Assignment 5

Problem Statement: Clustering of the iris dataset

Objective :

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

```
#IMPORTING LIBRARIES

import pandas as pd
import numpy as np
import seaborn as sns

from sklearn import preprocessing #for scaling data to fit
from sklearn.mixture import GaussianMixture #for implementing Guassian Mix
import matplotlib
import matplotlib.pyplot as plt #for plotting graphs
from sklearn.decomposition import PCA #for PCA
from sklearn.metrics.cluster import adjusted_rand_score #for calculating s
```

```
#LOADING THE DATASET

iris = pd.read_csv('/content/Iris (1).csv')
iris
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
0	1	5.1	3.5	1.4	0.2
1	2	4.9	3.0	1.4	0.2
2	3	4.7	3.2	1.3	0.2
3	4	4.6	3.1	1.5	0.2
4	5	5.0	3.6	1.4	0.2
...	...	...	...	...	...

```
iris.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 6 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Id             150 non-null    int64  
 1   SepalLengthCm  150 non-null    float64
 2   SepalWidthCm   150 non-null    float64
 3   PetalLengthCm  150 non-null    float64
 4   PetalWidthCm   150 non-null    float64
 5   Species        150 non-null    object 
dtypes: float64(4), int64(1), object(1)
memory usage: 7.2+ KB
```

```
iris.describe()
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	Petal
count	150.000000	150.000000	150.000000	150.000000	15
mean	75.500000	5.843333	3.054000	3.758667	
std	43.445368	0.828066	0.433594	1.764420	
min	1.000000	4.300000	2.000000	1.000000	
25%	38.250000	5.100000	2.800000	1.600000	
50%	75.500000	5.800000	3.000000	4.350000	
75%	112.750000	6.400000	3.300000	5.100000	
max	150.000000	7.900000	4.400000	6.900000	

```
#DROPPING UNWANTED COLUMN

iris.drop('Id', axis=1, inplace=True)
iris
```

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Sp
0	5.1	3.5	1.4	0.2	Iris-
1	4.9	3.0	1.4	0.2	Iris-
2	4.7	3.2	1.3	0.2	Iris-
3	4.6	3.1	1.5	0.2	Iris-
4	5.0	3.6	1.4	0.2	Iris-
...	...	...	...	...	
145	6.7	3.0	5.2	2.3	Iris-v
146	6.3	2.5	5.0	1.9	Iris-v
147	6.5	3.0	5.2	2.0	Iris-v
148	6.2	3.4	5.4	2.3	Iris-v
149	5.9	3.0	5.1	1.8	Iris-v

150 rows × 5 columns

```
#CHECKING IF NULL VALUES PRESENT

iris.isnull().sum()
```

```
SepalLengthCm    0
SepalWidthCm     0
PetalLengthCm    0
PetalWidthCm     0
Species          0
dtype: int64
```

```
#Split data into features (x) and labels(y)

X = iris.iloc[:, 0:4]
Y = iris.iloc[:, -1]
```

X

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2
...	...	...	...	...
145	6.7	3.0	5.2	2.3
146	6.3	2.5	5.0	1.9
147	6.5	3.0	5.2	2.0
148	6.2	3.4	5.4	2.3
149	5.9	3.0	5.1	1.8

150 rows × 4 columns

Y

```
0          Iris-setosa
1          Iris-setosa
2          Iris-setosa
3          Iris-setosa
4          Iris-setosa
              ...
145    Iris-virginica
146    Iris-virginica
147    Iris-virginica
148    Iris-virginica
149    Iris-virginica
Name: Species, Length: 150, dtype: object
```

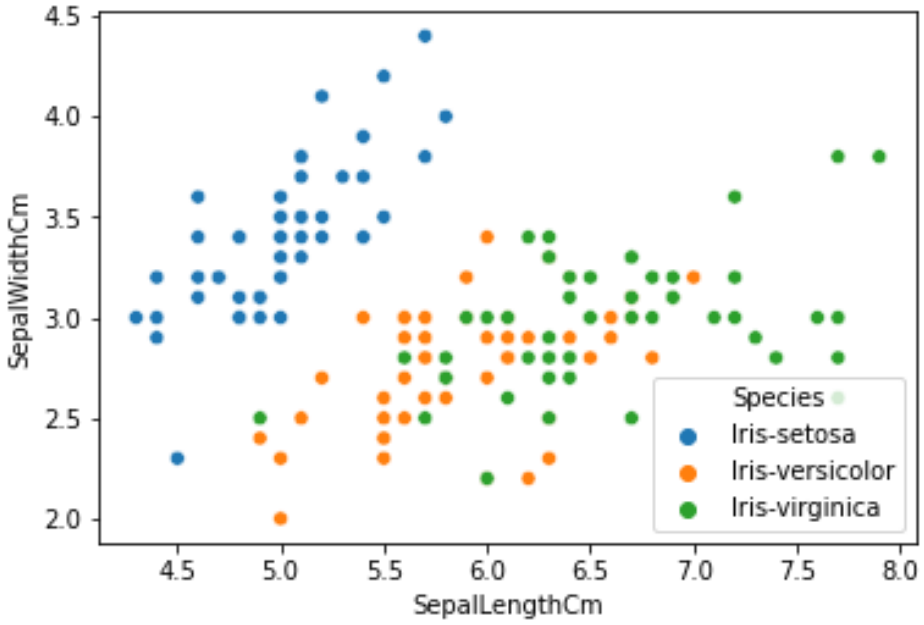
Visualize Data

```
iris['Species'].value_counts()
```

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

```
sns.scatterplot(iris['SepalLengthCm'], iris['SepalWidthCm'], hue=iris['Sp
```

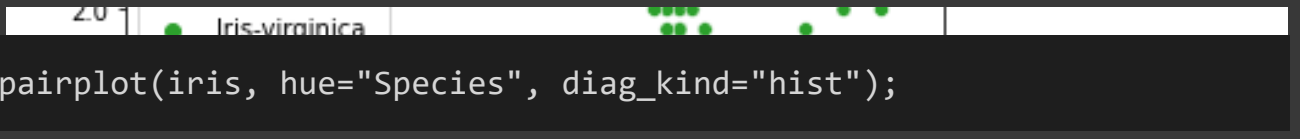
```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: Fu
  FutureWarning
```




```
sns.scatterplot(iris['PetalLengthCm'], iris['PetalWidthCm'], hue=iris['Sp
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: Fu
```

Species are nearly linearly separable with petal size, but sepal sizes are more mixed. So plotting a scatter matrix showing each pair of features in the data.




```
sns.pairplot(iris, hue="Species", diag_kind="hist");
```

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.

Feature Scaling

The data is unbalanced (eg sepallength ~4x petalwidth), so should do feature scaling, otherwise the larger features will dominate the others in clustering, etc.

```
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)
```

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
8	-1.748856	-0.356361	-1.341272	-1.312977
40	-1.021849	1.032057	-1.398138	-1.181504
83	0.189830	-0.819166	0.762759	0.527645
87	0.553333	-1.744778	0.364699	0.133226
2	-1.385353	0.337848	-1.398138	-1.312977

```
X_scaled
```

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
0	-0.900681	1.032057	-1.341272	-1.312977
1	-1.143017	-0.124958	-1.341272	-1.312977
2	-1.385353	0.337848	-1.398138	-1.312977
3	-1.506521	0.106445	-1.284407	-1.312977
4	-1.021849	1.263460	-1.341272	-1.312977
...	...	...	...	...

```
Y
```

```
0         Iris-setosa
1         Iris-setosa
2         Iris-setosa
3         Iris-setosa
4         Iris-setosa
             ...
145    Iris-virginica
146    Iris-virginica
147    Iris-virginica
148    Iris-virginica
149    Iris-virginica
Name: Species, Length: 150, dtype: object
```

1. Perform clustering of the iris dataset based on all variables using Gaussian mixture models.

```
nclusters = 3
gmm = GaussianMixture(n_components=nclusters)
gmm.fit(X_scaled)
```

```
GaussianMixture(covariance_type='full', init_params='kmeans', max_it
                means_init=None, n_components=3, n_init=1, precisior
                random_state=None, reg_covar=1e-06, tol=0.001, verbc
                verbose_interval=10, warm_start=False, weights_init=
```

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

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 1, 2, 1, 2, 1, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 1, 2, 2,
       2, 2, 2, 2, 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])
```

2) Using PCA to visualize clusters.

```
ndimensions = 2

pca = PCA(n_components=ndimensions, random_state=142)
pca.fit(X_scaled)
X_pca_array = pca.transform(X_scaled)
X_pca = pd.DataFrame(X_pca_array, columns=['PC1','PC2']) # PC=principal co
X_pca.sample(5)
```

	PC1	PC2
143	2.043308	0.864685
3	-2.304197	-0.575368
17	-2.190179	0.514304
98	-0.457013	-1.539465
68	1.215303	-1.633356

```
y_id_array = pd.Categorical(iris['Species']).codes

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

	PC1	PC2	ClusterGMM	SpeciesId
44	-2.133373	1.171432	0	0
62	0.551634	-1.772582	2	1
144	2.001691	1.048550	1	2
93	-0.373628	-2.017932	2	1
113	1.253087	1.167391	1	2

So now we can make a 2d scatterplot of the clusters first define a plot fn.

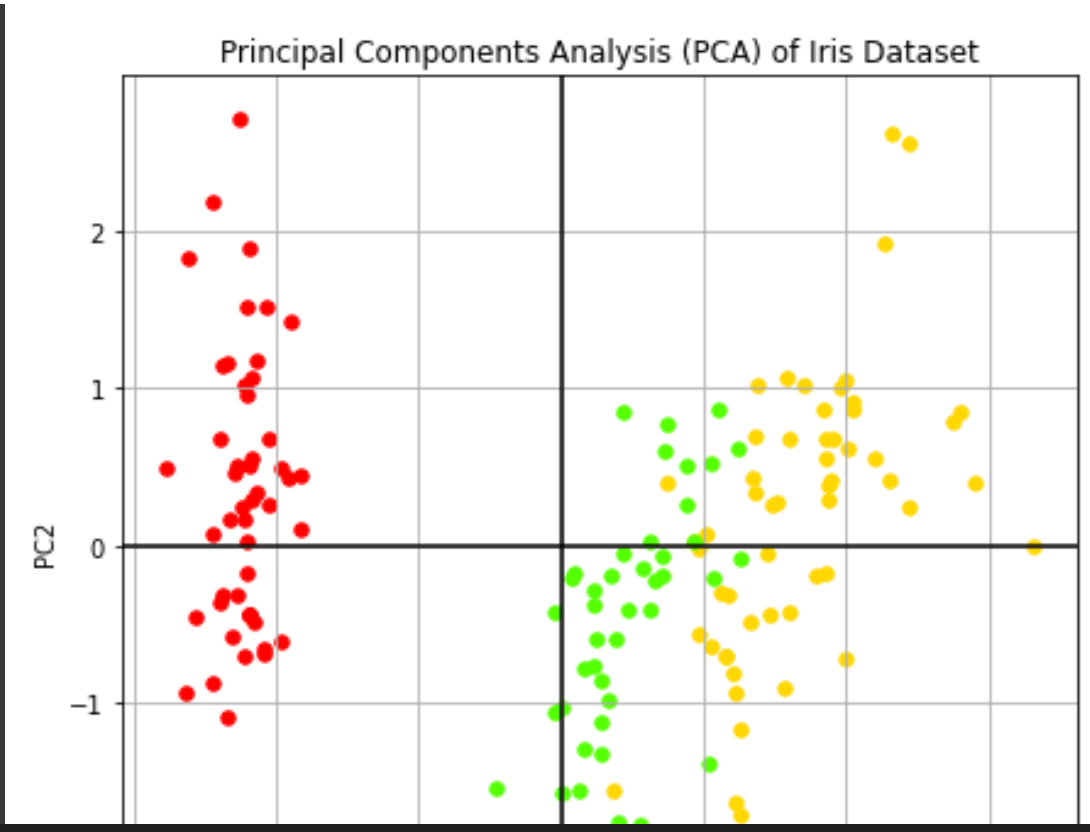
```
def plotData(df, groupby):
    "make a scatterplot of the first two principal components of the data,

    # making a figure with just one subplot.
    fig, ax = plt.subplots(figsize = (7,7))

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

    # Using pandas to plot each cluster on the same graph.
    for i, cluster in df.groupby(groupby):
        cluster.plot(ax = ax, # need to pass this so all scatterplots are
                     kind = 'scatter',
                     x = 'PC1', y = 'PC2',
                     color = cmap(i/(nclusters-1)), # cmap maps a number
                     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");
```

```
df_plot['ClusterGMM'] = y_cluster_gmm
plotData(df_plot, 'ClusterGMM')
```




```
#Checking score

score = adjusted_rand_score(Y, y_cluster_gmm)
score
```

```
0.456660389383244 94
```

Conclusion :

GMM clustering matched the true labels closely.