project3

April 19, 2022

CSCI 347

Project 3

Names: Moiyad Alfawwar, Michael Roduin, Philip Ghede

0.1 Problem 1: Think about the data

This data set comes from comes from the daily measures of sensors in an urban waste water treatment plant. The data helps us classify the operational state of the plant in order to predict faults through the state variables of the plant at each of the stages of the treatment process.

This multivariate data set is interesting because of the importance of predicting failures in the operational state of treatment plants and the likes through data analysis, as a means to avoid ecological catastrophes resulting from possible failures. We are also interested in determining how difficult it is to notice fluctuations in performance and output, in order to predict future failures and ensure quality of output.

This data set has 527 instances or observations and 38 Attributes. The site mentioned that there were no missing values. However, that was not our experience with the data. There are plenty of missing values. We used forward filling as well as backward filling. We used pandas Dataframe then converted the data back to numpy.

We can use many clustering techniques to evaluate this data including K-Means, and DBSCAN, and expect a minimum of 3 clusters, up to the 13, based on the class distribution described in data set description. Upon analysis, resulting clusters will allow us to determine deviation in performance in general and when influenced by varying conditions. Clustering is an unsupervised machine learning method used for grouping similar data points. With the help of clustering, we can classify data into structures that can be evaluated and manipulated easily. Given our data set, we might find it useful to identify clusters based on varying input, output, or performance to determine possible risk factors, etc. We expect to find many clusters of varying size and intensity depending on our attributes.

0.2 Part 2: Write Python code for clustering

[ ]:
```
# libraries
import numpy as np
from sklearn.datasets import make_blobs
from sklearn.cluster import DBSCAN
from sklearn.decomposition import PCA
```

```
from sklearn.datasets import load_boston
import math
import matplotlib.pyplot as plt
import warnings
from sklearn.cluster import KMeans
import pandas as pd
import random
```

1. (10 points) A function that implements the k-means clustering algorithm. The function should take a data matrix, a number of clusters k, and a convergence parameter ϵ, as input, and return the representatives (means) as well as the clusters found using k-means. If the distance is the same between a point and more than one representative (mean), then assign the point to the mean corresponding to the cluster with the lowest index.

[ ]:
```
#Get distance between 2 points
def distance(mean_point, distance_point):
    return math.sqrt(sum([(mean_point[index] - distance_point[index]) ** 2 for␣
 ↪index in range(len(mean_point))]))

#Create Random starting values for kmeans
def randomize(ranges, k):
    output = []
    while k > 0:
        output.append(tuple([random.uniform(ranges[column][0],␣
 ↪ranges[column][1]) for column in range(len(ranges))]))
        k -= 1
    return output

#Find mean based on index subset
def mean(matrix, indexes):
    array = list(zip(*[list(matrix[index, :]) for index in indexes]))
    array = [list(item) for item in array]
    return tuple([sum(index)/len(index) for index in array])

#Return index of closest point in cluster points to point(used to find closest␣
 ↪cluster)
def closest(cluster_points, point):
    distances = [distance(cluster_point, point) for cluster_point in␣
 ↪cluster_points]
    return distances.index(min(distances))

#Uses Kmeans Algorithm to return cluster points and matrix row indexes in each␣
 ↪cluster
def kmeans(matrix, k = 2, e = 10000):
    ranges = [(min(matrix[:,column]), max(matrix[:,column]))  for column in␣
 ↪range(len(matrix[0]))]
    start_points = []
```

```
    end_points = randomize(ranges, k)
    clusters = []
    while start_points != end_points and e > 0:
        start_points = end_points
        clusters = [[] for index in range(k)]
        for row in range(len(matrix)):
            clusters[closest(start_points, tuple(matrix[row]))].append(row)
        end_points = [mean(matrix, item) for item in clusters]
        e -= 1
    return end_points, clusters
```

2. (10 points) A function that implements the DBSCAN clustering algorithm. The function should take a data matrix and the parameters minpts and ϵ, as input, and return the clusters found using DBSCAN, and for each data point a label of core, border, or noise point.

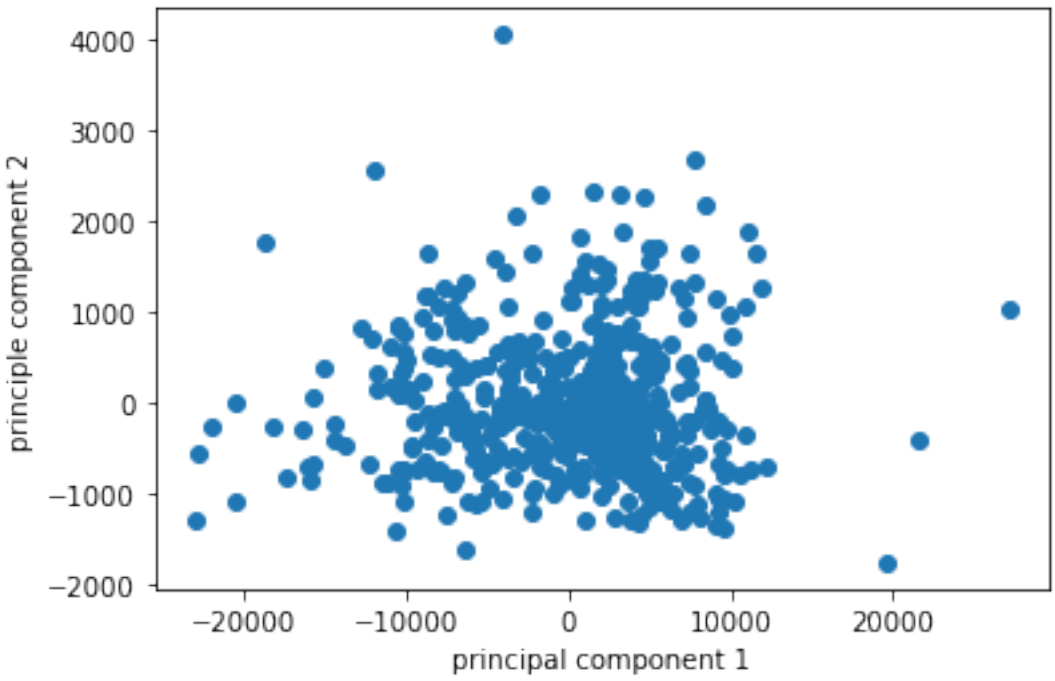
[ ]:
```
#Get distance between 2 points
def distance(mean_point, distance_point):
    return math.sqrt(sum([(mean_point[index] - distance_point[index]) ** 2 for␣
 ↪index in range(len(mean_point))]))

#Return if in neighborhood
def within_neighborhood(point_1, point_2, max_distance):
    return distance(point_1, point_2) <= max_distance

class Point():
    #Point is a tuple of point values
    def __init__(self, point):
        self.point = point
        self.label =  "noise"
        self.cluster = []

    def get_cluster(self, cluster = [], loop = 0):
        length = len(cluster)
        new = [self] + self.cluster
        cluster = cluster + new
        cluster = list(set(cluster))
        if len(cluster) == length:
            return cluster
        for item in new:
            cluster = item.get_cluster(cluster = cluster, loop = loop+1)
        return cluster


def dbscan(matrix, minpts = 5, e = 10000):
    points = [Point(tuple(matrix[row,:])) for row in range(len(matrix))]
    for item in points:
        for other in points:
```

```
            if within_neighborhood(item.point, other.point, minpts):
                item.cluster.append(other)
                other.cluster.append(item)
    for item in points:
        item.cluster = list(set(item.cluster))
        if len(item.cluster) >= minpts:
            item.label = "core"
        else:
            for other in item.cluster:
                if other.label == "core":
                    item.label = "border"
                    break
    output =  list(set([tuple(point.get_cluster()) for point in points]))
    return [list(item) for item in output]
```

0.3 Part 3: Analyze your data

1. (4 points) Use sklearn's PCA implementation to linearly transform the data to two dimensions. Create a scatter plot of the data, with the x-axis corresponding to coordinates of the data along the first principal component, and the y-axis corresponding to coordinates of the data along the second principal component. Does it look like there are clusters in these two dimensions? If so, how many would you say there are?

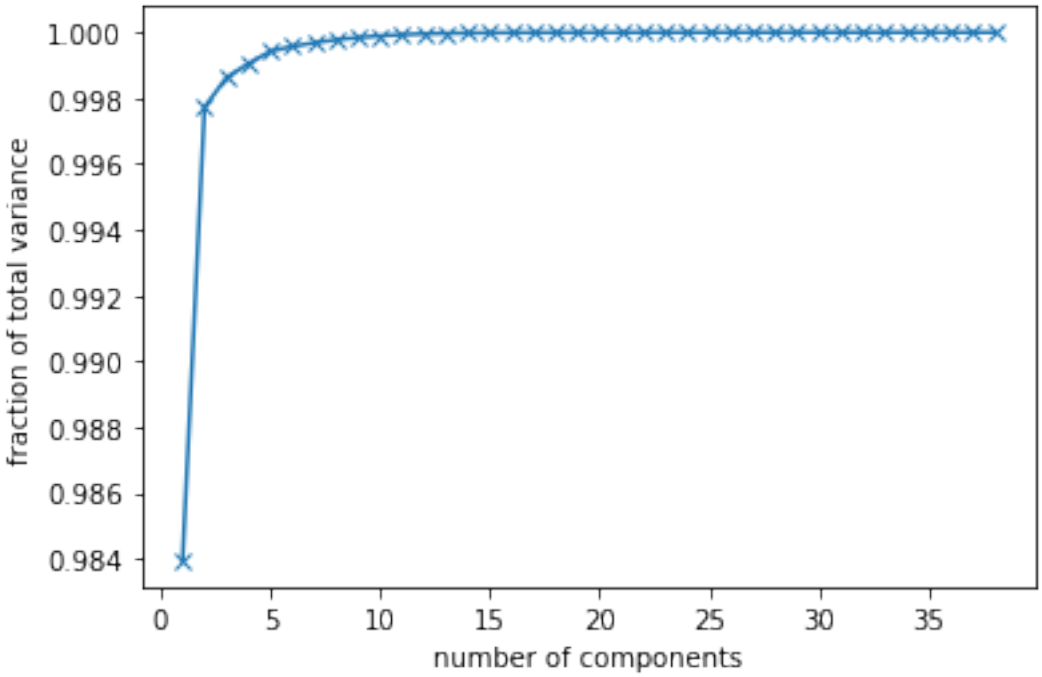
[ ]:
```
Data = np.genfromtxt('data/water-treatment.data', delimiter=",")
pca = PCA(n_components=2)
Data2 = np.delete(Data, [0],1)
D3 = pd.DataFrame(Data2)
D3 = D3.fillna(method='ffill')
D3 = D3.fillna(method='bfill')
D3.isnull().sum()
D3 = D3.to_numpy()
# D3 = Data2[:, ~np.isnan(Data2).any(axis=0)]
pca_data = pca.fit_transform(D3)
# Data2
pca_data
plt.scatter(pca_data[:,0],pca_data[:,1])
plt.xlabel('principal component 1')
plt.ylabel('principle component 2')
plt.show()
```



It doesn't look like there are clusters. But if there is it would be 2 at most.

2. (3 points) Use sklearn's PCA implementation to linearly transform the data, without specifying the number of components to use. Create a plot with r, the number of components (i.e., dimensionality), on the x-axis, and f (r), the fraction of total variance captured in the first r principal components, on the y-axis. Based on this plot, choose a number of principal components to reduce the dimensionality of the data. Report how many principal components will be used as well as the faction of total variance captured using this many components.

[ ]:
```
pca2 = PCA()
pca_data2 = pca2.fit_transform(D3)

explained_variance=pca2.explained_variance_ratio_[0]+pca2.
  ↪explained_variance_ratio_[1]
plt.plot(range(1,39),np.cumsum(pca2.explained_variance_ratio_), marker='x')
plt.xlabel('number of components')
plt.ylabel('fraction of total variance')
plt.show()

print("The explained variance using two principal components: ",␣
  ↪explained_variance)
```




```
The explained variance using two principal components:  0.9977245150851312
```

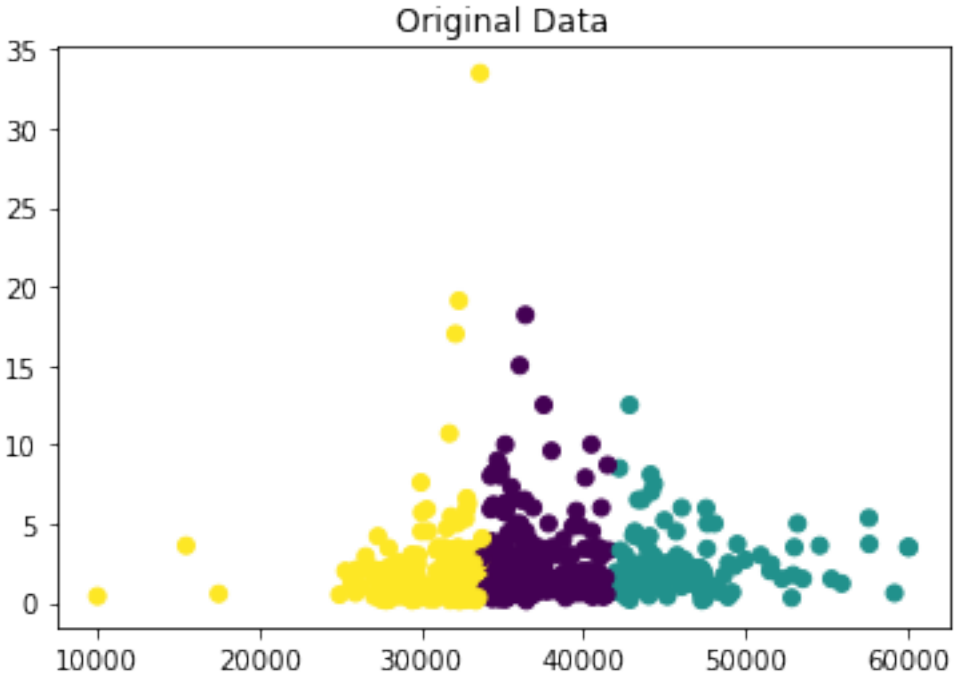
3. (5 points) For both the original and the reduced-dimensionality data obtained using PCA in question 3.2, do the following: Experiment with a range of values for the number of clusters, k, that you pass as input to the k-means function, to find clusters in the chosen data set. Use at least 5 different values of k. For each value of k, report the value of the objective function for that choice of k.

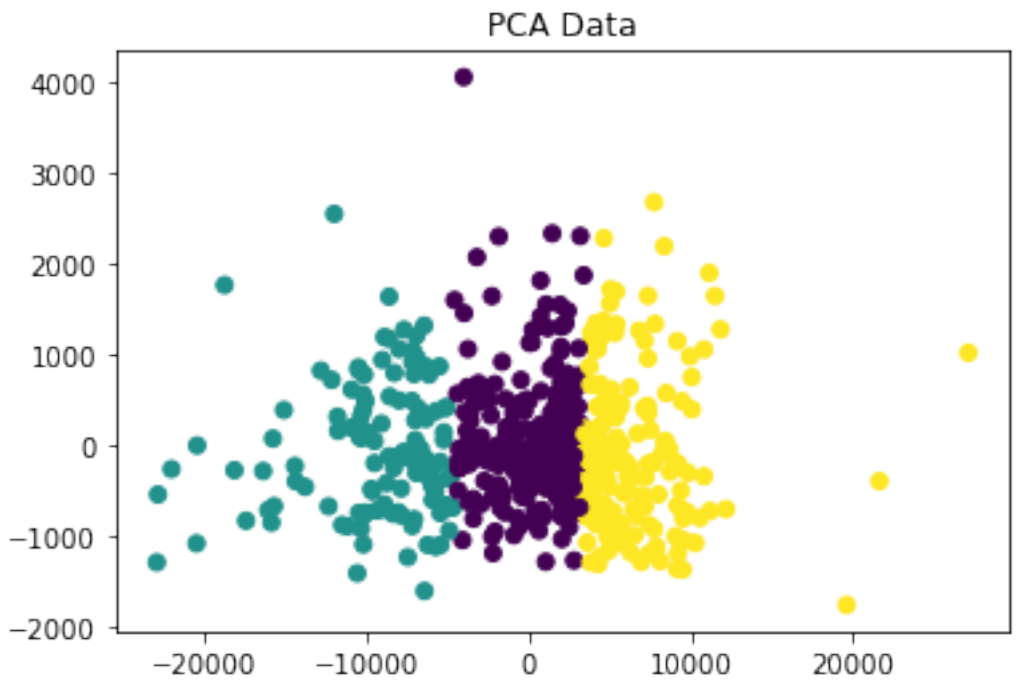
[ ]:
```
kmeans1 = KMeans(n_clusters=3, init='k-means++', max_iter=300, random_state=0)
kmeans2 = KMeans(n_clusters=6, init='k-means++', max_iter=300, random_state=0)
kmeans3 = KMeans(n_clusters=8, init='k-means++', max_iter=300, random_state=0)
kmeans4 = KMeans(n_clusters=10, init='k-means++', max_iter=300, random_state=0)
kmeans5 = KMeans(n_clusters=15, init='k-means++', max_iter=300, random_state=0)


pred_labels1 = kmeans1.fit_predict(D3)
pred_labels_pca1 = kmeans1.fit_predict(pca_data2)
pred_labels2 = kmeans2.fit_predict(D3)
pred_labels_pca2 = kmeans2.fit_predict(pca_data2)
pred_labels3 = kmeans3.fit_predict(D3)
pred_labels_pca3 = kmeans3.fit_predict(pca_data2)
pred_labels4 = kmeans4.fit_predict(D3)
pred_labels_pca4 = kmeans4.fit_predict(pca_data2)
pred_labels5 = kmeans5.fit_predict(D3)
pred_labels_pca5 = kmeans5.fit_predict(pca_data2)
pred_labels1[:300]
```

```
pred_labels_pca1[:100]

pred_labels_pca5[:100]


print("Objective function result of 3 clusters: ", kmeans1.inertia_)
print("Objective function result of 6 clusters: ", kmeans2.inertia_)
print("Objective function result of 8 clusters: ", kmeans3.inertia_)
print("Objective function result of 10 clusters: ", kmeans4.inertia_)
print("Objective function result of 15 clusters: ", kmeans5.inertia_)
```

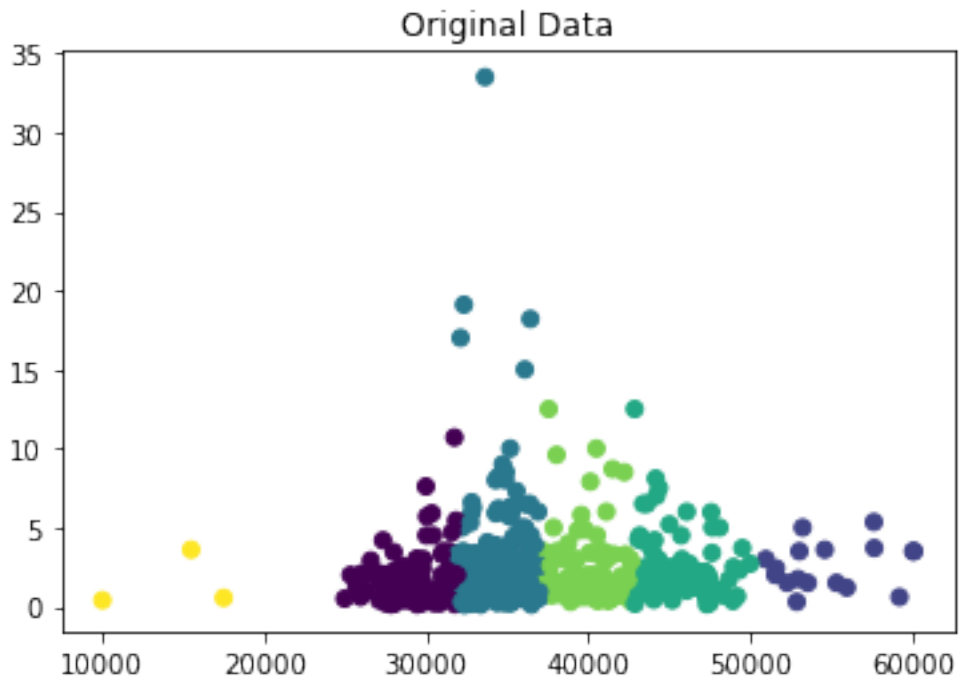
```
Objective function result of 3 clusters:  5236701736.875474
Objective function result of 6 clusters:  1831174016.14705
Objective function result of 8 clusters:  1131607953.0024855
Objective function result of 10 clusters:  849288910.5025187
Objective function result of 15 clusters:  559585010.4365393
```

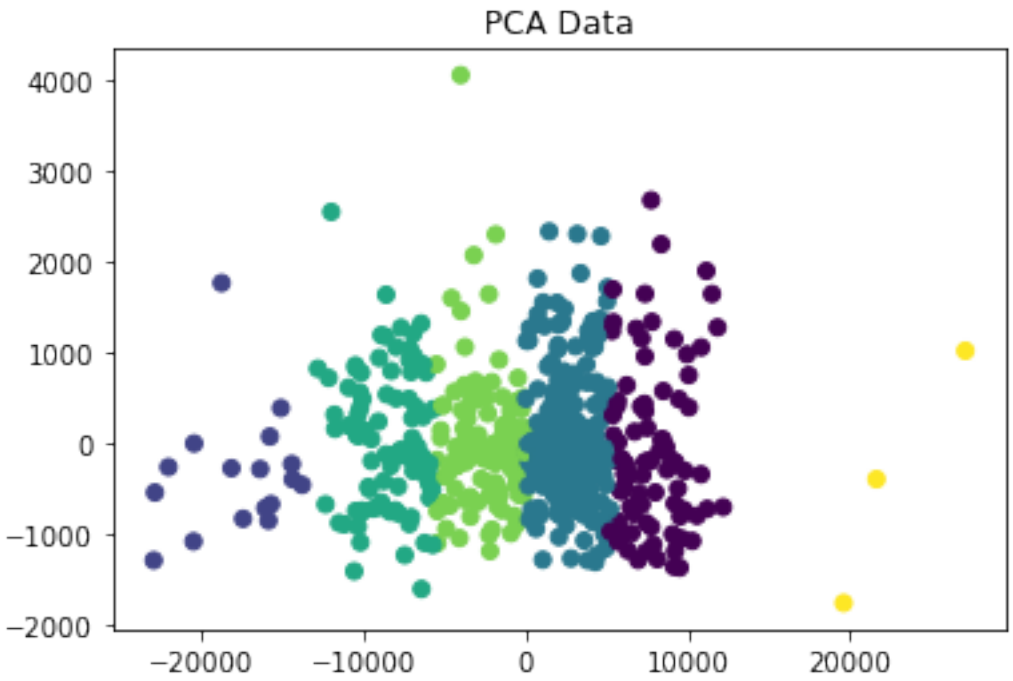
[ ]:
```
# 3 clusters
plt.scatter(D3[:,0], D3[:,1], c=pred_labels1)
plt.title("Original Data")
plt.show()
plt.scatter(pca_data[:,0],pca_data[:,1], c=pred_labels_pca1)
plt.title("PCA Data")
plt.show()
```





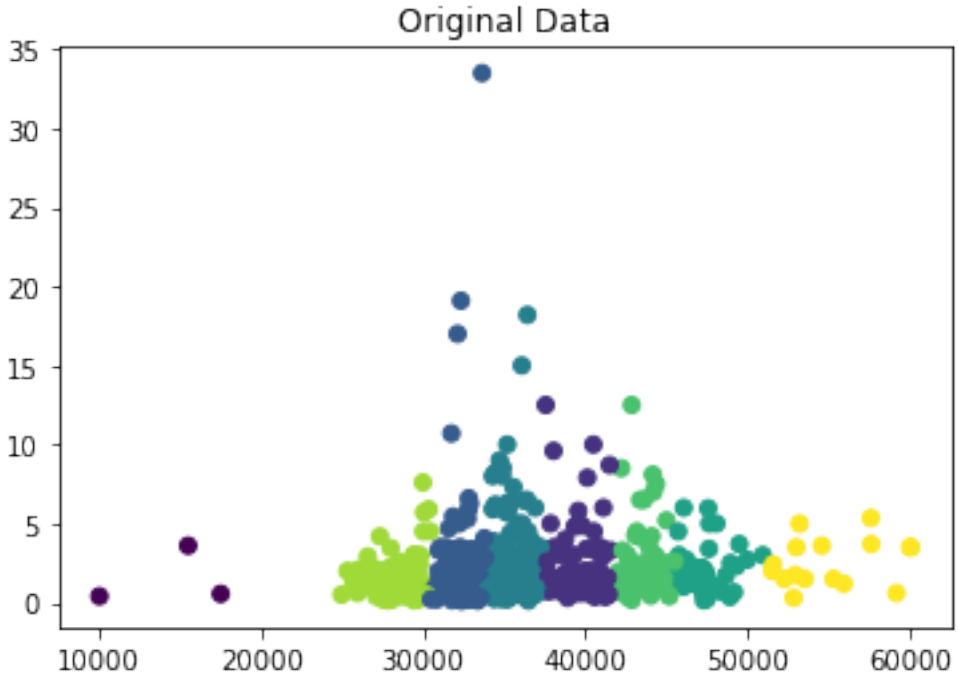
[ ]:
```
# 6 clusters
plt.scatter(D3[:,0], D3[:,1], c=pred_labels2)
plt.title("Original Data")
plt.show()
plt.scatter(pca_data[:,0],pca_data[:,1], c=pred_labels_pca2)
plt.title("PCA Data")
plt.show()
```

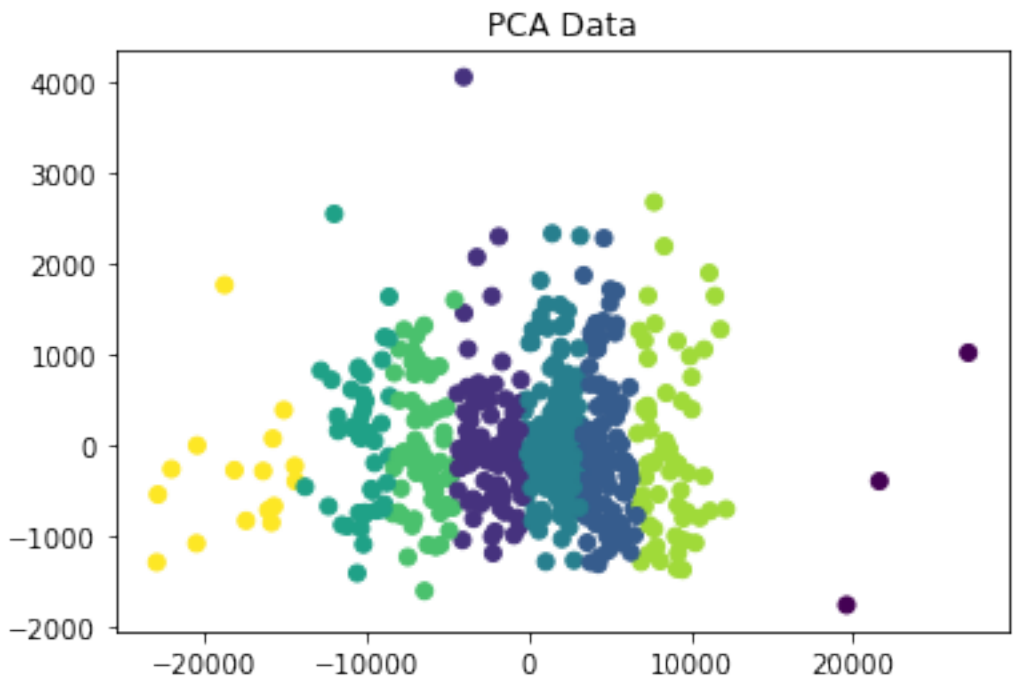




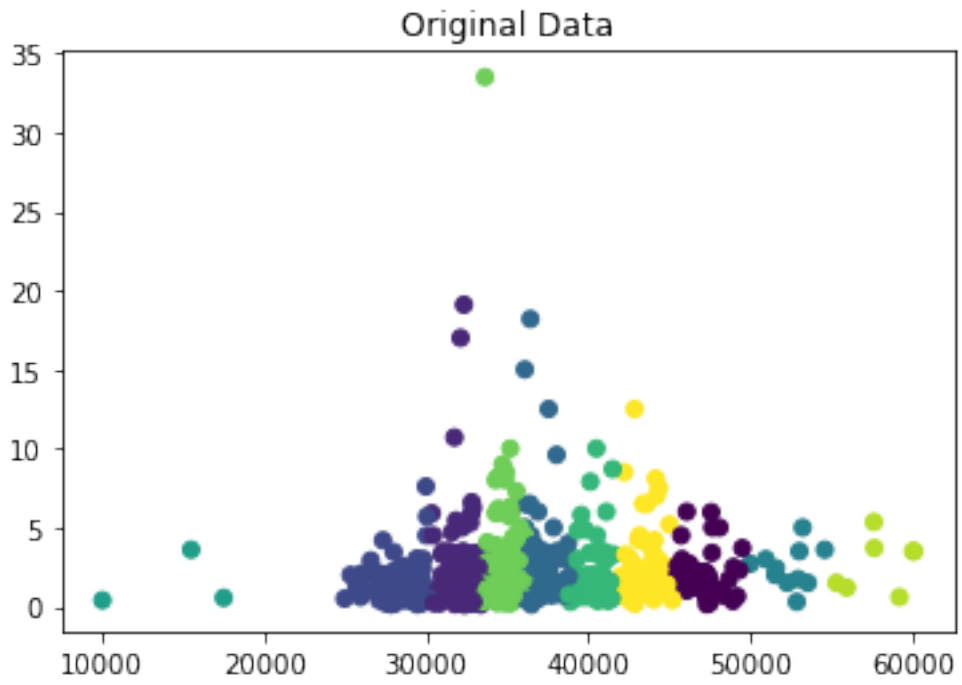
[ ]:
```
# 8 clusters
plt.scatter(D3[:,0], D3[:,1], c=pred_labels3)
```

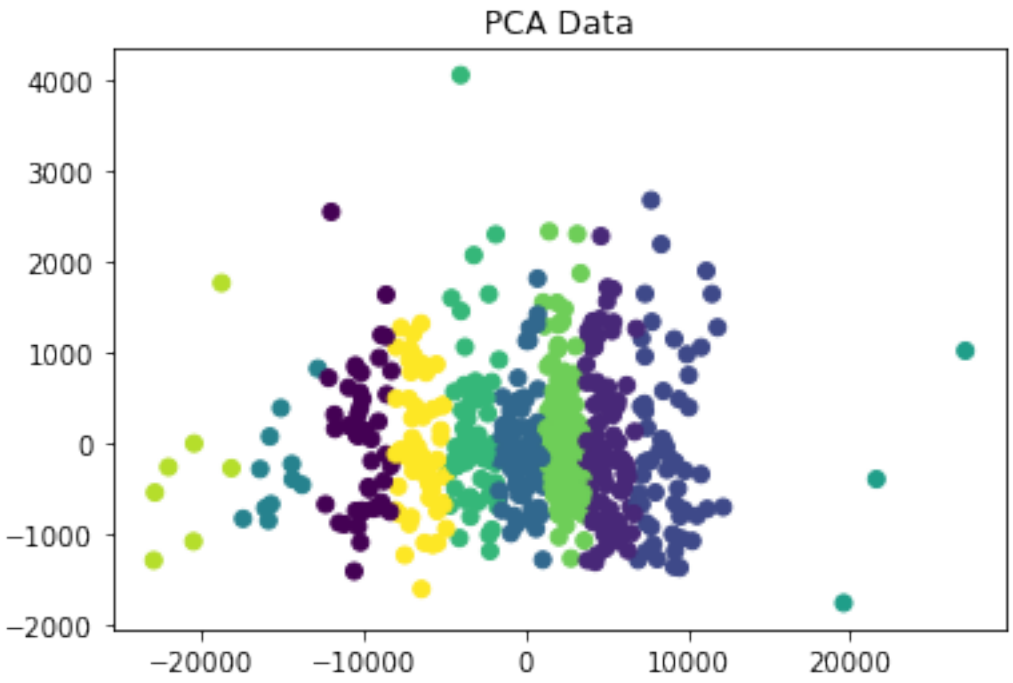
```
plt.title("Original Data")
plt.show()
plt.scatter(pca_data[:,0],pca_data[:,1], c=pred_labels_pca3)
plt.title("PCA Data")
plt.show()
```





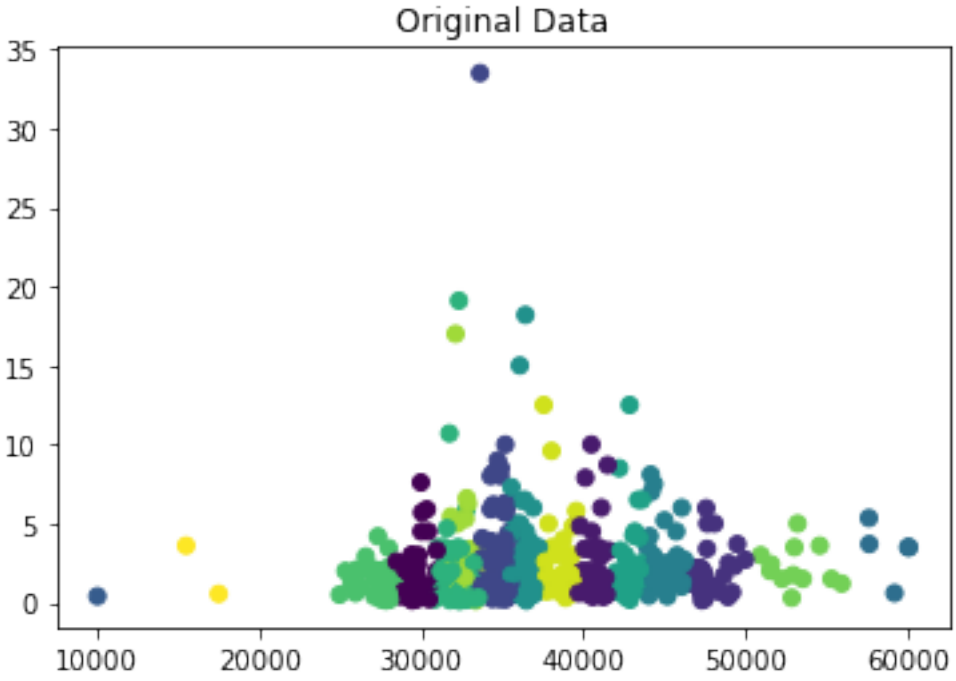
[ ]:
```
# 10 clusters
plt.scatter(D3[:,0], D3[:,1], c=pred_labels4)
plt.title("Original Data")
plt.show()
plt.scatter(pca_data[:,0],pca_data[:,1], c=pred_labels_pca4)
plt.title("PCA Data")
plt.show()
```

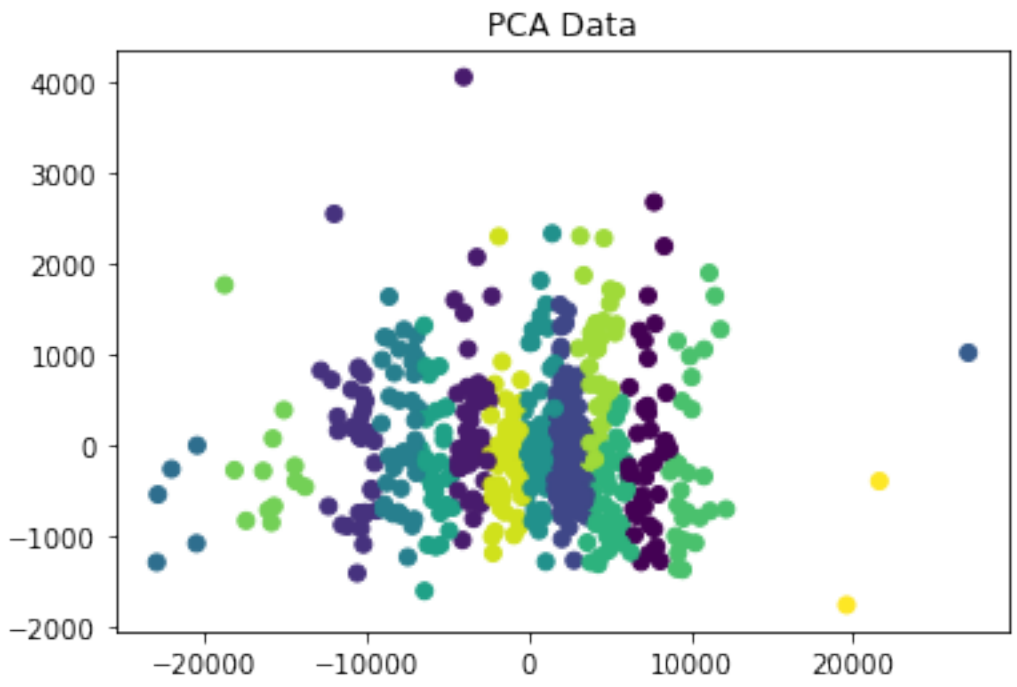




[ ]:
```
# 15 clusters
plt.scatter(D3[:,0], D3[:,1], c=pred_labels5)
```

```
plt.title("Original Data")
plt.show()
plt.scatter(pca_data[:,0],pca_data[:,1], c=pred_labels_pca5)
plt.title("PCA Data")
plt.show()
```





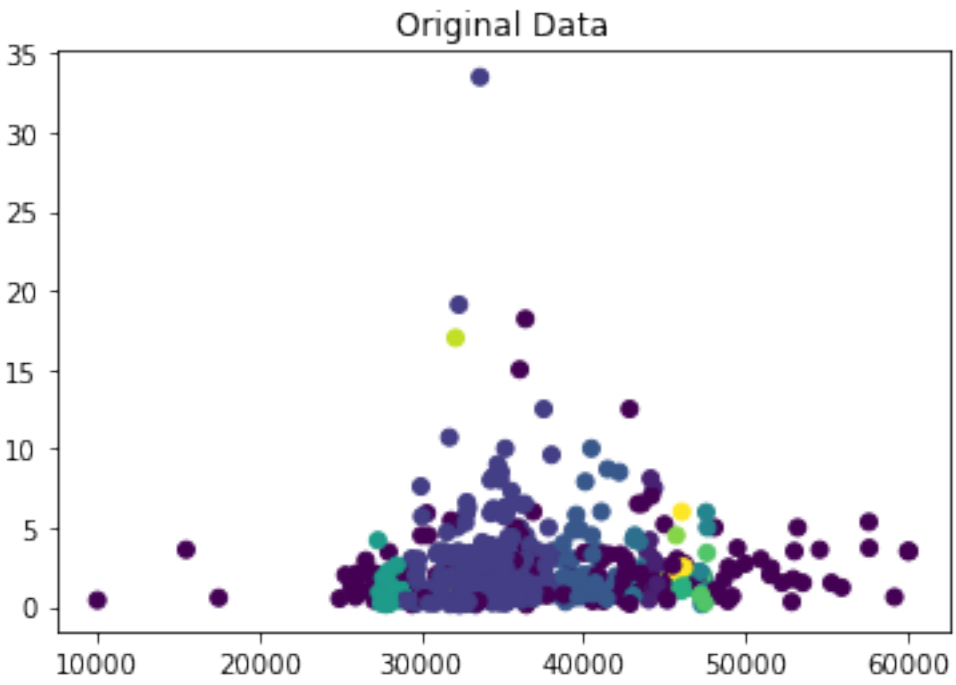
4. (5 points) For both the original and the reduced-dimensionality data obtained using PCA in question 3.2, do the following: Experiment with a range of values for the minpts and ϵ input parameters to the DBSCAN function to find clusters in the chosen data set. First, keep ϵ fixed and try out a range of different values for minpts. Then keep minpts fixed, and try a range of values for ϵ. Use at least 5 values of ϵ and at least 5 values of minpts. Report the number of clusters found for each (minpts, ϵ) pair tested.

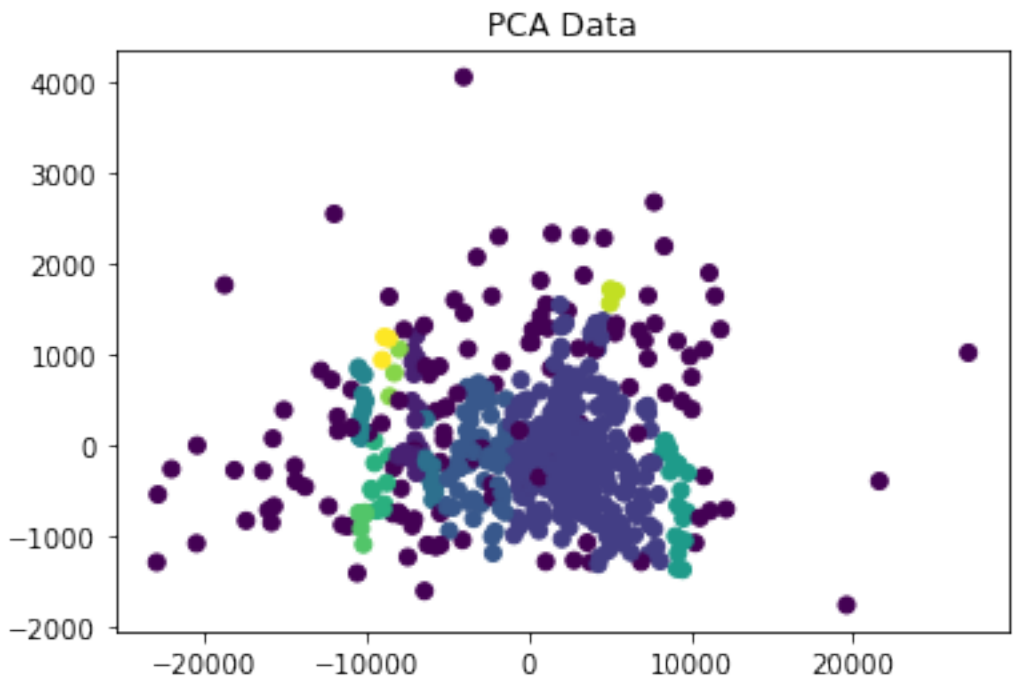
[ ]:
```
# Minsamples
dbs1 = DBSCAN(eps=500, min_samples=3)
dbs2 = DBSCAN(eps=500, min_samples=4)
dbs3 = DBSCAN(eps=500, min_samples=5)
dbs4 = DBSCAN(eps=500, min_samples=6)
dbs5 = DBSCAN(eps=500, min_samples=7)


# Eps
dbs_e1 = DBSCAN(eps=275, min_samples=5)
dbs_e2 = DBSCAN(eps=299, min_samples=5)
dbs_e3 = DBSCAN(eps=300, min_samples=5)
dbs_e4 = DBSCAN(eps=425, min_samples=5)
dbs_e5 = DBSCAN(eps=500, min_samples=5)
```

-

[ ]:
```
# dbs_e1 = DBSCAN(eps=290, min_samples=5)
# Min samples 3
dbs1 = DBSCAN(eps=500, min_samples=3)
dbs1_labels = dbs1.fit_predict(D3)
print("Original Data's Number of Clusters at min_samples=3:" ,␣
 ↪max(dbs1_labels)+1)
pca1_labels = dbs1.fit_predict(pca_data2)
print("PCA Data's Number of Clusters at min_samples=3:" , max(pca1_labels)+1)

plt.scatter(D3[:,0], D3[:,1],c=dbs1_labels)
plt.title("Original Data")
plt.show()

plt.scatter(pca_data2[:,0], pca_data2[:,1],c=pca1_labels)
plt.title("PCA Data")
plt.show()
```

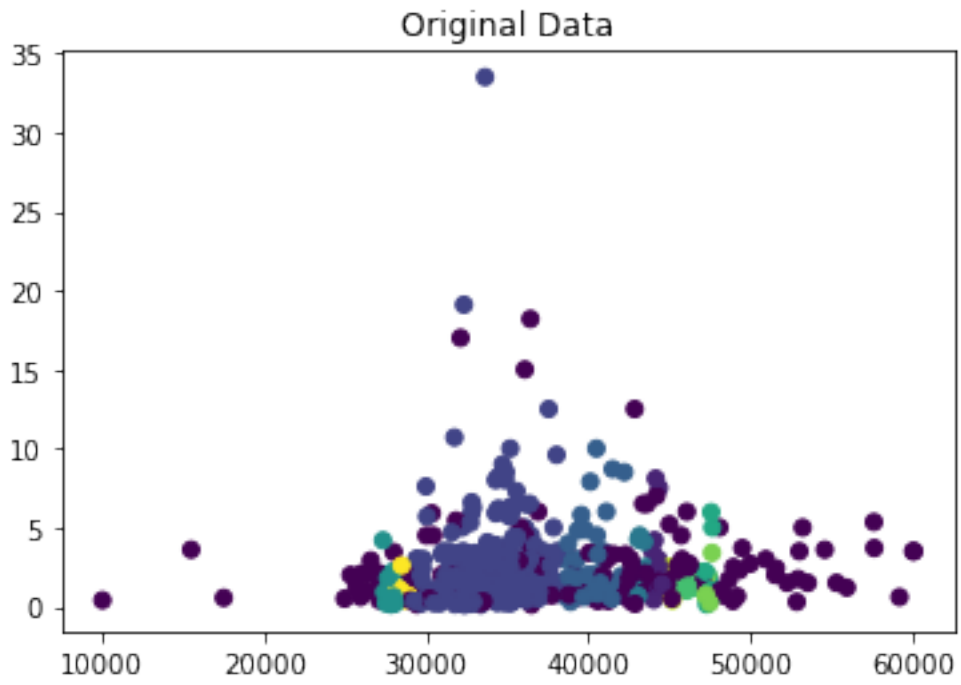
```
Original Data's Number of Clusters at min_samples=3: 11
PCA Data's Number of Clusters at min_samples=3: 11
```

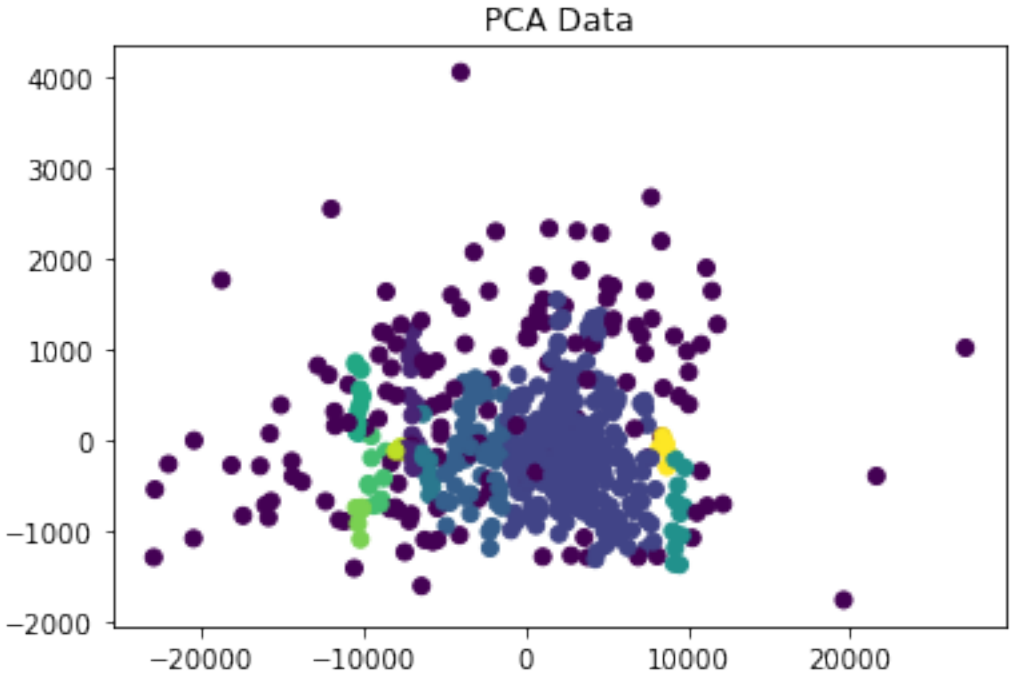




[ ]:
```
dbs2 = DBSCAN(eps=500, min_samples=4)
dbs2_labels = dbs2.fit_predict(D3)
print("Original Data's Number of Clusters at min_samples=4:" ,␣
 ↪max(dbs2_labels)+1)
pca2_labels = dbs2.fit_predict(pca_data2)
print("PCA Data's Number of Clusters at min_samples=4:" , max(pca2_labels)+1)

plt.scatter(D3[:,0], D3[:,1],c=dbs2_labels)
plt.title("Original Data")
plt.show()

plt.scatter(pca_data2[:,0], pca_data2[:,1],c=pca2_labels)
plt.title("PCA Data")
plt.show()
```

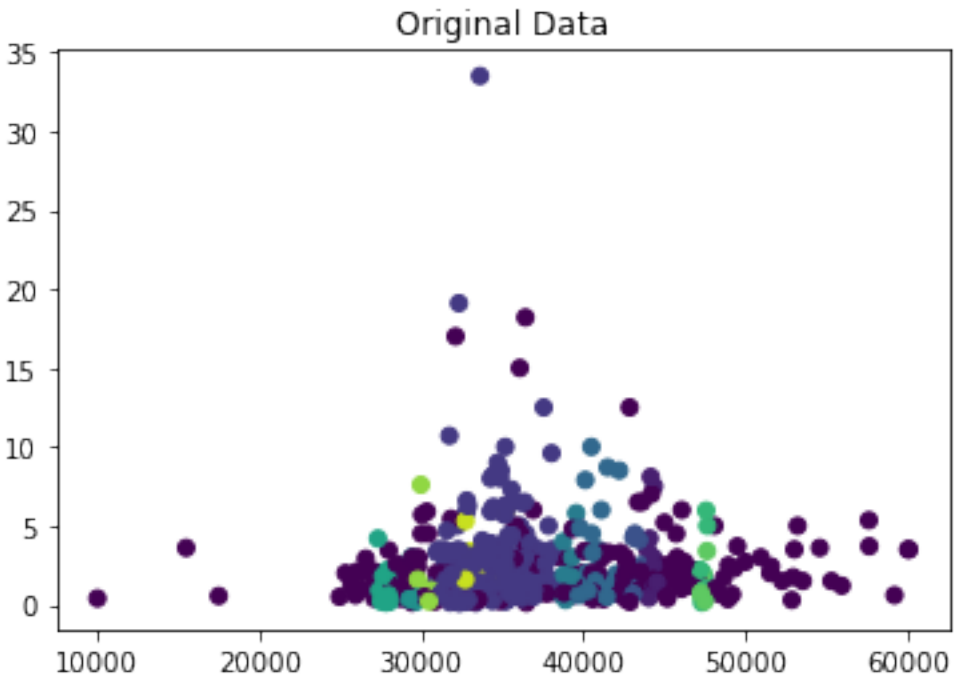
```
Original Data's Number of Clusters at min_samples=4: 10
PCA Data's Number of Clusters at min_samples=4: 10
```

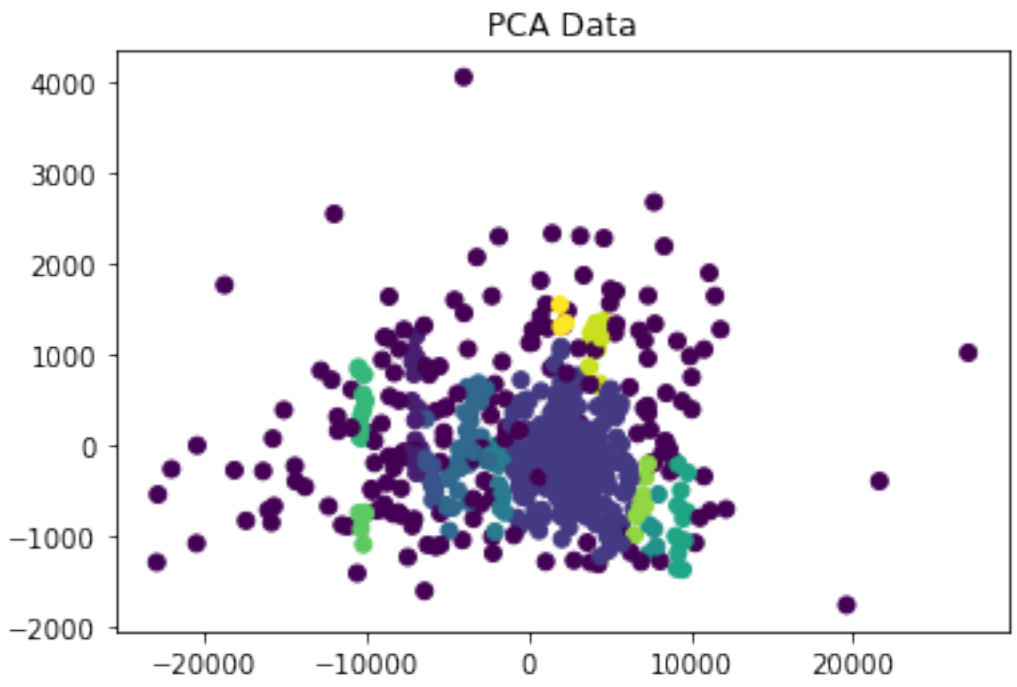

Original Data
35
30
25
20
15
10
5
0
10000
20000
30000
40000
50000
60000


PCA Data
4000
3000
2000
1000
0
−1000
−2000
−20000
−10000
0
10000
20000

[ ]:
```
dbs3 = DBSCAN(eps=500, min_samples=5)
dbs3_labels = dbs3.fit_predict(D3)
print("Original Data's Number of Clusters at min_samples=5:" ,␣
 ↪max(dbs3_labels)+1)
pca3_labels = dbs3.fit_predict(pca_data2)
print("PCA Data's Number of Clusters at min_samples=5:" , max(pca3_labels)+1)

plt.scatter(D3[:,0], D3[:,1],c=dbs3_labels)
plt.title("Original Data")
plt.show()

plt.scatter(pca_data2[:,0], pca_data2[:,1],c=pca3_labels)
plt.title("PCA Data")
plt.show()
```

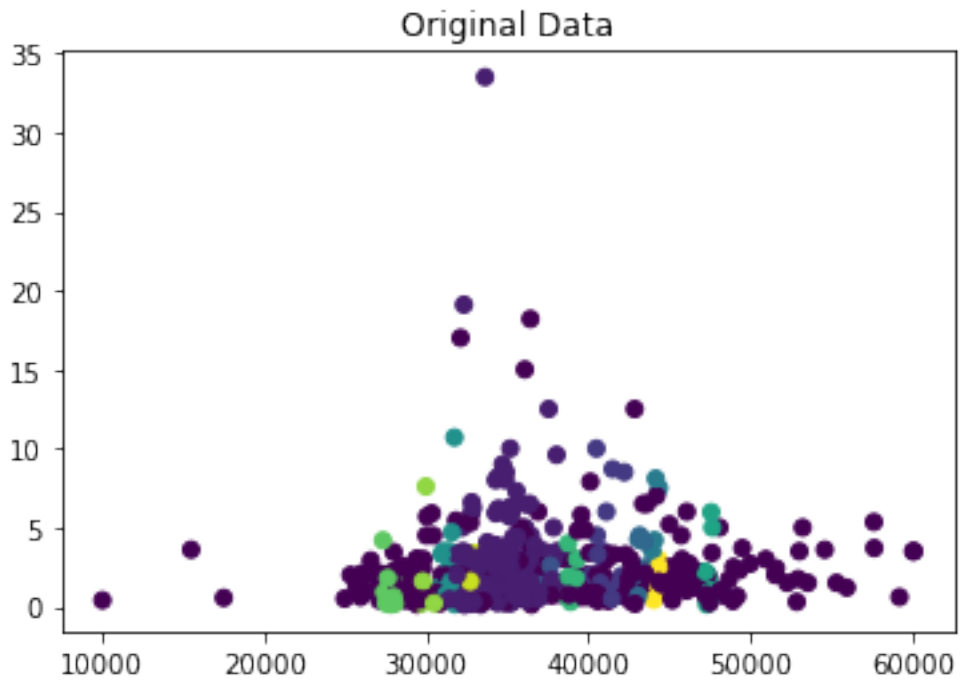
```
Original Data's Number of Clusters at min_samples=5: 12
PCA Data's Number of Clusters at min_samples=5: 12
```

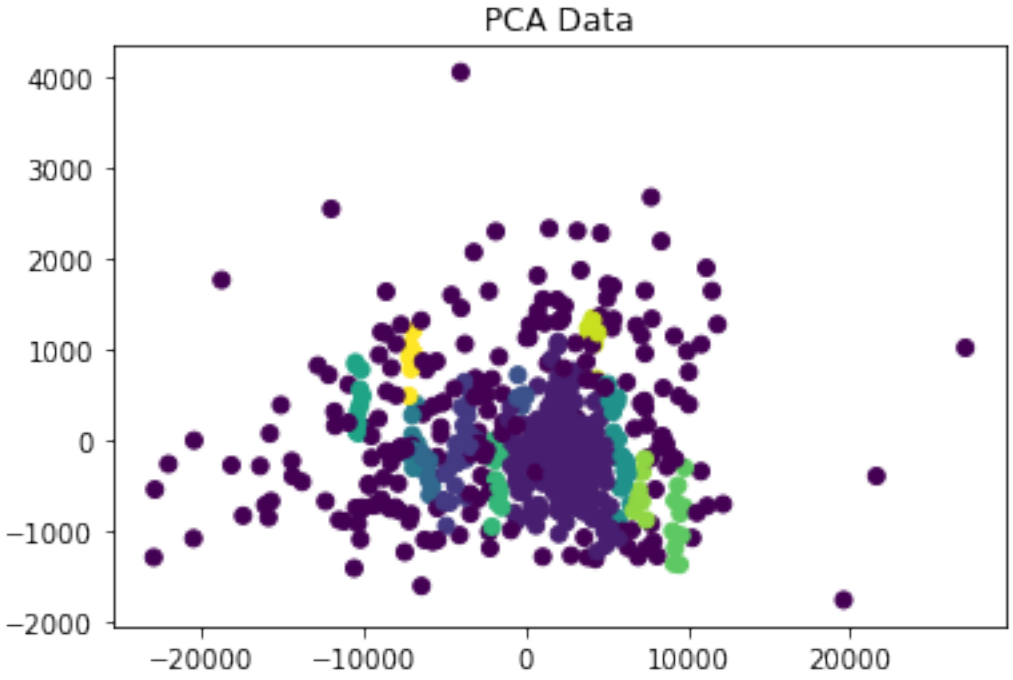




[ ]:
```
dbs4 = DBSCAN(eps=500, min_samples=6)
dbs4_labels = dbs4.fit_predict(D3)
print("Original Data's Number of Clusters at min_samples=6:" ,␣
 ↪max(dbs4_labels)+1)
pca4_labels = dbs4.fit_predict(pca_data2)
print("PCA Data's Number of Clusters at min_samples=6:" , max(pca4_labels)+1)

plt.scatter(D3[:,0], D3[:,1],c=dbs4_labels)
plt.title("Original Data")
plt.show()

plt.scatter(pca_data2[:,0], pca_data2[:,1],c=pca4_labels)
plt.title("PCA Data")
plt.show()
```

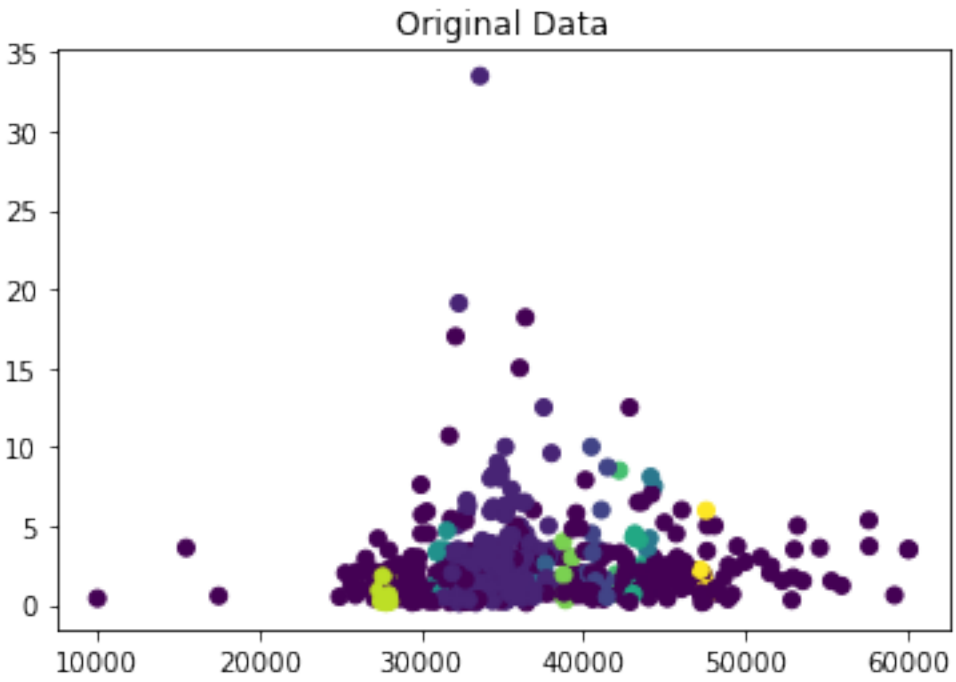
```
Original Data's Number of Clusters at min_samples=6: 12
PCA Data's Number of Clusters at min_samples=6: 12
```

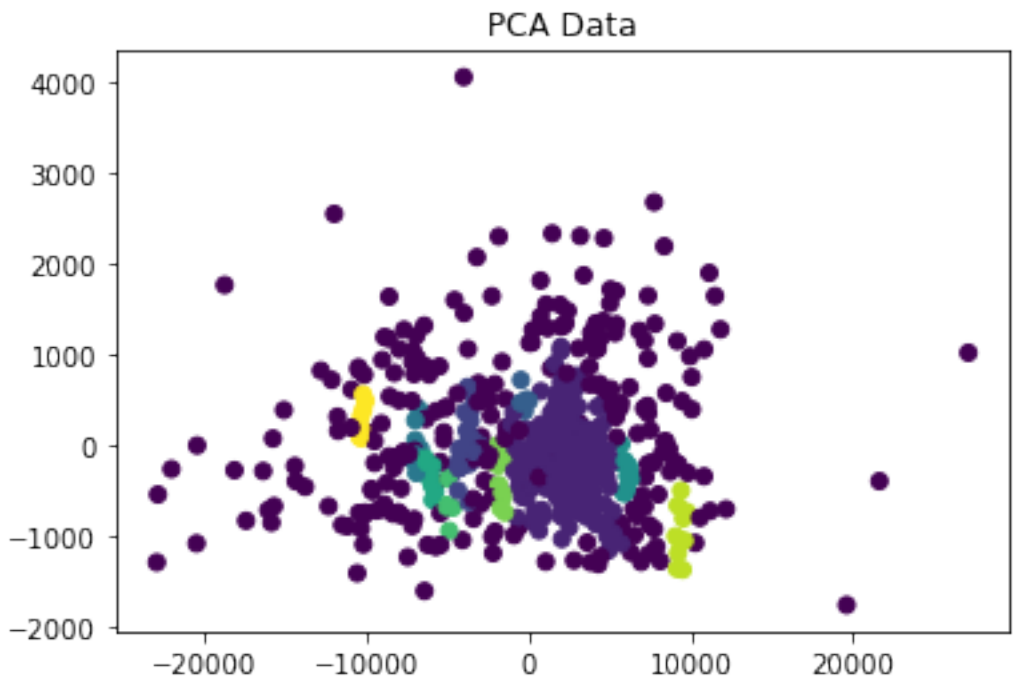

Original Data
35
30
25
20
15
10
5
0
10000
20000
30000
40000
50000
60000


PCA Data
4000
3000
2000
1000
0
−1000
−2000
−20000
−10000
0
10000
20000

[ ]:
```
dbs5 = DBSCAN(eps=500, min_samples=7)
dbs5_labels = dbs5.fit_predict(D3)
print("Original Data's Number of Clusters at min_samples=7:" ,␣
 ↪max(dbs5_labels)+1)
pca5_labels = dbs5.fit_predict(pca_data2)
print("PCA Data's Number of Clusters at min_samples=7:" , max(pca5_labels)+1)

plt.scatter(D3[:,0], D3[:,1],c=dbs5_labels)
plt.title("Original Data")
plt.show()

plt.scatter(pca_data2[:,0], pca_data2[:,1],c=pca5_labels)
plt.title("PCA Data")
plt.show()
```

```
Original Data's Number of Clusters at min_samples=7: 10
PCA Data's Number of Clusters at min_samples=7: 10
```

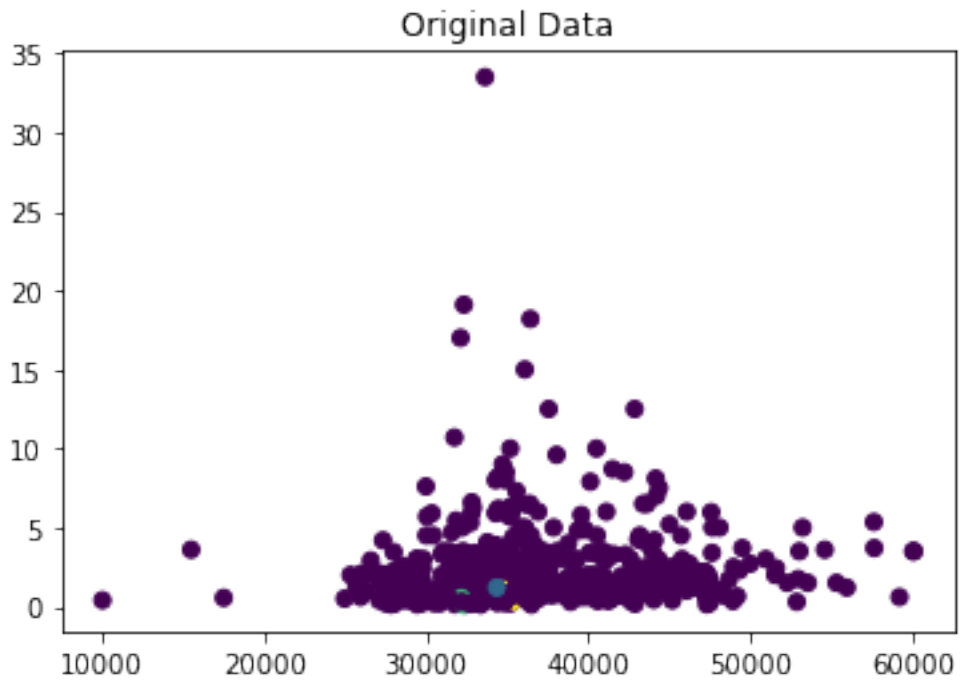


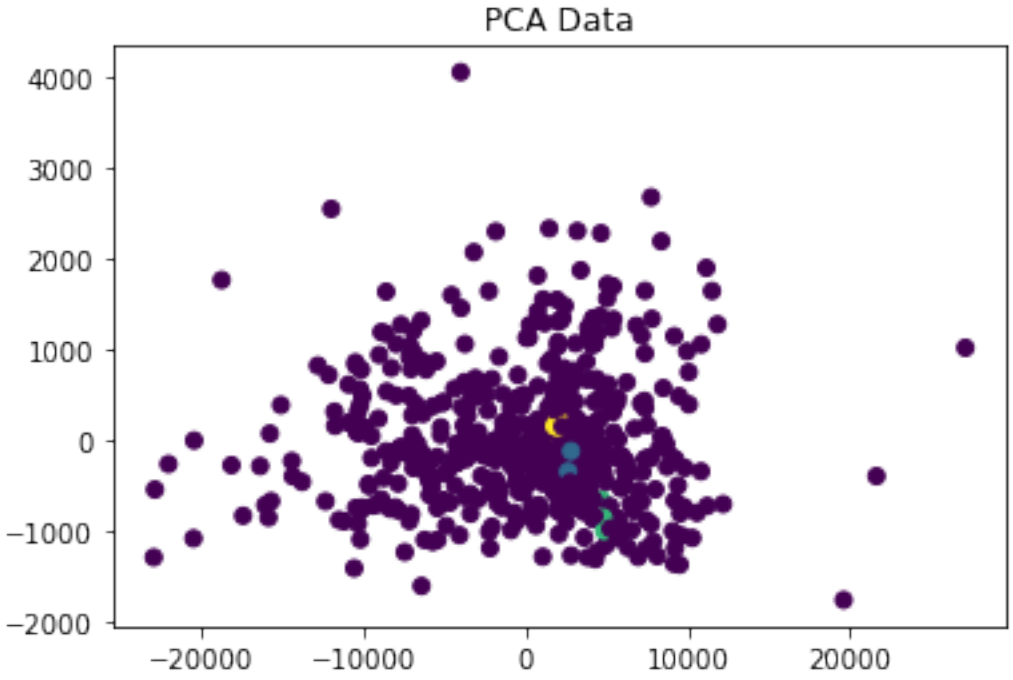


-

[ ]:
```
dbs_e1 = DBSCAN(eps=275, min_samples=5)
dbs_e1_labels = dbs_e1.fit_predict(D3)
print("Original Data's Number of Clusters at eps=7:" , max(dbs_e1_labels)+1)
pca1_e1_labels = dbs_e1.fit_predict(pca_data2)
print("PCA Data's Number of Clusters at eps=7:" , max(pca1_e1_labels)+1)
plt.scatter(D3[:,0], D3[:,1],c=dbs_e1_labels)
plt.title("Original Data")
plt.show()

plt.scatter(pca_data2[:,0], pca_data2[:,1],c=pca1_e1_labels)
plt.title("PCA Data")
plt.show()
```

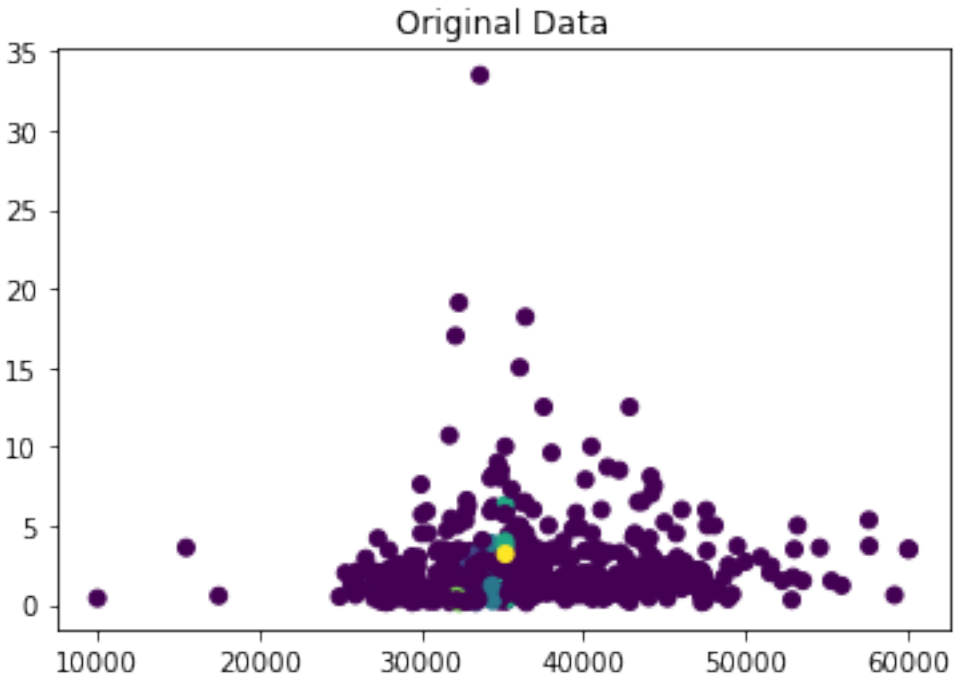
```
Original Data's Number of Clusters at eps=7: 3
PCA Data's Number of Clusters at eps=7: 3
```

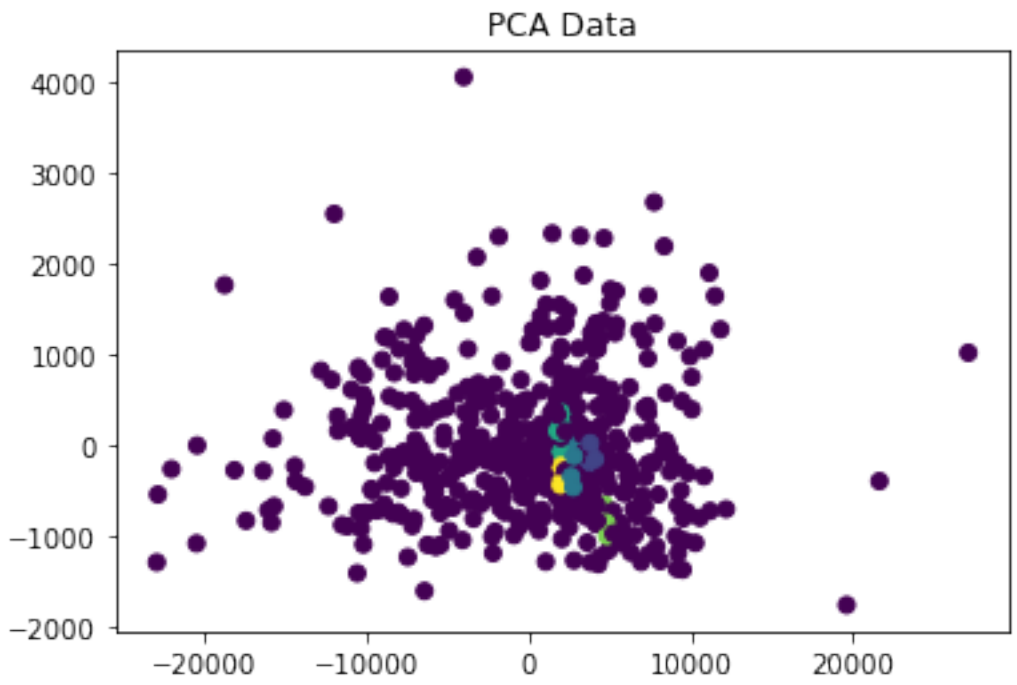

Original Data
35
30
25
20
15
10
5
0
10000
20000
30000
40000
50000
60000


PCA Data
4000
3000
2000
1000
0
−1000
−2000
−20000
−10000
0
10000
20000

[ ]:
```
dbs_e2 = DBSCAN(eps=299, min_samples=5)
dbs_e2_labels = dbs_e2.fit_predict(D3)
print("Original Data's Number of Clusters at eps=7:" , max(dbs_e2_labels)+1)
pca1_e2_labels = dbs_e2.fit_predict(pca_data2)
print("PCA Data's Number of Clusters at eps=7:" , max(pca1_e2_labels)+1)

plt.scatter(D3[:,0], D3[:,1],c=dbs_e2_labels)
plt.title("Original Data")
plt.show()

plt.scatter(pca_data2[:,0], pca_data2[:,1],c=pca1_e2_labels)
plt.title("PCA Data")
plt.show()
```

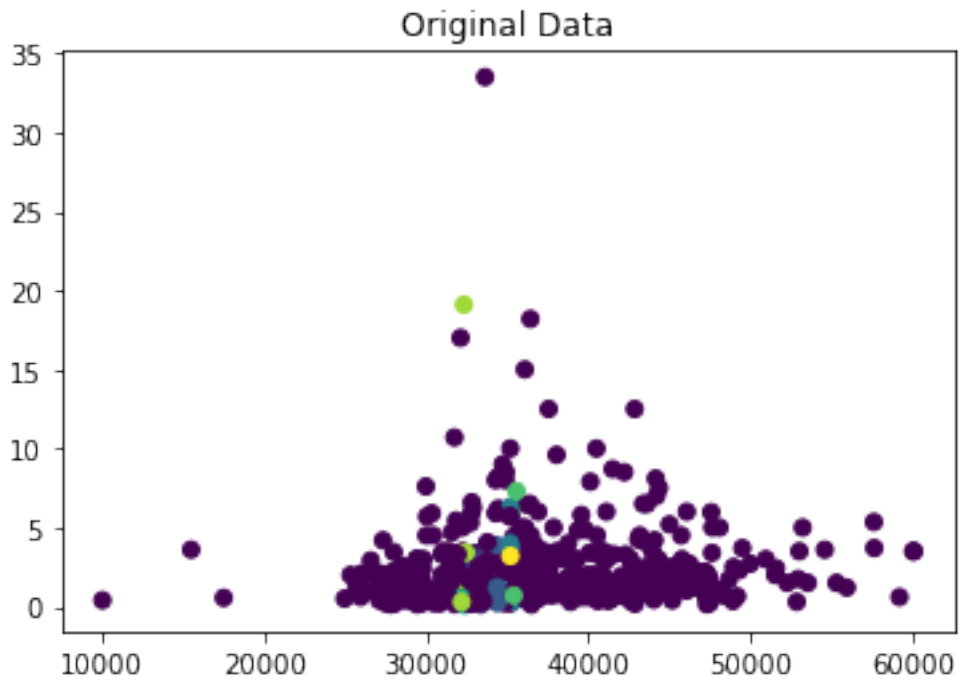
```
Original Data's Number of Clusters at eps=7: 5
PCA Data's Number of Clusters at eps=7: 5
```

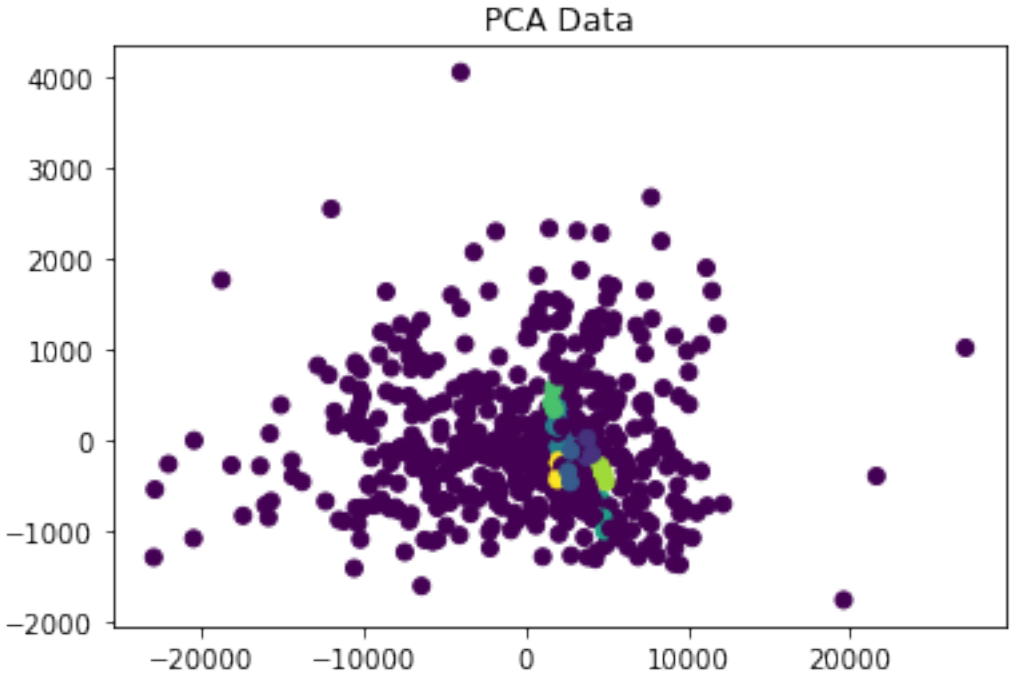




[ ]:
```
dbs_e3 = DBSCAN(eps=300, min_samples=5)
dbs_e3_labels = dbs_e3.fit_predict(D3)
print("Original Data's Number of Clusters at eps=7:" , max(dbs_e3_labels)+1)
pca1_e3_labels = dbs_e3.fit_predict(pca_data2)
print("PCA Data's Number of Clusters at eps=7:" , max(pca1_e3_labels)+1)

plt.scatter(D3[:,0], D3[:,1],c=dbs_e3_labels)
plt.title("Original Data")
plt.show()

plt.scatter(pca_data2[:,0], pca_data2[:,1],c=pca1_e3_labels)
plt.title("PCA Data")
plt.show()
```

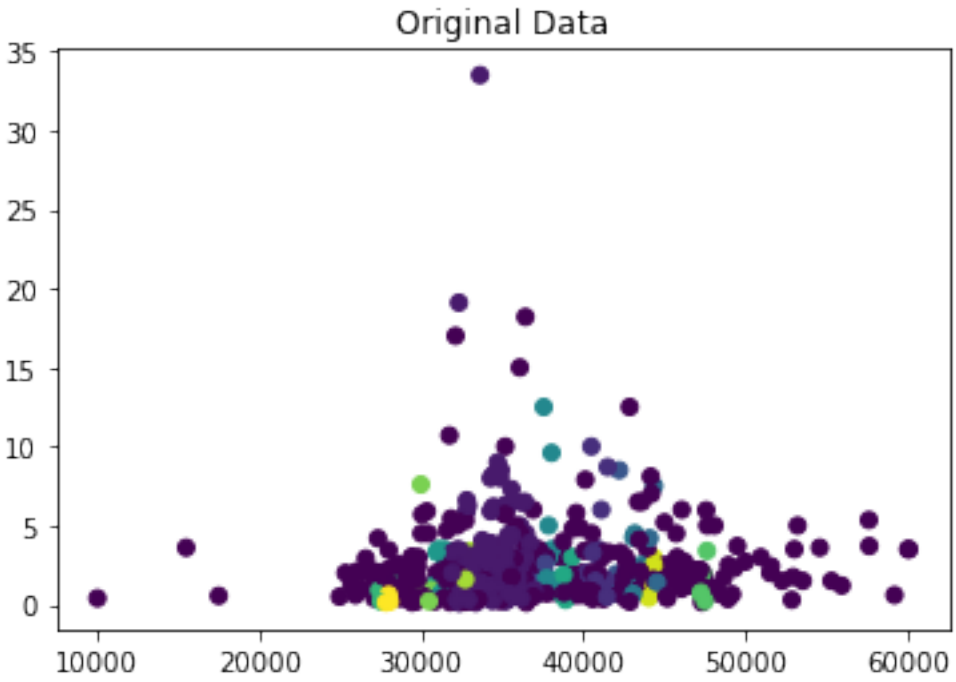
```
Original Data's Number of Clusters at eps=7: 7
PCA Data's Number of Clusters at eps=7: 7
```

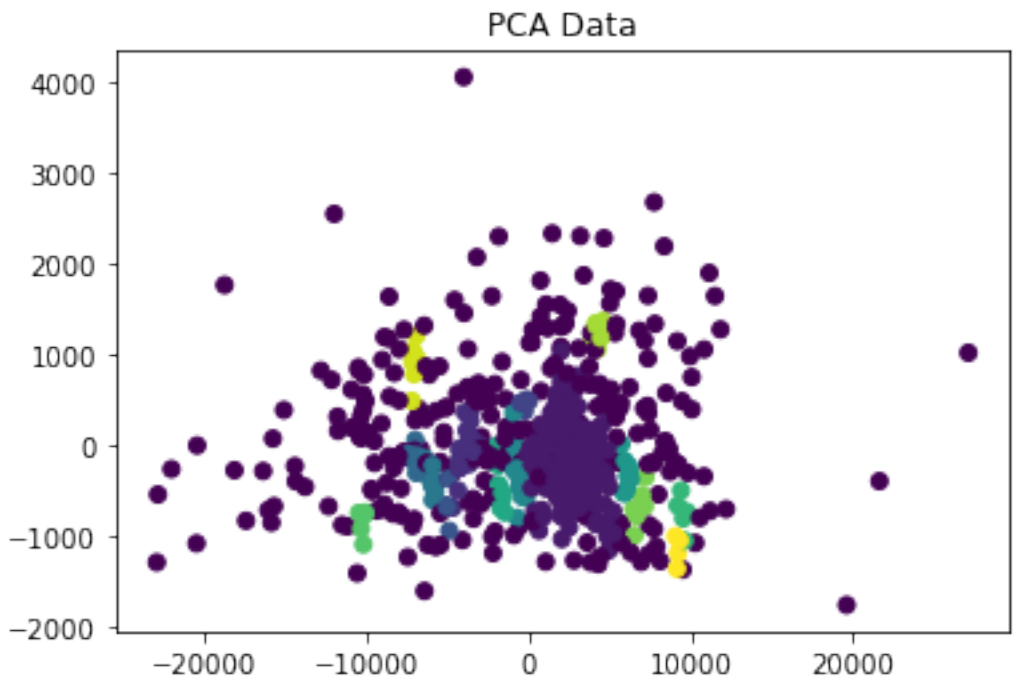

Original Data
35
30
25
20
15
10
5
0
10000
20000
30000
40000
50000
60000


PCA Data
4000
3000
2000
1000
0
−1000
−2000
−20000
−10000
0
10000
20000

[ ]:
```
dbs_e4 = DBSCAN(eps=425, min_samples=5)
dbs_e4_labels = dbs_e4.fit_predict(D3)
print("Original Data's Number of Clusters at eps=7:" , max(dbs_e4_labels)+1)
pca1_e4_labels = dbs_e4.fit_predict(pca_data2)
print("PCA Data's Number of Clusters at eps=7:" , max(pca1_e4_labels)+1)

plt.scatter(D3[:,0], D3[:,1],c=dbs_e4_labels)
plt.title("Original Data")
plt.show()

plt.scatter(pca_data2[:,0], pca_data2[:,1],c=pca1_e4_labels)
plt.title("PCA Data")
plt.show()
```

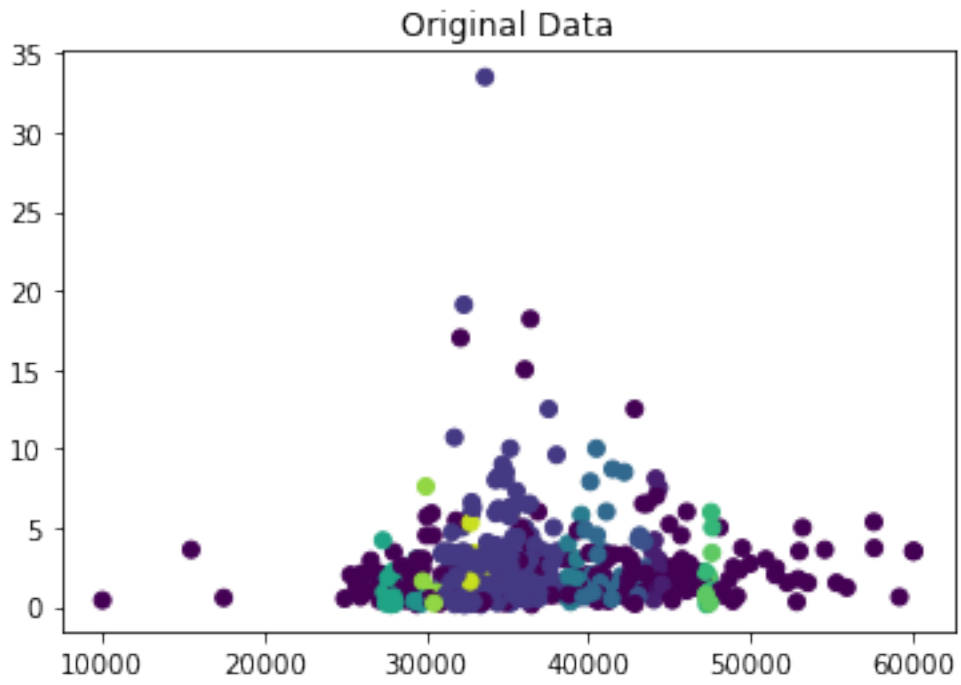
```
Original Data's Number of Clusters at eps=7: 15
PCA Data's Number of Clusters at eps=7: 15
```

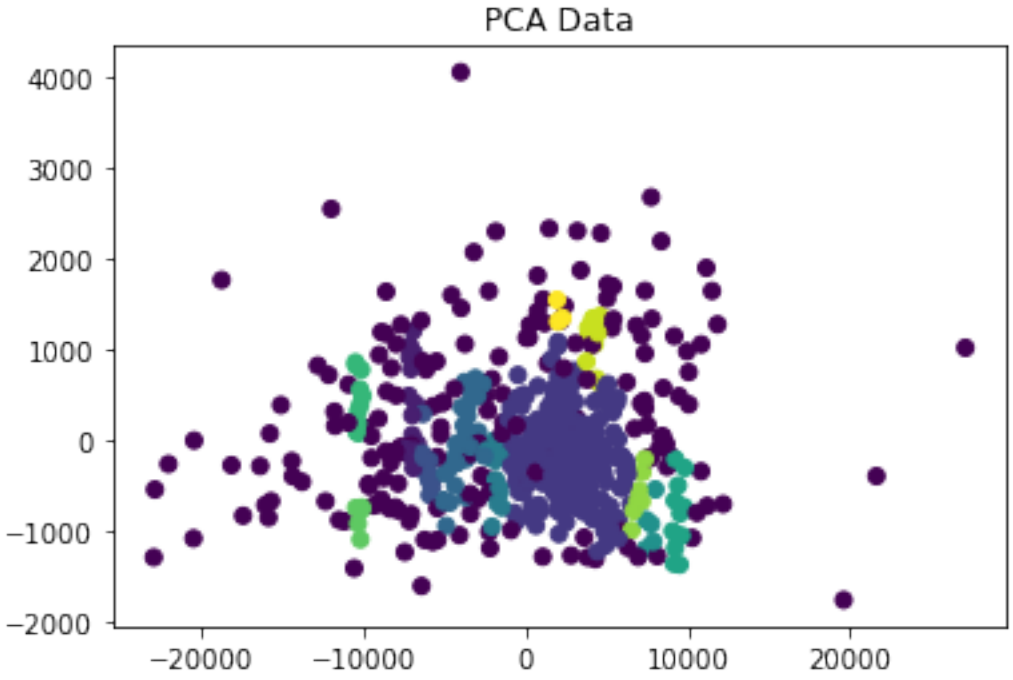




[ ]:
```
dbs_e5 = DBSCAN(eps=500, min_samples=5)
dbs_e5_labels = dbs_e5.fit_predict(D3)
print("Original Data's Number of Clusters at eps=7:" , max(dbs_e5_labels)+1)
pca1_e5_labels = dbs_e5.fit_predict(pca_data2)
print("PCA Data's Number of Clusters at eps=7:" , max(pca1_e5_labels)+1)

plt.scatter(D3[:,0], D3[:,1],c=dbs_e5_labels)
plt.title("Original Data")
plt.show()

plt.scatter(pca_data2[:,0], pca_data2[:,1],c=pca1_e5_labels)
plt.title("PCA Data")
plt.show()
```

```
Original Data's Number of Clusters at eps=7: 12
PCA Data's Number of Clusters at eps=7: 12
```


Original Data
35
30
25
20
15
10
5
0
10000
20000
30000
40000
50000
60000


PCA Data
4000
3000
2000
1000
0
−1000
−2000
−20000
−10000
0
10000
20000

5. (Extra credit - 3 points): Create a plot of clustering precision for each value of k used in question 3.3, each value of ϵ used in question 3.4, and each value of minpts used in question 3.4, for both the original and reduced-dimensionality data.

Added them with 3.3 and 3.4