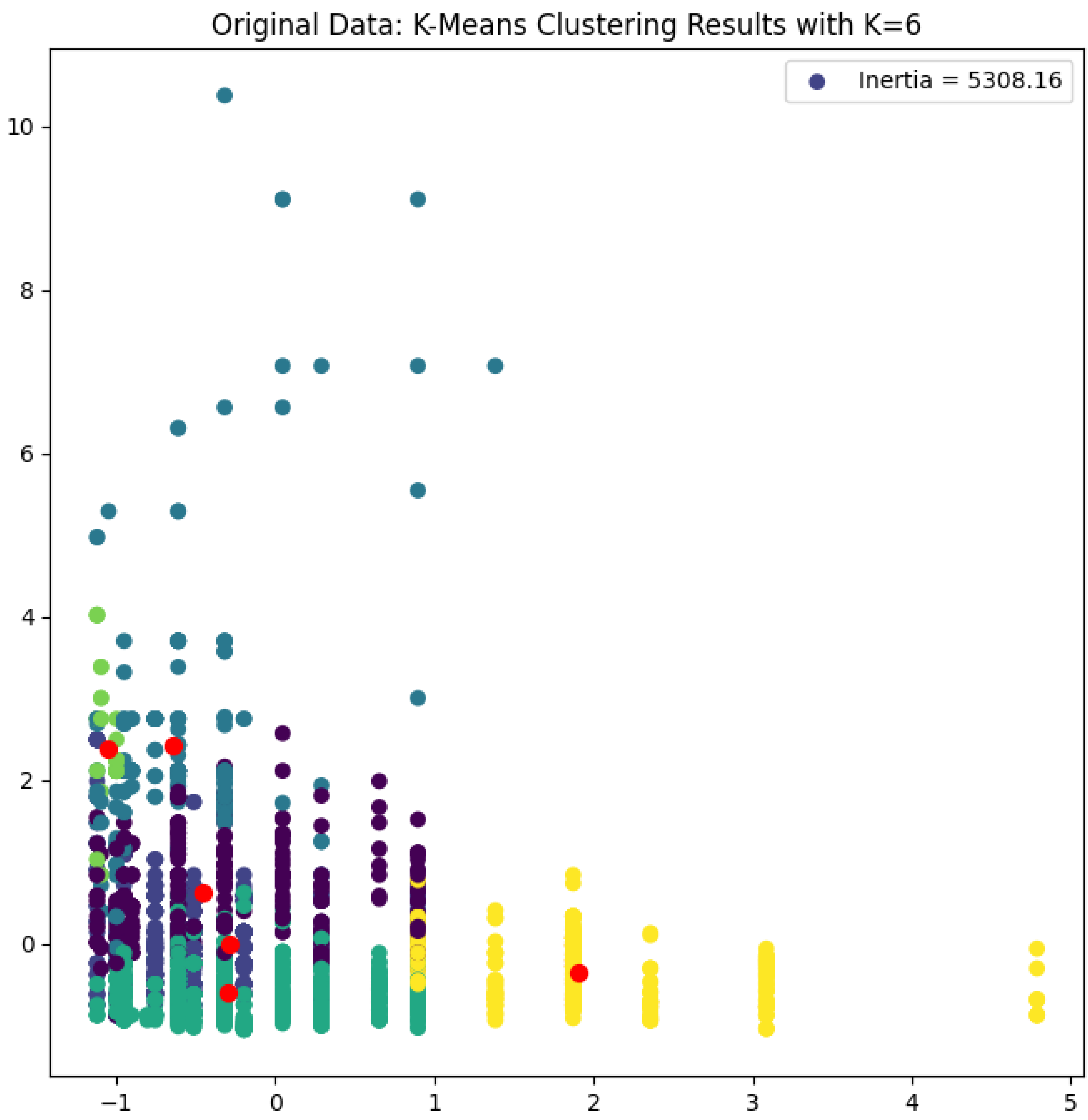
```
for i in range(2,7):
    dfkmeans = KMeans(n_clusters=i,
                      random_state=0,
                      n_init=20).fit(df_scaled)
    label = dfkmeans.fit_predict(df_scaled)
    fig, ax = plt.subplots(1, 1, figsize=(8,8))
    ax.scatter(df_scaled[:,0], df_scaled[:,1], c=label, label='Inertia = '+"{:.2f}".format(dfk
    ax.set_title("Original Data: K-Means Clustering Results with K="+str(i))
    plt.scatter(dfkmeans.cluster_centers_[:,0], dfkmeans.cluster_centers_[:,1], s=50, c='red')
    leg=plt.legend()
    i += 1
```


Original Data: K-Means Clustering Results with K=2
● Inertia = 11976.64
10
8
6
4
2
0
−1
0
1
2
3
4
5


Original Data: K-Means Clustering Results with K=3
Inertia = 9220.12
10
8
6
4
2
0
−1
0
1
2
3
4
5

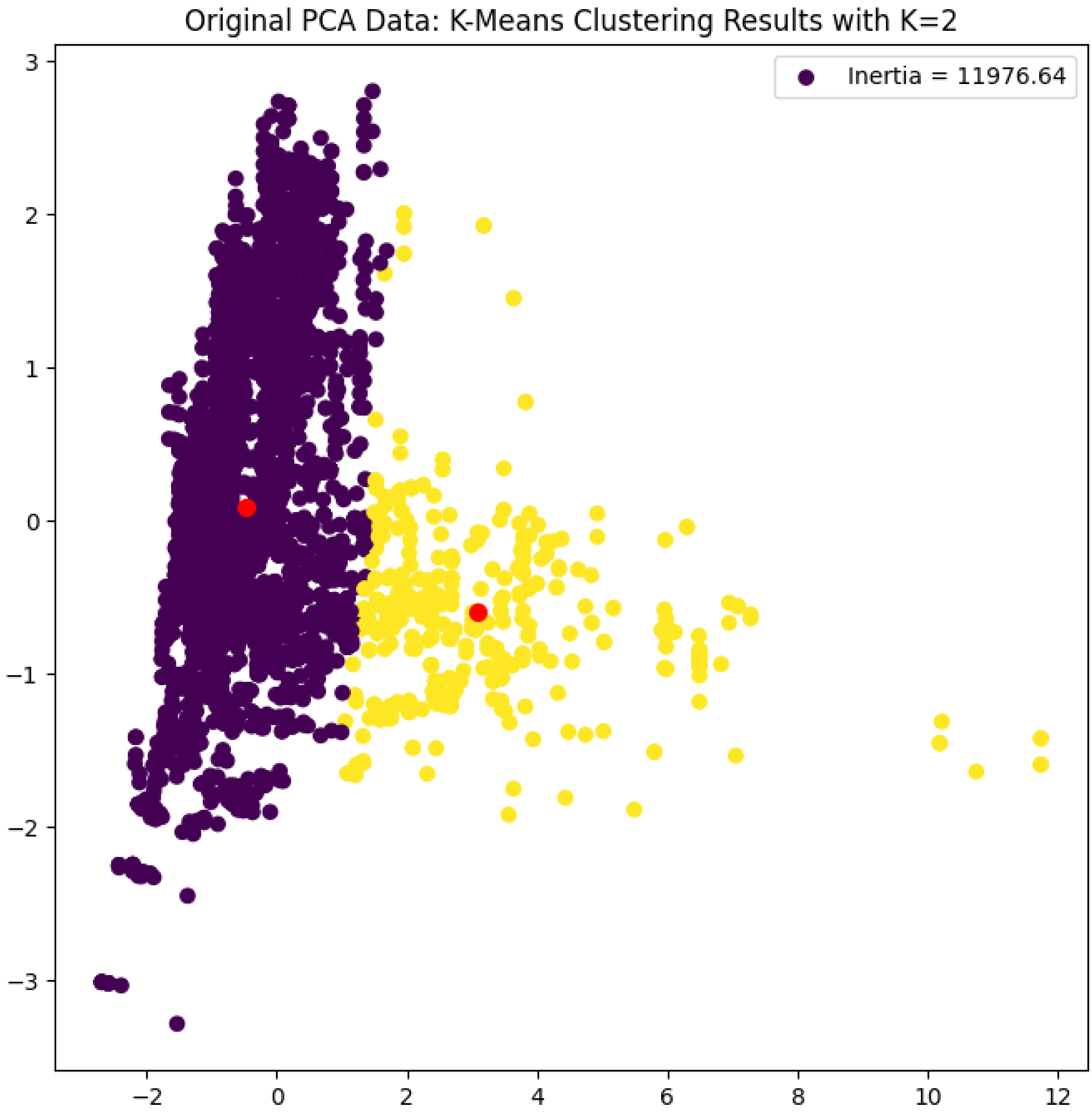

Original Data: K-Means Clustering Results with K=4
Inertia = 7240.36
10
8
6
4
2
0
−1
0
1
2
3
4
5

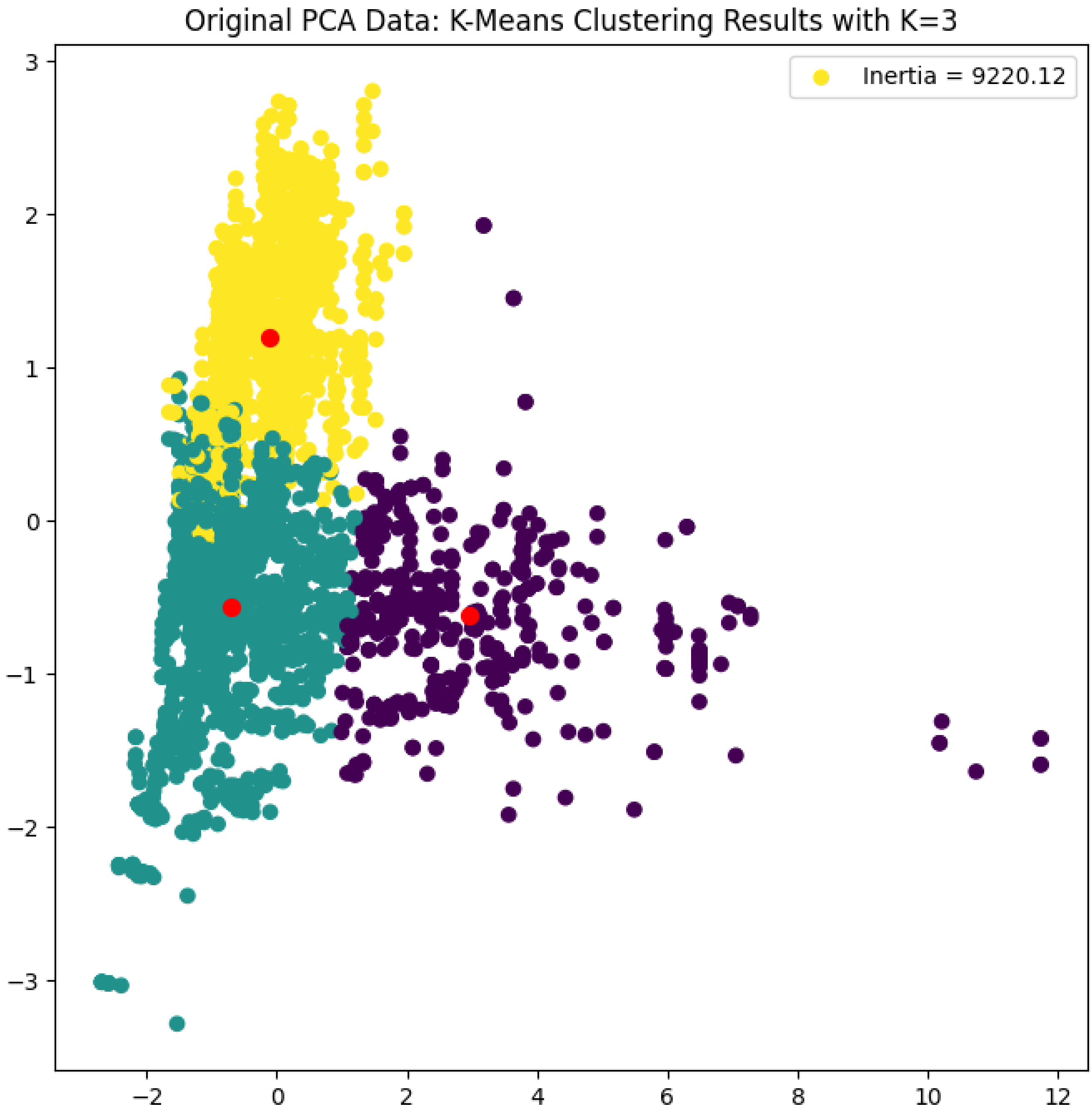

Original Data: K-Means Clustering Results with K=5
● Inertia = 5918.93
10
8
6
4
2
0
−1
0
1
2
3
4
5

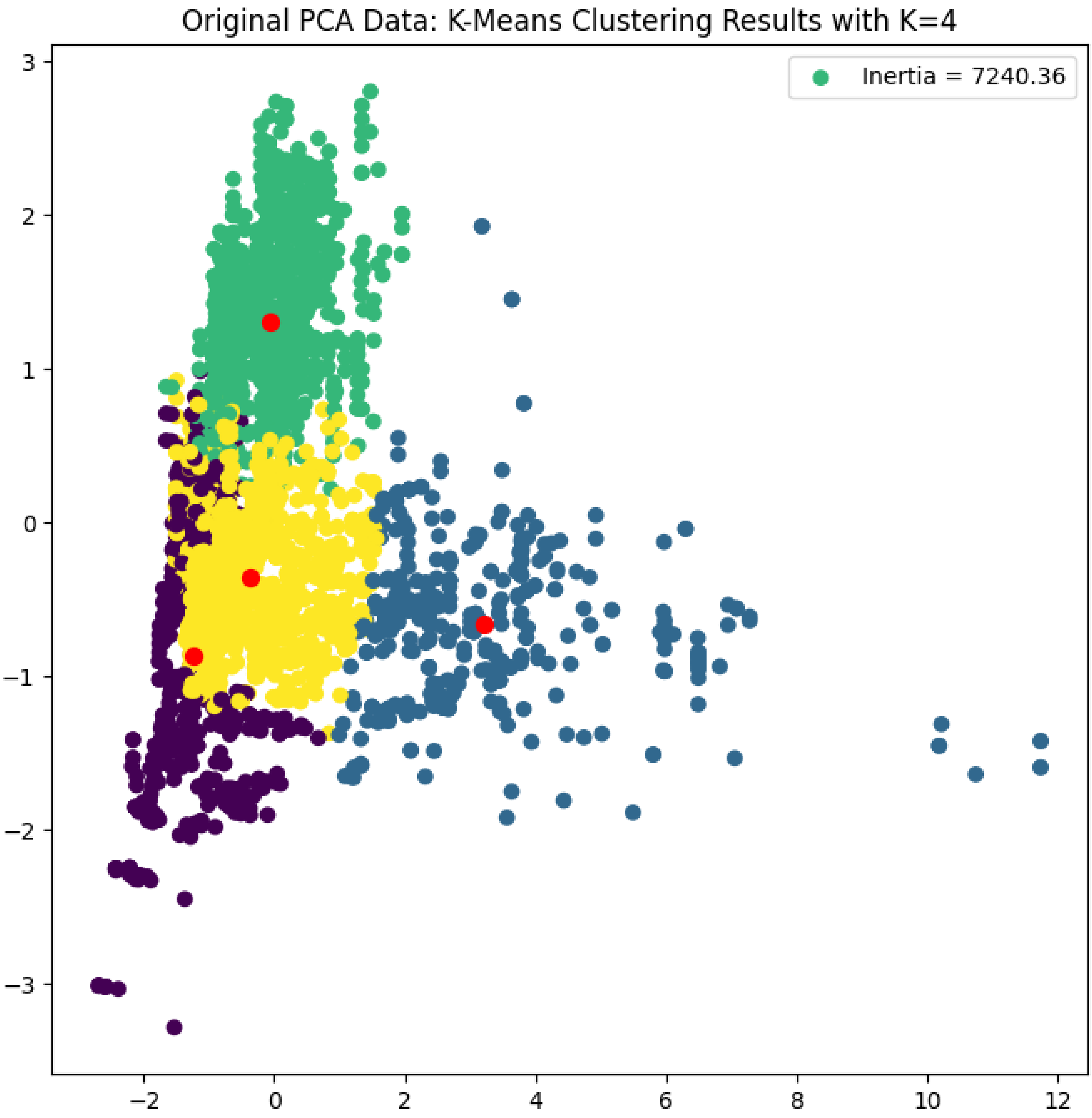


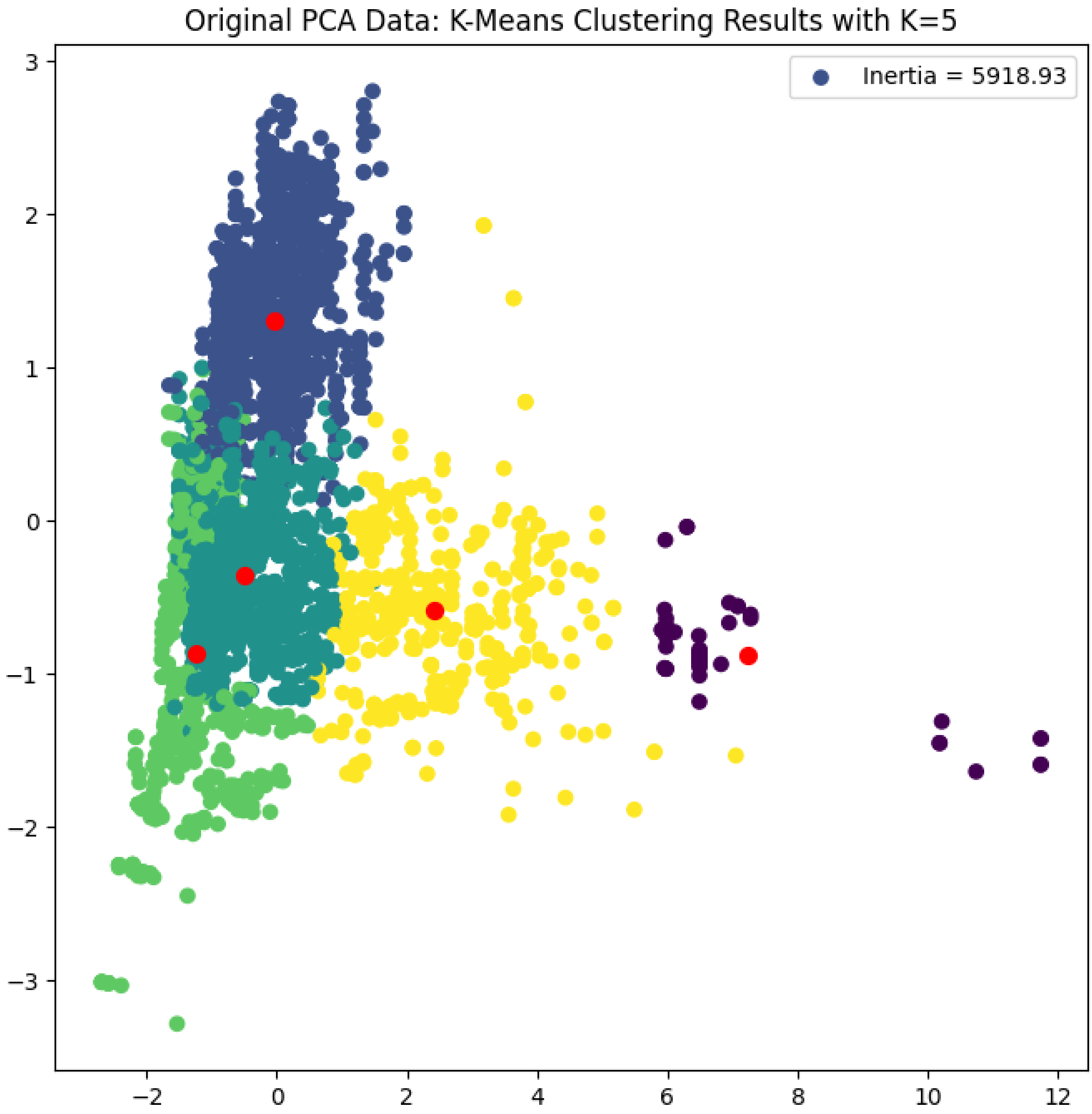
In [ ]:

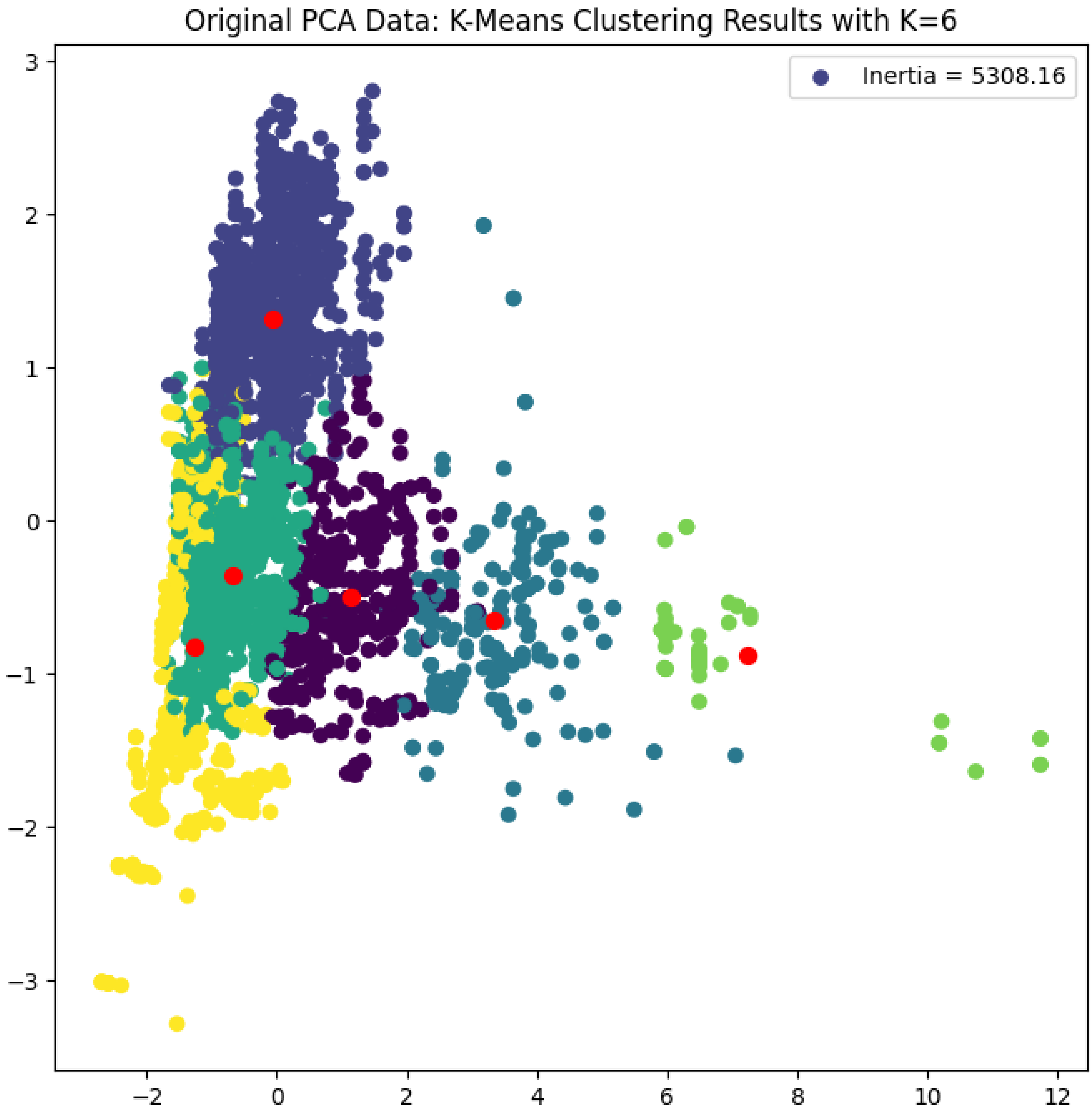
```
for i in range(2,7):
    dfkmeans = KMeans(n_clusters=i,
                      random_state=0,
                      n_init=20).fit(scores)
    label = dfkmeans.fit_predict(scores)
    fig, ax = plt.subplots(1, 1, figsize=(8,8))
    ax.scatter(scores[:,0], scores[:,1], c=label, label='Inertia = '+"{:.2f}".format(dfkmeans.
    ax.set_title("Original PCA Data: K-Means Clustering Results with K="+str(i))
    plt.scatter(dfkmeans.cluster_centers_[:,0], dfkmeans.cluster_centers_[:,1], s=50, c='red')
    leg=plt.legend()
    i += 1
```


Original PCA Data: K-Means Clustering Results with K=2
● Inertia = 11976.64
3
2
1
0
−1
−2
−3
−2
0
2
4
6
8
10
12

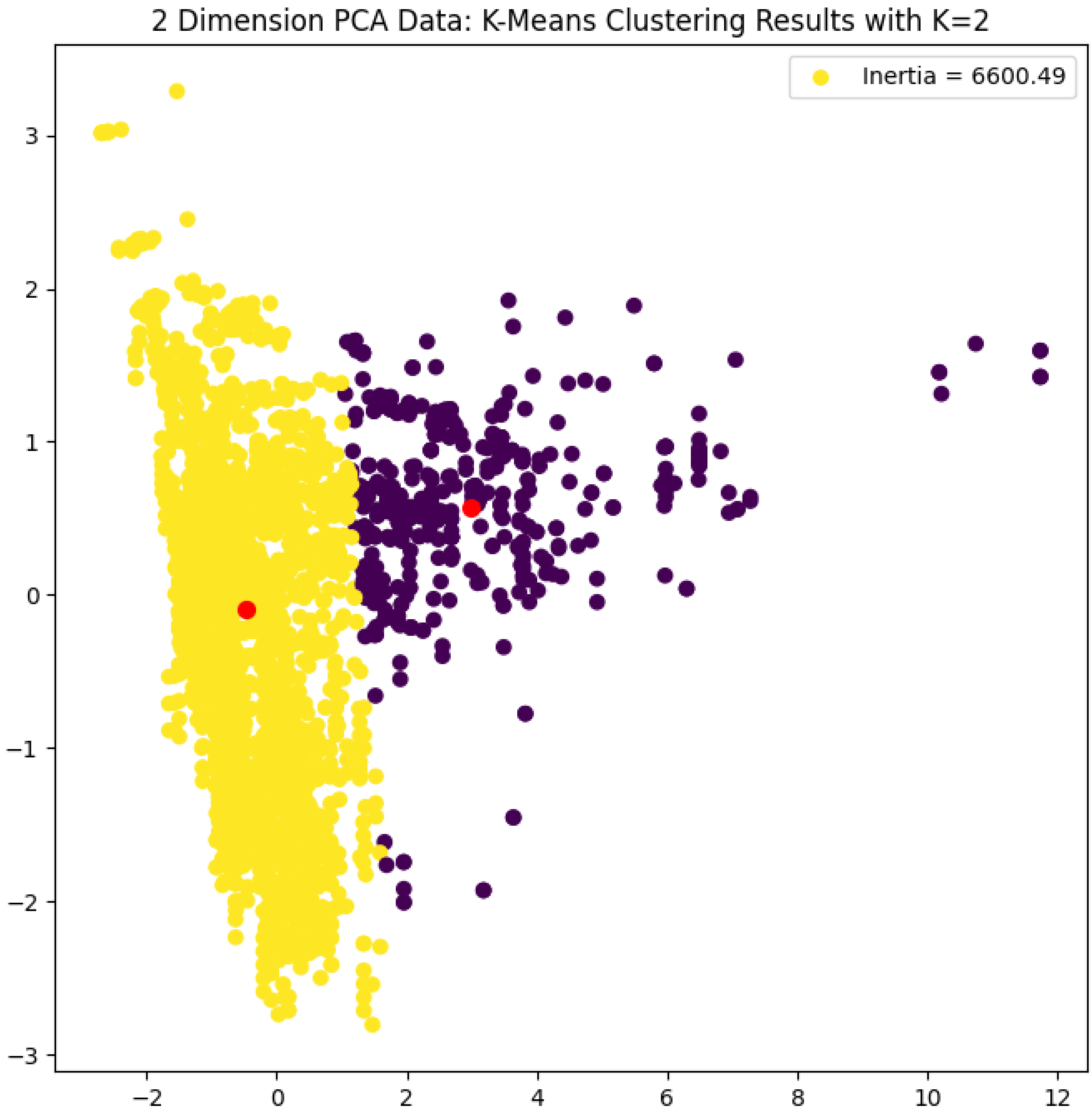

Original PCA Data: K-Means Clustering Results with K=3
Inertia = 9220.12
3
2
1
0
−1
−2
−3
−2
0
2
4
6
8
10
12

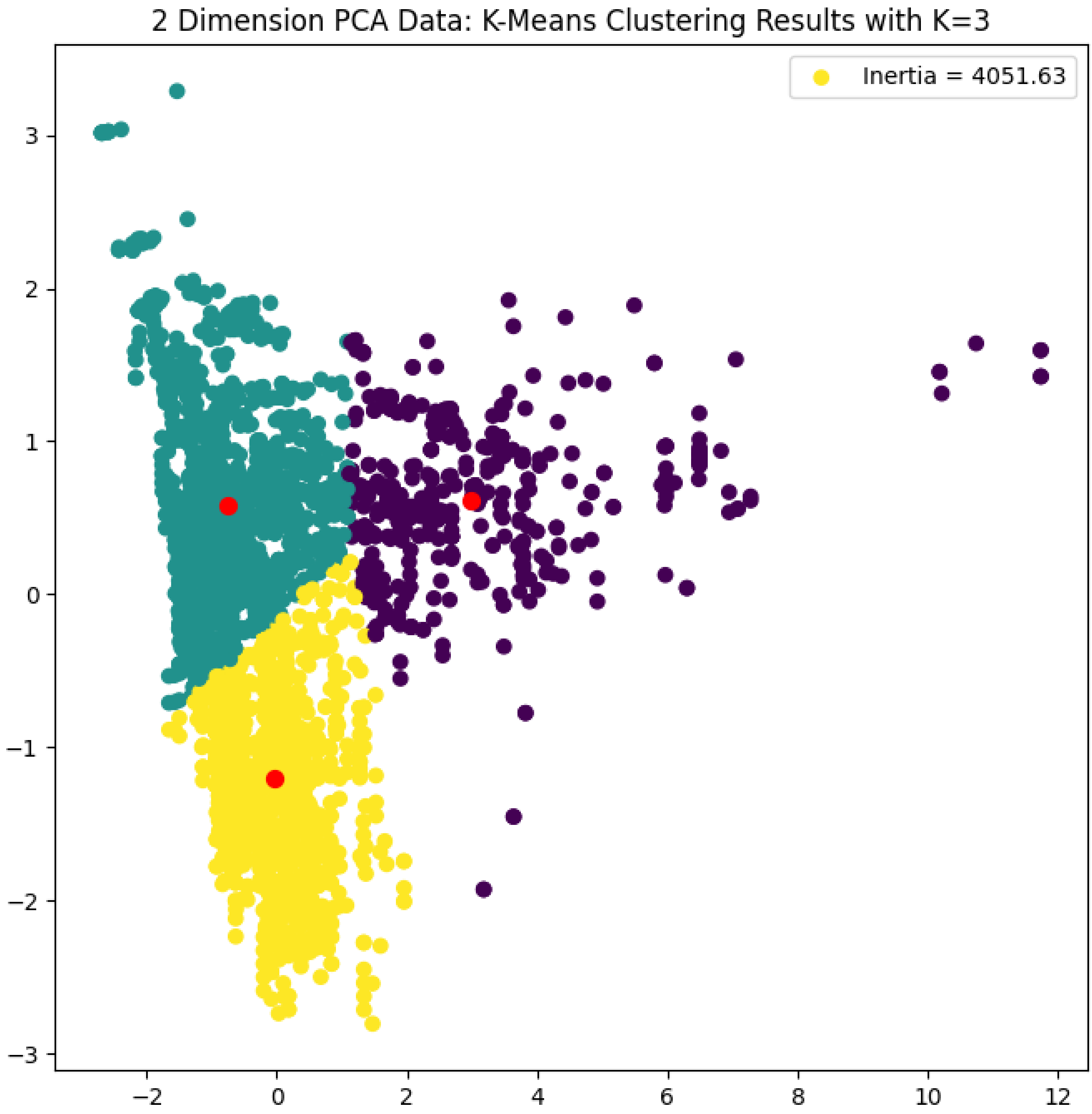

Original PCA Data: K-Means Clustering Results with K=4
Inertia = 7240.36
3
2
1
0
−1
−2
−3
−2
0
2
4
6
8
10
12

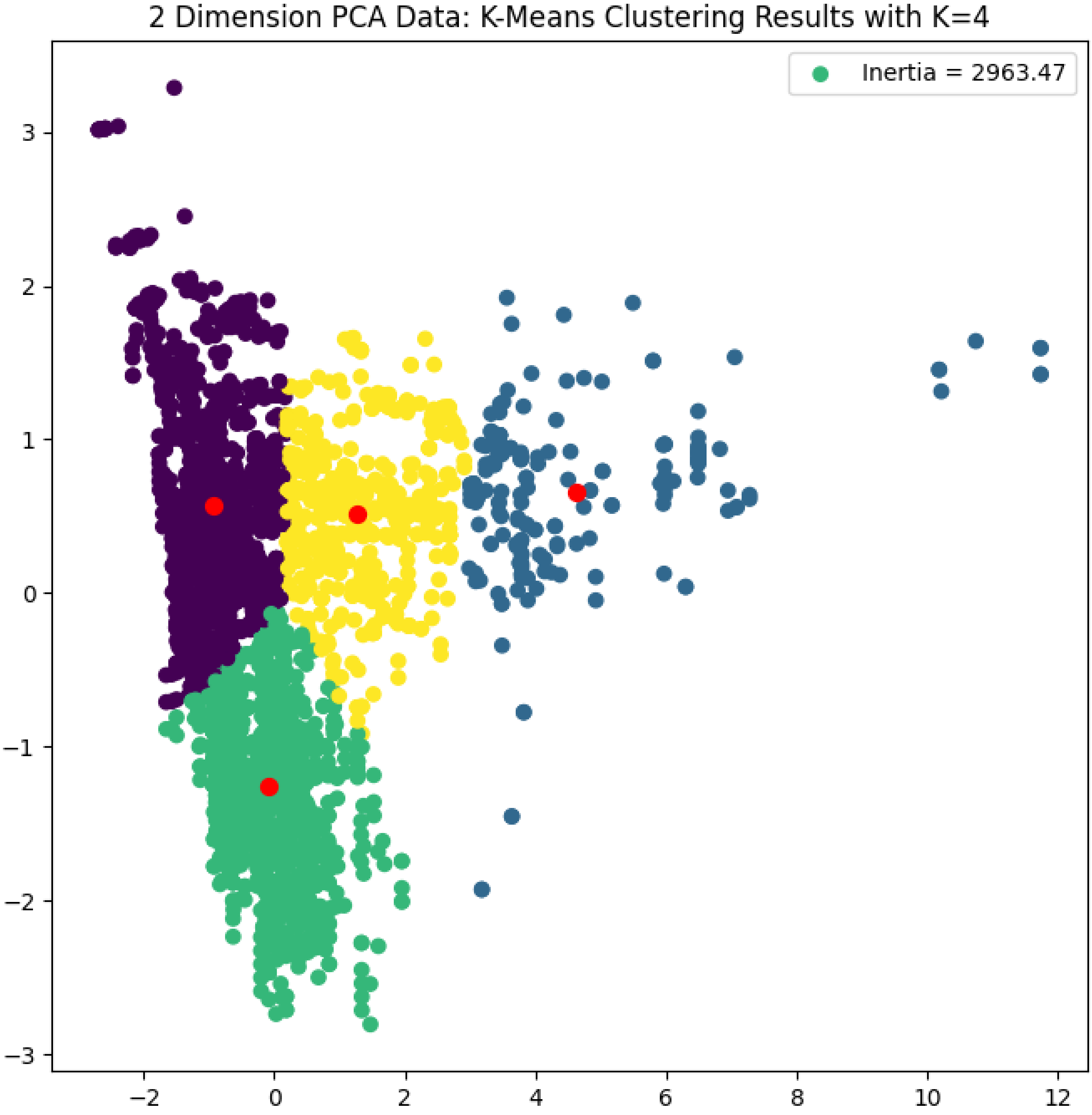

Original PCA Data: K-Means Clustering Results with K=5
● Inertia = 5918.93
3
2
1
0
−1
−2
−3
−2
0
2
4
6
8
10
12

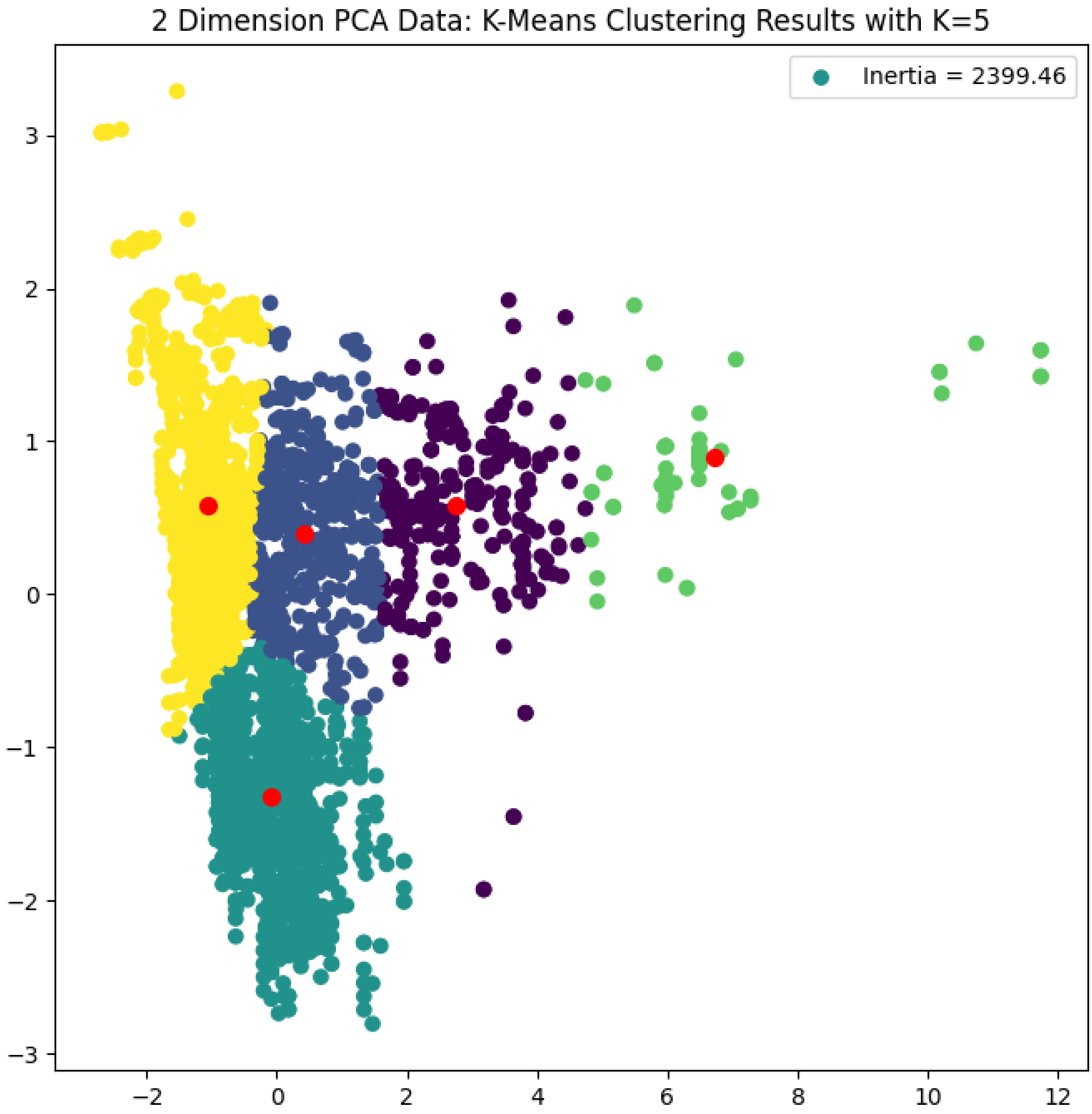


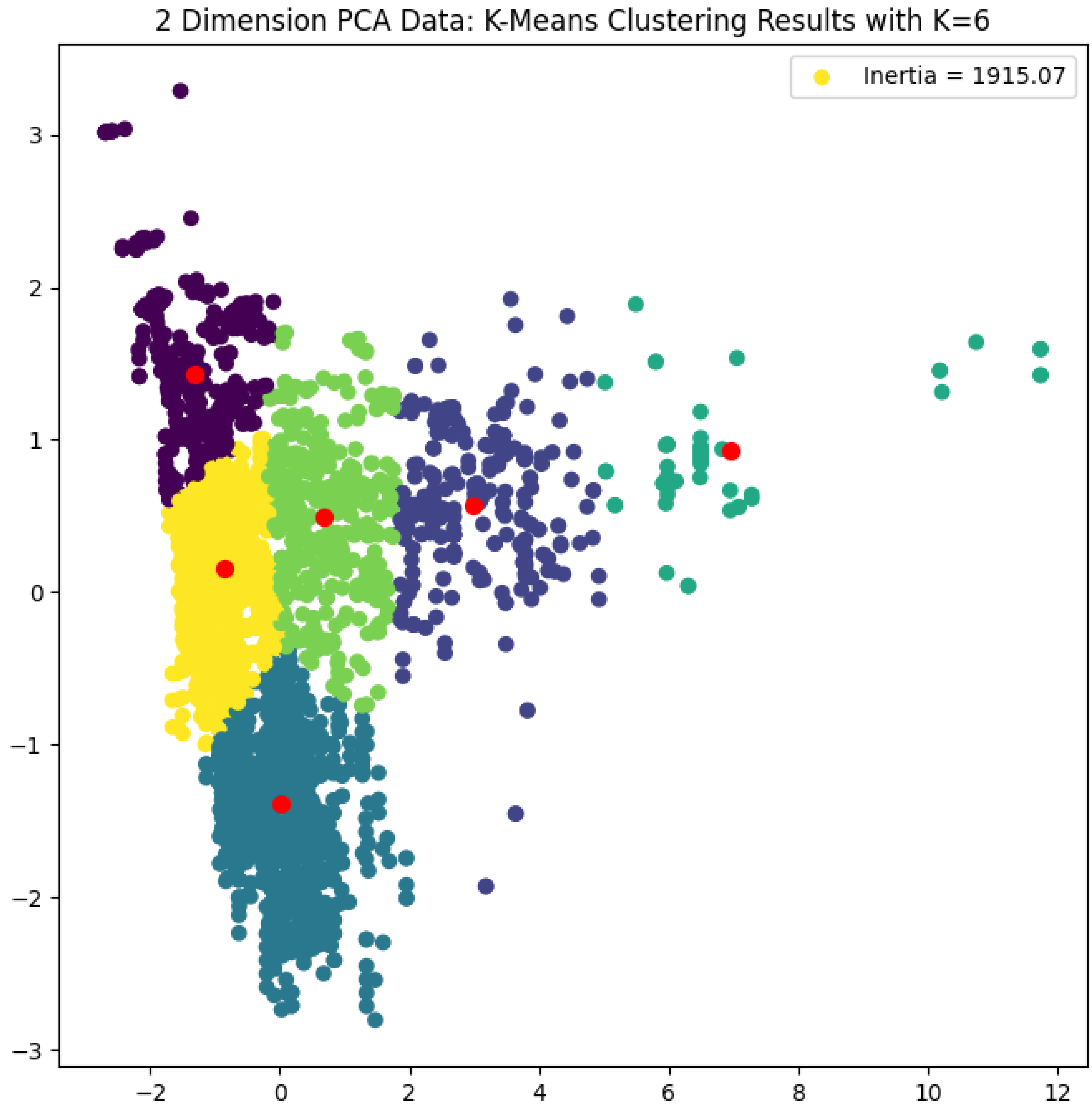
```
In [ ]: for i in range(2,7):
            dfkmeans = KMeans(n_clusters=i,
                              random_state=0,
                              n_init=20).fit(scores2)
            label = dfkmeans.fit_predict(scores2)
            fig, ax = plt.subplots(1, 1, figsize=(8,8))
            ax.scatter(scores2[:,0], scores2[:,1], c=label, label='Inertia = '+"{:.2f}".format(dfkmean
            ax.set_title("2 Dimension PCA Data: K-Means Clustering Results with K="+str(i))
            plt.scatter(dfkmeans.cluster_centers_[:,0], dfkmeans.cluster_centers_[:,1], s=50, c='red')
            leg=plt.legend()
            i += 1
```


2 Dimension PCA Data: K-Means Clustering Results with K=2
Inertia = 6600.49
3
2
1
0
−1
−2
−3
−2
0
2
4
6
8
10
12


2 Dimension PCA Data: K-Means Clustering Results with K=3
Inertia = 4051.63
3
2
1
0
−1
−2
−3
−2
0
2
4
6
8
10
12


2 Dimension PCA Data: K-Means Clustering Results with K=4
● Inertia = 2963.47
3
2
1
0
−1
−2
−3
−2
0
2
4
6
8
10
12


2 Dimension PCA Data: K-Means Clustering Results with K=5
● Inertia = 2399.46
3
2
1
0
−1
−2
−3
−2
0
2
4
6
8
10
12

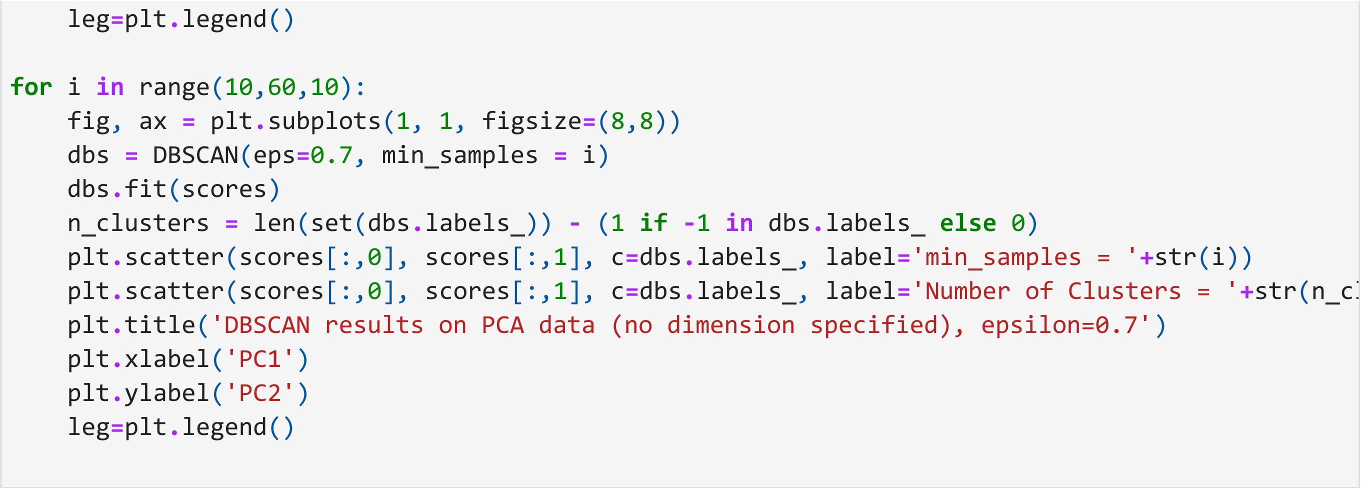


4.4 DBSCAN Analysis

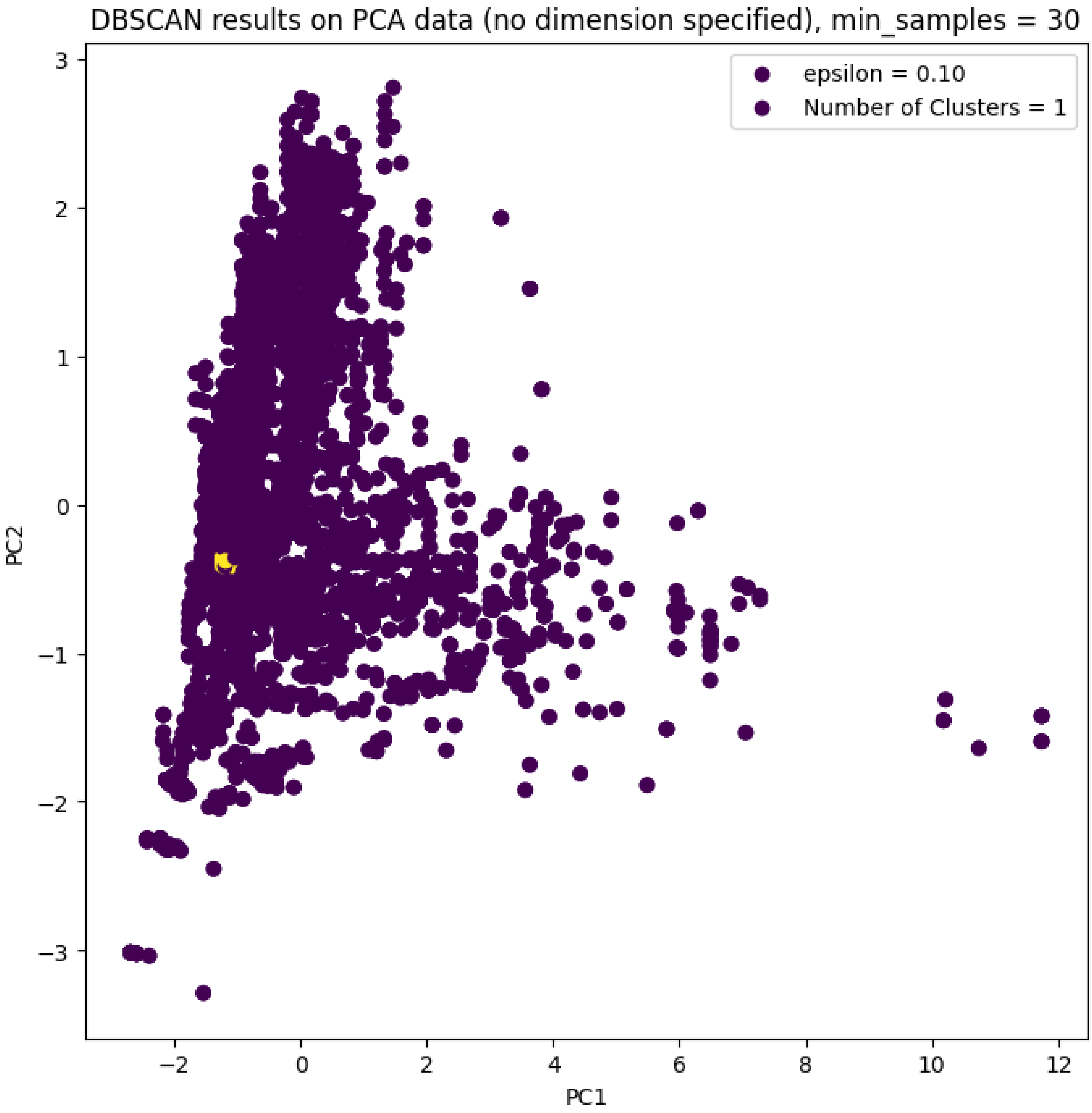
The base values of epsilon and min_samples we used to iterate through are 0.7 and 30, respectively.

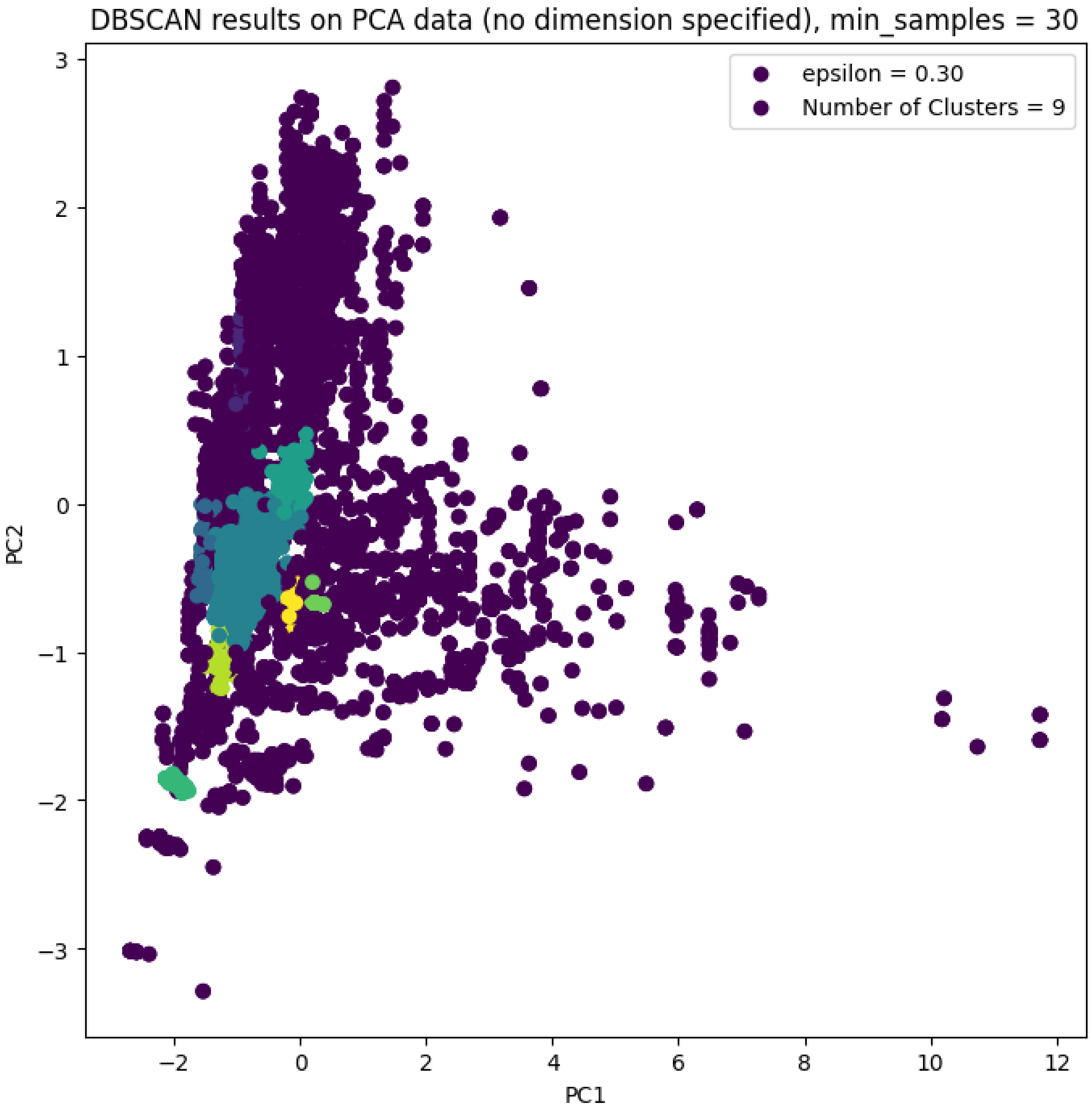
In [ ]:

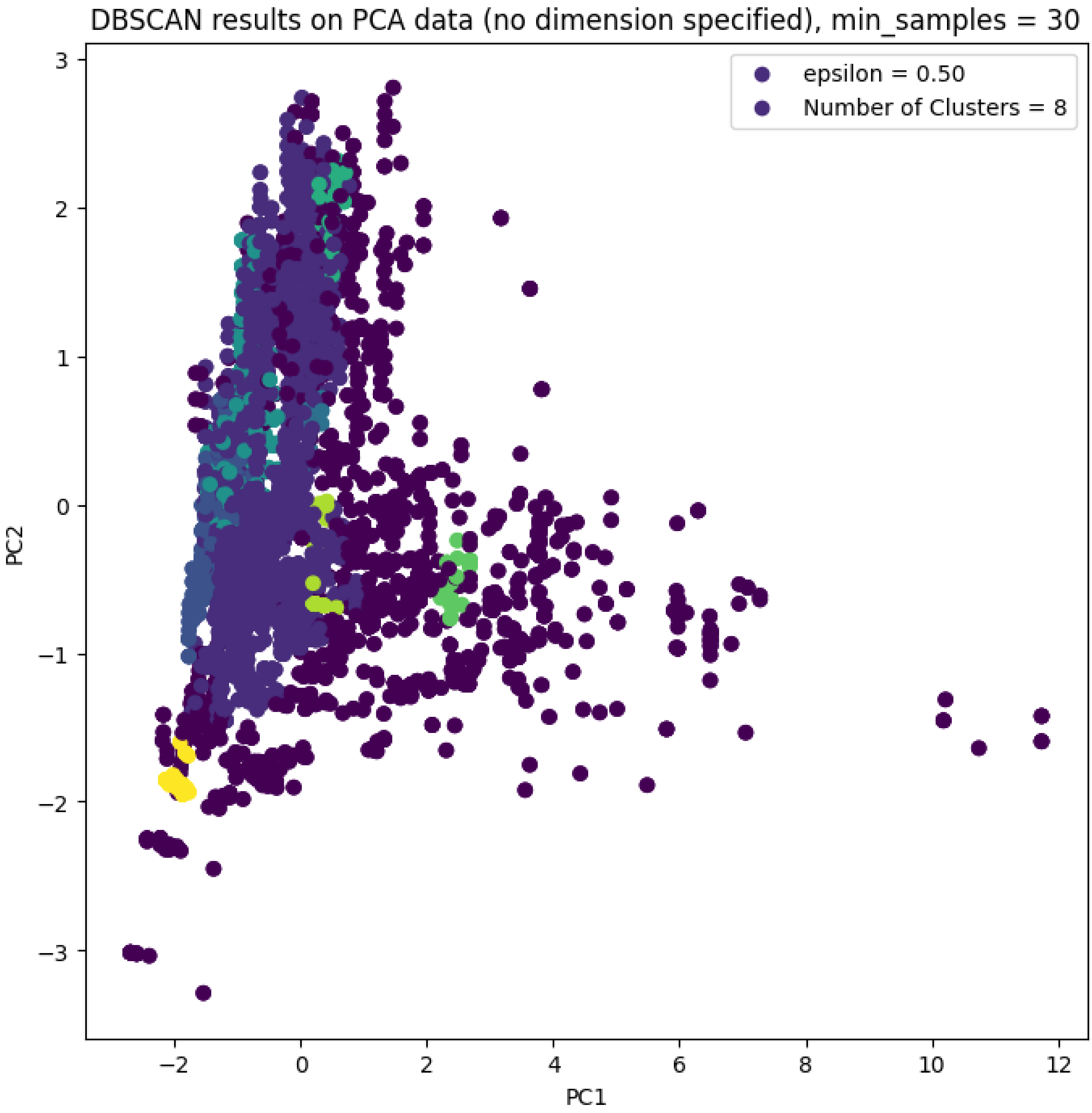
```
for i in np.arange(0.1,1,0.2):
    fig, ax = plt.subplots(1, 1, figsize=(8,8))
    dbs = DBSCAN(eps=i, min_samples = 30)
    dbs.fit(scores)
    n_clusters = len(set(dbs.labels_)) - (1 if -1 in dbs.labels_ else 0)
    plt.scatter(scores[:,0], scores[:,1], c=dbs.labels_, label='epsilon = '+"{:.2f}".format(i)
    plt.scatter(scores[:,0], scores[:,1], c=dbs.labels_, label='Number of Clusters = '+str(n_c
    plt.title('DBSCAN results on PCA data (no dimension specified), min_samples = 30')
    plt.xlabel('PC1')
    plt.ylabel('PC2')
```

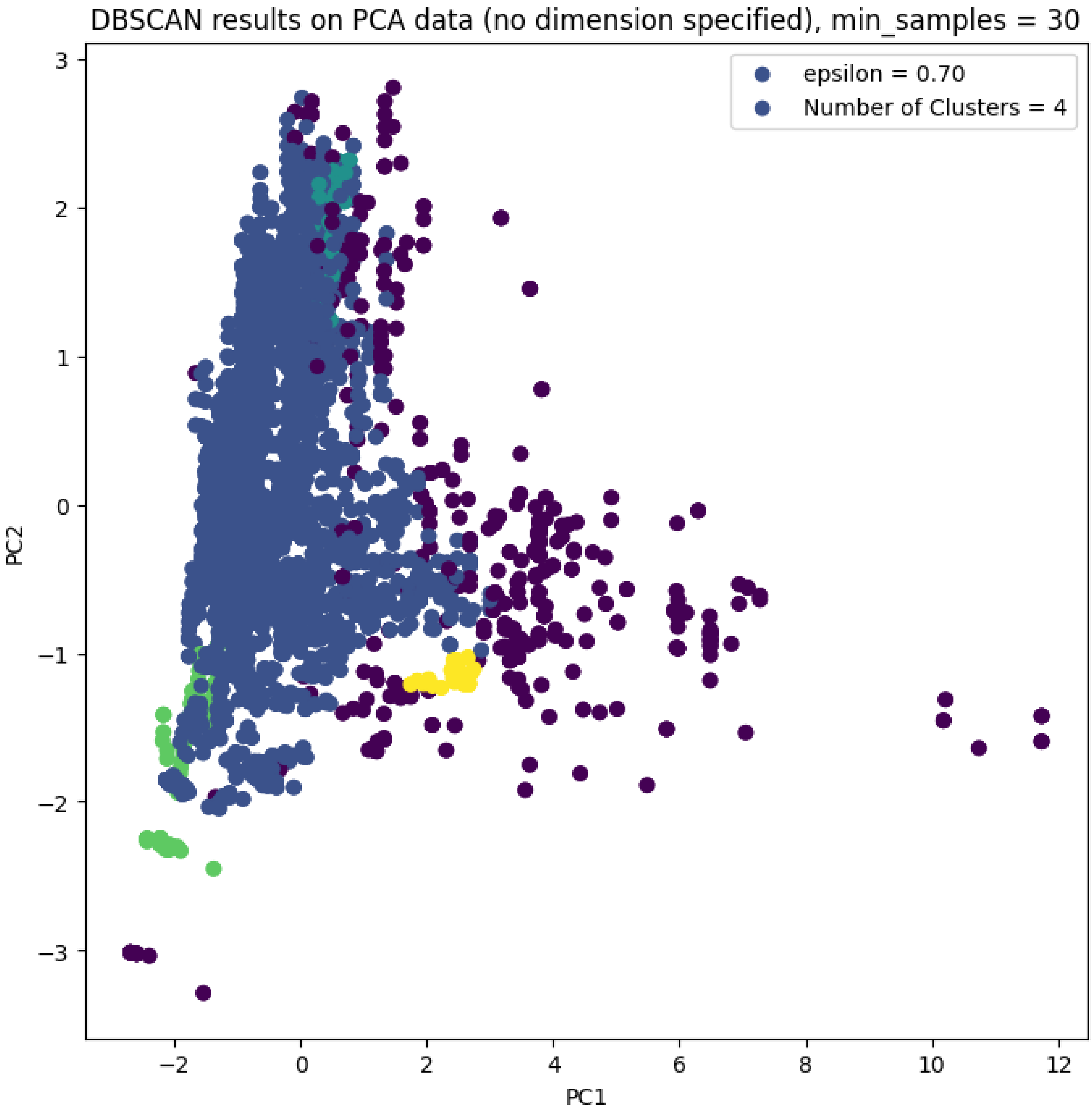


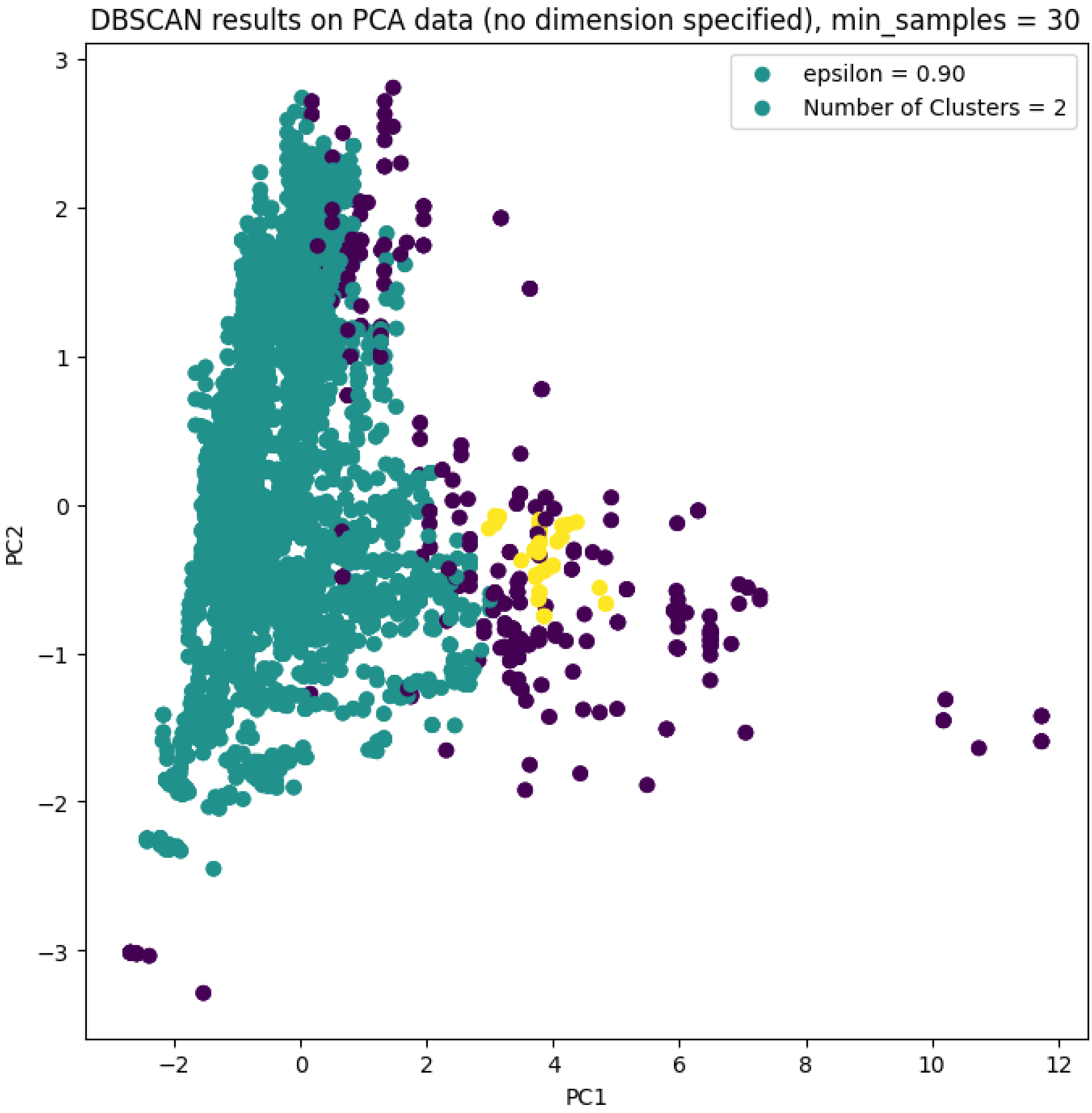
```
    leg=plt.legend()

for i in range(10,60,10):
    fig, ax = plt.subplots(1, 1, figsize=(8,8))
    dbs = DBSCAN(eps=0.7, min_samples = i)
    dbs.fit(scores)
    n_clusters = len(set(dbs.labels_)) - (1 if -1 in dbs.labels_ else 0)
    plt.scatter(scores[:,0], scores[:,1], c=dbs.labels_, label='min_samples = '+str(i))
    plt.scatter(scores[:,0], scores[:,1], c=dbs.labels_, label='Number of Clusters = '+str(n_c
    plt.title('DBSCAN results on PCA data (no dimension specified), epsilon=0.7')
    plt.xlabel('PC1')
    plt.ylabel('PC2')
    leg=plt.legend()
```

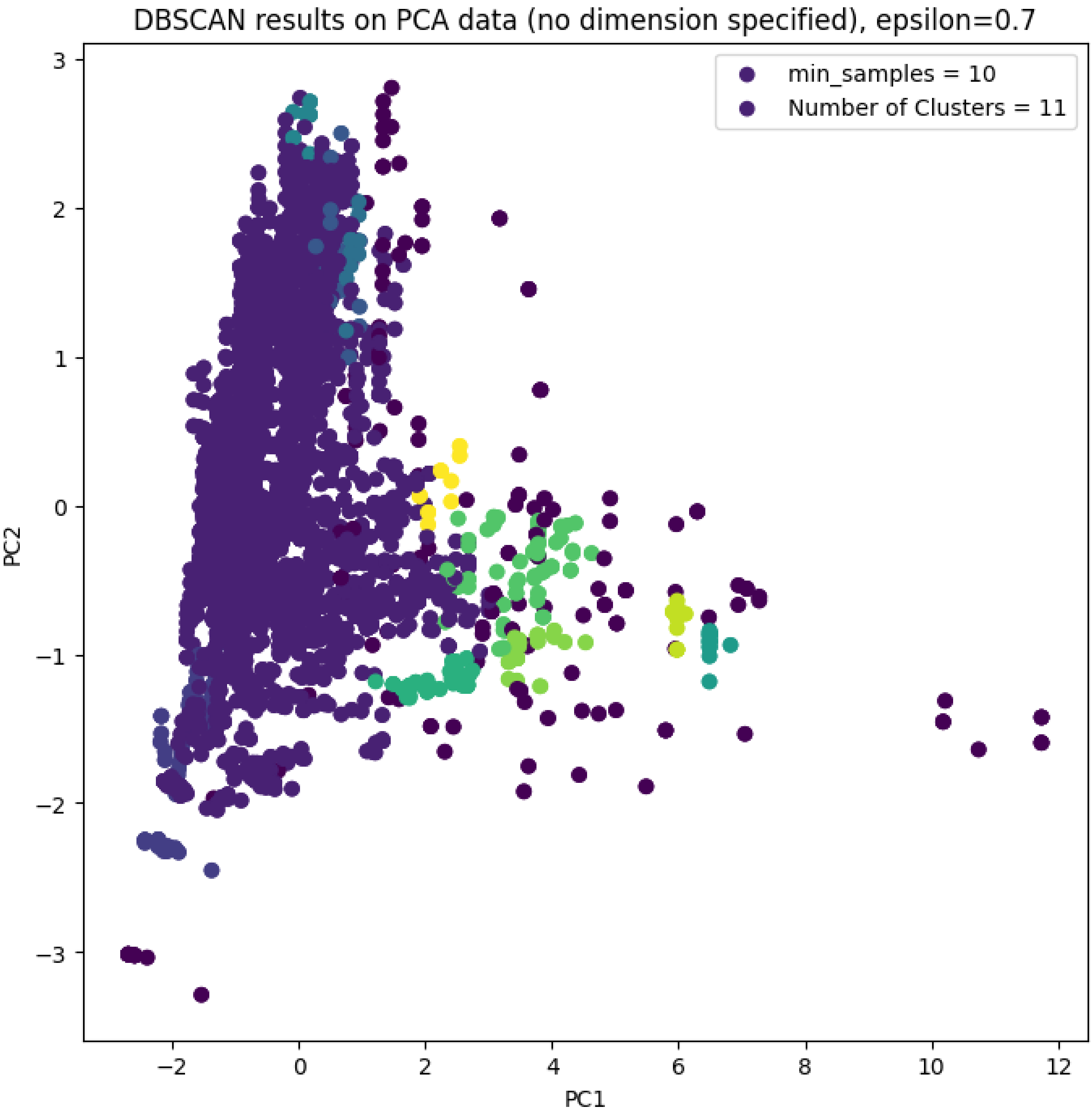


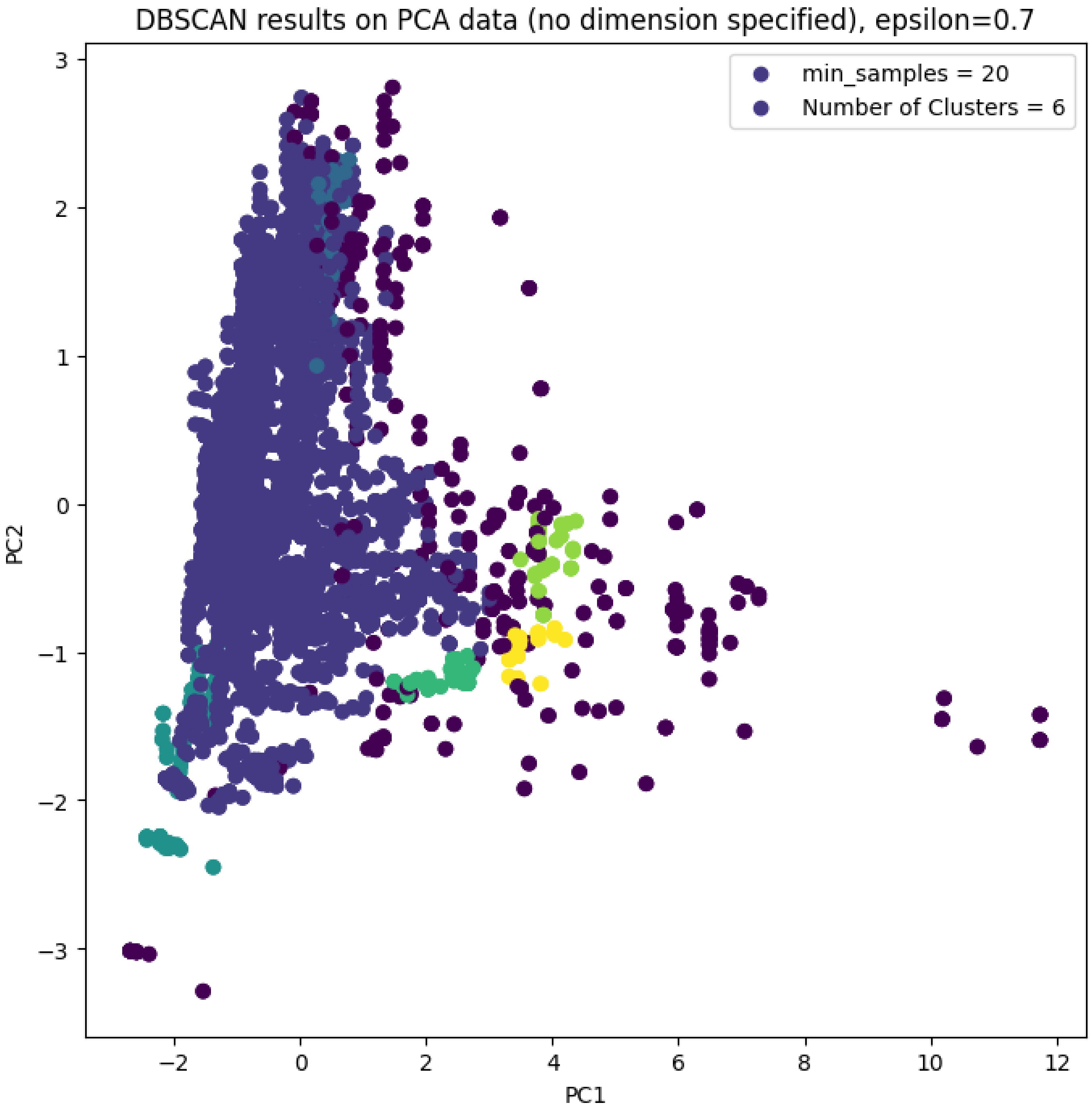

DBSCAN results on PCA data (no dimension specified), min_samples = 30
epsilon = 0.30
Number of Clusters = 9
PC2
PC1
3
2
1
0
−1
−2
−3
−2
0
2
4
6
8
10
12

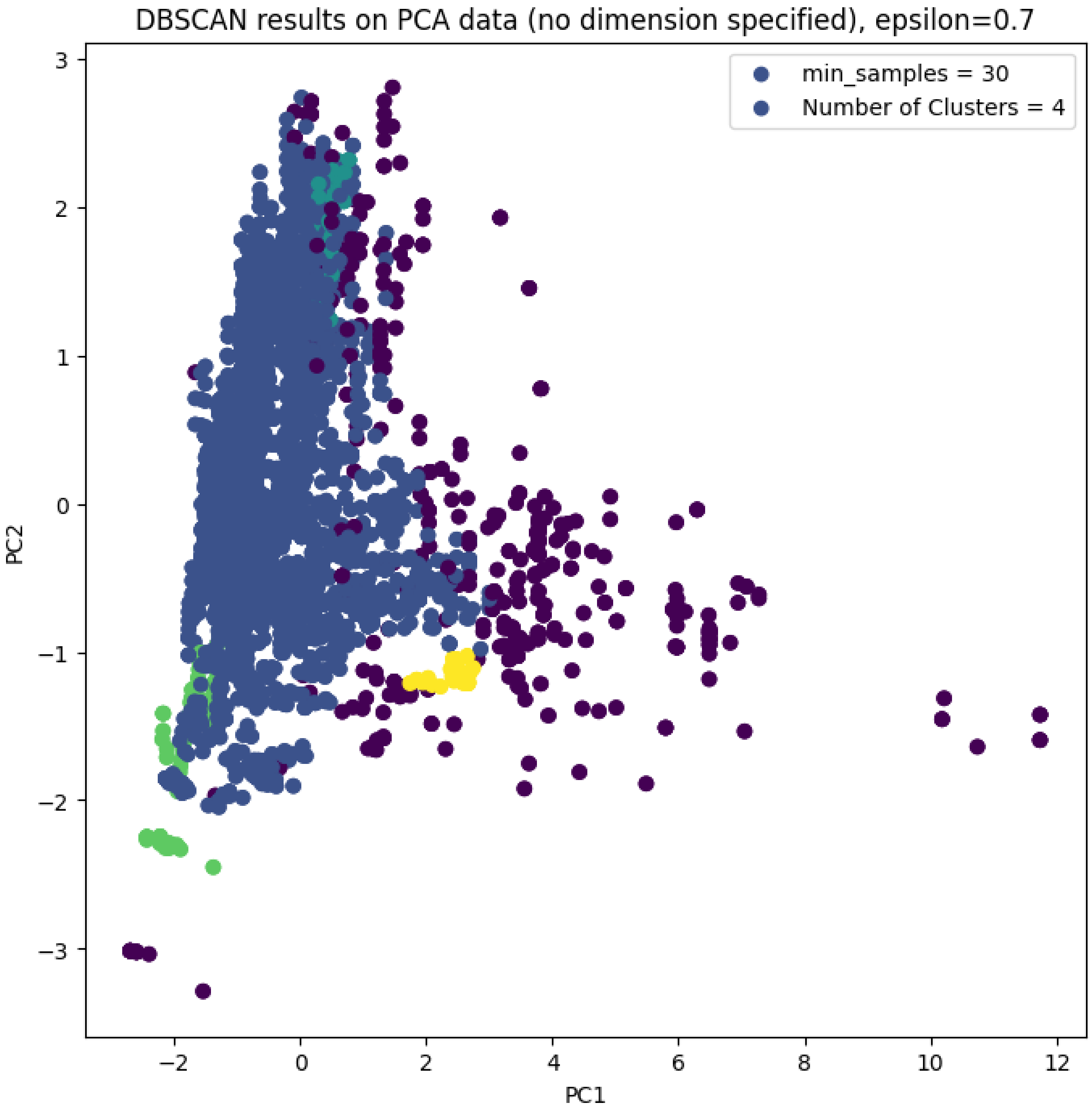

DBSCAN results on PCA data (no dimension specified), min_samples = 30
● epsilon = 0.50
● Number of Clusters = 8
PC2
3
2
1
0
−1
−2
−3
PC1
−2
0
2
4
6
8
10
12

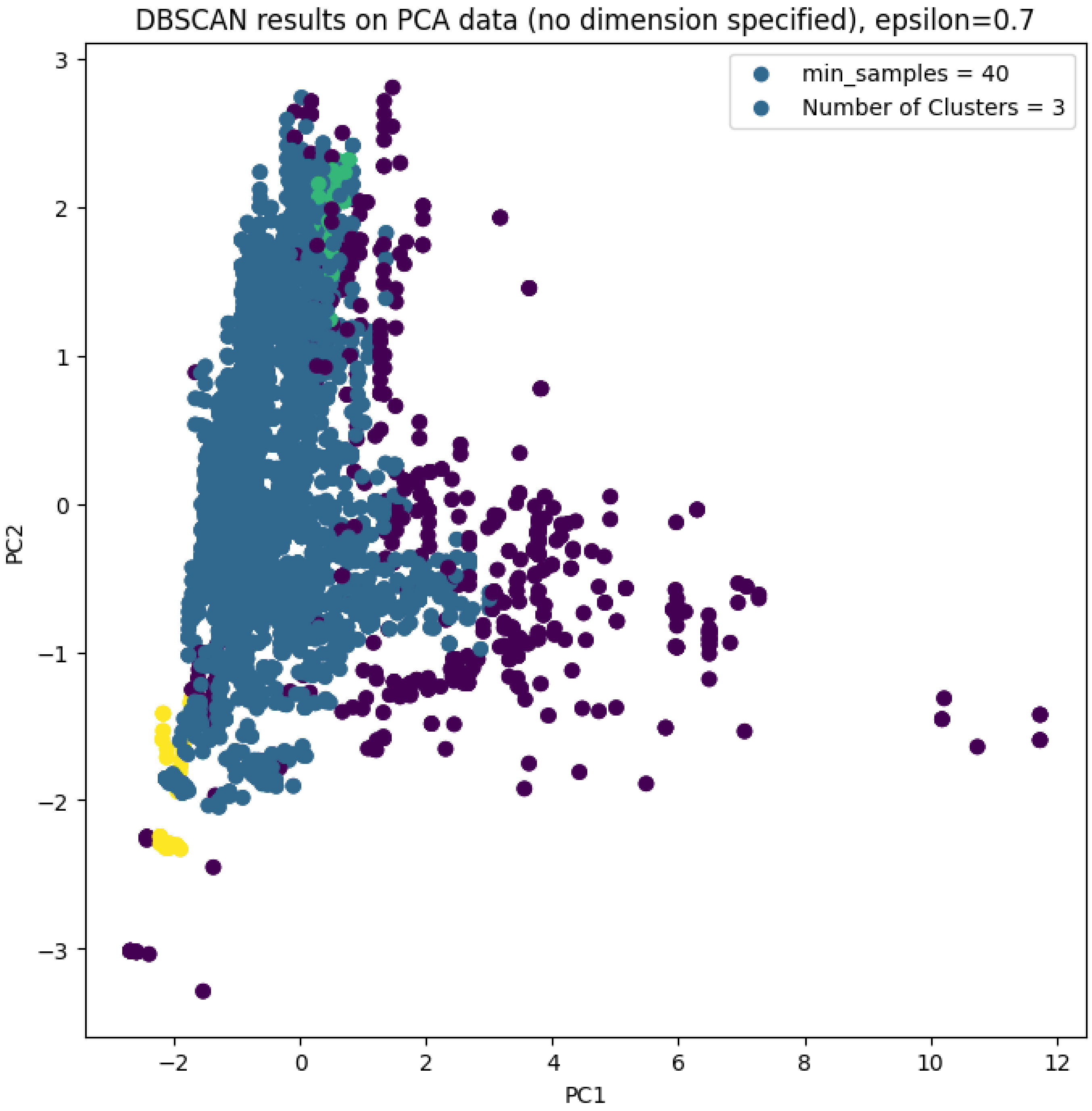

DBSCAN results on PCA data (no dimension specified), min_samples = 30
● epsilon = 0.70
● Number of Clusters = 4
PC2
3
2
1
0
−1
−2
−3
PC1
−2
0
2
4
6
8
10
12

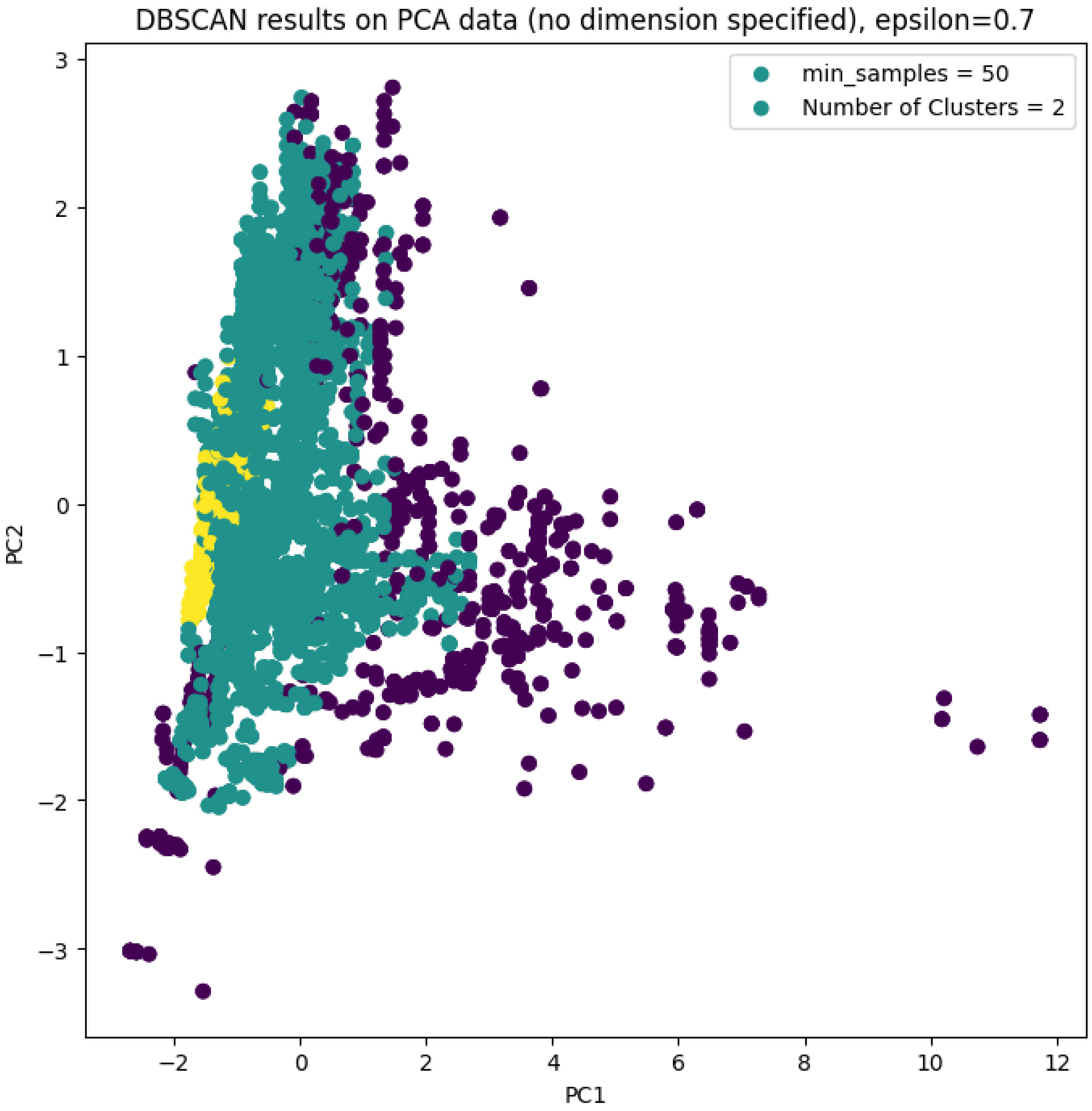

DBSCAN results on PCA data (no dimension specified), min_samples = 30
● epsilon = 0.90
● Number of Clusters = 2
PC1
PC2
−2
0
2
4
6
8
10
12
3
2
1
0
−1
−2
−3


DBSCAN results on PCA data (no dimension specified), epsilon=0.7
● min_samples = 10
● Number of Clusters = 11
3
2
1
0
−1
−2
−3
PC2
−2
0
2
4
6
8
10
12
PC1


DBSCAN results on PCA data (no dimension specified), epsilon=0.7
● min_samples = 20
● Number of Clusters = 6
PC2
3
2
1
0
−1
−2
−3
PC1
−2
0
2
4
6
8
10
12

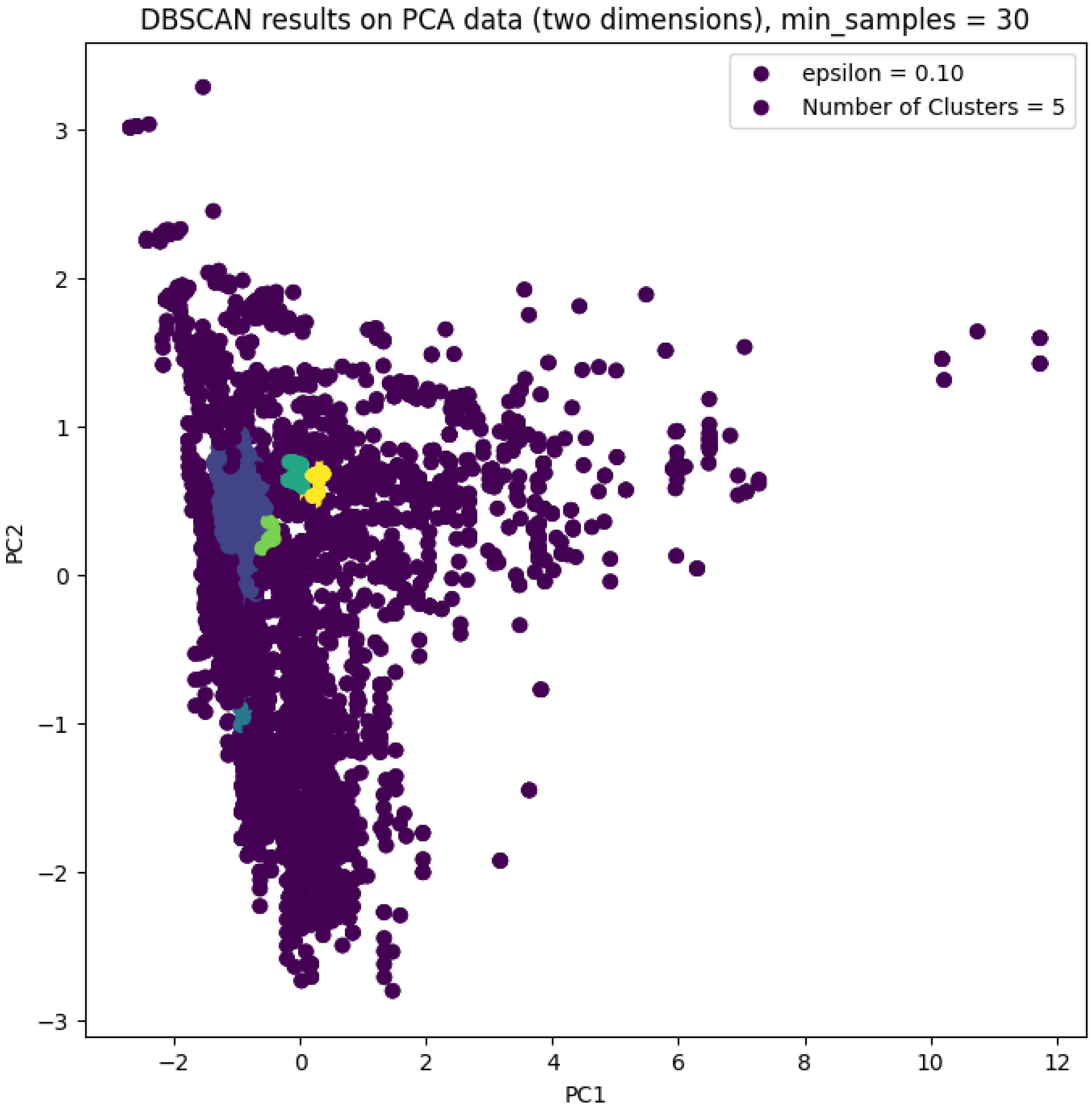

DBSCAN results on PCA data (no dimension specified), epsilon=0.7
● min_samples = 30
● Number of Clusters = 4
PC2
3
2
1
0
−1
−2
−3
PC1
−2
0
2
4
6
8
10
12

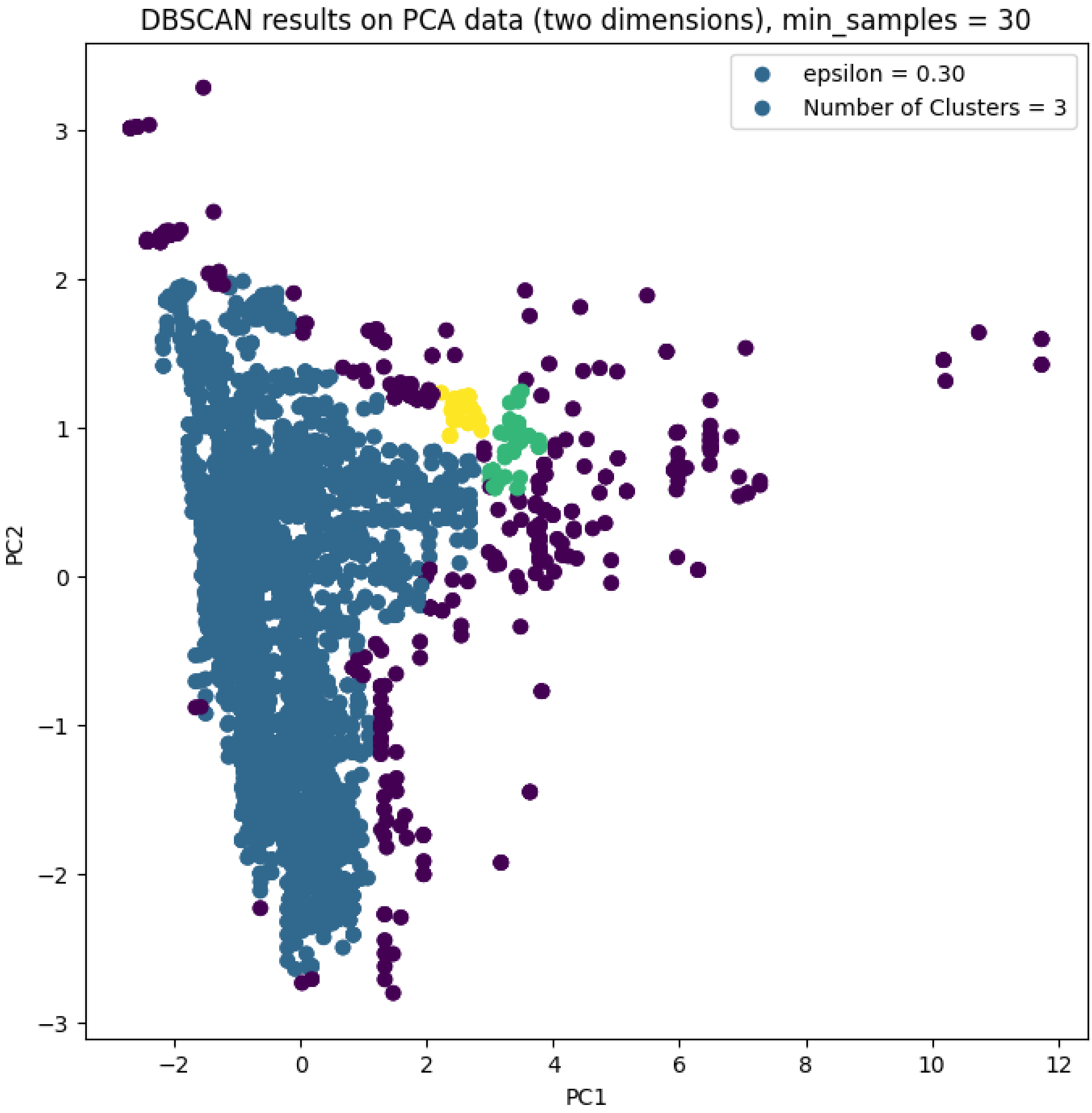

DBSCAN results on PCA data (no dimension specified), epsilon=0.7
min_samples = 40
Number of Clusters = 3
PC2
PC1
3
2
1
0
−1
−2
−3
−2
0
2
4
6
8
10
12

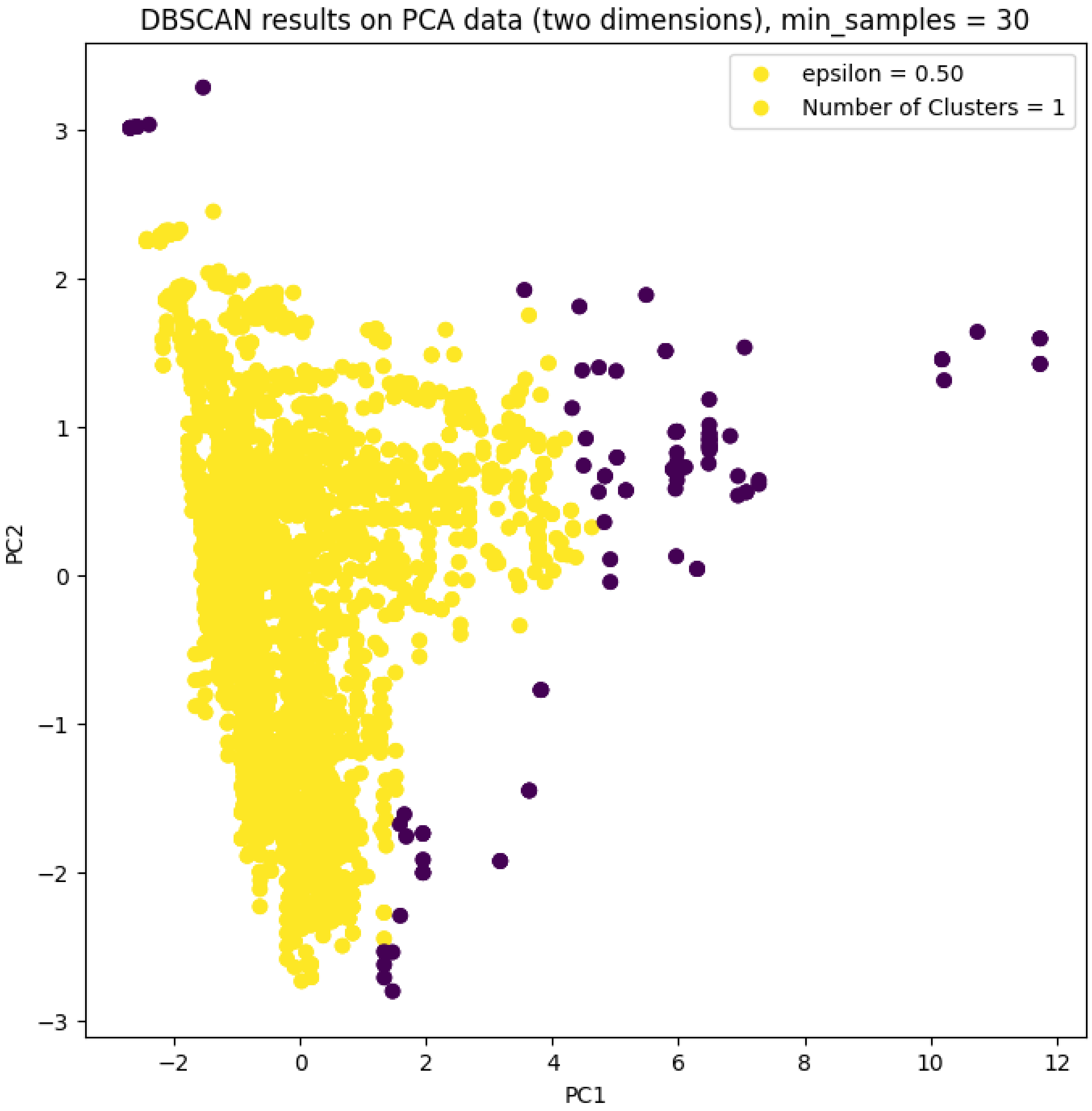


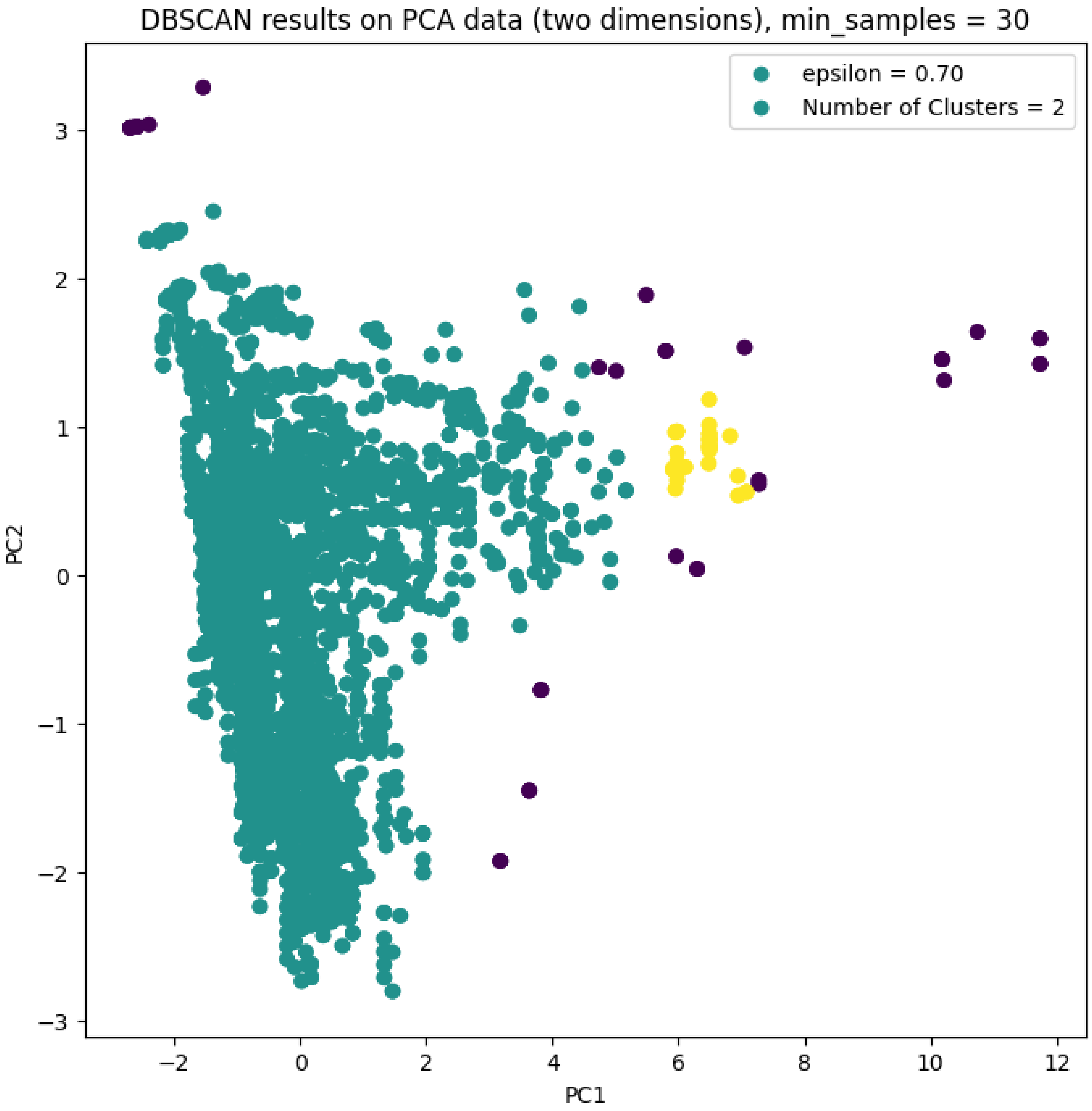
```
In [ ]: for i in np.arange(0.1,1,0.2):
            fig, ax = plt.subplots(1, 1, figsize=(8,8))
            dbs = DBSCAN(eps=i, min_samples = 30)
            dbs.fit(scores2)
            n_clusters = len(set(dbs.labels_)) - (1 if -1 in dbs.labels_ else 0)
            plt.scatter(scores2[:,0], scores2[:,1], c=dbs.labels_, label='epsilon = '+"{:.2f}".format(
            plt.scatter(scores2[:,0], scores2[:,1], c=dbs.labels_, label='Number of Clusters = '+str(n_
            plt.title('DBSCAN results on PCA data (two dimensions), min_samples = 30')
            plt.xlabel('PC1')
            plt.ylabel('PC2')
            leg=plt.legend()

        for i in range(10,60,10):
            fig, ax = plt.subplots(1, 1, figsize=(8,8))
```

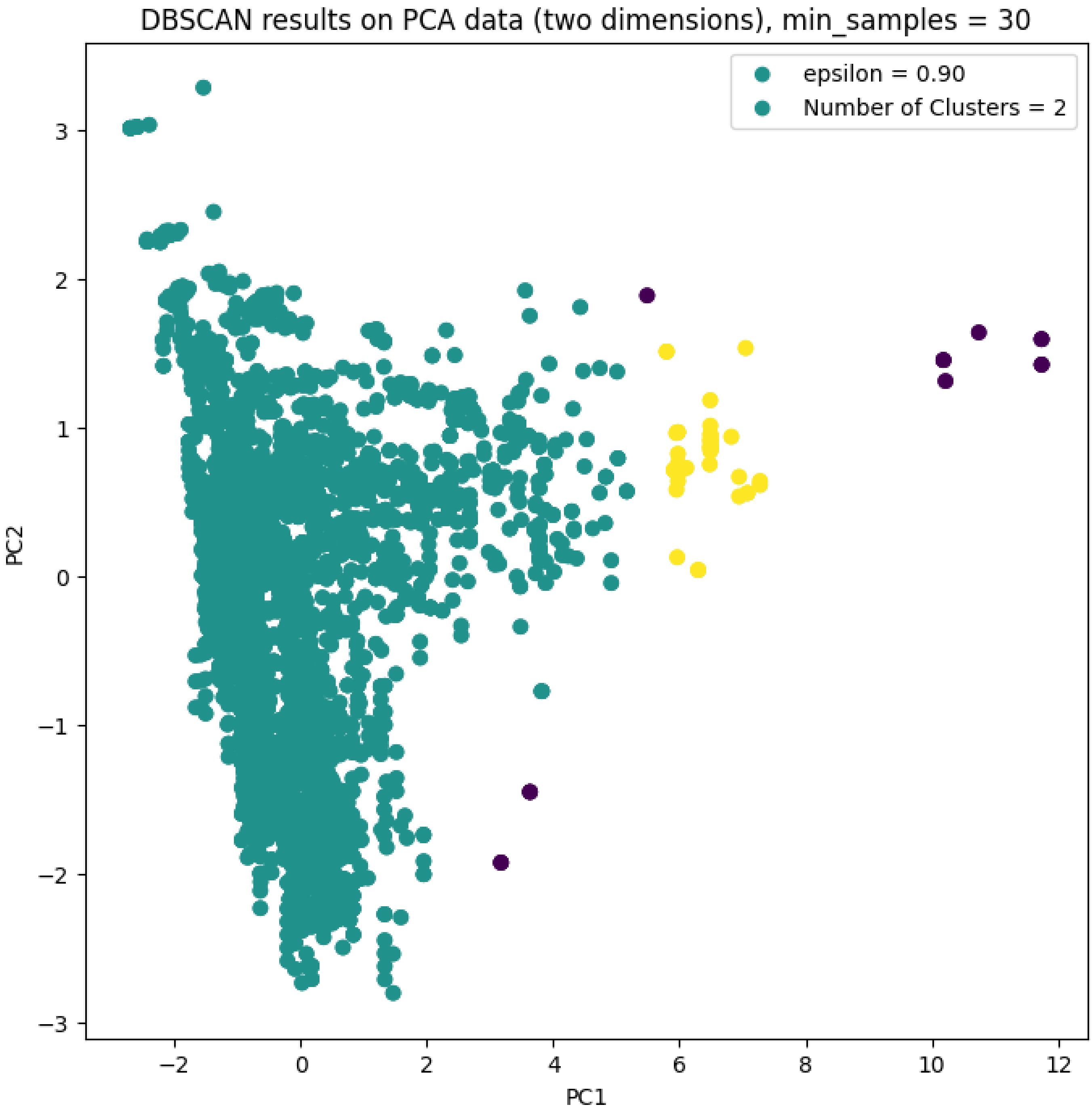
```
    dbs = DBSCAN(eps=0.7, min_samples = i)
    dbs.fit(scores2)
    n_clusters = len(set(dbs.labels_)) - (1 if -1 in dbs.labels_ else 0)
    plt.scatter(scores2[:,0], scores2[:,1], c=dbs.labels_, label='min_samples = '+str(i))
    plt.scatter(scores2[:,0], scores2[:,1], c=dbs.labels_, label='Number of Clusters = '+str(n_
    plt.title('DBSCAN results on PCA data (two dimensions), epsilon=0.7')
    plt.xlabel('PC1')
    plt.ylabel('PC2')
    leg=plt.legend()
```

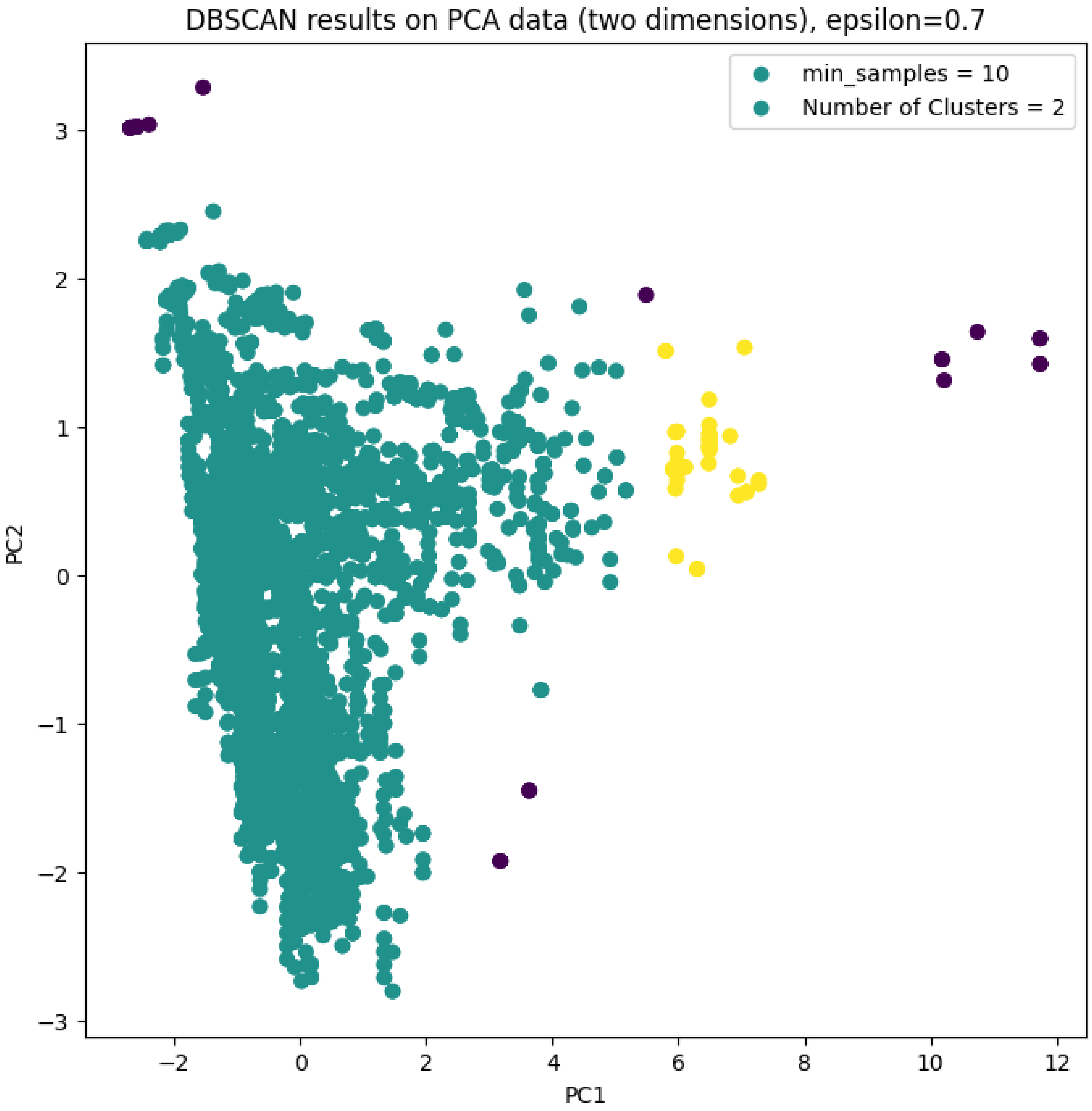


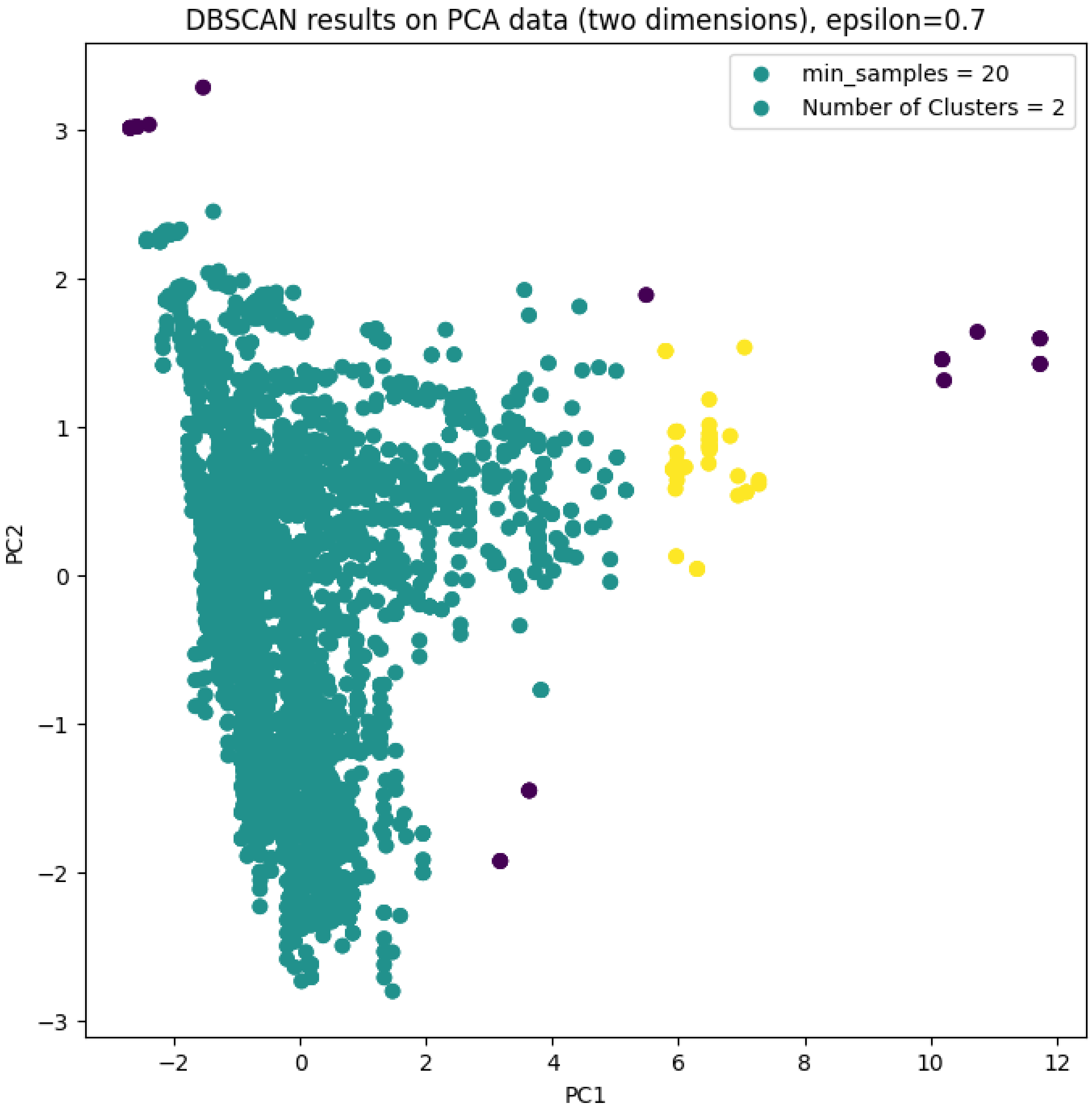

DBSCAN results on PCA data (two dimensions), min_samples = 30
● epsilon = 0.30
● Number of Clusters = 3
PC2
3
2
1
0
−1
−2
−3
PC1
−2
0
2
4
6
8
10
12

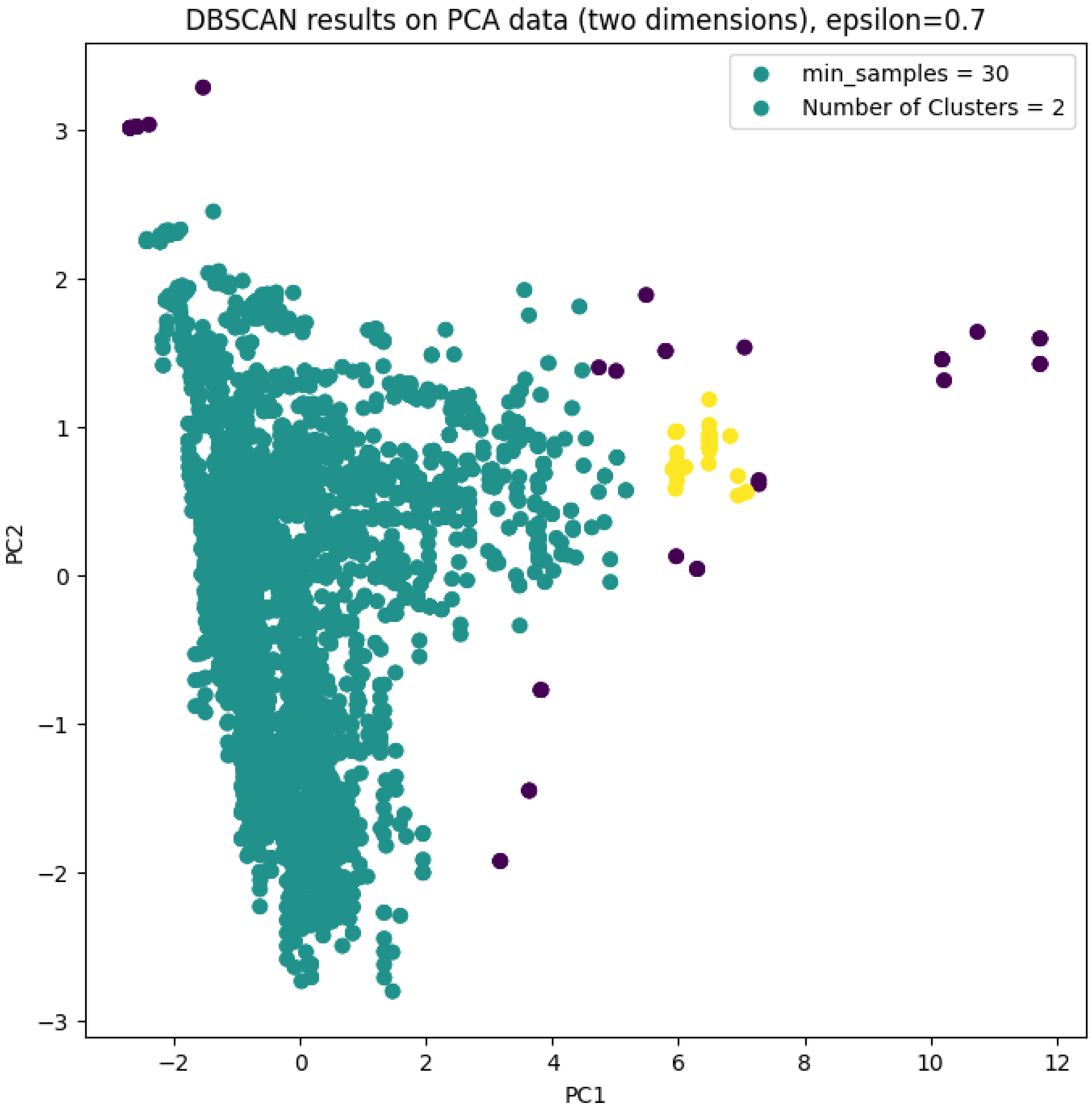

DBSCAN results on PCA data (two dimensions), min_samples = 30
● epsilon = 0.50
● Number of Clusters = 1
PC2
3
2
1
0
−1
−2
−3
PC1
−2
0
2
4
6
8
10
12

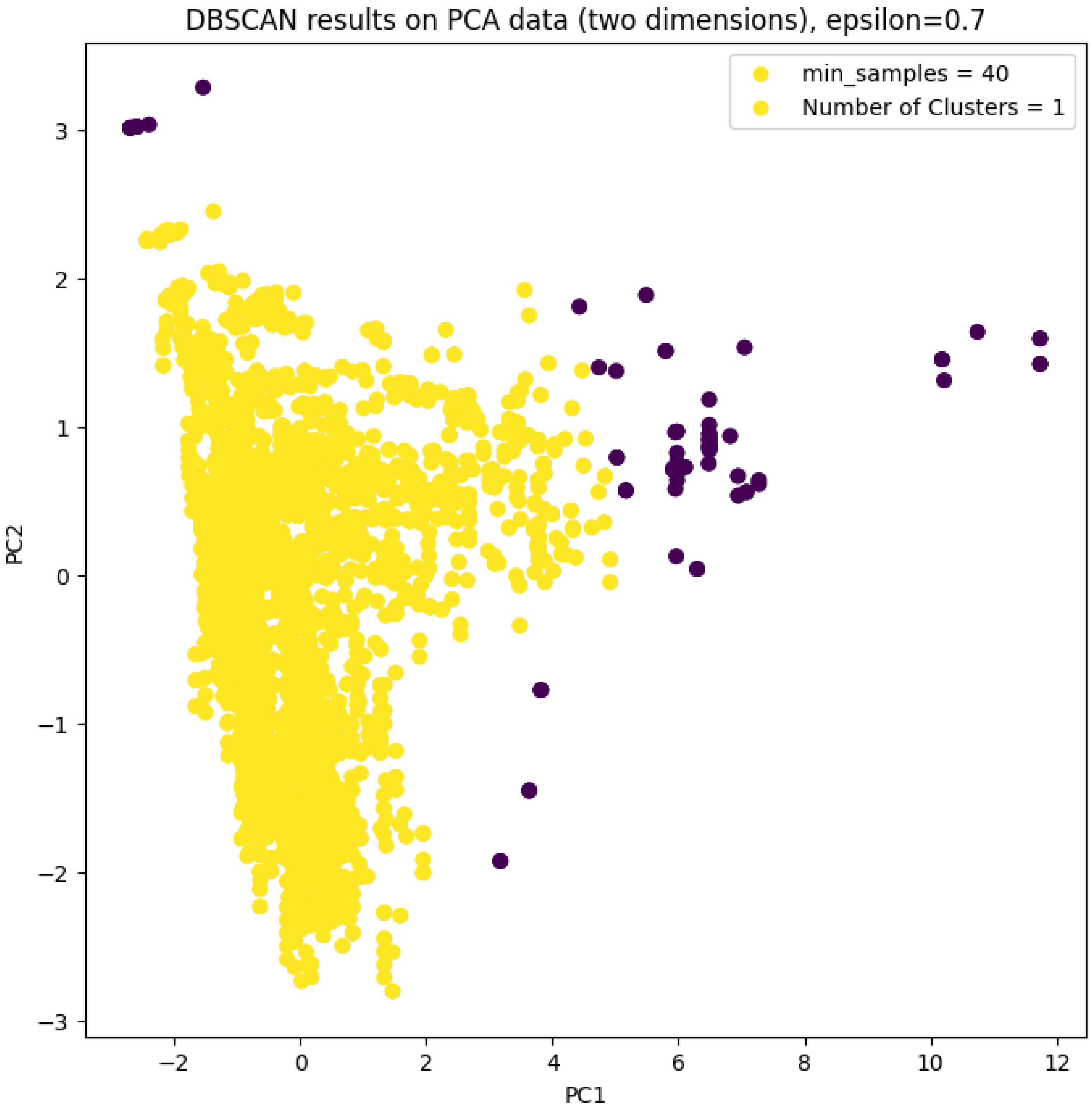

DBSCAN results on PCA data (two dimensions), min_samples = 30
● epsilon = 0.70
● Number of Clusters = 2
PC1
PC2
−2
0
2
4
6
8
10
12
−3
−2
−1
0
1
2
3

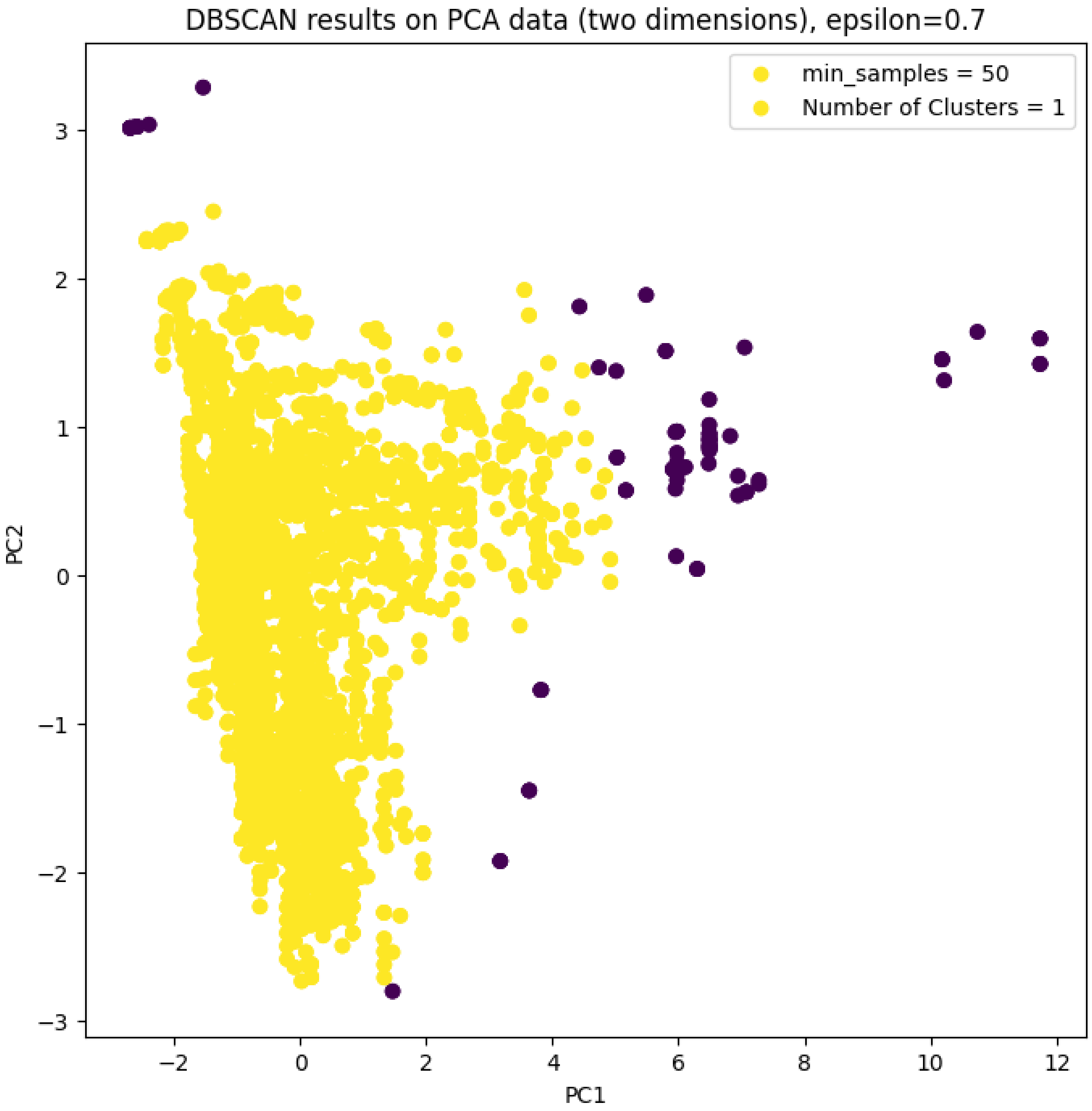

DBSCAN results on PCA data (two dimensions), min_samples = 30
● epsilon = 0.90
● Number of Clusters = 2
PC2
3
2
1
0
−1
−2
−3
PC1
−2
0
2
4
6
8
10
12


DBSCAN results on PCA data (two dimensions), epsilon=0.7
● min_samples = 10
● Number of Clusters = 2
PC2
PC1
3
2
1
0
−1
−2
−3
−2
0
2
4
6
8
10
12


DBSCAN results on PCA data (two dimensions), epsilon=0.7
● min_samples = 20
● Number of Clusters = 2
PC2
3
2
1
0
−1
−2
−3
PC1
−2
0
2
4
6
8
10
12


DBSCAN results on PCA data (two dimensions), epsilon=0.7
min_samples = 30
Number of Clusters = 2
PC1
PC2
−2
0
2
4
6
8
10
12
3
2
1
0
−1
−2
−3


DBSCAN results on PCA data (two dimensions), epsilon=0.7
● min_samples = 40
● Number of Clusters = 1
PC2
3
2
1
0
−1
−2
−3
PC1
−2
0
2
4
6
8
10
12



Works Cited

1. Sun, Yifan et al. "Summarizing CPU and GPU Design Trends with Product Data", July 13, 2020.