GRIP : The Sparks Foundation

Data Science and Business Analytics

Author : Kuldeep Dwivedi

Task 2 : Prediction Using Unsupervised Learning.

Dataset : Iris.csv (https://bit.ly/3kXTdox)

Algorithm used here : K-Means Clustering

In [9]:
```
%matplotlib inline

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

from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

from sklearn.cluster import KMeans
from sklearn.decomposition import PCA

df=pd.read_csv('Iris.csv')
df
```

Out[9]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa
...	...	...	...	...	...	...
145	146	6.7	3.0	5.2	2.3	Iris-virginica
146	147	6.3	2.5	5.0	1.9	Iris-virginica
147	148	6.5	3.0	5.2	2.0	Iris-virginica
148	149	6.2	3.4	5.4	2.3	Iris-virginica
149	150	5.9	3.0	5.1	1.8	Iris-virginica

150 rows × 6 columns

K-Means is considered an unsupervised learning algorthm. This means you only need a features

matrix. In the iris dataset, there are four features. In this notebook, the features matrix will only be two features as it is easier to visualize clusters in two dimensions. KMeans is a popular clustering algorithm that we can use to find structure in our data.

In [12]:
```
df.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    int32
dtypes: float64(4), int32(1), int64(1)
memory usage: 6.6 KB
```

Arrange Data into Feature Matrix

In [15]:
```
features = ['PetalLengthCm','PetalWidthCm']

# Create features matrix
x = df.loc[:, features].values

x
```

Out[15]:
```
array([[1.4, 0.2],
       [1.4, 0.2],
       [1.3, 0.2],
       [1.5, 0.2],
       [1.4, 0.2],
       [1.7, 0.4],
       [1.4, 0.3],
       [1.5, 0.2],
       [1.4, 0.2],
       [1.5, 0.1],
       [1.5, 0.2],
       [1.6, 0.2],
       [1.4, 0.1],
       [1.1, 0.1],
       [1.2, 0.2],
       [1.5, 0.4],
       [1.3, 0.4],
       [1.4, 0.3],
       [1.7, 0.3],
       [1.5, 0.3],
       [1.7, 0.2],
       [1.5, 0.4],
       [1. , 0.2],
       [1.7, 0.5],
       [1.9, 0.2],
       [1.6, 0.2],
       [1.6, 0.4],
       [1.5, 0.2],
       [1.4, 0.2],
       [1.6, 0.2],
       [1.6, 0.2],
       [1.5, 0.4],
```

```
       [1.5, 0.1],
       [1.4, 0.2],
       [1.5, 0.1],
       [1.2, 0.2],
       [1.3, 0.2],
       [1.5, 0.1],
       [1.3, 0.2],
       [1.5, 0.2],
       [1.3, 0.3],
       [1.3, 0.3],
       [1.3, 0.2],
       [1.6, 0.6],
       [1.9, 0.4],
       [1.4, 0.3],
       [1.6, 0.2],
       [1.4, 0.2],
       [1.5, 0.2],
       [1.4, 0.2],
       [4.7, 1.4],
       [4.5, 1.5],
       [4.9, 1.5],
       [4. , 1.3],
       [4.6, 1.5],
       [4.5, 1.3],
       [4.7, 1.6],
       [3.3, 1. ],
       [4.6, 1.3],
       [3.9, 1.4],
       [3.5, 1. ],
       [4.2, 1.5],
       [4. , 1. ],
       [4.7, 1.4],
       [3.6, 1.3],
       [4.4, 1.4],
       [4.5, 1.5],
       [4.1, 1. ],
       [4.5, 1.5],
       [3.9, 1.1],
       [4.8, 1.8],
       [4. , 1.3],
       [4.9, 1.5],
       [4.7, 1.2],
       [4.3, 1.3],
       [4.4, 1.4],
       [4.8, 1.4],
       [5. , 1.7],
       [4.5, 1.5],
       [3.5, 1. ],
       [3.8, 1.1],
       [3.7, 1. ],
       [3.9, 1.2],
       [5.1, 1.6],
       [4.5, 1.5],
       [4.5, 1.6],
       [4.7, 1.5],
       [4.4, 1.3],
       [4.1, 1.3],
       [4. , 1.3],
       [4.4, 1.2],
       [4.6, 1.4],
       [4. , 1.2],
       [3.3, 1. ],
       [4.2, 1.3],
       [4.2, 1.2],
       [4.2, 1.3],
```

```
       [4.3, 1.3],
       [3. , 1.1],
       [4.1, 1.3],
       [6. , 2.5],
       [5.1, 1.9],
       [5.9, 2.1],
       [5.6, 1.8],
       [5.8, 2.2],
       [6.6, 2.1],
       [4.5, 1.7],
       [6.3, 1.8],
       [5.8, 1.8],
       [6.1, 2.5],
       [5.1, 2. ],
       [5.3, 1.9],
       [5.5, 2.1],
       [5. , 2. ],
       [5.1, 2.4],
       [5.3, 2.3],
       [5.5, 1.8],
       [6.7, 2.2],
       [6.9, 2.3],
       [5. , 1.5],
       [5.7, 2.3],
       [4.9, 2. ],
       [6.7, 2. ],
       [4.9, 1.8],
       [5.7, 2.1],
       [6. , 1.8],
       [4.8, 1.8],
       [4.9, 1.8],
       [5.6, 2.1],
       [5.8, 1.6],
       [6.1, 1.9],
       [6.4, 2. ],
       [5.6, 2.2],
       [5.1, 1.5],
       [5.6, 1.4],
       [6.1, 2.3],
       [5.6, 2.4],
       [5.5, 1.8],
       [4.8, 1.8],
       [5.4, 2.1],
       [5.6, 2.4],
       [5.1, 2.3],
       [5.1, 1.9],
       [5.9, 2.3],
       [5.7, 2.5],
       [5.2, 2.3],
       [5. , 1.9],
       [5.2, 2. ],
       [5.4, 2.3],
       [5.1, 1.8]])
```

In [18]:

```
from sklearn import preprocessing
le=preprocessing.LabelEncoder()

df.Species=le.fit_transform(df.Species.values)
df.Species
```

Out[18]:

```
0      0
1      0
2      0
```

```
3      0
4      0
      ..
145    2
146    2
147    2
148    2
149    2
Name: Species, Length: 150, dtype: int64
```

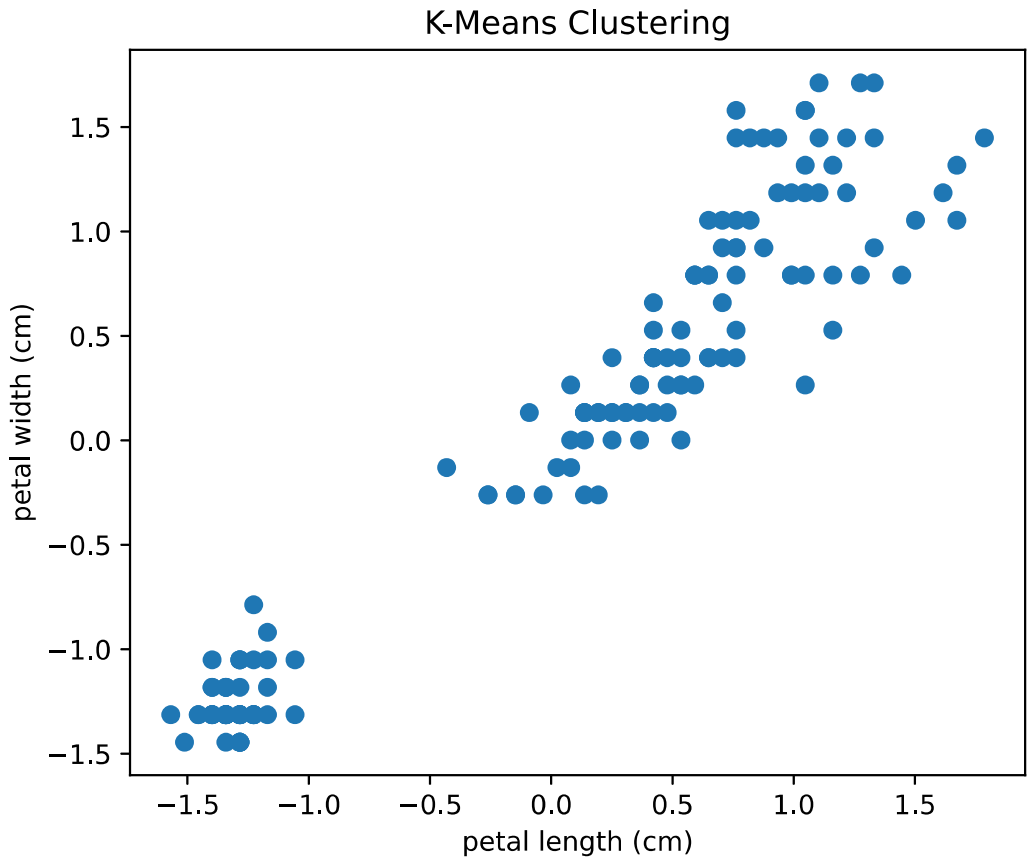
Standardize the data

In [21]:
```
x=StandardScaler().fit_transform(x)
```

Plot data to estimate number of clusters

In [24]:
```
X=pd.DataFrame(x,columns=features)
plt.figure(figsize=(6,5))
plt.scatter(X['PetalLengthCm'], X['PetalWidthCm'])
plt.xlabel('petal length (cm)')
plt.ylabel('petal width (cm)');
plt.title('K-Means Clustering')
```

Out[24]:
```
Text(0.5, 1.0, 'K-Means Clustering')
```



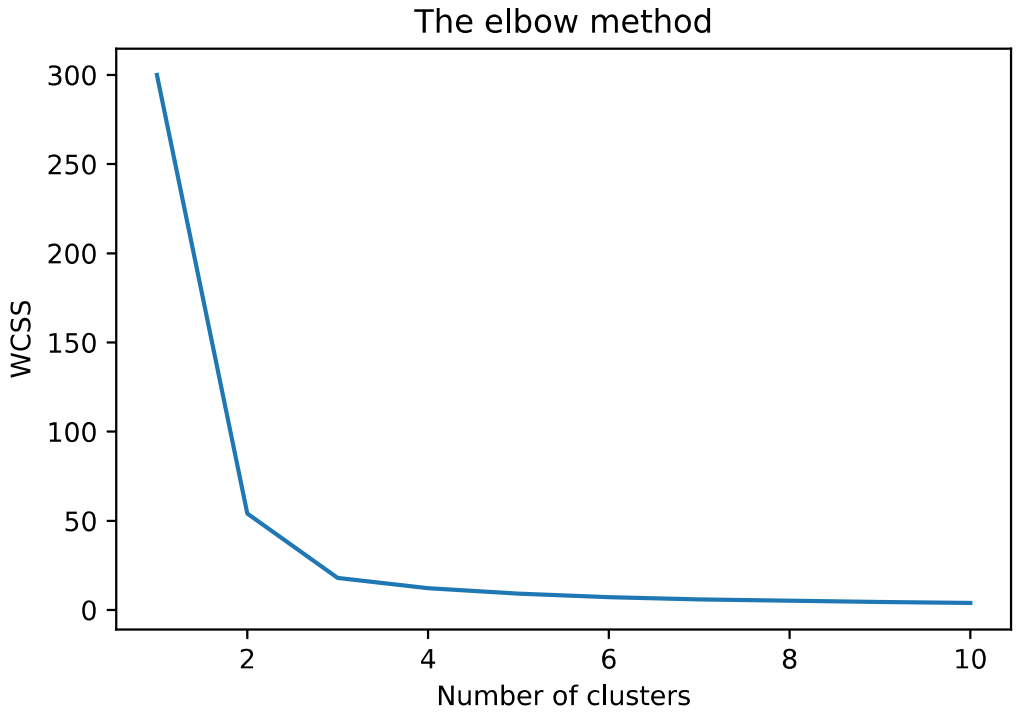
Finding the optimum number of clusters for K-means clustering

In [27]:
```
# Finding the optimum number of clusters for k-means classification
wcss = []

for i in range(1, 11):
```

```
    kmeans = KMeans(n_clusters = i, init = 'k-means++',
                    max_iter = 300, n_init = 10, random_state = 0)
    kmeans.fit(x)
    wcss.append(kmeans.inertia_)

# Plotting the results onto a line graph,
# `allowing us to observe 'The elbow'
plt.plot(range(1, 11), wcss)
plt.title('The elbow method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS') # Within cluster sum of squares
plt.show()
```



It is called 'The elbow method' from the above graph, the optimum clusters is where the elbow occurs. This is when the within cluster sum of squares (WCSS) doesn't decrease significantly with every iteration. From this we choose the number of clusters as '3'.

K-Means Clustering

In [30]:
```
# Make an instance of KMeans with 3 clusters
kmeans = KMeans(n_clusters=3, random_state=1)

# Fit only on a features matrix
kmeans.fit(x)
```

Out[30]:
```
KMeans(n_clusters=3, random_state=1)
```

In [33]:
```
# Get labels and cluster centroids
labels = kmeans.labels_
centroids = kmeans.cluster_centers_

labels
```

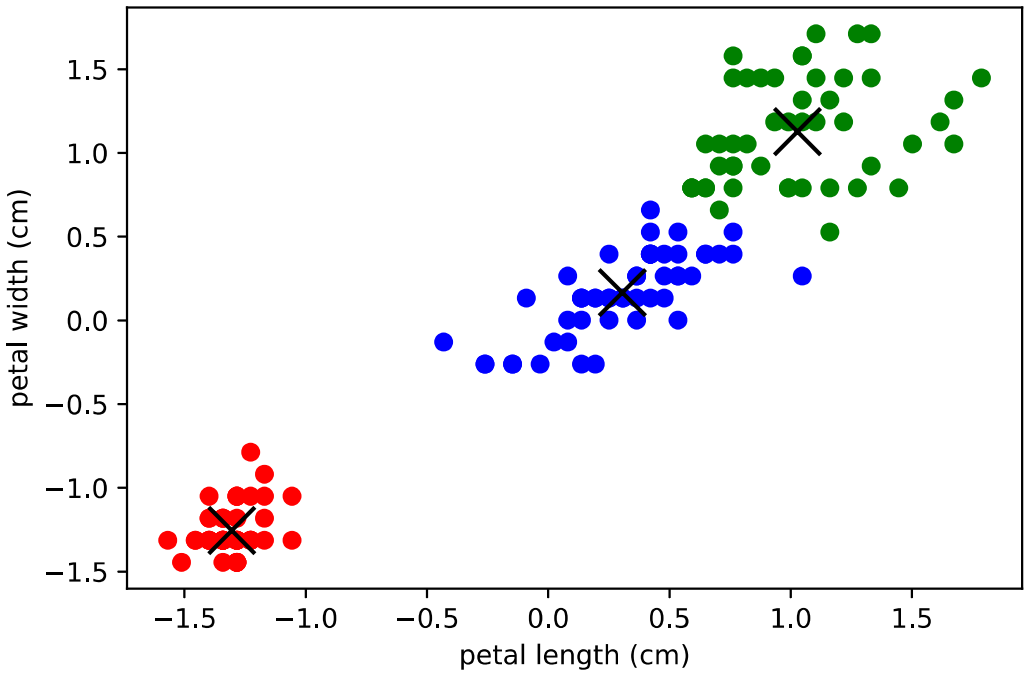
Out[33]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1])
```

In [36]:
```
centroids
```

Out[36]:
```
array([[-1.30487835, -1.25512862],
       [ 1.02813193,  1.12749028],
       [ 0.30564587,  0.16609419]])
```

In [39]:
```
colormap = np.array(['r', 'g', 'b'])
plt.scatter(X['PetalLengthCm'], X['PetalWidthCm'], c=colormap[labels])
plt.scatter(centroids[:,0], centroids[:,1], s = 300, marker = 'x', c = 'k')

plt.xlabel('petal length (cm)')
plt.ylabel('petal width (cm)');
```



Visually Evaluate the clusters and compare the species

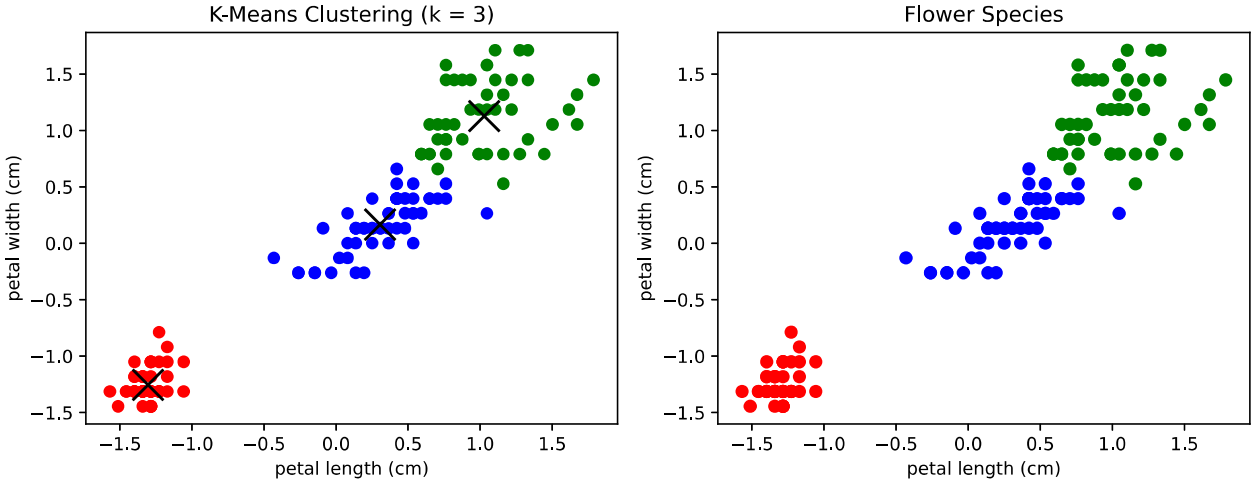
In [42]:
```
plt.figure(figsize=(10,4))

plt.subplot(1, 2, 1)
plt.scatter(X['PetalLengthCm'], X['PetalWidthCm'], c=colormap[labels])
plt.scatter(centroids[:,0], centroids[:,1], s = 300, marker = 'x', c = 'k')
plt.xlabel('petal length (cm)')
plt.ylabel('petal width (cm)')
plt.title('K-Means Clustering (k = 3)')

plt.subplot(1, 2, 2)
plt.scatter(X['PetalLengthCm'], X['PetalWidthCm'], c=colormap[labels], s=40)
plt.xlabel('petal length (cm)')
plt.ylabel('petal width (cm)')
```

```
plt.title('Flower Species')

plt.tight_layout()
```



They look pretty similar. Looks like KMeans picked up flower differences with only two features and not the labels. The colors are different in the two graphs simply because KMeans gives out a arbitrary cluster number and the iris dataset has an arbitrary number in the target column.

PCA Projection in 2D

The original data has 4 columns (sepal length, sepal width, petal length, and petal width). The code below projects the original data which is 4 dimensional into 2 dimensions. Note that after dimensionality reduction, there usually isn't a particular meaning assigned to each principal component. The new components are just the two main dimensions of variation

In [45]:
```
pca = PCA(n_components=2)

# Fit and transform the data
principalComponents = pca.fit_transform(x)

principalDf = pd.DataFrame(data = principalComponents, columns = ['principal component

df=pd.read_csv('Iris.csv')
```

2D Projection

In [48]:
```
finalDf = pd.concat([principalDf, df[['Species']]], axis = 1)

finalDf
```

Out[48]:

	principal component 1	principal component 2	Species
0	-1.876838	0.020008	Iris-setosa
1	-1.876838	0.020008	Iris-setosa
2	-1.917048	0.060218	Iris-setosa
3	-1.836627	-0.020202	Iris-setosa
4	-1.876838	0.020008	Iris-setosa

	principal component 1	principal component 2	Species
...	...	...	...
145	1.603421	0.444297	Iris-virginica
146	1.151139	0.152856	Iris-virginica
147	1.324525	0.165401	Iris-virginica
148	1.683841	0.363877	Iris-virginica
149	1.098384	0.019680	Iris-virginica

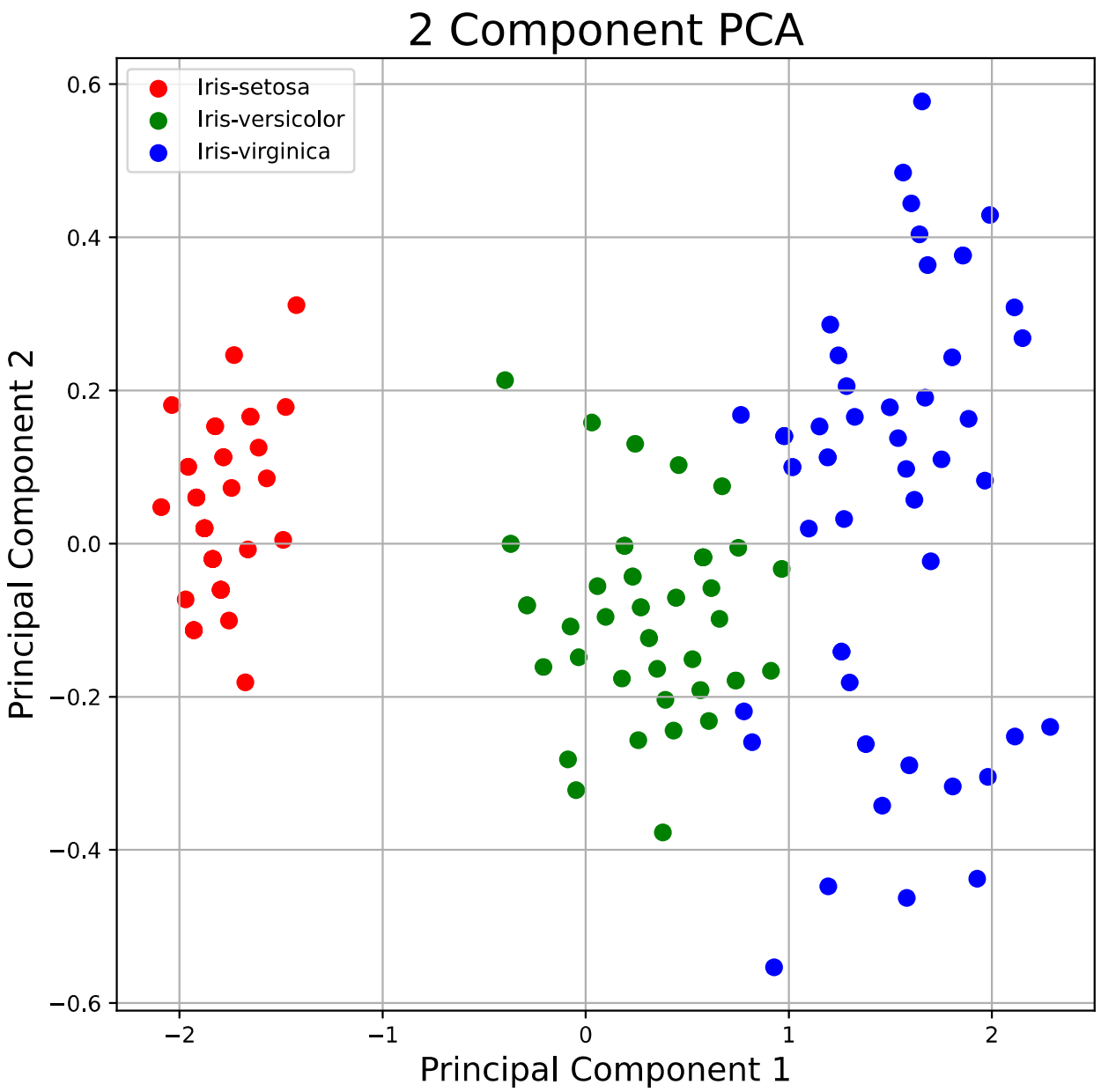
150 rows × 3 columns

In [50]:

```
fig, ax = plt.subplots(nrows = 1, ncols = 1, figsize = (8,8));
targets = df.loc[:, 'Species'].unique()
colors = ['r', 'g', 'b']

for target, color in zip(targets,colors):
    indicesToKeep = finalDf['Species'] == target
    ax.scatter(finalDf.loc[indicesToKeep, 'principal component 1']
               , finalDf.loc[indicesToKeep, 'principal component 2']
               , c = color
               , s = 50)

ax.set_xlabel('Principal Component 1', fontsize = 15)
ax.set_ylabel('Principal Component 2', fontsize = 15)
ax.set_title('2 Component PCA', fontsize = 20)
ax.legend(targets)
ax.grid()
```



From the graph, it looks like the setosa class is well separated from the versicolor and virginica classes.

Explained Varience

The explained variance tells us how much information (variance) can be attributed to each of the principal components. This is important as while you can convert 4 dimensional space to 2 dimensional space, you lose some of the variance (information) when you do this.

In [51]:
```
pca.explained_variance_ratio_
```

Out[51]:
```
array([0.98137855, 0.01862145])
```

In [52]:
```
sum(pca.explained_variance_ratio_)
```

Out[52]:
```
1.0
```

Together, the two principal components contain 100% of the information. The first principal component contains about 98% of the variance. The second principal component contains about 1.8% of the variance. PCA can be used to help visualize our data. Thank you!