Importing libraries and Iris dataset

```
%matplotlib inline
```

```
from sklearn.cluster import KMeans
from sklearn import metrics
from scipy.spatial.distance import cdist
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import sklearn
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import AgglomerativeClustering
```

```
columns=['sepal_length','sepal_width','petal_length','petal_width','species']
df = pd.read_csv(r"C:/Users/Aadya/Downloads/iris_dataset.csv")
```

```
df.columns = columns
df.head()
```

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

Data Exploration

```
#Information about the dataset
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   sepal_length  150 non-null    float64
 1   sepal_width   150 non-null    float64
 2   petal_length  150 non-null    float64
 3   petal_width   150 non-null    float64
 4   species       150 non-null    object 
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
```

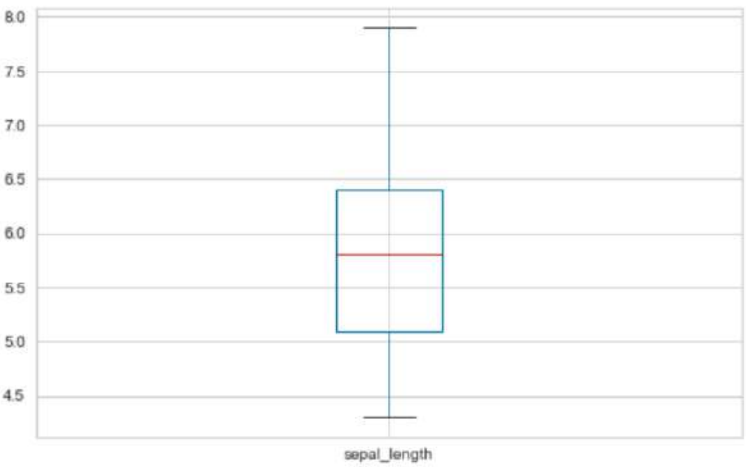
```
#Data types of the dataset columns
df.dtypes
```

```
sepal_length    float64
sepal_width     float64
petal_length    float64
petal_width     float64
species          object
dtype: object
```

```
#Total memory used by the dataset
df.memory_usage().sum()
```

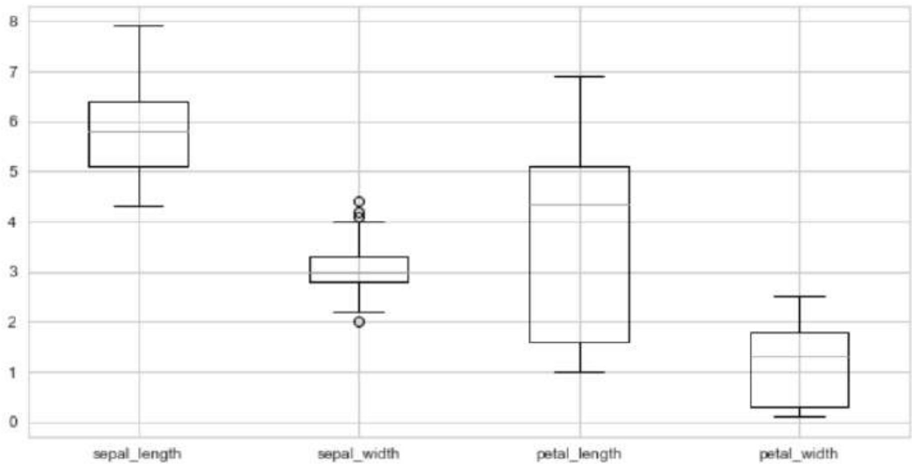
```
6128
```

```
#Boxplot of a column
df['sepal_length'].plot.box(figsize=(8,5));
```

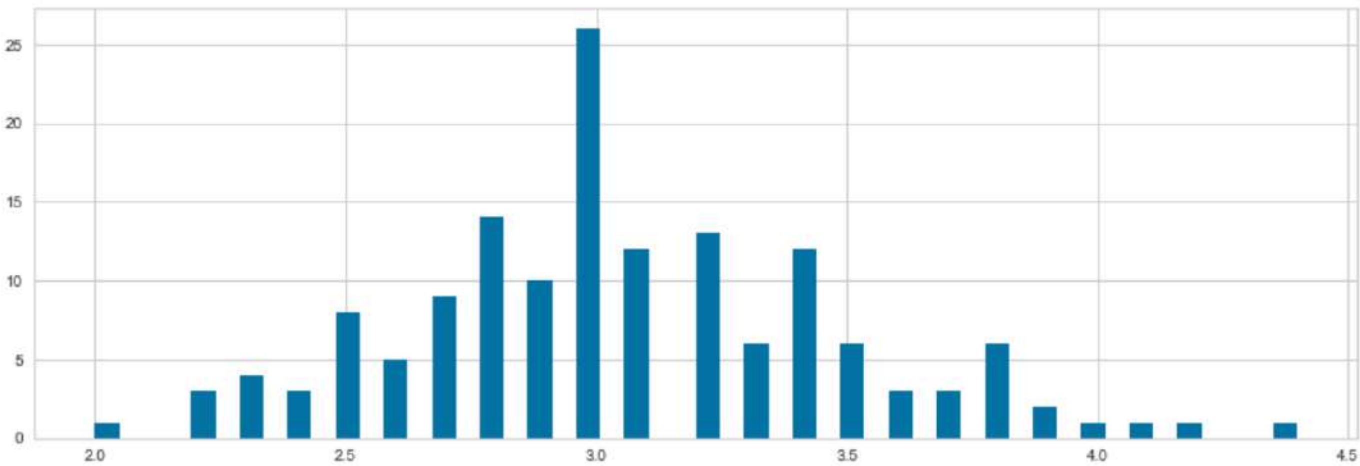


```python
#Boxplot of all the columns with numerical data
df.boxplot(figsize=(10,5))
```

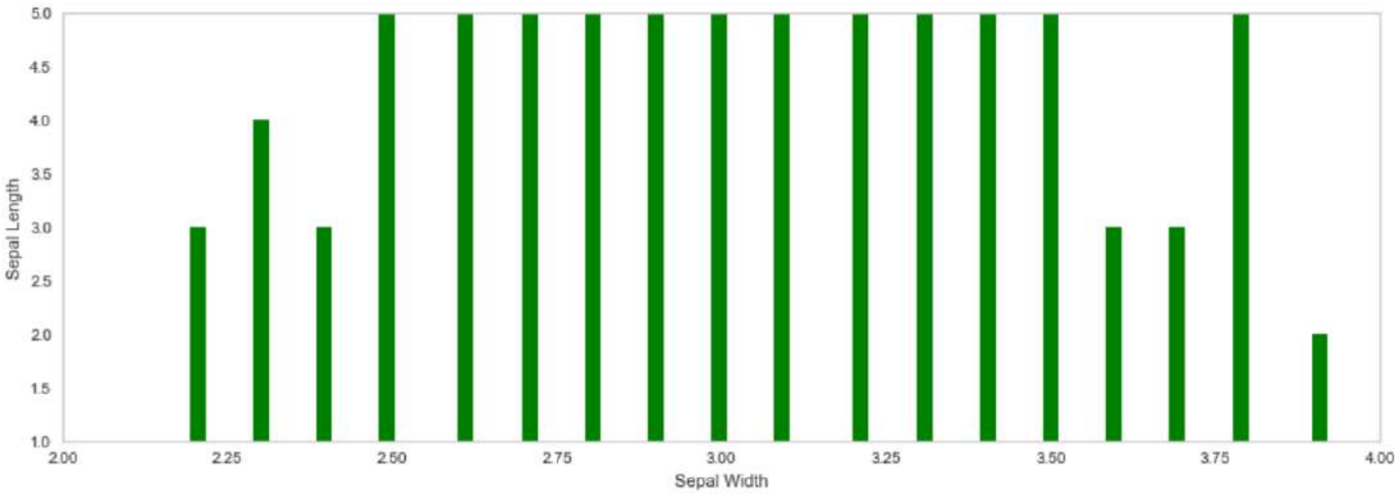
<AxesSubplot:>



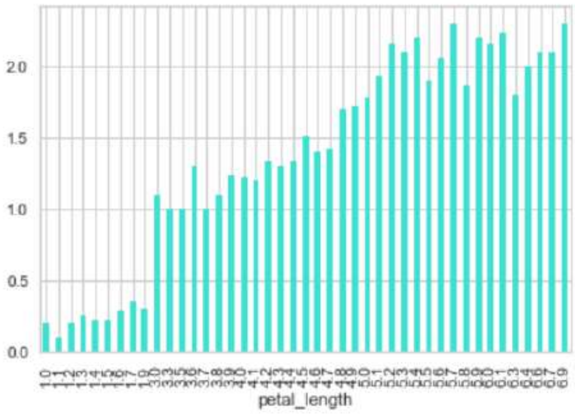
```
#Histogram
df['sepal_width'].hist(bins=50, figsize=(15, 5));
```



```
ax = df['sepal_width'].hist(bins=100, grid=False, color='green', figsize=(15, 5)) # grid turned off and color changed
ax.set_xlabel('Sepal Width')
ax.set_ylabel('Sepal Length')
ax.set_xlim(2,4) #limiting display range to 2-4 for the x-axis
ax.set_ylim(1,5); #limiting display range to 1-5 for the y-axis
```



```
#Barplot with Petal width as dependent variable
df_petal_width = df.groupby('petal_length')['petal_width'].mean()
df_petal_width[:].plot.bar(color='turquoise');
```



Data Cleaning

```
#Check if there are missing values in the dataset
df.isnull().sum().sum()
```

```
0
```

```
#Check if there are duplicate rows in the dataset
df.duplicated().sum()
```

```
3
```

```
#Removing duplicates from the dataset
df.drop_duplicates(keep="first",inplace=True)
```

```
df.isnull().sum().sum()
```

```
0
```

```
#Check if duplicate rows have been removed successfully from the dataset
df.duplicated().sum()
```

```
0
```

```
#No. of rows in the dataset after cleaning
print(len(df.axes[0]))
```

```
147
```

```python
#Statistics for all the dataset columns
df.describe()
```

	sepal_length	sepal_width	petal_length	petal_width
count	147.000000	147.000000	147.000000	147.000000
mean	5.856463	3.055782	3.780272	1.208844
std	0.829100	0.437009	1.759111	0.757874
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.400000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

```python
#Variance
df.var()
```

```
sepal_length    0.687407
sepal_width     0.190977
petal_length    3.094471
petal_width     0.574373
dtype: float64
```

```python
#Skewness
df.skew()
```

```
sepal_length    0.292560
sepal_width     0.324351
petal_length   -0.293763
petal_width    -0.113479
dtype: float64
```

```
#Kurtosis
df.kurtosis()
```

```
sepal_length   -0.556956
sepal_width     0.246838
petal_length   -1.374462
petal_width    -1.317760
dtype: float64
```

Label encoding to convert all columns into numbers

```
df['species'].replace({'Iris-setosa':0,'low':1,'Iris-versicolor':2,'Iris-virginica':3},inplace=True)
```

K-means clustering using all the data columns

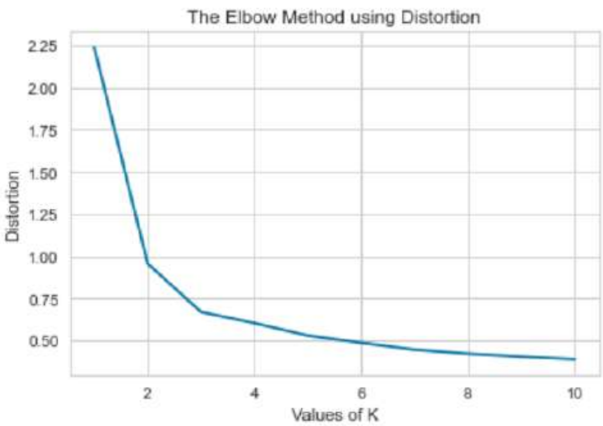
K-means using Elbow method

```
#K-means Model building for Elbow method
distortions=[]
inertias=[]
mapping1={}
mapping2={}

for k in range(1,11):

    # Building and fitting the model
    kmeanModel=KMeans(n_clusters=k).fit(df)
    kmeanModel.fit(df)

    #Calculating Distortion and Inertia for each K-value
    distortions.append(sum(np.min(cdist(df, kmeanModel.cluster_centers_,'euclidean'), axis=1))/df.shape[0])
    inertias.append(kmeanModel.inertia_)

    #Mapping for Distortion and Inertia for each K-value
    mapping1[k]=sum(np.min(cdist(df, kmeanModel.cluster_centers_,'euclidean'), axis=1))/df.shape[0]
    mapping2[k]=kmeanModel.inertia_
```

```
#Mapping for Distortion
for key, val in mapping1.items():
    print(f'{key} : {val}')
```

```
1 : 2.238203224334289
2 : 0.9564315803752368
3 : 0.6694722230259413
4 : 0.6030171894117745
5 : 0.5297687777358332
6 : 0.4865629284645346
7 : 0.44619708645375866
8 : 0.4223347412370906
9 : 0.40451830634398167
10 : 0.38934686466070795
```

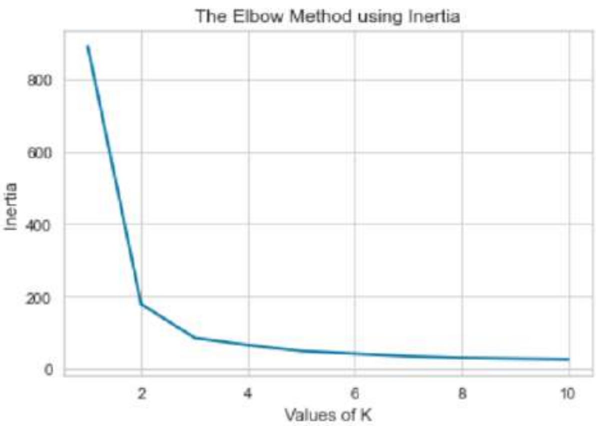
```
#Distortion vs Values of K graph
plt.plot(K, distortions, 'bx-')
plt.xlabel('Values of K')
plt.ylabel('Distortion')
plt.title('The Elbow Method using Distortion')
plt.show()
```



```python
#Mapping for Inertia
for key, val in mapping2.items():
    print(f'{key} : {val}')
```

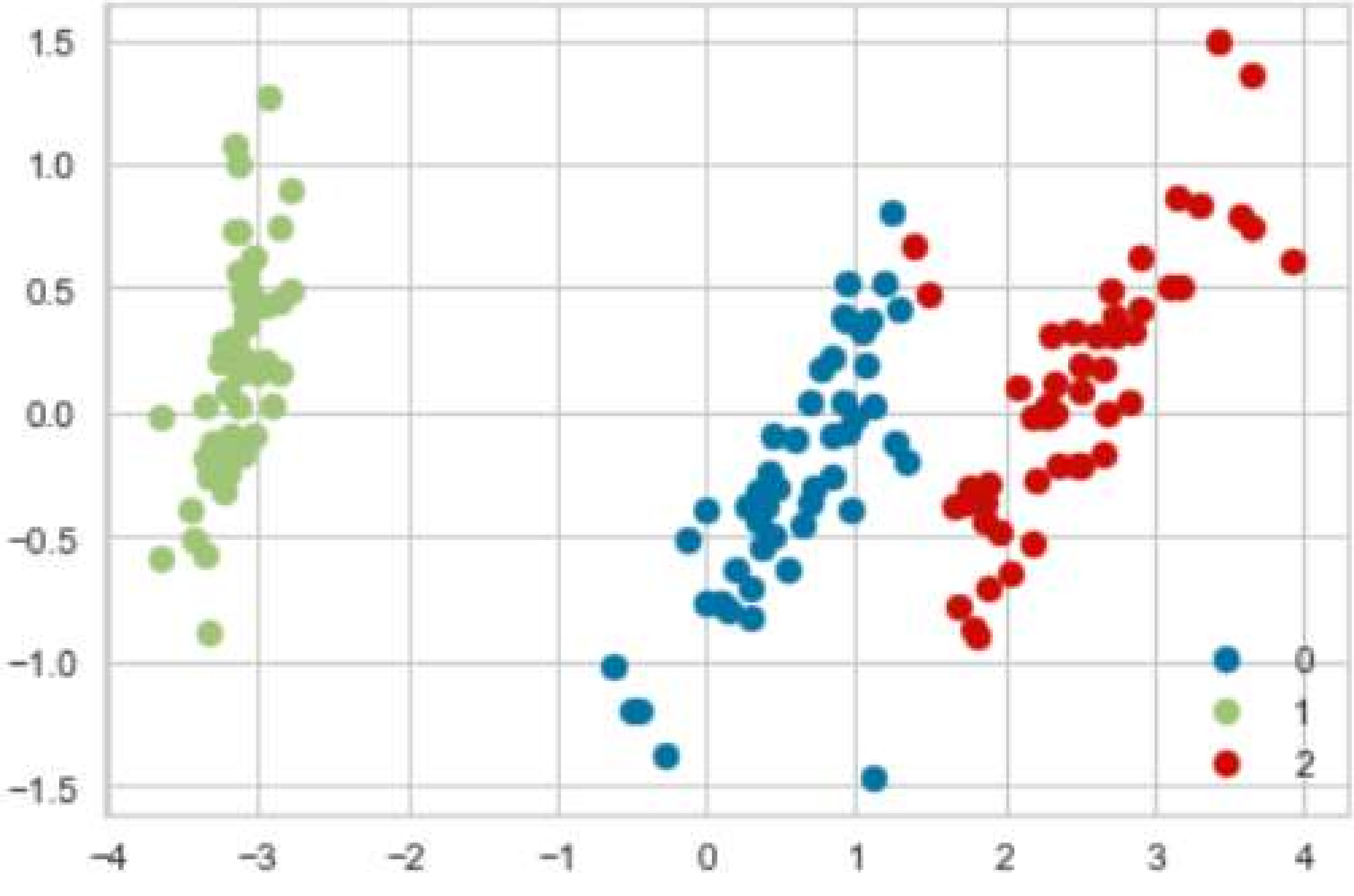
```
1 : 889.8680272108843
2 : 179.1697095959596
3 : 86.0105759803922
4 : 66.29249149659869
5 : 49.70517735355013
6 : 42.26973423185699
7 : 35.25026222891444
8 : 31.007034715686917
9 : 28.893206001113903
10 : 27.001097939677535
```

```
#Inertia vs Values of K graph
plt.plot(K, inertias, 'bx-')
plt.xlabel('Values of K')
plt.ylabel('Inertia')
plt.title('The Elbow Method using Inertia')
plt.show()
```



From both Distortion and Inertia plots it can be observed that k = 3 will be an optimal choice.

```python
#K-means clustering for K-value from Elbow method

from sklearn.decomposition import PCA
#Dimensionality reduction through PCA
pca=PCA(2)

#Transform the data
data=pca.fit_transform(df)

#Initialize the class object
kmeans=KMeans(n_clusters= 3)

#Predict the cluster labels
label=kmeans.fit_predict(data)

#Getting unique labels
u_labels=np.unique(label)

#Plotting the results
for i in u_labels:
    plt.scatter(data[label==i,0], data[label==i,1], label=i)
plt.legend()
plt.show()
```


1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−4
−3
−2
−1
0
1
2
3
4
0
1
2

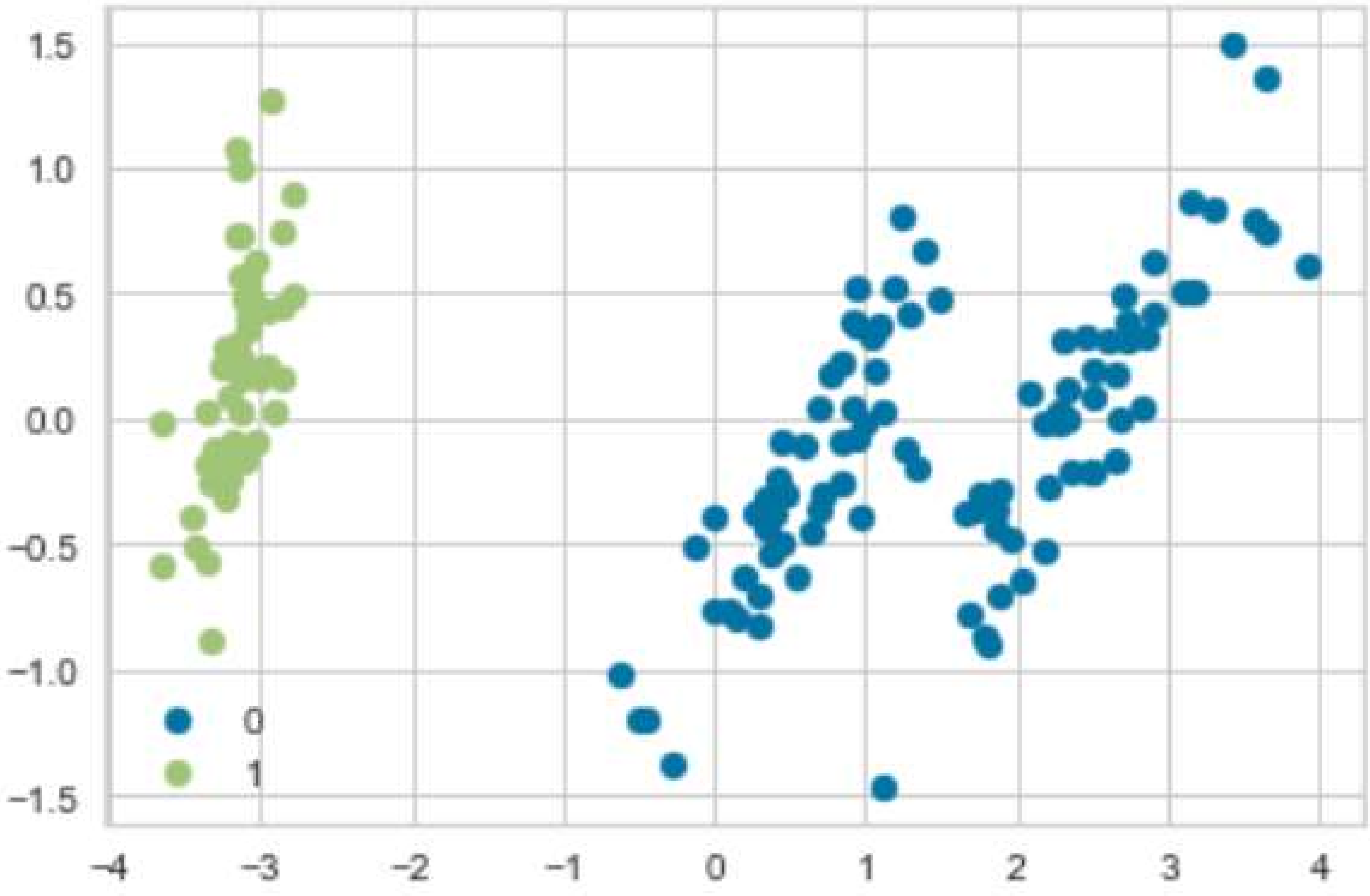
K-means using Silhouette method

```
#K-means Model building for Silhouette method
from sklearn.metrics import silhouette_samples, silhouette_score
for n in range(2,11):
    km=KMeans(n_clusters=n, random_state=42)

    # Fit the KMeans model
    km.fit_predict(df)

    # Calculate Silhoutte Score
    score=silhouette_score(df, km.labels_, metric='euclidean')
    print('Silhouette Score for {} : {}'.format(n,score))
```

```
Silhouette Score for 2 : 0.712221990514557
Silhouette Score for 3 : 0.5908280725198171
Silhouette Score for 4 : 0.5440469717715031
Silhouette Score for 5 : 0.5225389754263405
Silhouette Score for 6 : 0.521023858892796
Silhouette Score for 7 : 0.3899770113978547
Silhouette Score for 8 : 0.37983877100050145
Silhouette Score for 9 : 0.356063591699158
Silhouette Score for 10 : 0.3447360826592199
```

Hence, the Silhouette score for k = 2 is highest i.e. k = 2 is the optimal number of clusters.

```python
#K-means clustering for K-value from Silhouette method

from sklearn.decomposition import PCA
#Dimensionality reduction through PCA
pca=PCA(2)

#Transform the data
data=pca.fit_transform(df)

#Initialize the class object
kmeans=KMeans(n_clusters= 2)

#Predict the cluster labels
label=kmeans.fit_predict(data)

#Getting unique labels
u_labels=np.unique(label)

#Plotting the results
for i in u_labels:
    plt.scatter(data[label==i,0], data[label==i,1], label=i)
plt.legend()
plt.show()
```


1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−4
−3
−2
−1
0
1
2
3
4
0
1

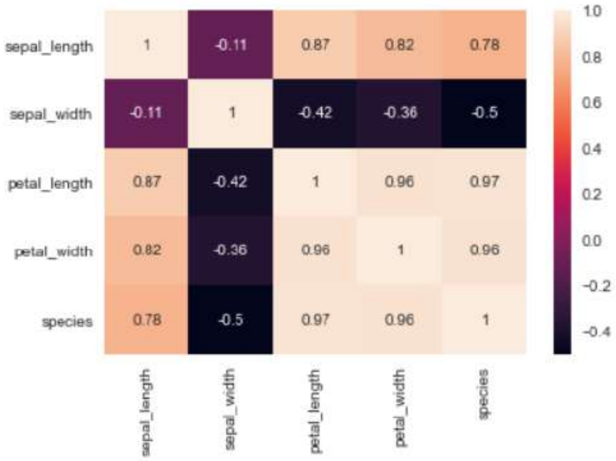
K-means clustering by selecting top 2 dimensions highly correlated to the target column "species" for better cluster visualization

Data Selection

```
import seaborn as sns
```

```
sns.heatmap(df.corr('pearson'),annot=True)
```

```
<AxesSubplot:>
```



Since, "sepal_length" and "sepal_width" are the least correlated to the target "species" hence they are dropped.

```
df=df.drop(['sepal_length','sepal_width'],axis=1)
```

```
df.replace('', np.nan, inplace=True)
```

```
df.dropna(inplace=True)
```

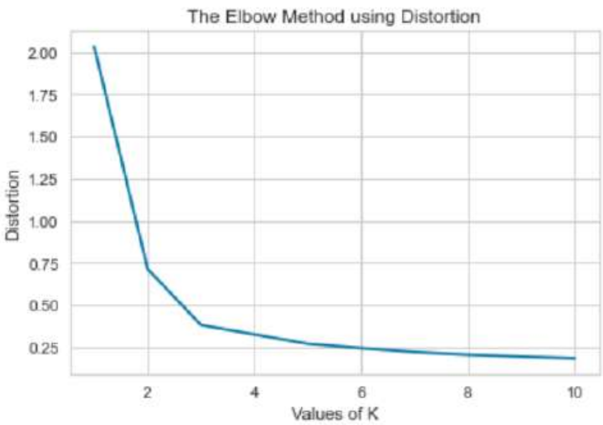
K-means using Elbow method

```
#K-means Model building for Elbow method
distortions=[]
inertias=[]
mapping1={}
mapping2={}

for k in range(1,11):

    # Building and fitting the model
    kmeanModel=KMeans(n_clusters=k).fit(df)
    kmeanModel.fit(df)

    #Calculating Distortion and Inertia for each K-value
    distortions.append(sum(np.min(cdist(df, kmeanModel.cluster_centers_,'euclidean'), axis=1))/df.shape[0])
    inertias.append(kmeanModel.inertia_)

    #Mapping for Distortion and Inertia for each K-value
    mapping1[k]=sum(np.min(cdist(df, kmeanModel.cluster_centers_,'euclidean'), axis=1))/df.shape[0]
    mapping2[k]=kmeanModel.inertia_
```

```
#Mapping for Distortion
for key, val in mapping1.items():
    print(f'{key} : {val}')
```

```
1 : 2.0316803566715484
2 : 0.7130839019972075
3 : 0.3802982348816473
4 : 0.32489966143835736
5 : 0.2695074250678213
6 : 0.2434150626582915
7 : 0.22068234204619736
8 : 0.20441326981066948
9 : 0.19352743680639897
10 : 0.18347323349710112
```

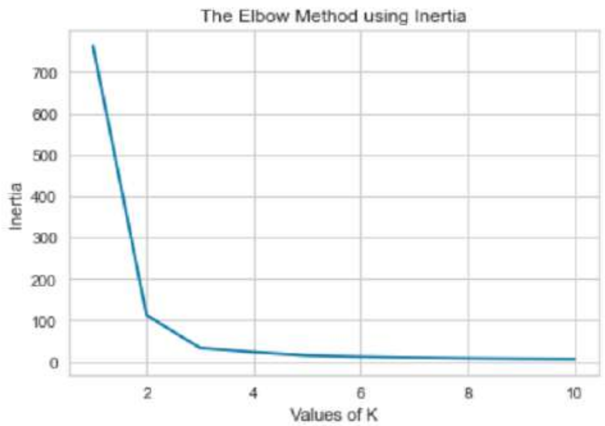
```
#Distortion vs Values of K graph
plt.plot(K, distortions, 'bx-')
plt.xlabel('Values of K')
plt.ylabel('Distortion')
plt.title('The Elbow Method using Distortion')
plt.show()
```



```python
#Mapping for Inertia
for key, val in mapping2.items():
    print(f'{key} : {val}')
```

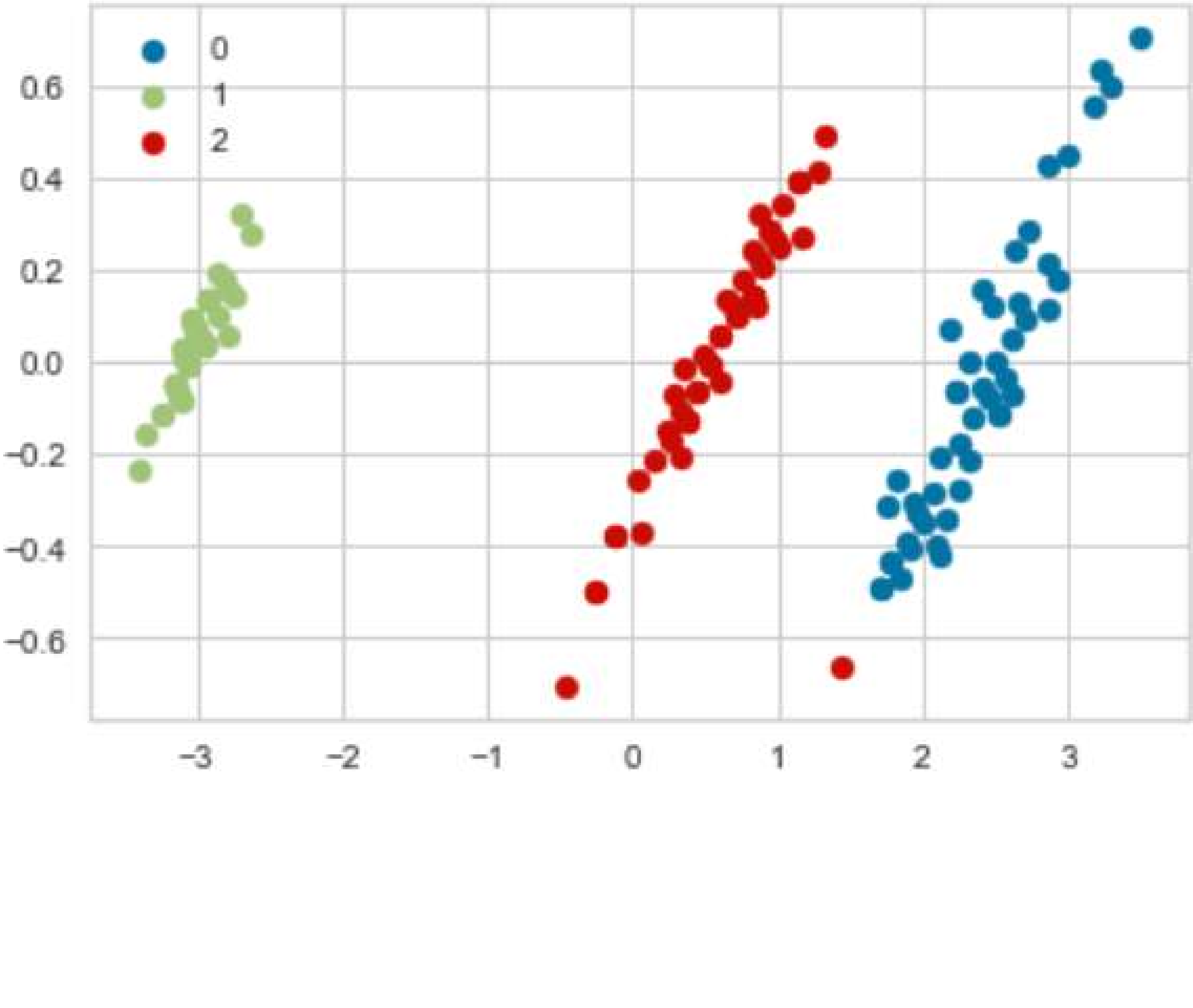
```
1 : 761.6240816326532
2 : 111.99007575757574
3 : 33.039154411764706
4 : 23.14159297658864
5 : 14.960490379186023
6 : 11.925914862914862
7 : 9.862746031746031
8 : 8.227603690599047
9 : 7.12346844675792
10 : 6.3626424233661085
```

```python
#Inertia vs Values of K graph
plt.plot(K, inertias, 'bx-')
plt.xlabel('Values of K')
plt.ylabel('Inertia')
plt.title('The Elbow Method using Inertia')
plt.show()
```



From both Distortion and Inertia plots it can be observed that k = 3 will be an optimal choice.

```python
#K-means clustering for K-value from Elbow method

from sklearn.decomposition import PCA
#Dimensionality reduction through PCA
pca=PCA(2)

#Transform the data
data=pca.fit_transform(df)

#Initialize the class object
kmeans=KMeans(n_clusters= 3)

#Predict the cluster labels
label=kmeans.fit_predict(data)

#Getting unique labels
u_labels=np.unique(label)

#Plotting the results
for i in u_labels:
    plt.scatter(data[label==i,0], data[label==i,1], label=i)
plt.legend()
plt.show()
```


0
1
2
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−3
−2
−1
0
1
2
3

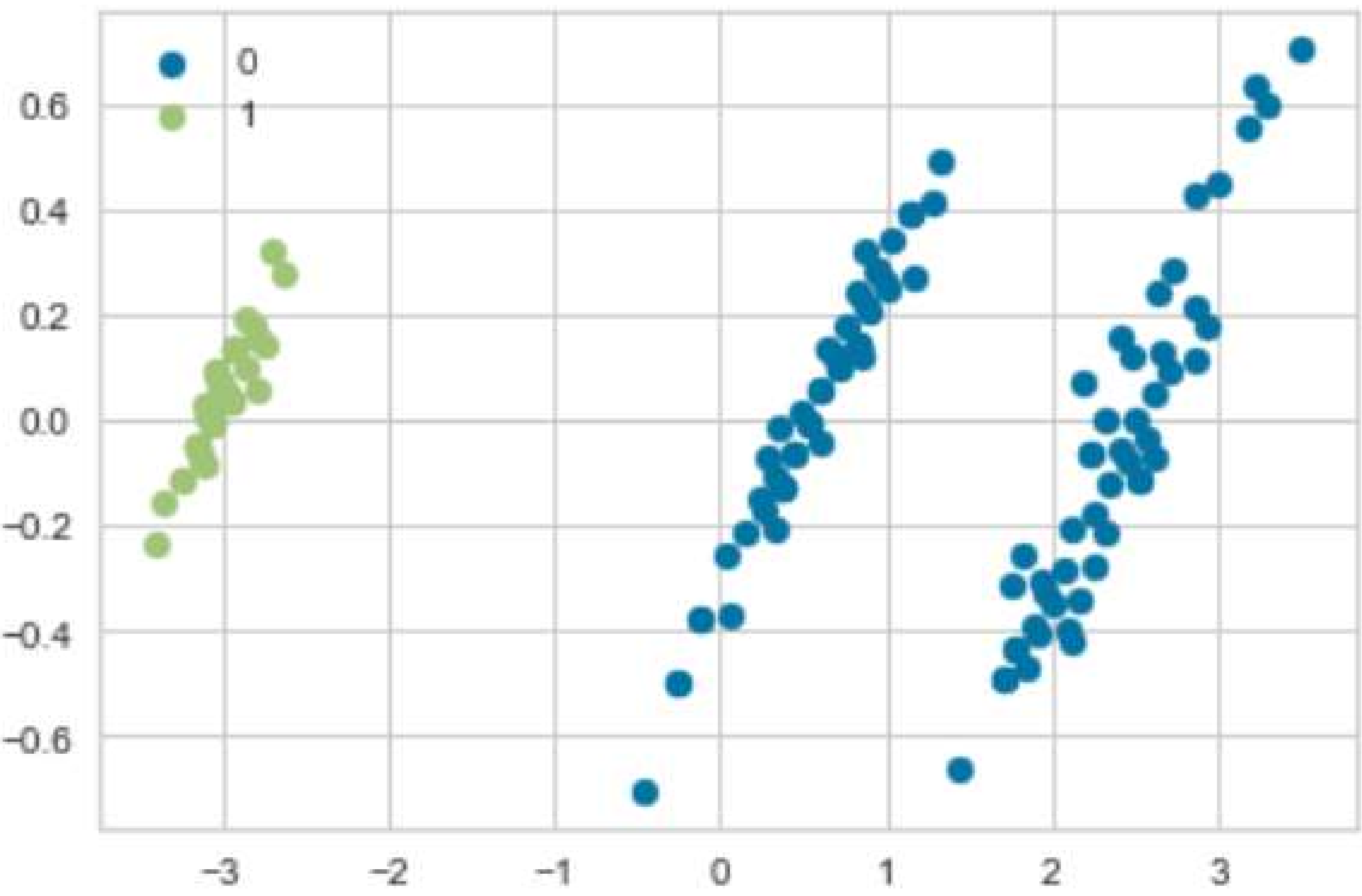
K-means using Silhouette method

```
#K-means Model building for Silhouette method
from sklearn.metrics import silhouette_samples, silhouette_score
for n in range(2,11):
    km=KMeans(n_clusters=n, random_state=42)

    # Fit the KMeans model
    km.fit_predict(df)

    # Calculate Silhoutte Score
    score=silhouette_score(df, km.labels_, metric='euclidean')
    print('Silhouette Score for {} : {}'.format(n,score))
```

```
Silhouette Score for 2 : 0.780759974279947
Silhouette Score for 3 : 0.7234947790338877
Silhouette Score for 4 : 0.6513734671362285
Silhouette Score for 5 : 0.6196970665679534
Silhouette Score for 6 : 0.6097647735483723
Silhouette Score for 7 : 0.6070990018083023
Silhouette Score for 8 : 0.6161518970713937
Silhouette Score for 9 : 0.6056227673235245
Silhouette Score for 10 : 0.4338603630481673
```

Hence, the Silhouette score for k = 2 is highest i.e. k = 2 is the optimal number of clusters.

```python
#K-means clustering for K-value from Silhouette method

from sklearn.decomposition import PCA
#Dimensionality reduction through PCA
pca=PCA(2)

#Transform the data
data=pca.fit_transform(df)

#Initialize the class object
kmeans=KMeans(n_clusters= 2)

#Predict the cluster labels
label=kmeans.fit_predict(data)

#Getting unique labels
u_labels=np.unique(label)

#Plotting the results
for i in u_labels:
    plt.scatter(data[label==i,0], data[label==i,1], label=i)
plt.legend()
plt.show()
```



0
1
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−3
−2
−1
0
1
2
3