Homework 2

Please submit both an .ipynb version and an .html version (for easier visualization on Canvas)

In [1]:
```
# Make sure you are running on Colab
if 'google.colab' in str(get_ipython()):
    print('Running on CoLab')
    !pip install yellowbrick
    !pip install gower
    !pip install scikit-learn-extra
    !pip install kmodes
```

In [2]:
```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt # for data visualization
import seaborn as sns # for statistical data visualization
from sklearn.decomposition import PCA
from sklearn.preprocessing import MinMaxScaler,StandardScaler
import plotly.express as px
# %matplotlib inline
import warnings
warnings.filterwarnings('ignore') # Ignore warnings
```

PCA and Clustering

Delta Airlines' website have data on all of their aircraft in a certain site section. In this homework, we will investigate the different aircraft in Delta's fleet. Which planes are similar? Which are dissimilar?

In [3]:
```
# We begin by loading inspecting the data
delta_data = pd.read_csv('delta.csv', index_col=0).reset_index()
```

In [4]:
```
delta_data.head()
```

Out[4]:

	Aircraft	Seat Width (Club)	Seat Pitch (Club)	Seat (Club)	Seat Width (First Class)	Seat Pitch (First Class)	Seats (First Class)	Seat Width (Business)	Seat Pitch (Business)	Seats (Business)	.
0	Airbus A319	0.0	0	0	21.0	36.0	12	0.0	0.0	0	.
1	Airbus A319 VIP	19.4	44	12	19.4	40.0	28	21.0	59.0	14	.
2	Airbus A320	0.0	0	0	21.0	36.0	12	0.0	0.0	0	.
3	Airbus A320 32-R	0.0	0	0	21.0	36.0	12	0.0	0.0	0	.
4	Airbus A330-200	0.0	0	0	0.0	0.0	0	21.0	60.0	32	.

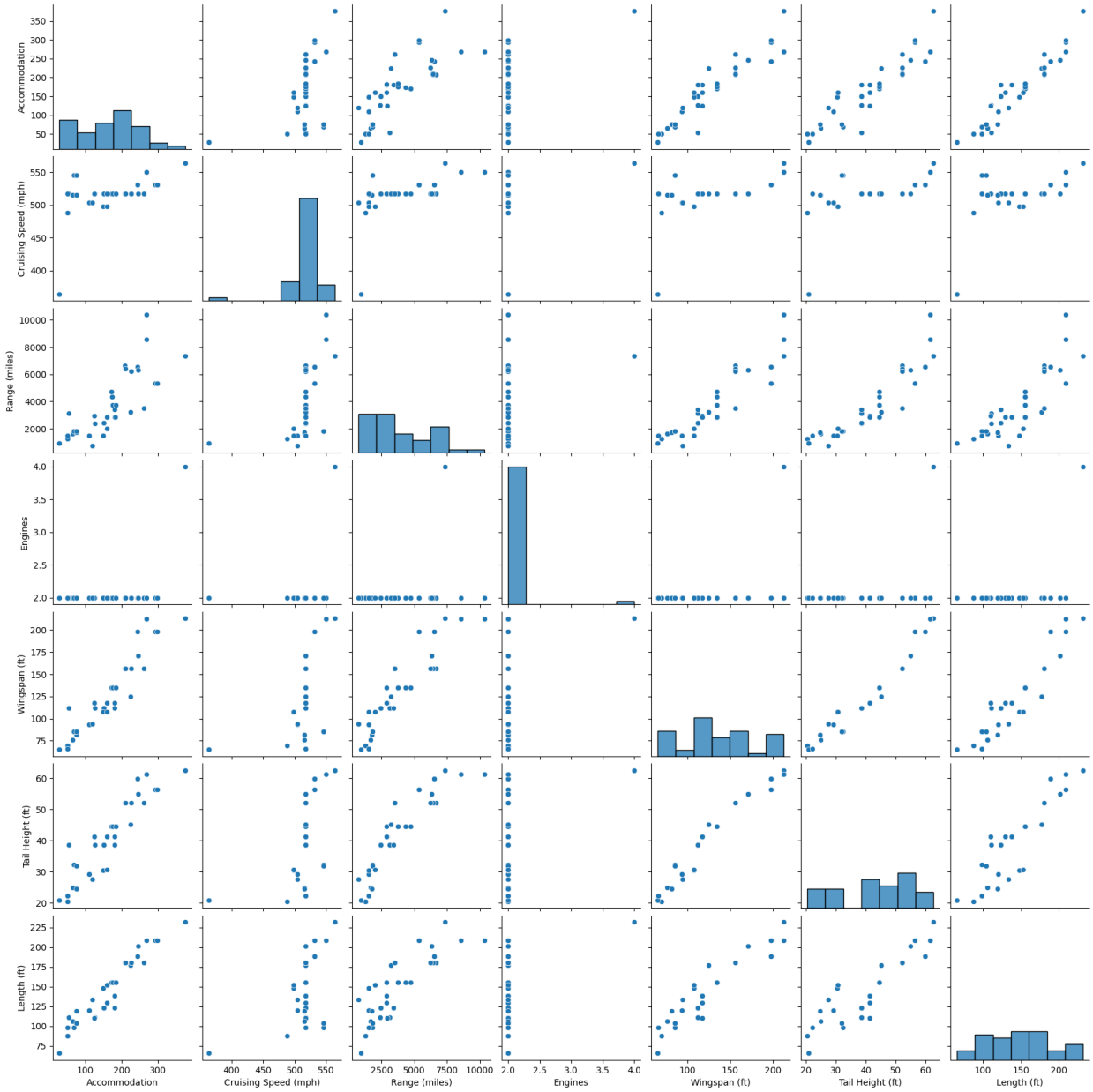
5 rows × 34 columns

Exercise 1: check correlation with visualization

(1 point) Visualize pairwise correlation using scatterplot for columns `'Accommodation', 'Cruising Speed (mph)', 'Range (miles)', 'Engines', 'Wingspan (ft)', 'Tail Height (ft)', 'Length (ft)'` (i.e., columns 16-23 in `delta_data`)

In [5]:
```
comp_df = pd.DataFrame(delta_data,columns=['Accommodation', 'Cruising Speed (mp
sns.pairplot(comp_df)
```

Out[5]:
```
<seaborn.axisgrid.PairGrid at 0x13870d960>
```



Exercise 2: Prepare X

(1 point) Identify all binary columns (i.e., a binary column is the column that only contains `0` and `1` as values)

(1 point) Generate a dataframe `X` removing these binary columns and column `Aircraft` (since this is the `y`)

In [6]:
```
binary_columns = []
for column in delta_data.columns:
    unique_values = delta_data[column].unique()
    if len(unique_values) == 2 and 0 in unique_values and 1 in unique_values:
        binary_columns.append(column)

print(binary_columns)
```

```
['Wifi', 'Video', 'Power', 'Satellite', 'Flat-bed', 'Sleeper', 'Club', 'First Class', 'Business', 'Eco Comfort', 'Economy']
```

In [7]:
```
columns_to_remove = binary_columns + ["Aircraft"]

X = delta_data.drop(columns=columns_to_remove)
```

In [8]:
```
X
```

Out[8]:

	Seat Width (Club)	Seat Pitch (Club)	Seat (Club)	Seat Width (First Class)	Seat Pitch (First Class)	Seats (First Class)	Seat Width (Business)	Seat Pitch (Business)	Seats (Business)	Seat Width (Eco Comfort)
0	0.0	0	0	21.0	36.0	12	0.0	0.00	0	17.20
1	19.4	44	12	19.4	40.0	28	21.0	59.00	14	0.00
2	0.0	0	0	21.0	36.0	12	0.0	0.00	0	17.20
3	0.0	0	0	21.0	36.0	12	0.0	0.00	0	17.20
4	0.0	0	0	0.0	0.0	0	21.0	60.00	32	18.00
5	0.0	0	0	0.0	0.0	0	21.0	80.00	34	18.00
6	0.0	0	0	0.0	0.0	0	21.0	80.00	34	18.00
7	0.0	0	0	0.0	0.0	0	20.0	60.00	34	18.00
8	0.0	0	0	19.6	37.0	12	0.0	0.00	0	18.10
9	0.0	0	0	21.0	37.0	12	0.0	0.00	0	17.20
10	0.0	0	0	21.0	37.0	16	0.0	0.00	0	17.20
11	0.0	0	0	21.0	38.0	16	0.0	0.00	0	17.20
12	0.0	0	0	21.0	37.0	20	0.0	0.00	0	17.20
13	0.0	0	0	0.0	0.0	0	20.5	82.00	48	17.20
14	0.0	0	0	21.0	40.0	24	0.0	0.00	0	17.20
15	0.0	0	0	0.0	0.0	0	21.0	54.50	16	17.20
16	0.0	0	0	21.0	37.0	22	0.0	0.00	0	17.20
17	0.0	0	0	21.0	37.0	22	0.0	0.00	0	17.20
18	0.0	0	0	21.0	37.5	24	0.0	0.00	0	17.20
19	0.0	0	0	21.0	45.0	22	0.0	0.00	0	17.20
20	0.0	0	0	0.0	0.0	0	21.0	38.00	26	17.20
21	0.0	0	0	21.0	37.0	24	0.0	0.00	0	17.20
22	0.0	0	0	0.0	0.0	0	18.5	60.00	30	17.90
23	0.0	0	0	0.0	0.0	0	21.0	78.95	36	17.90
24	0.0	0	0	18.5	37.5	30	0.0	0.00	0	17.90
25	0.0	0	0	18.5	37.5	30	0.0	0.00	0	17.90
26	0.0	0	0	0.0	0.0	0	21.0	78.95	36	17.90
27	0.0	0	0	18.5	60.0	36	0.0	0.00	0	17.90
28	0.0	0	0	0.0	0.0	0	21.0	78.95	26	17.90
29	0.0	0	0	0.0	0.0	0	21.0	78.95	26	17.90
30	0.0	0	0	0.0	0.0	0	21.0	78.95	40	17.90
31	0.0	0	0	0.0	0.0	0	21.0	78.00	45	18.50
32	0.0	0	0	0.0	0.0	0	21.0	78.00	45	18.50

	Seat Width (Club)	Seat Pitch (Club)	Seat (Club)	Seat Width (First Class)	Seat Pitch (First Class)	Seats (First Class)	Seat Width (Business)	Seat Pitch (Business)	Seats (Business)	Seat Width (Eco Comfort)
33	0.0	0	0	0.0	0.0	0	0.0	0.00	0	0.00
34	0.0	0	0	0.0	0.0	0	0.0	0.00	0	0.00
35	0.0	0	0	19.6	36.0	9	0.0	0.00	0	17.30
36	0.0	0	0	19.6	37.0	12	0.0	0.00	0	17.30
37	0.0	0	0	0.0	0.0	0	0.0	0.00	0	0.00
38	0.0	0	0	20.0	37.0	9	0.0	0.00	0	18.25
39	0.0	0	0	20.0	37.0	12	0.0	0.00	0	18.25
40	0.0	0	0	0.0	0.0	0	0.0	0.00	0	0.00
41	0.0	0	0	19.6	37.0	16	0.0	0.00	0	18.10
42	0.0	0	0	19.6	37.0	16	0.0	0.00	0	18.10
43	0.0	0	0	19.6	38.0	16	0.0	0.00	0	18.10

44 rows x 22 columns

Exercise 3: PCA

(1 point) Standardize X

(1 point) conduct PCA

(1 point) calculate the amount of varialce each principal component explain, as well as the cumulative sum

(1 point) visualize the cumulative sum of explained variance and set the xticks as the number of components

In [15]:
```
X_normalized = MinMaxScaler().fit(X).transform(X)
```

In [16]:
```
pca = PCA().fit(X_normalized)
```

In [17]:
```
#the amount of varialce each principal component explain
pca.explained_variance_ratio_
```

Out[17]:
```
array([5.99690587e-01, 2.26458950e-01, 9.40984759e-02, 3.73443328e-02,
       1.43784344e-02, 7.86312504e-03, 5.43385504e-03, 4.40748715e-03,
       3.72523645e-03, 2.19432926e-03, 1.25207519e-03, 9.95571232e-04,
       8.74118421e-04, 5.24802577e-04, 3.89720810e-04, 1.87827044e-04,
       7.66219420e-05, 5.95000513e-05, 3.64437261e-05, 8.50632331e-06,
       5.09624504e-33, 3.58832495e-35])
```

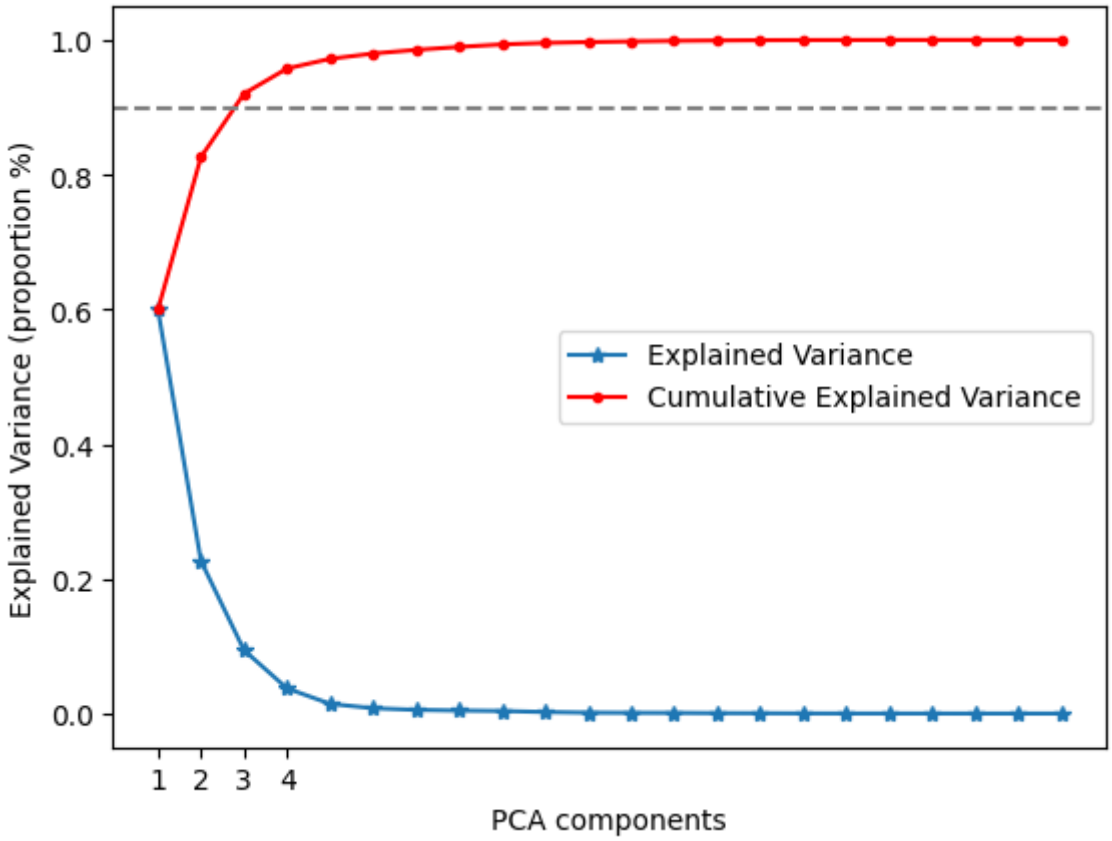
In [18]:
```
#the cumulative sum of varialce each principal component explain
exp_var = pca.explained_variance_ratio_
```

```
exp_var_cumsum = np.cumsum(exp_var)
exp_var_cumsum
```

Out[18]:
```
array([0.59969059, 0.82614954, 0.92024801, 0.95759235, 0.97197078,
       0.9798339 , 0.98526776, 0.98967525, 0.99340048, 0.99559481,
       0.99684689, 0.99784246, 0.99871658, 0.99924138, 0.9996311 ,
       0.99981893, 0.99989555, 0.99995505, 0.99999149, 1.        ,
       1.        , 1.        ])
```

In [19]:
```
#visualize the cumulative sum of explained variance and set the xticks as the r
plt.plot(range(1, len(exp_var)+1),
        exp_var,'*-',label='Explained Variance')

plt.plot(range(1, len(exp_var)+1),
        exp_var_cumsum,'r.-',label='Cumulative Explained Variance')



plt.legend()
ax = plt.gca()
ax.set_xticks([1,2,3,4])
ax.set_xlabel('PCA components')
ax.set_ylabel('Explained Variance (proportion %)')

# which shows that the first two PCs accounts for more than 90% of the variance
plt.axhline(0.9,linestyle='--',color='grey')
```

Out[19]:
```
<matplotlib.lines.Line2D at 0x13e6ddcc0>
```



Exercise 4: Clustering

So, now that we've simplified the complex data set into a lower dimensional space we can visualize and work with, we will use clustering to find patterns in the data, in our case, the aircraft which are most similar?

(1 point) Pick the number of components which explain 90% (or a little greater) of the variation and represent the transformed data (projected onto the selected principal components) as `components`

(1 point) Using `components` as the input, run k-means for k from `2` to `15` (with `random_state=1`), and visualize SSE by elbow curve.

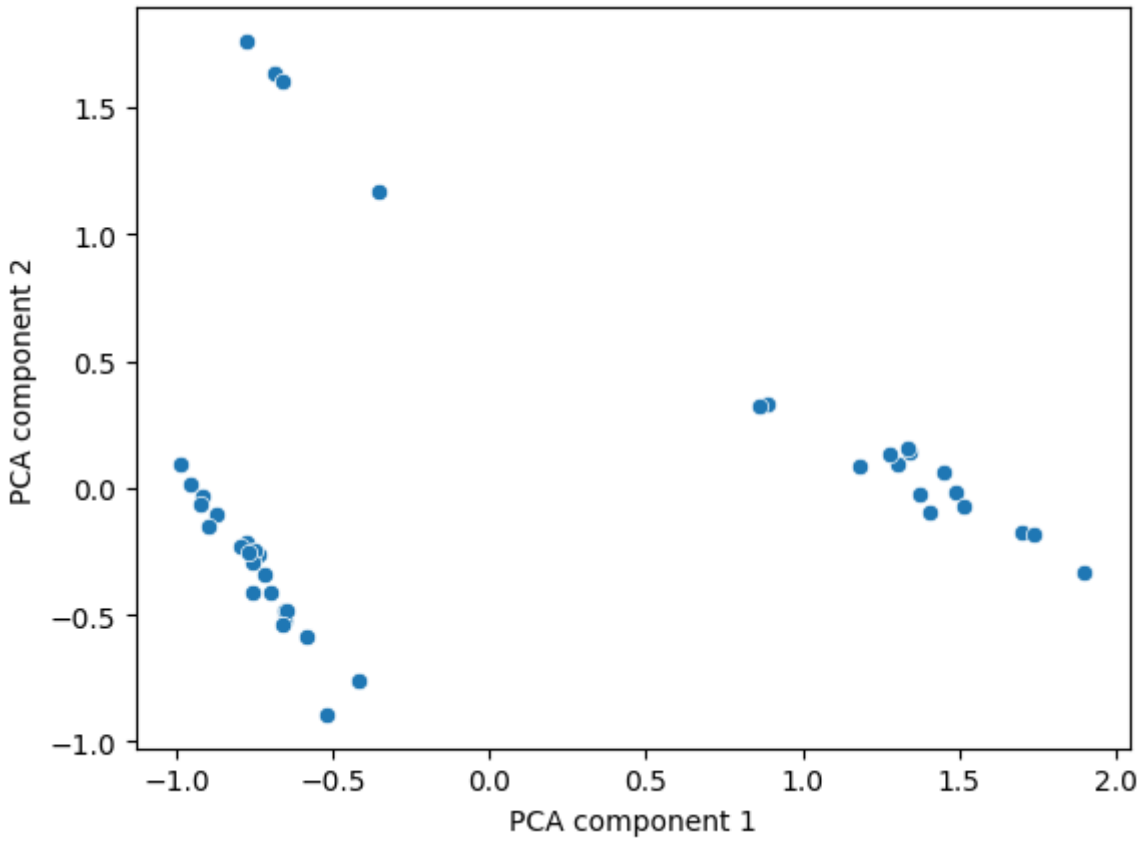
(1 point) Select the best k using elbow curve and run clustering with the selected k to get the cluster assignment

(Extra credit: 1 point) Visualize the cluster membership by scatter plots (for each pair of principle components as `x` and `y` axes)

In [20]:
```
# Pick the number of components which explain 90% (or a little greater) of the
pca = PCA(n_components=3)
components_3 = pca.fit_transform(X_normalized)
```

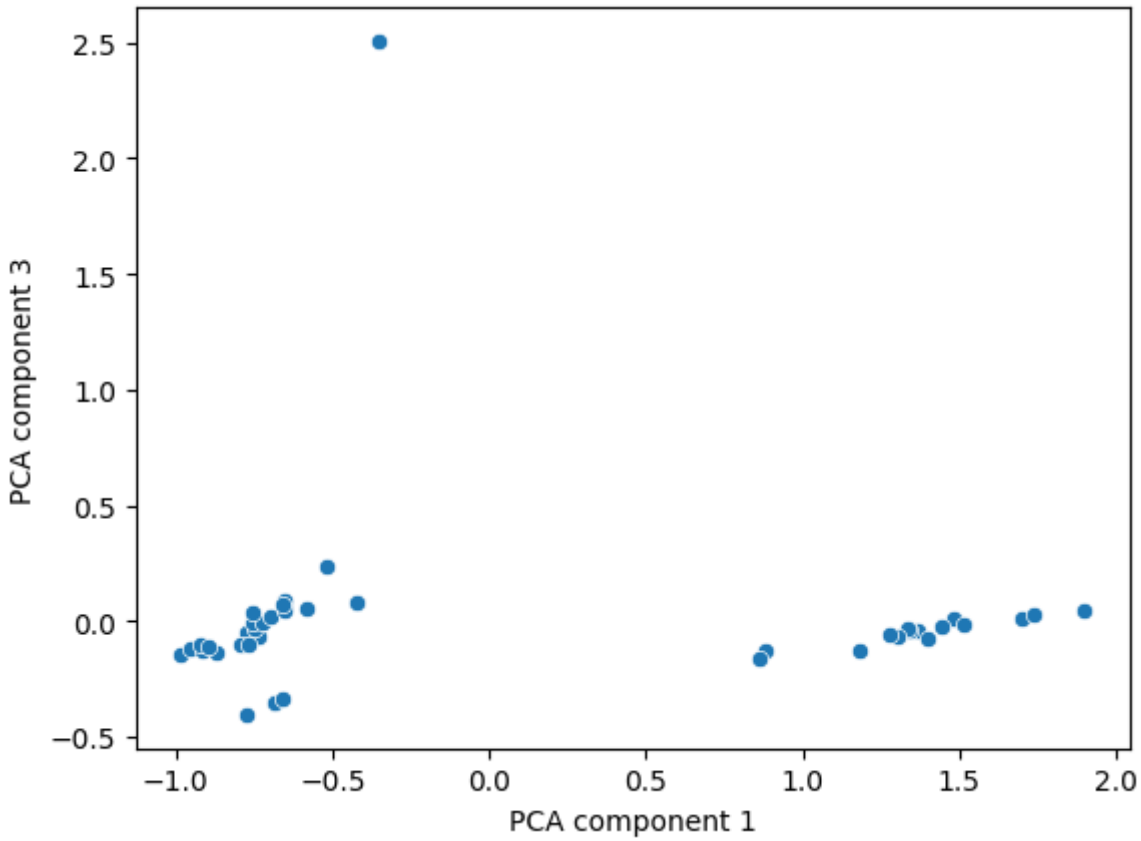
In [21]:
```
sns.scatterplot(x = components_3[:,0], y=components_3[:,1])
plt.xlabel('PCA component 1')
plt.ylabel('PCA component 2')
```

Out[21]:
```
Text(0, 0.5, 'PCA component 2')
```



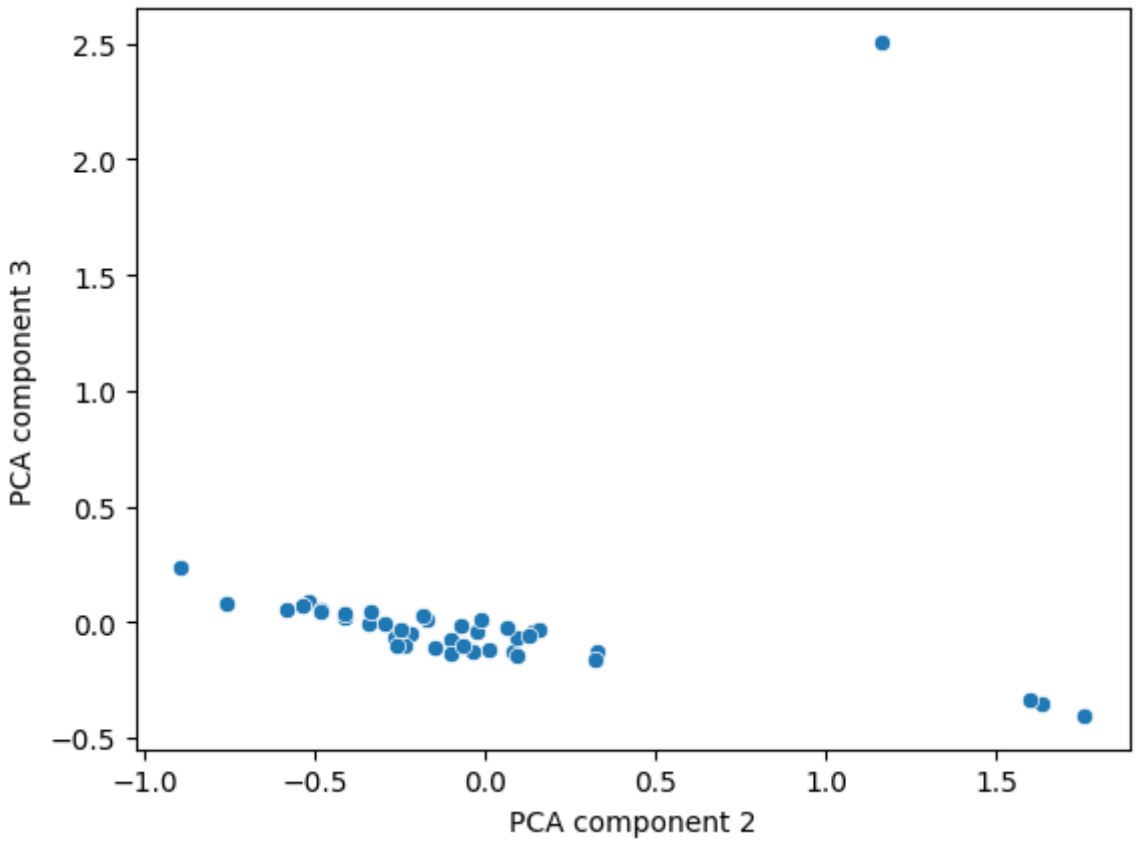
In [22]:
```
sns.scatterplot(x = components_3[:,0], y=components_3[:,2])
plt.xlabel('PCA component 1')
plt.ylabel('PCA component 3')
```

Out[22]:
```
Text(0, 0.5, 'PCA component 3')
```



In [23]:
```
sns.scatterplot(x = components_3[:,1], y=components_3[:,2])
plt.xlabel('PCA component 2')
plt.ylabel('PCA component 3')
```

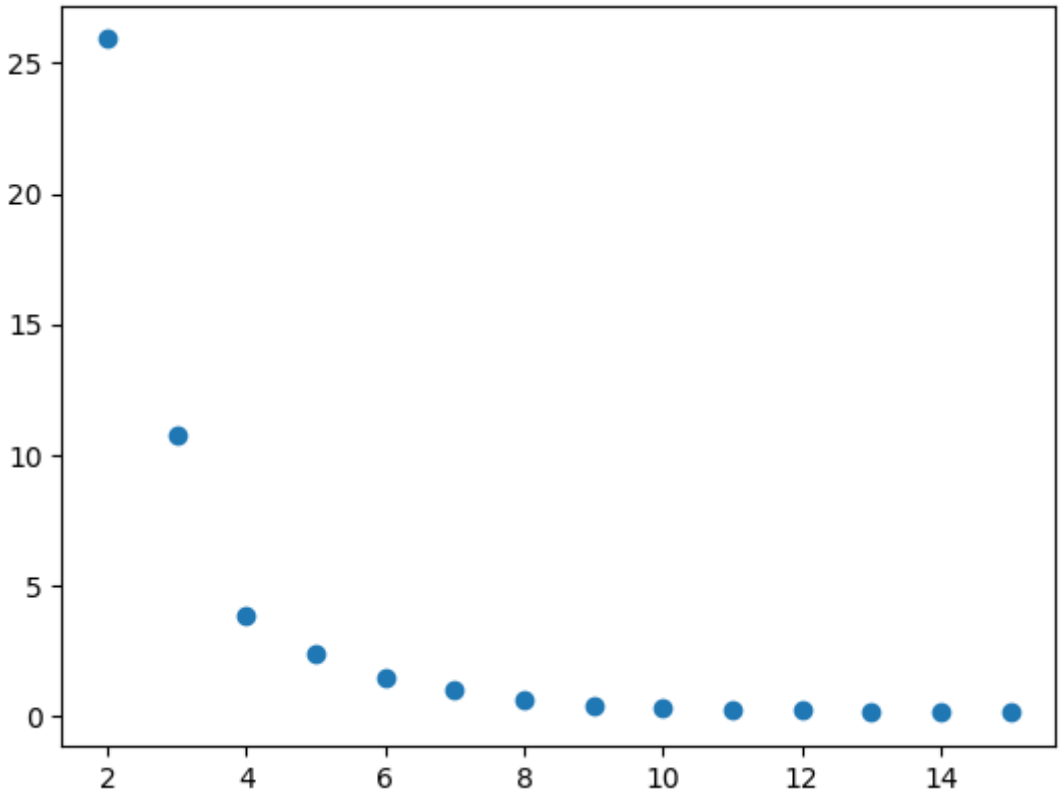
Out[23]:
```
Text(0, 0.5, 'PCA component 3')
```



In [24]:
```
#Using components as the input, run k-means for k from 2 to 15 (with random_sta
```

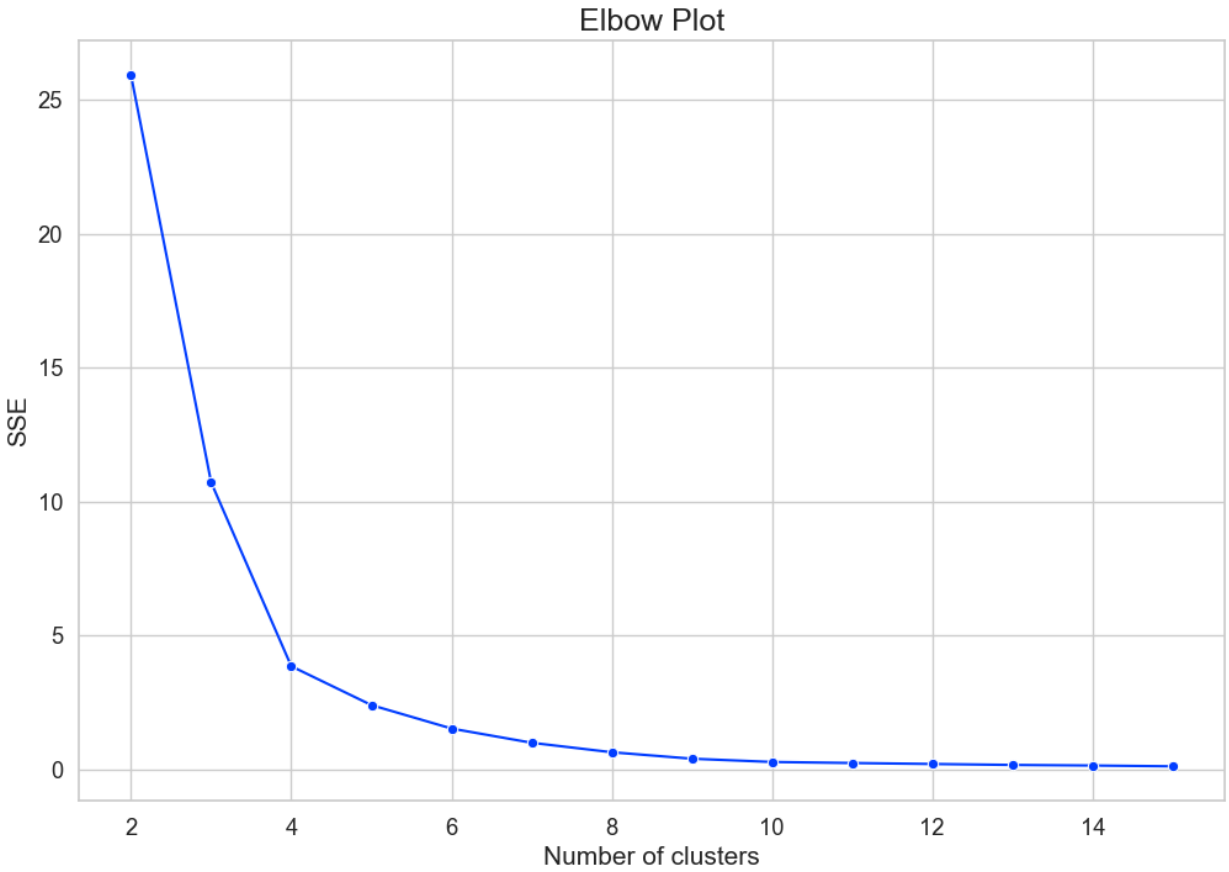
In [25]:
```
from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans

scores = []

for k in range(2, 16):

    kmeans = KMeans(n_clusters = k)
    # Training the KMeans model on the normalized iris data.
    kmeans.fit(components_3)
    # Appending the negative score of the model to the scores list. The negativ
    scores.append(-kmeans.score(components_3))

# Plotting the scores for the range of clusters from 1 to 5.
plt.scatter(list(range(2, 16)), scores)
```

Out[25]:
```
<matplotlib.collections.PathCollection at 0x13f742590>
```



In [26]:

```
# Function for plotting elbow curve
def plot_elbow_curve(start, end, data):
    no_of_clusters = list(range(start, end+1))
    cost_values = []

    for k in no_of_clusters:
        kmeans = KMeans(n_clusters = k, random_state = 1)
        kmeans.fit_predict(data)
        cost_values.append(kmeans.inertia_)

    sns.set_theme(style="whitegrid", palette="bright", font_scale=1.2)
    plt.figure(figsize=(12, 8))
    ax = sns.lineplot(x=no_of_clusters, y=cost_values, marker="o", dashes=False
    ax.set_title('Elbow Plot', fontsize=18)
    ax.set_xlabel('Number of clusters', fontsize=15)
    ax.set_ylabel('SSE', fontsize=15)
    plt.plot();

plot_elbow_curve(start=2,end=15,data = components_3)
```



In [27]:
```
#Select the best k using elbow curve and run clustering with the selected k to
```

Based on the elbow plot, we can see that before cluster = 4, the SSE drops a lot, but after cluster = 4, the SSE drops not too much,
so we choose cluster = 4 as the number of the cluster

In [28]:
```
kmeans = KMeans(n_clusters = 4)
kmeans.fit(components_3)
```

Out[28]:
```
KMeans
KMeans(n_clusters=4)
```

In [29]:
```
#Visualize the cluster membership by scatter plots (for each pair of principle
```

In [30]:
```
cluster = kmeans.fit_predict(X)

X['Cluster'] = cluster

X.head()
```

Out[30]:

	Seat Width (Club)	Seat Pitch (Club)	Seat (Club)	Seat Width (First Class)	Seat Pitch (First Class)	Seats (First Class)	Seat Width (Business)	Seat Pitch (Business)	Seats (Business)	Seat Width (Eco Comfort)
0	0.0	0	0	21.0	36.0	12	0.0	0.0	0	17.2
1	19.4	44	12	19.4	40.0	28	21.0	59.0	14	0.0
2	0.0	0	0	21.0	36.0	12	0.0	0.0	0	17.2
3	0.0	0	0	21.0	36.0	12	0.0	0.0	0	17.2
4	0.0	0	0	0.0	0.0	0	21.0	60.0	32	18.0

5 rows × 23 columns

In [38]:
```
X['Cluster'].unique()
```

Out[38]:
```
array([0, 1, 3, 2], dtype=int32)
```

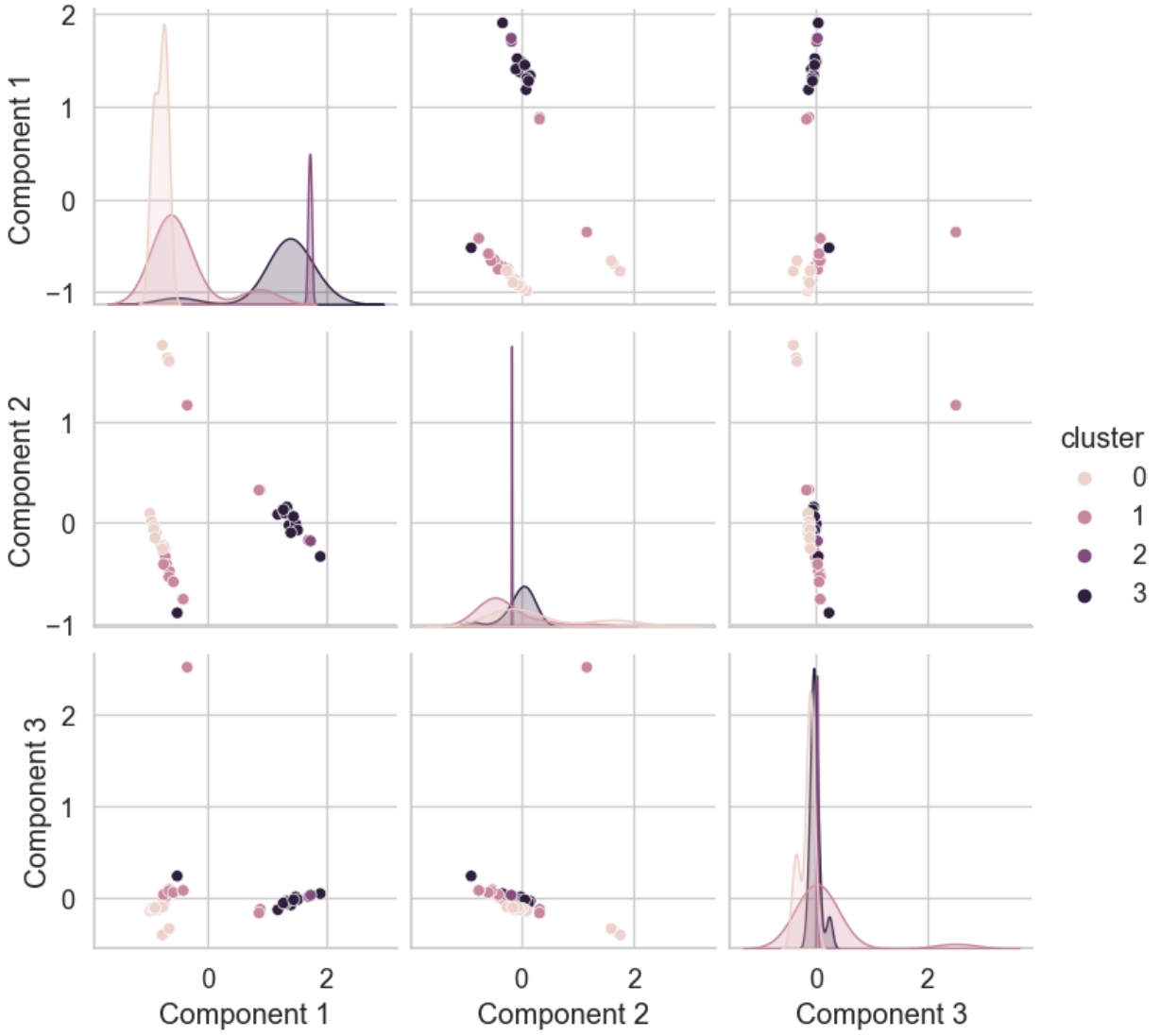
In [41]:
```
n_components = 3
components = pca.fit_transform(X_normalized)[:, :n_components]
```

In [42]:
```
components
```

Out[42]:
```
array([[-0.77466193, -0.21700729, -0.05355604],
       [-0.35226585,  1.16500076,  2.50730036],
       [-0.73733865, -0.2634676 , -0.06844264],
       [-0.73733865, -0.2634676 , -0.06844264],
       [ 1.37047615, -0.02529343, -0.03735896],
       [ 1.48556093, -0.01374743,  0.01117171],
       [ 1.51617041, -0.07366905, -0.01791195],
       [ 1.40183588, -0.09829394, -0.07752105],
       [-0.868668  , -0.10111375, -0.1336597 ],
       [-0.74769565, -0.24559031, -0.03609813],
       [-0.75344767, -0.29498722, -0.00991356],
       [-0.7204656 , -0.34206006, -0.0046536 ],
       [-0.69723216, -0.41366629,  0.01945511],
       [ 1.89965131, -0.33493074,  0.04540639],
       [-0.65290643, -0.51957737,  0.0909585 ],
       [ 0.88393359,  0.32712588, -0.1240362 ],
       [-0.65480761, -0.48044357,  0.05384107],
       [-0.65058242, -0.48190108,  0.04914975],
       [-0.75373969, -0.4109464 ,  0.03478625],
       [-0.6618634 , -0.53679833,  0.07593217],
       [ 0.86292227,  0.32343072, -0.16534424],
       [-0.58495708, -0.58611543,  0.05862966],
       [ 1.18108388,  0.08292413, -0.12922966],
       [ 1.34320516,  0.14019767, -0.04099879],
       [-0.42013889, -0.75781804,  0.07976157],
       [-0.42013889, -0.75781804,  0.07976157],
       [ 1.33255971,  0.15705407, -0.03557833],
       [-0.52110108, -0.89332543,  0.23689738],
       [ 1.30402865,  0.09416289, -0.06425664],
       [ 1.27576751,  0.12907932, -0.05643265],
       [ 1.4471444 ,  0.06133449, -0.02134079],
       [ 1.69964438, -0.17381433,  0.01395651],
       [ 1.73621642, -0.18048987,  0.02911536],
       [-0.68521893,  1.63349135, -0.34940691],
       [-0.68521893,  1.63349135, -0.34940691],
       [-0.98356099,  0.09138576, -0.14223844],
       [-0.95384144,  0.01179537, -0.11491241],
       [-0.77360417,  1.76062425, -0.40421758],
       [-0.91343473, -0.03524607, -0.12917053],
       [-0.91954989, -0.0675653 , -0.10520977],
       [-0.65834354,  1.5984617 , -0.33481665],
       [-0.79362836, -0.2334027 , -0.10538464],
       [-0.76729883, -0.2569345 , -0.09900036],
       [-0.89715117, -0.15006853, -0.10758358]])
```

In [43]:
```
labels = {
    str(i): f"PC {i+1} ({var:.1f}%)"
    for i, var in enumerate(pca.explained_variance_ratio_ * 100)
}

comp_df = pd.DataFrame(components,columns=['Component 1','Component 2','Compone
comp_df['cluster']=X['Cluster']
sns.pairplot(comp_df,hue='cluster')
```

Out[43]:
```
<seaborn.axisgrid.PairGrid at 0x14b5f29b0>
```


Component 1
Component 2
Component 3
cluster
0
1
2
3
Component 1
Component 2
Component 3

In [45]:
```
comp_df
```

Out[45]:

	Component 1	Component 2	Component 3	cluster
0	-0.774662	-0.217007	-0.053556	0
1	-0.352266	1.165001	2.507300	1
2	-0.737339	-0.263468	-0.068443	0
3	-0.737339	-0.263468	-0.068443	0
4	1.370476	-0.025293	-0.037359	3
5	1.485561	-0.013747	0.011172	3
6	1.516170	-0.073669	-0.017912	3
7	1.401836	-0.098294	-0.077521	3
8	-0.868668	-0.101114	-0.133660	0
9	-0.747696	-0.245590	-0.036098	1
10	-0.753448	-0.294987	-0.009914	1
11	-0.720466	-0.342060	-0.004654	1
12	-0.697232	-0.413666	0.019455	1
13	1.899651	-0.334931	0.045406	3
14	-0.652906	-0.519577	0.090958	1
15	0.883934	0.327126	-0.124036	1
16	-0.654808	-0.480444	0.053841	1
17	-0.650582	-0.481901	0.049150	1
18	-0.753740	-0.410946	0.034786	1
19	-0.661863	-0.536798	0.075932	1
20	0.862922	0.323431	-0.165344	1
21	-0.584957	-0.586115	0.058630	1
22	1.181084	0.082924	-0.129230	3
23	1.343205	0.140198	-0.040999	3
24	-0.420139	-0.757818	0.079762	1
25	-0.420139	-0.757818	0.079762	1
26	1.332560	0.157054	-0.035578	3
27	-0.521101	-0.893325	0.236897	3
28	1.304029	0.094163	-0.064257	3
29	1.275768	0.129079	-0.056433	3
30	1.447144	0.061334	-0.021341	3
31	1.699644	-0.173814	0.013957	2
32	1.736216	-0.180490	0.029115	2
33	-0.685219	1.633491	-0.349407	0
34	-0.685219	1.633491	-0.349407	0

	Component 1	Component 2	Component 3	cluster
35	-0.983561	0.091386	-0.142238	0
36	-0.953841	0.011795	-0.114912	0
37	-0.773604	1.760624	-0.404218	0
38	-0.913435	-0.035246	-0.129171	0
39	-0.919550	-0.067565	-0.105210	0
40	-0.658344	1.598462	-0.334817	0
41	-0.793628	-0.233403	-0.105385	0
42	-0.767299	-0.256935	-0.099000	0
43	-0.897151	-0.150069	-0.107584	0