Assignment No :- 05

Title :-

Perform clustering of the iris dataset based on all variables using Gaussian mixture models. Use PCA to visualize clusters.

In [2]:
```
import numpy as np
import pandas as pd
import matplotlib as mpl
import matplotlib.pyplot as plt
import seaborn as sns
sns.set(style="white", color_codes=True)
import warnings
warnings.filterwarnings("ignore")
%matplotlib inline
```

In [3]:
```
data = pd.read_csv('Iris.csv')
data = data.drop('Id', axis=1) # get rid of the Id column - don't need it
data.sample(5)
```

Out[3]:

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
50	7.0	3.2	4.7	1.4	Iris-versicolor
115	6.4	3.2	5.3	2.3	Iris-virginica
109	7.2	3.6	6.1	2.5	Iris-virginica
146	6.3	2.5	5.0	1.9	Iris-virginica
135	7.7	3.0	6.1	2.3	Iris-virginica

In [4]:
```
# split data into features (X) and labels (y)
X = data.iloc[:,0:4]
y = data.iloc[:,-1]
print(X.sample(5))
print(y.sample(5))
```

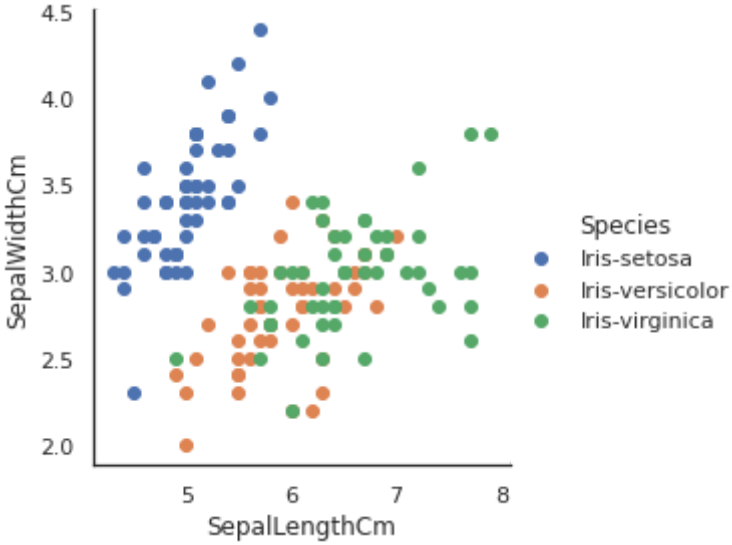
```
     SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm
65             6.7           3.1            4.4           1.4
109            7.2           3.6            6.1           2.5
143            6.8           3.2            5.9           2.3
18             5.7           3.8            1.7           0.3
83             6.0           2.7            5.1           1.6
78     Iris-versicolor
131     Iris-virginica
17         Iris-setosa
82     Iris-versicolor
41         Iris-setosa
Name: Species, dtype: object
```

In [5]:
```
data["Species"].value_counts()
```

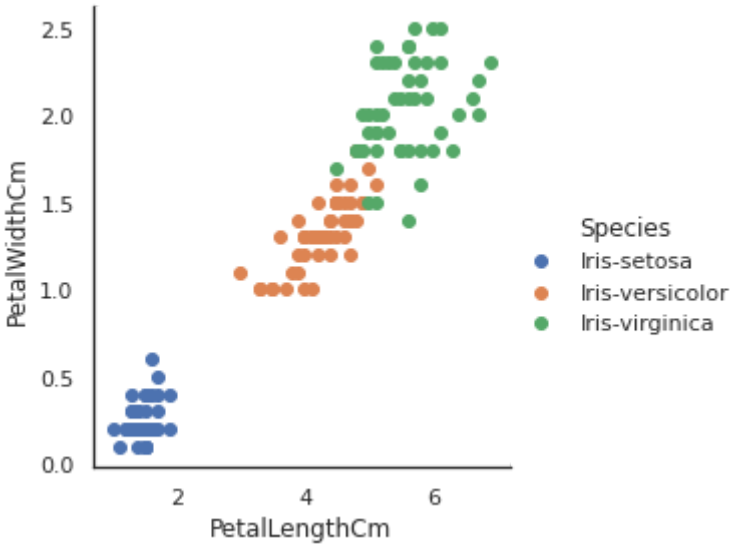
Out[5]:
```
Iris-setosa        50
Iris-versicolor    50
```

```
Iris-virginica     50
Name: Species, dtype: int64
```

In [6]:
```
sns.FacetGrid(data, hue="Species", size=4) \
   .map(plt.scatter, "SepalLengthCm", "SepalWidthCm") \
   .add_legend();
```

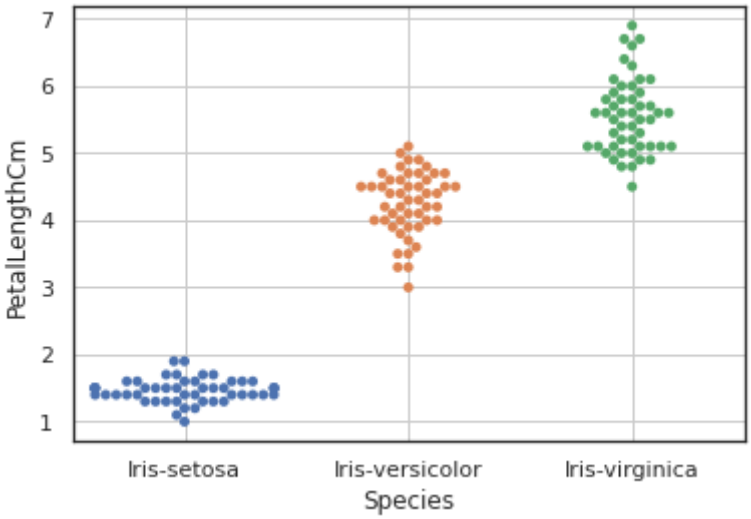


In [7]:
```
sns.FacetGrid(data, hue="Species", size=4) \
   .map(plt.scatter, "PetalLengthCm", "PetalWidthCm") \
   .add_legend();
```



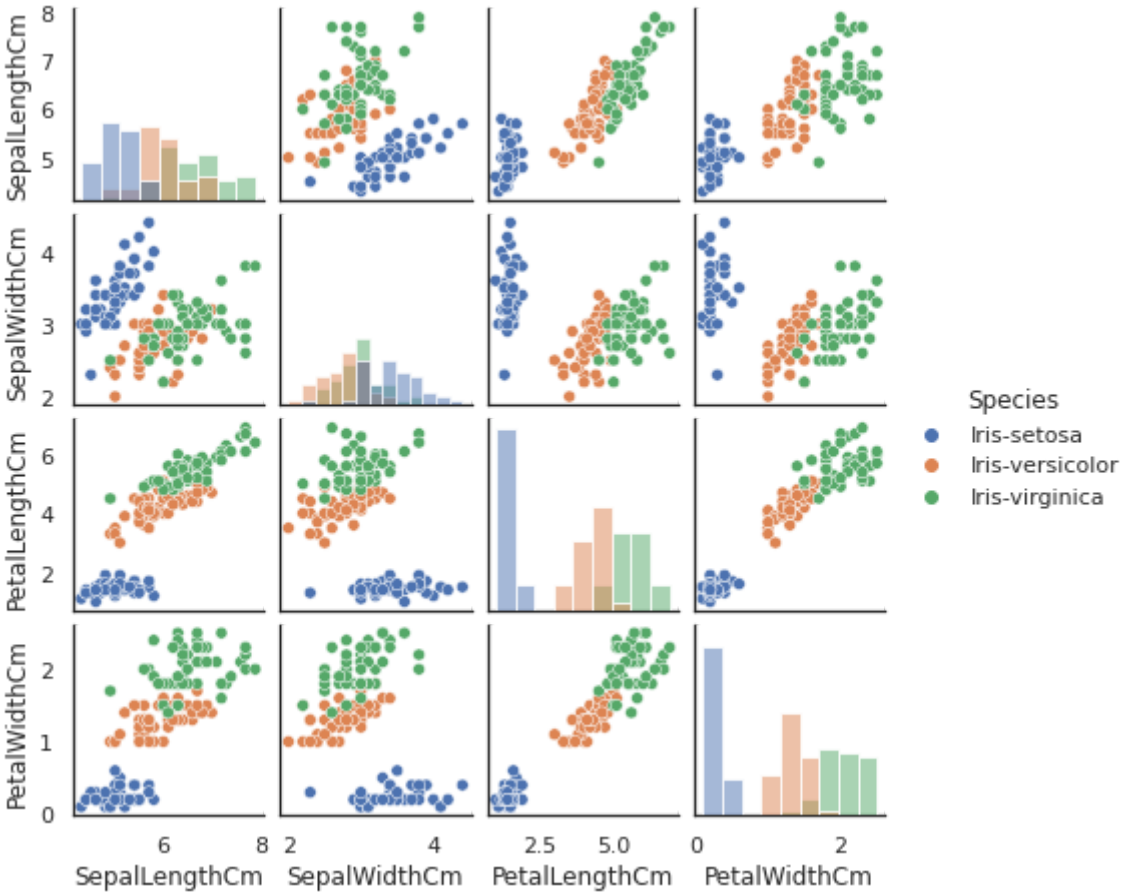
Note that the species are nearly linearly separable with petal size, but sepal sizes are more mixed.but a clustering algorithm might have a hard time realizing that there were three separate species, which we happen to know in advance - usually if you're doing exploratory data analysis (EDA), you don't know this, e.g. if you were looking for different groups of customers. it might not matter too much though - e.g. the versicolor and virginica species seem to be very similar, so it might be just as well for your purposes to lump them together.

In [9]:
```
# show petal length distributions in a swarm plot -
# just shows one dimension of the data, so not as useful as the previous plot
sns.swarmplot(x="Species", y="PetalLengthCm", data=data)
plt.grid()
```



In [10]:

```
# make a scatter matrix showing each pair of features in the data.
# seaborn can show the species labels as different colors, but
# normally with EDA you wouldn't have that information.
# note: the semicolon at the end just hides a line of text output.
sns.pairplot(data, hue="Species", diag_kind="hist", size=1.6);
```



So again, this shows how similar versicolor and virginica are, at least with the given features.but there could be features that you didn't measure that would more clearly separate the species. it's the same for any unsupervised learning - you need to have the right features to separate the groups in the best way.

Feature Scaling

In [12]:

```
#The data is unbalanced (eg sepallength ~4x petalwidth), so should do feature

from sklearn import preprocessing

scaler = preprocessing.StandardScaler()

scaler.fit(X)
X_scaled_array = scaler.transform(X)
X_scaled = pd.DataFrame(X_scaled_array, columns = X.columns)

X_scaled.sample(5)
```

Out[12]:

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
55	-0.173674	-0.587764	0.421564	0.133226
118	2.249683	-1.050569	1.786341	1.447956
149	0.068662	-0.124958	0.762759	0.790591
31	-0.537178	0.800654	-1.284407	-1.050031
37	-1.143017	0.106445	-1.284407	-1.444450

K-Means Clustering

In [13]:

```
# Try clustering on the 4d data and see if can reproduce the actual clusters.

# ie imagine we don't have the species labels on this data and wanted to
# divide the flowers into species. could set an arbitrary number of clusters
# and try dividing them up into similar clusters.

# we happen to know there are 3 species, so let's find 3 species and see
# if the predictions for each point matches the label in y.

from sklearn.cluster import KMeans

nclusters = 3 # this is the k in kmeans
seed = 0

km = KMeans(n_clusters=nclusters, random_state=seed)
km.fit(X_scaled)

# predict the cluster for each data point
y_cluster_kmeans = km.predict(X_scaled)
y_cluster_kmeans
```

Out[13]:

```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 2, 2, 2, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 2,
       0, 0, 0, 0, 2, 0, 0, 0, 0, 2, 2, 2, 0, 0, 0, 0, 0, 0, 0, 2, 2, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 2, 2, 2, 2, 0, 2, 2, 2,
       2, 2, 2, 0, 0, 2, 2, 2, 2, 0, 2, 0, 2, 0, 2, 2, 0, 2, 2, 2, 2, 2,
       2, 0, 0, 2, 2, 2, 0, 2, 2, 2, 0, 2, 2, 2, 0, 2, 2, 0], dtype=int32)
```

ordinarily, when you don't have the actual labels, you might use silhouette analysis to determine a good number of clusters k to use.

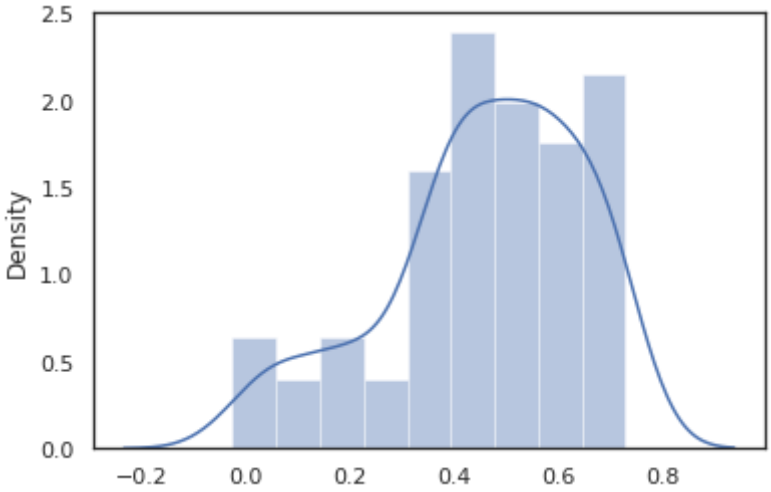
i.e. you would just run that same code for different values of k and print the value for the silhouette score.

let's see what that value is for the case we just did, k=3.

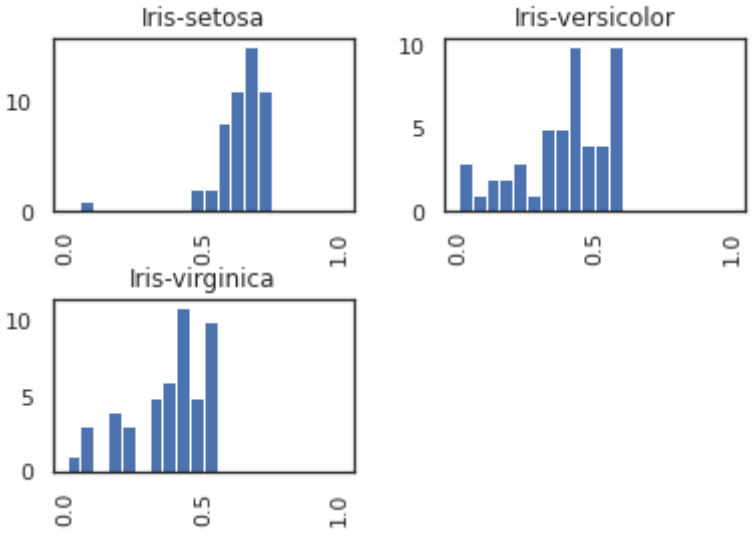
In [14]:
```
from sklearn import metrics
score = metrics.silhouette_score(X_scaled, y_cluster_kmeans)
score
```

Out[14]:
```
0.4589717867018717
```

In [15]:
```
# note that this is the mean over all the samples - there might be some clust
# that are well separated and others that are closer together.

# so let's look at the distribution of silhouette scores...

scores = metrics.silhouette_samples(X_scaled, y_cluster_kmeans)
sns.distplot(scores);
```

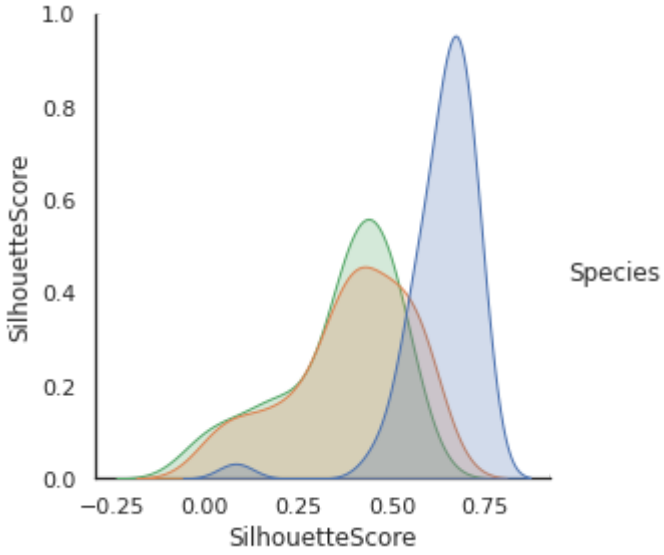


In [16]:
```
# can we add the species info to that plot?
# well, can plot them separately using pandas -
df_scores = pd.DataFrame()
df_scores['SilhouetteScore'] = scores
df_scores['Species'] = data['Species']
df_scores.hist(by='Species', column='SilhouetteScore', range=(0,1.0), bins=20
```



In [19]:
```
# so as expected, versicolor and virginica have lower silhouette scores than
# the more separate setosas, because they are closer together.
```

```
# can we put them all on one histogram?
# yes, with a bit of a hack - it's not in seaborn yet -
# see https://github.com/mwaskom/seaborn/issues/861
sns.pairplot(df_scores, hue="Species", size=4);
```



so you can see that the blue species have higher silhouette scores (the legend doesn't show the colors though... so the pandas plot is more useful). note that if we used the best mean silhouette score to try to find the best number of clusters k, we'd end up with 2 clusters, because the mean silhouette score in that case would be largest, since the clusters would be better separated. but, that's using k-means - gmm might give better results...

Principal Component Analysis (PCA)

So that was clustering on the orginal 4d data.

if you have a lot of features it can be helpful to do some feature reduction to avoid the curse of dimensionality (i.e. needing exponentially more data to do accurate predictions as the number of features grows).you can do this with Principal Component Analysis (PCA), which remaps the data to a new (smaller) coordinate system which tries to account for the most information possible. you can *also* use PCA to visualize the data by reducing the features to 2 dimensions and making a scatterplot. it kind of mashes the data down into 2d, so can lose information - but in this case it's just going from 4d to 2d, so not losing too much info. so let's just use it to visualize the data...

In [20]:
```
# mash the data down into 2 dimensions

from sklearn.decomposition import PCA

ndimensions = 2

pca = PCA(n_components=ndimensions, random_state=seed)
pca.fit(X_scaled)
X_pca_array = pca.transform(X_scaled)
X_pca = pd.DataFrame(X_pca_array, columns=['PC1','PC2']) # PC=principal compo
X_pca.sample(5)
```

Out[20]:

	PC1	PC2

	PC1	PC2
94	0.282944	-0.853951
16	-2.202750	1.513750
78	0.662126	-0.224346
6	-2.445711	0.074563
105	2.754197	0.788432

In [45]:

```
# so that gives us new 2d coordinates for each data point.
# at this point, if you don't have labelled data,
# you can add the k-means cluster ids to this table and make a
# colored scatterplot.

# we do actually have labels for the data points, but let's imagine
# we don't, and use the predicted labels to see what the predictions look lik
# first, convert species to an arbitrary number

#y_id_array = pd.Categorical.from_array(data['Species']).codes

y_id_array = data['Species'] = data['Species'].replace({0:'setosa',1:'versicc

df_plot = X_pca.copy()
df_plot['ClusterKmeans'] = y_cluster_kmeans
df_plot['SpeciesId'] = y_id_array # also add actual labels so we can use it i
df_plot.sample(5)
```

Out[45]:

	PC1	PC2	ClusterKmeans	SpeciesId
79	-0.047282	-1.057212	0	Iris-versicolor
59	0.004968	-1.029401	0	Iris-versicolor
100	1.847673	0.871697	2	Iris-virginica
114	1.464062	-0.444148	0	Iris-virginica
23	-1.820412	0.106751	1	Iris-setosa

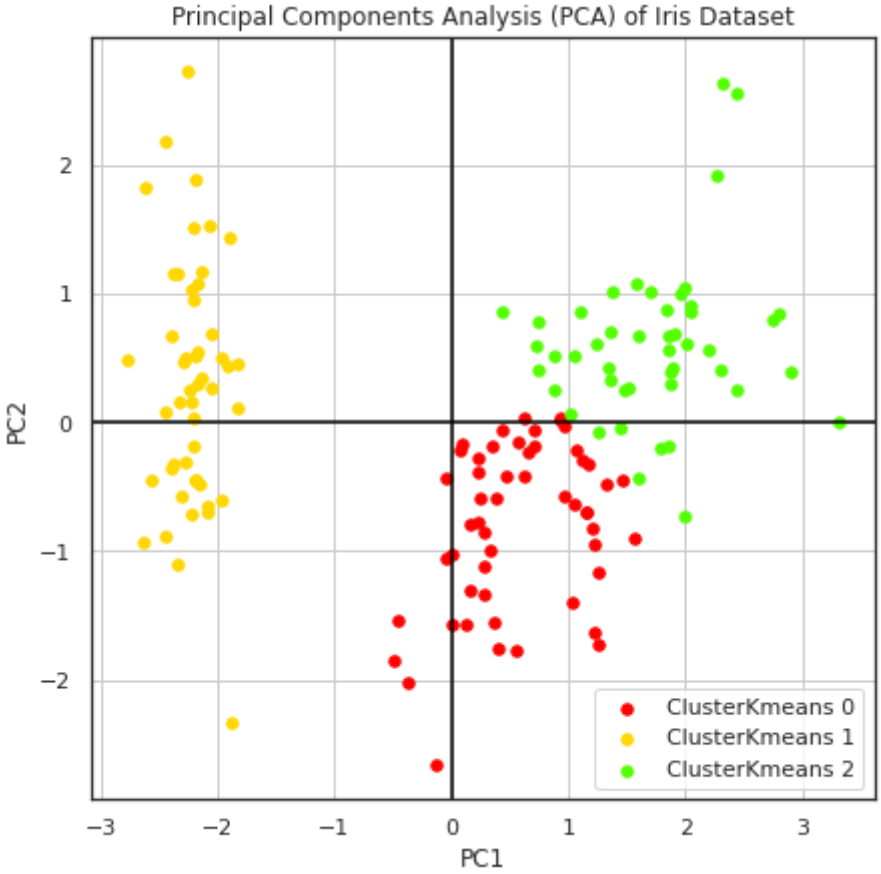
In [46]:

```
# so now we can make a 2d scatterplot of the clusters
# first define a plot fn

def plotData(df, groupby):
    "make a scatterplot of the first two principal components of the data, cc

    # make a figure with just one subplot.
    # you can specify multiple subplots in a figure,
    # in which case ax would be an array of axes,
    # but in this case it'll just be a single axis object.
    fig, ax = plt.subplots(figsize = (7,7))

    # color map
    cmap = mpl.cm.get_cmap('prism')

    # we can use pandas to plot each cluster on the same graph.
    # see http://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFr
    for i, cluster in df.groupby(groupby):
        cluster.plot(ax = ax, # need to pass this so all scatterplots are on
                     kind = 'scatter',
                     x = 'PC1', y = 'PC2',
                     color = cmap(i/(nclusters-1)), # cmap maps a number to a
```

```
                    label = "%s %i" % (groupby, i),
                    s=30) # dot size
    ax.grid()
    ax.axhline(0, color='black')
    ax.axvline(0, color='black')
    ax.set_title("Principal Components Analysis (PCA) of Iris Dataset");
```

In [47]:

```
# plot the clusters each datapoint was assigned to
plotData(df_plot, 'ClusterKmeans')
```



so the k-means clustering *did not* find the correct clusterings!

q. so what do these dimensions mean?

They're the principal components, which pick out the directions of maximal variation in the original data.

PC1 finds the most variation, PC2 the second-most.

The rest of the data is basically thrown away when the data is reduced down to 2d.

q. if these principal components represent some latent (hidden) features, what would those be?

maybe size (area) of the petals and sepals?

Gaussian Mixture Model (GMM) Clustering

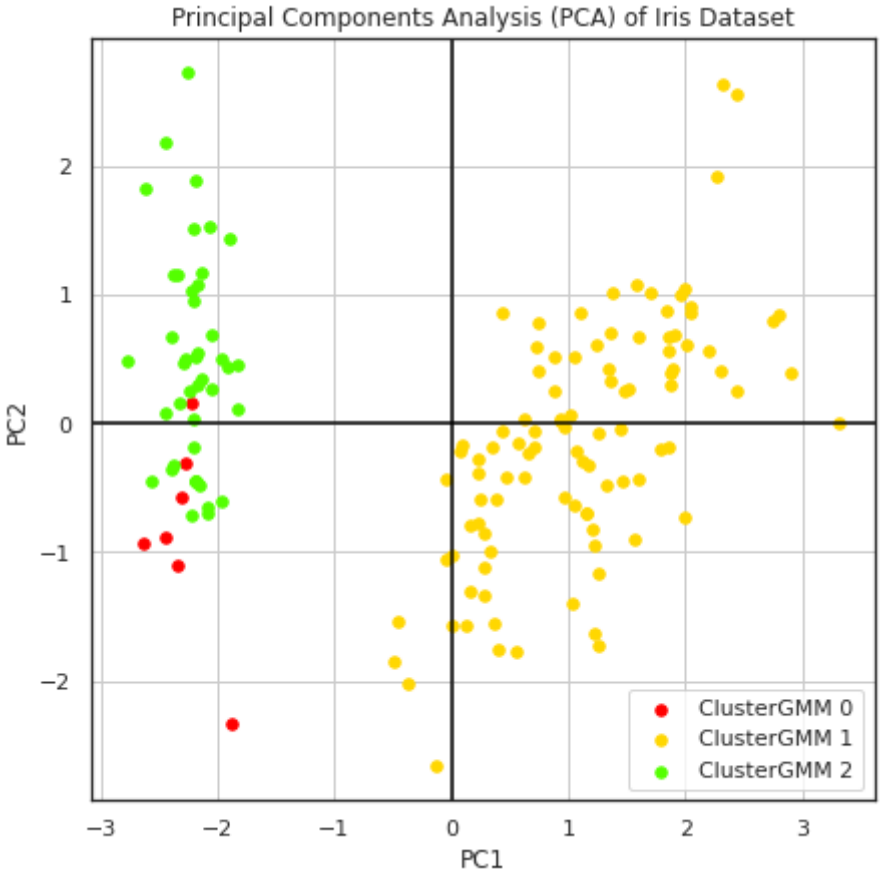
In [49]:

```
# now let's try GMM clustering, which tries to fit normally-distributed clust
# and might be the case when measuring things like petal and sepal sizes...
```

```
from sklearn.mixture import GaussianMixture

gmm = GaussianMixture(n_components=nclusters)
gmm.fit(X_scaled)

# predict the cluster for each data point
y_cluster_gmm = gmm.predict(X_scaled)
y_cluster_gmm
```

Out[49]:
```
array([2, 2, 2, 0, 2, 2, 2, 2, 0, 2, 2, 2, 2, 0, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 0, 2, 2, 2, 2, 0, 2, 2, 2, 2, 2, 2, 2, 2, 0, 2, 2, 0, 2, 2,
       2, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1])
```

In [50]:
```
# add the GMM clusters to our data table and plot them
df_plot['ClusterGMM'] = y_cluster_gmm
plotData(df_plot, 'ClusterGMM')
```



The GMM did much better at finding the actual species clusters! how did it do that?

GMM tries to fit normally distributed clusters, which is probably the case with this data,

So it fit it better. k-means is biased towards spherically distributed clusters.

Comparing k-Means and GMM clustering

In [52]:
```
# q. so how much better did the GMM do versus the K-means clustering? ie quar

# you can't just compare the SpeciesId with the cluster numbers, because they
```

```
# both arbitrarily assigned integers.

# but you can use the *adjusted Rand score* to quantify the goodness of the c
# as compared with SpeciesId (the true labels).

# e.g. this will give a perfect score of 1.0, even though the labels are reve
# adjusted_rand_score([0,0,1,1], [1,1,0,0]) # => 1.0

# see http://scikit-learn.org/stable/modules/generated/sklearn.metrics.adjust

from sklearn.metrics.cluster import adjusted_rand_score

# first let's see how the k-means clustering did -
score = adjusted_rand_score(y, y_cluster_kmeans)
score
```

Out[52]:
```
0.6201351808870379
```

In [53]:
```
# now the GMM clustering -
score = adjusted_rand_score(y, y_cluster_gmm)
score
```

Out[53]:
```
0.5073487662737015
```

Conclusion

Principal Component Analysis (PCA) is useful for visualizing high-dimensional datasets, as it can compress it down to 2 dimensions. It's also useful for reducing the dimensionality of high-dimensional datasets, which require exponentially more data as the number of dimensions increase, but we didn't need to do that in this case because the dataset was rather small.

k-Means Clustering is biased towards spherical distributions of clusters, and makes hard assignments to clusters, but is very fast (linear in number of features and data points).

Gaussian Mixture Model (GMM) Clustering handles ellipsoidal distributions, and makes 'soft' assignments to clusters, but is much slower than k-means for large datasets.

For this dataset, which was measuring what were probably normally distributed features, the GMM clustering worked better at finding the actual species labels, as measured by the adjusted Rand score.