chapter-3-1

February 2, 2022

Run in Google Colab

0.1 What is Unsupervised Learning

Unsupervised learning uses machine learning algorithms to analyze and cluster unlabeled datasets. These algorithms discover hidden patterns or data groupings without the need for human intervention. Its ability to discover similarities and differences in information make it the ideal solution for exploratory data analysis, cross-selling strategies, customer segmentation, and image recognition.

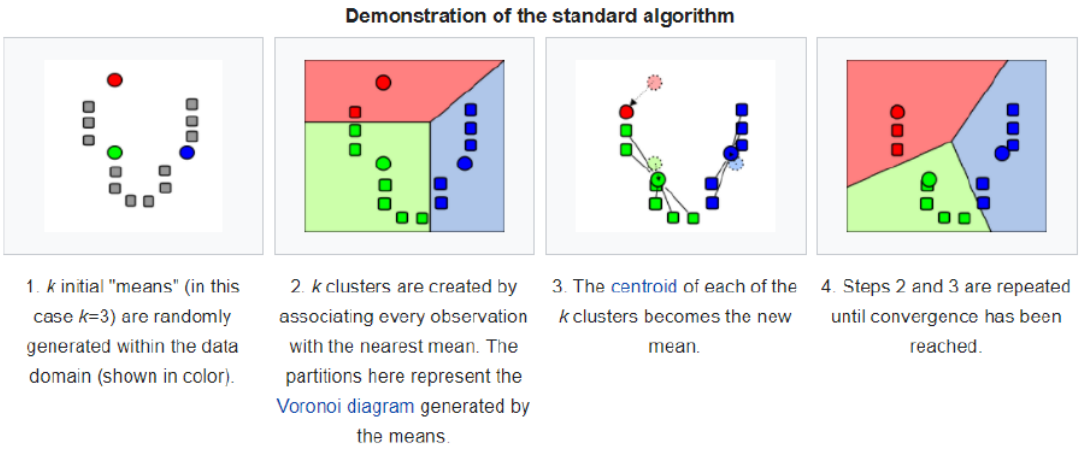
0.1.1 *k*-Means Clustering

k-means clustering is one of the simplest and popular unsupervised machine learning algorithms. Typically, unsupervised algorithms make inferences from datasets using only input vectors without referring to known, or labelled, outcomes. The objective of K-means is simple: group similar data points together and discover underlying patterns. To achieve this objective, K-means looks for a fixed number (k) of clusters in a dataset.

A cluster refers to a collection of data points aggregated together because of certain similarities. You'll define a target number k, which refers to the number of centroids you need in the dataset. A centroid is the imaginary or real location representing the center of the cluster.

Every data point is allocated to each of the clusters through reducing the in-cluster sum of squares. In other words, the K-means algorithm identifies k number of centroids, and then allocates every data point to the **nearest** cluster, while keeping the centroids as small as possible.

The 'means' in the K-means refers to averaging of the data; that is, finding the centroid.



How the K-means algorithm works To process the learning data, the K-means algorithm in data mining starts with a first group of randomly selected centroids, which are used as the beginning points for every cluster, and then performs iterative (repetitive) calculations to optimize the positions of the centroids. It halts creating and optimizing clusters when either:

- The centroids have stabilized — there is no change in their values because the clustering has been successful.
- The defined number of iterations has been achieved.

A Distance Measure For clustering we need a distance measure. The simplest distance measure is the Euclidean Distance measure:

$$Distance = \sqrt{(x_B - x_B)^2 + (y_B - y_A)^2}$$

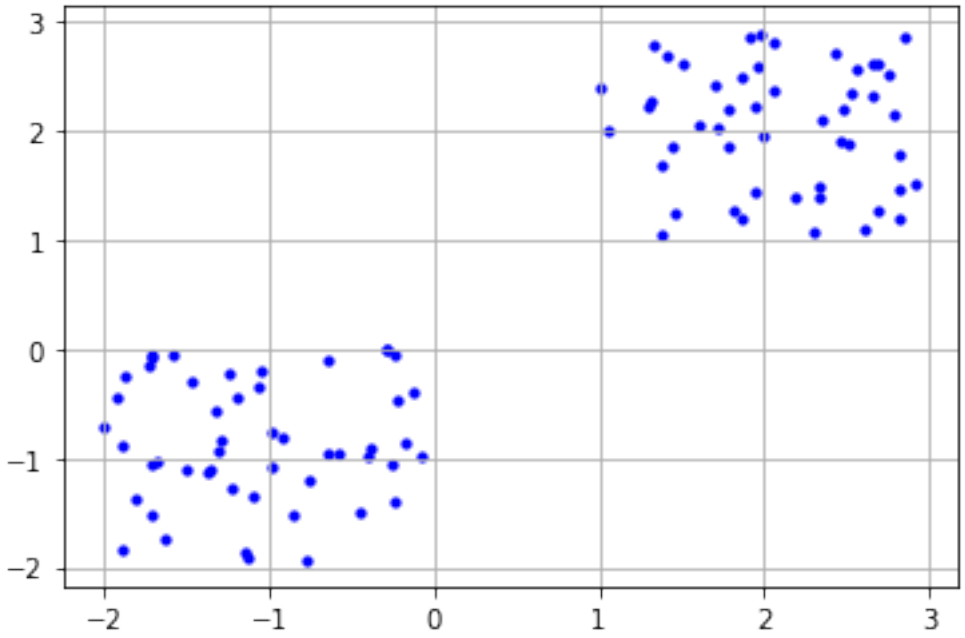
K-means algorithm example problem Let's see the steps on how the K-means machine learning algorithm works using the Python programming language. We'll use the Scikit-learn library and some random data to illustrate a K-means clustering simple explanation.

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
%matplotlib inline
```

Here is the code for generating some random data in a two-dimensional space:

[2]:
```
X  =    -2 * np.random.rand(100,2)
X1 = 1 + 2 * np.random.rand(50,2)
X[50:100, :] = X1
```

```
plt.scatter(X[ : , 0], X[ :, 1], s = 10, c = 'b')
plt.grid()
plt.show()
```



This give us two sets approximately centered about (-1,-1) and (2, 2). We'll use some of the available functions in the Scikit-learn library to process the randomly generated data.

[3]:
```
from sklearn.cluster import KMeans
Kmean = KMeans(n_clusters=2)
Kmean.fit(X)
```

[3]:
```
KMeans(n_clusters=2)
```

In this case, we arbitrarily gave k (n_clusters) an arbitrary value of two. Here is the output of the K-means parameters we get if we run the code:

[29]:
```
#KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
# n_clusters=2, n_init=10, n_jobs=1, precompute_distances='auto',
# random_state=None, tol=0.0001, verbose=0)
```

[30]:
```
KMeans(max_iter=300, n_clusters=2, tol=0.0001)
```

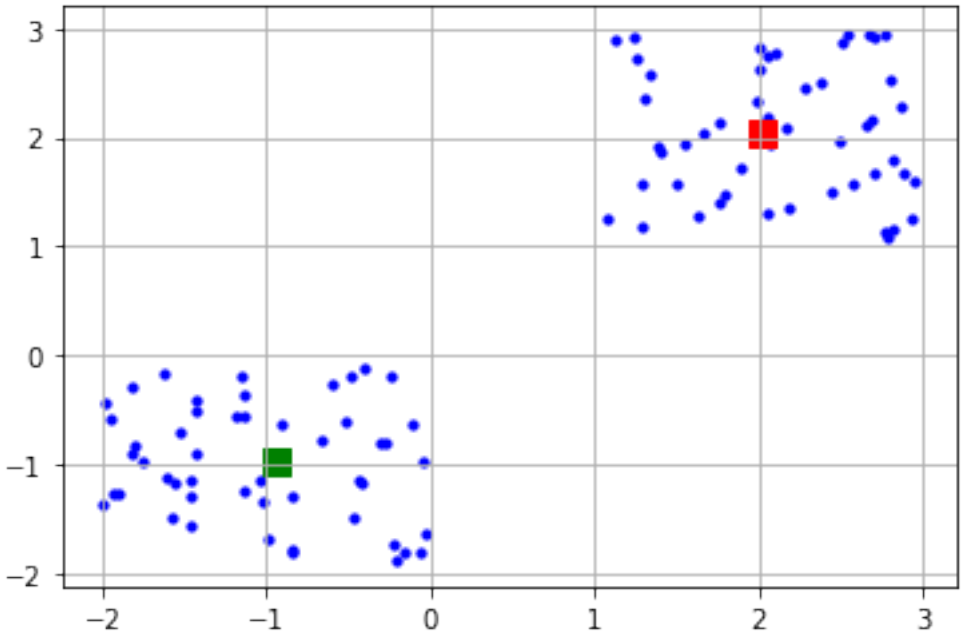
[30]:
```
KMeans(n_clusters=2)
```

[31]:
```
Kmean.cluster_centers_
```

[31]:
```
array([[ 2.11896647,  2.04141105],
       [-1.02802583, -0.98355376]])
```

Let's display the cluster centroids

[32]:
```
plt.scatter(X[ : , 0], X[ : , 1], s =10, c='b')
plt.scatter(-0.94665068, -0.97138368, s=100, c='g', marker='s')
plt.scatter( 2.01559419,  2.02597093, s=100, c='r', marker='s')
plt.grid()
plt.show()
```



Here is the code for getting the labels property of the K-means clustering example dataset; that is, how the data points are categorized into the two clusters.

[33]:
```
Kmean.labels_
```

[33]:
```
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, 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])
```

As you can see above, 50 data points belong to the 0 cluster while the rest belong to the 1 cluster.

For example, let's use the code below for predicting the cluster of a data point:

[34]:
```
sample_test=np.array([1.5,1.5])
second_test=sample_test.reshape(1, -1)
Kmean.predict(second_test)
```

[34]:
```
array([0])
```

0.1.2 Example 2 - Country Risk

[35]:
```
# loading packages

import os

import pandas as pd
import numpy as np

# plotting packages
%matplotlib inline
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
import matplotlib.cm as cm
import matplotlib.colors as clrs

# Kmeans algorithm from scikit-learn
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_samples, silhouette_score
```

The Country Risk Dataset (J. C. Hull, 2019, Chapter 2) Consider the problem of understanding the risk of countries for foreign investment. Among the features that can be used for this are:

- GDP growth rate (IMF)
- Corruption index (Transparency international)
- Peace index (Institute for Economics and Peace)
- Legal Risk Index (Property Rights Association)

Values for each of the features for 122 countries are found in the `countryriskdata.csv` (available here)

[36]:
```
if 'google.colab' in str(get_ipython()):
    from google.colab import files
    uploaded = files.upload()
    path = ''
else:
    path = './data/'
```

[37]:
```
# load raw data
raw = pd.read_csv(os.path.join(path, 'countryriskdata.csv'))

# check the raw data
print("Size of the dataset (row, col): ", raw.shape)
print("\nFirst 5 rows\n", raw.head(n=5))
```

```
Size of the dataset (row, col):  (122, 6)

First 5 rows
      Country Abbrev  Corruption  Peace  Legal  GDP Growth
0    Albania     AL          39  1.867  3.822       3.403
1    Algeria     DZ          34  2.213  4.160       4.202
2  Argentina     AR          36  1.957  4.568      -2.298
3    Armenia     AM          33  2.218  4.126       0.208
4  Australia     AU          79  1.465  8.244       2.471
```

The GDP growth rate (%) is typically a positive or negative number with a magnitude less than 10. The corruption index is on a scale from 0 (highly corrupt) to 100 (no corruption). The peace index is on a scale from 1 (very peaceful) to 5 (not at all peaceful). The legal risk index runs from 0 to 10 (with high values being favorable).

0.1.3 Simple exploratory analysis

Print summary statistics

Note that all features have quite different variances, and Corruption and Legal are highly correlated.

[38]:
```
# print summary statistics
print("\nSummary statistics\n", raw.describe())
print("\nCorrelation matrix\n", raw.corr())
```

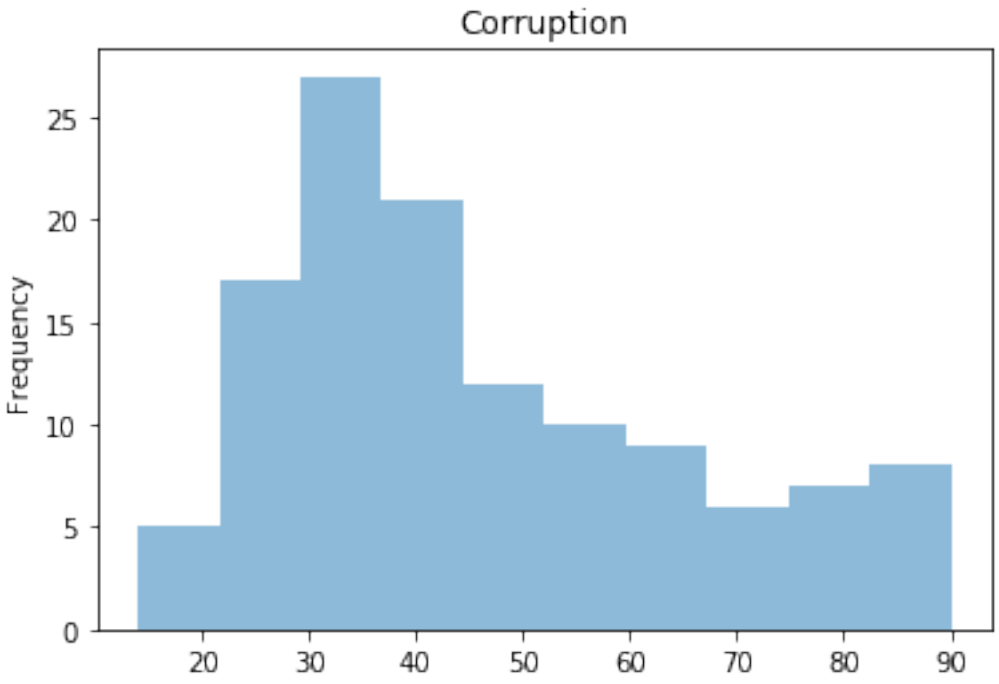
```
Summary statistics
        Corruption       Peace       Legal  GDP Growth
count  122.000000  122.000000  122.000000  122.000000
mean    46.237705    2.003730    5.598861    2.372566
std     19.126397    0.447826    1.487328    3.241424
min     14.000000    1.192000    2.728000  -18.000000
25%     31.250000    1.684750    4.571750    1.432250
50%     40.000000    1.969000    5.274000    2.496000
75%     58.750000    2.280500    6.476750    4.080000
max     90.000000    3.399000    8.633000    7.958000

Correlation matrix
             Corruption     Peace     Legal  GDP Growth
Corruption    1.000000 -0.700477  0.923589    0.102513
```

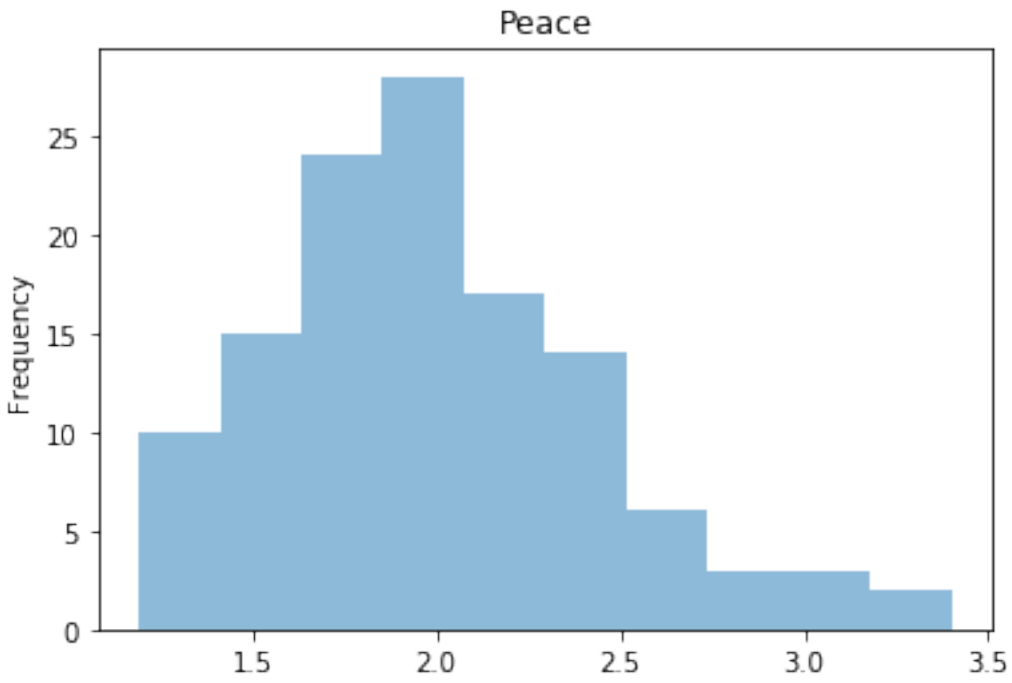
```
Peace         -0.700477  1.000000 -0.651961   -0.199855
Legal          0.923589 -0.651961  1.000000    0.123440
GDP Growth     0.102513 -0.199855  0.123440    1.000000
```

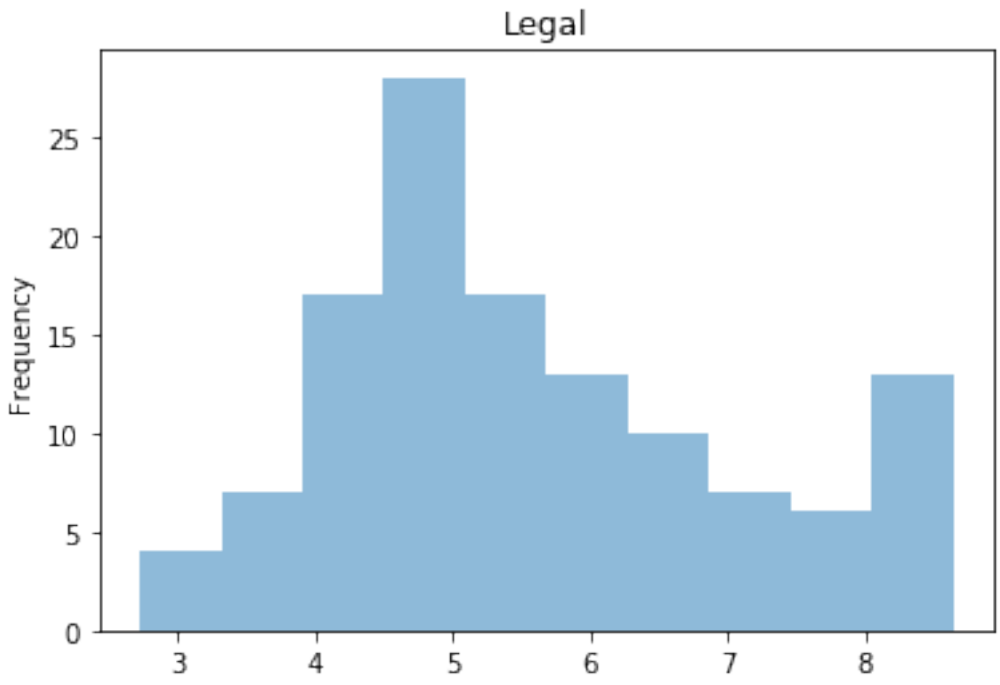
Plot histogram

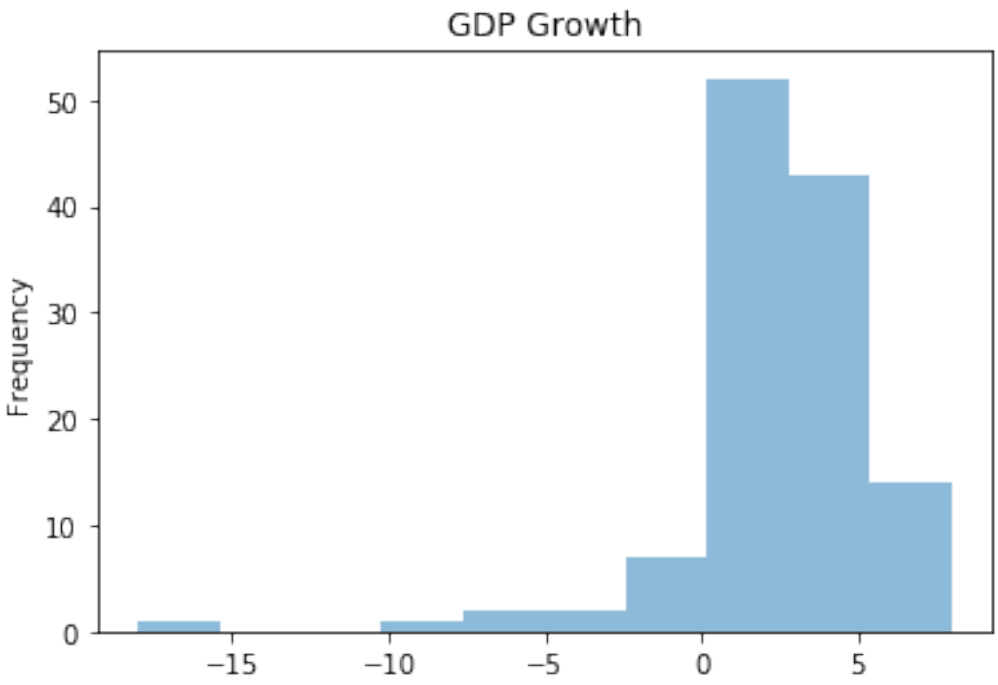
Note that distributions for GDP Growth is quite skewed.

[39]:
```
# plot histograms
plt.figure(1)
raw['Corruption'].plot(kind = 'hist', title = 'Corruption', alpha = 0.5)

plt.figure(2)
raw['Peace'].plot(kind = 'hist', title = 'Peace', alpha = 0.5)

plt.figure(3)
raw['Legal'].plot(kind = 'hist', title = 'Legal', alpha = 0.5)

plt.figure(4)
raw['GDP Growth'].plot(kind = 'hist', title = 'GDP Growth', alpha = 0.5)

plt.show()
```




Peace
Frequency
25
20
15
10
5
0
1.5
2.0
2.5
3.0
3.5


Legal
Frequency
25
20
15
10
5
0
3
4
5
6
7
8



0.1.4 K means cluster

Pick features & normalization

Since Corruption and Legal are highly correlated, we drop the Corruption variable, i.e., we pick three features for this analysis, Peace, Legal and GDP Grwoth. Let's normalize all the features, effectively making them equally weighted.

Ref. Feature normalization.

[40]:
```
X = raw[['Peace', 'Legal', 'GDP Growth']]
X = (X - X.mean()) / X.std()
print(X.head(5))
```

```
      Peace     Legal  GDP Growth
0 -0.305319 -1.194666    0.317896
1  0.467304 -0.967413    0.564392
2 -0.104348 -0.693096   -1.440899
3  0.478469 -0.990273   -0.667782
4 -1.202990  1.778450    0.030368
```

0.1.5 Perform elbow method

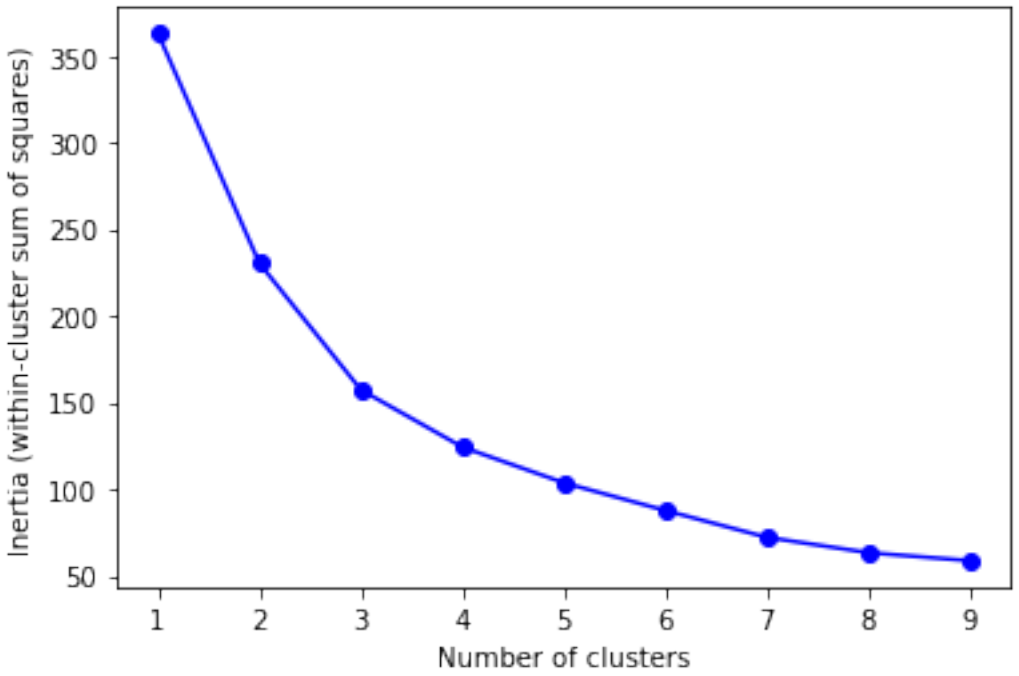
In cluster analysis, the elbow method is a heuristic used in determining the number of clusters in a data set. The method consists of plotting the explained variation as a function of the number of clusters, and picking the elbow of the curve as the number of clusters to use. The same method can be used to choose the number of parameters in other data-driven models, such as the number of principal components to describe a data set.

For example in the following picture k=4 is suggested

In our case, the marginal gain of adding one cluster dropped quite a bit from k=3 to k=4. We will choose k=3 (not a clear cut though).

Ref. Determining the number of clusters in a dataset.

[41]:
```
# https://stackoverflow.com/questions/41540751/
 ↪sklearn-kmeans-equivalent-of-elbow-method

Ks = range(1, 10)
inertia = [KMeans(i).fit(X).inertia_ for i in Ks]

fig = plt.figure()
plt.plot(Ks, inertia, '-bo')
plt.xlabel('Number of clusters')
plt.ylabel('Inertia (within-cluster sum of squares)')
plt.show()
```



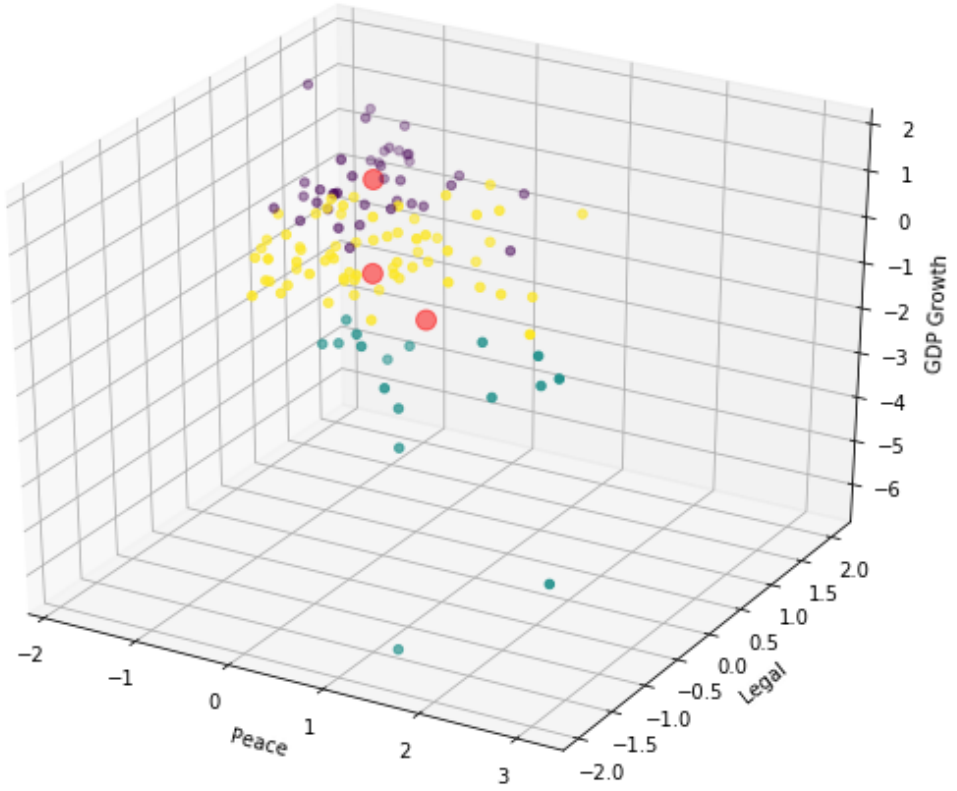
***k*-means with k=3**

[42]:
```
k = 3
kmeans = KMeans(n_clusters=k, random_state=0)
kmeans.fit(X)

# print inertia & cluster center
print("inertia for k=3 is", kmeans.inertia_)
print("cluster centers: ", kmeans.cluster_centers_)

# take a quick look at the result
y = kmeans.labels_
print("cluster labels: ", y)
```

```
inertia for k=3 is 157.551489241025
cluster centers:  [[-0.92810589  1.16641329 -0.01445833]
 [ 1.21562552 -1.01677118 -1.61496953]
 [ 0.25320926 -0.45186802  0.43127408]]
cluster labels:  [2 2 1 1 0 0 1 2 2 0 2 2 2 0 1 2 1 2 0 1 0 2 2 0 2 2 0 1 0 2 1
2 2 0 2 0 0
 2 2 0 2 2 2 2 0 0 2 2 2 0 2 0 2 0 2 2 2 0 2 2 1 1 0 2 2 0 2 2 0 2 2 2 2 2
 2 0 0 2 1 0 2 2 2 2 2 2 0 0 0 2 1 2 2 2 2 2 0 0 0 0 0 2 0 0 0 2 2 2 1 2 2
 2 1 0 0 0 0 1 2 1 2 1]
```

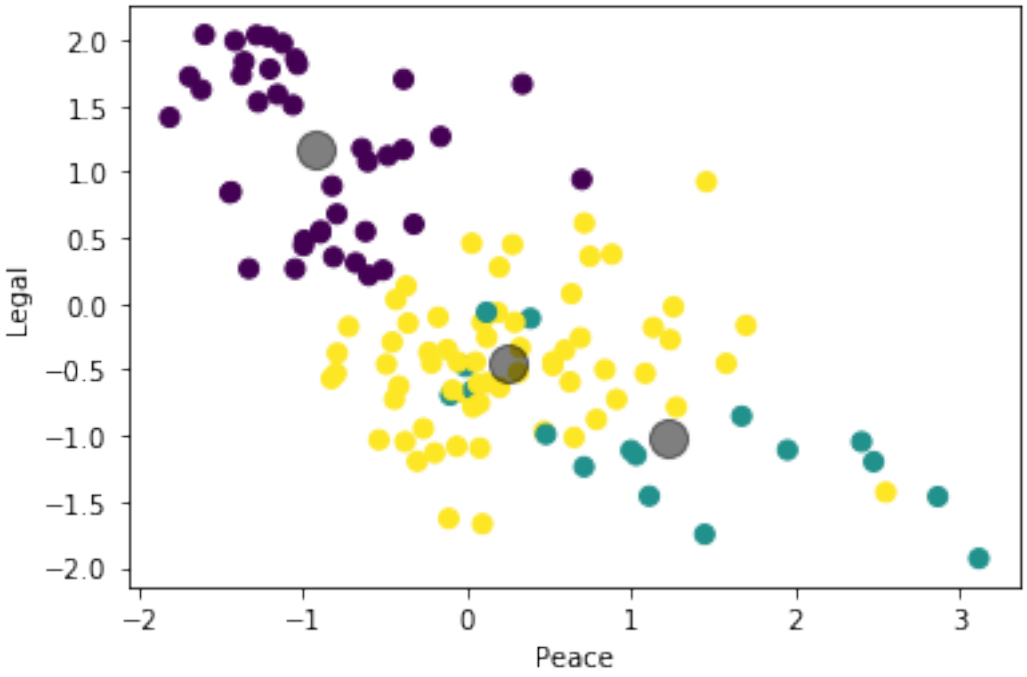
Visualize the result (3D plot)

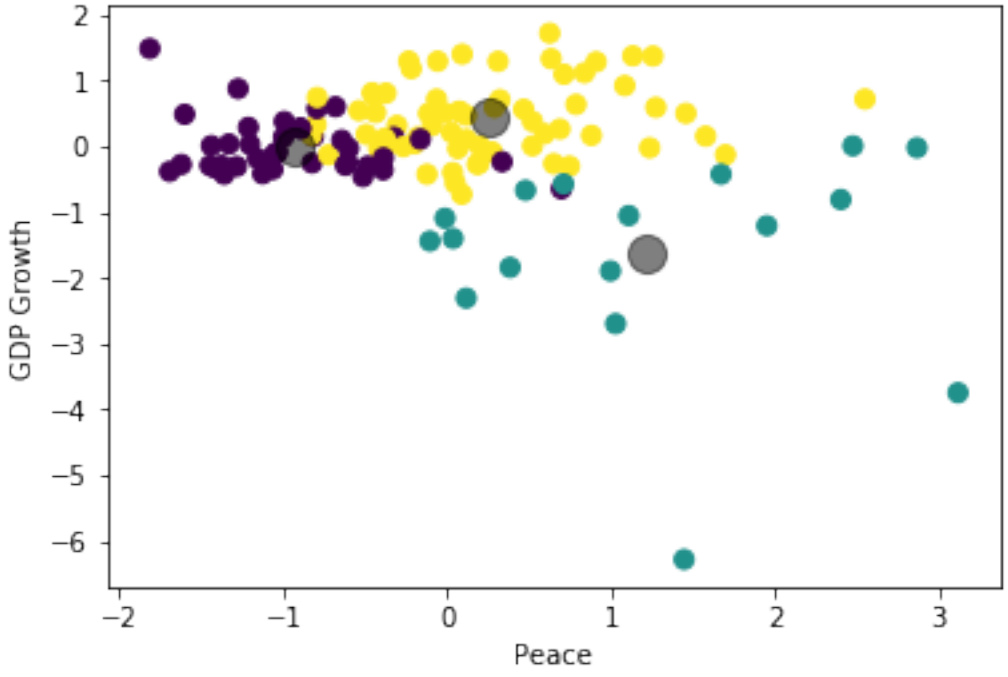
[43]:
```
# set up the color
norm = clrs.Normalize(vmin=0.,vmax=y.max())
cmap = cm.viridis

fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111, projection='3d')

ax.scatter(X.iloc[:,0], X.iloc[:,1], X.iloc[:,2], c=cmap(norm(y)), marker='o')

centers = kmeans.cluster_centers_
ax.scatter(centers[:, 0], centers[:, 1], c='red', s=100, alpha=0.5, marker='o')

ax.set_xlabel('Peace')
ax.set_ylabel('Legal')
ax.set_zlabel('GDP Growth')

plt.show()
```

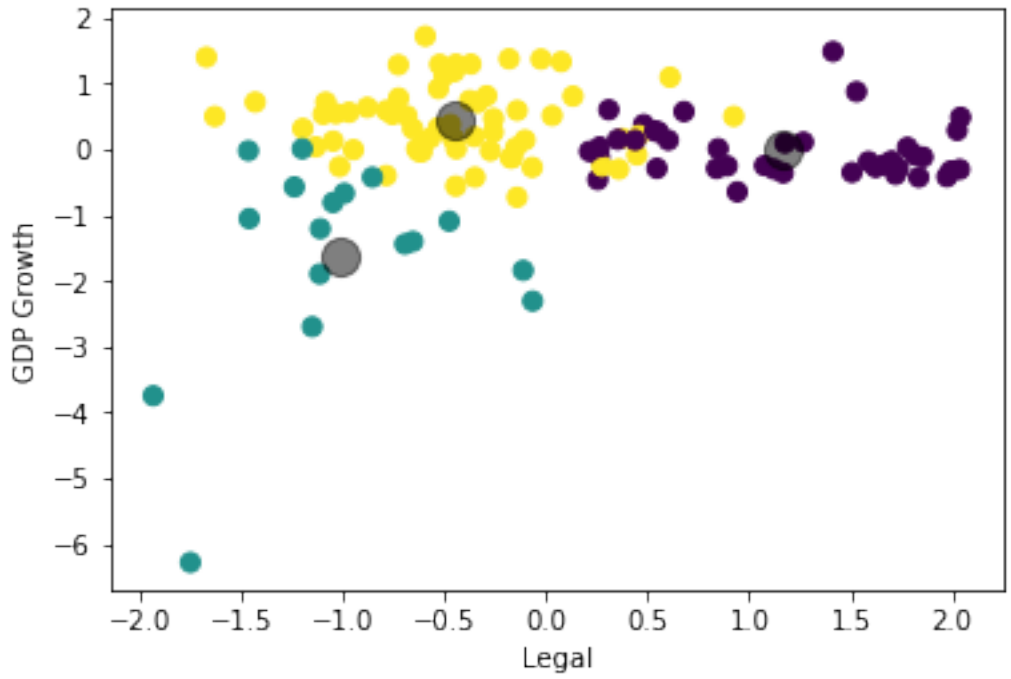


Visualize the result (3 2D plots)

[44]:
```
%matplotlib inline
import matplotlib.pyplot as plt

figs = [(0, 1), (0, 2), (1, 2)]
labels = ['Peace', 'Legal', 'GDP Growth']

for i in range(3):
    fig = plt.figure(i)
    plt.scatter(X.iloc[:,figs[i][0]], X.iloc[:,figs[i][1]], c=cmap(norm(y)),␣
 ↪s=50)
    plt.scatter(centers[:, figs[i][0]], centers[:, figs[i][1]], c='black',␣
 ↪s=200, alpha=0.5)
    plt.xlabel(labels[figs[i][0]])
    plt.ylabel(labels[figs[i][1]])

plt.show()
```


Legal
Peace
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−2
−1
0
1
2
3

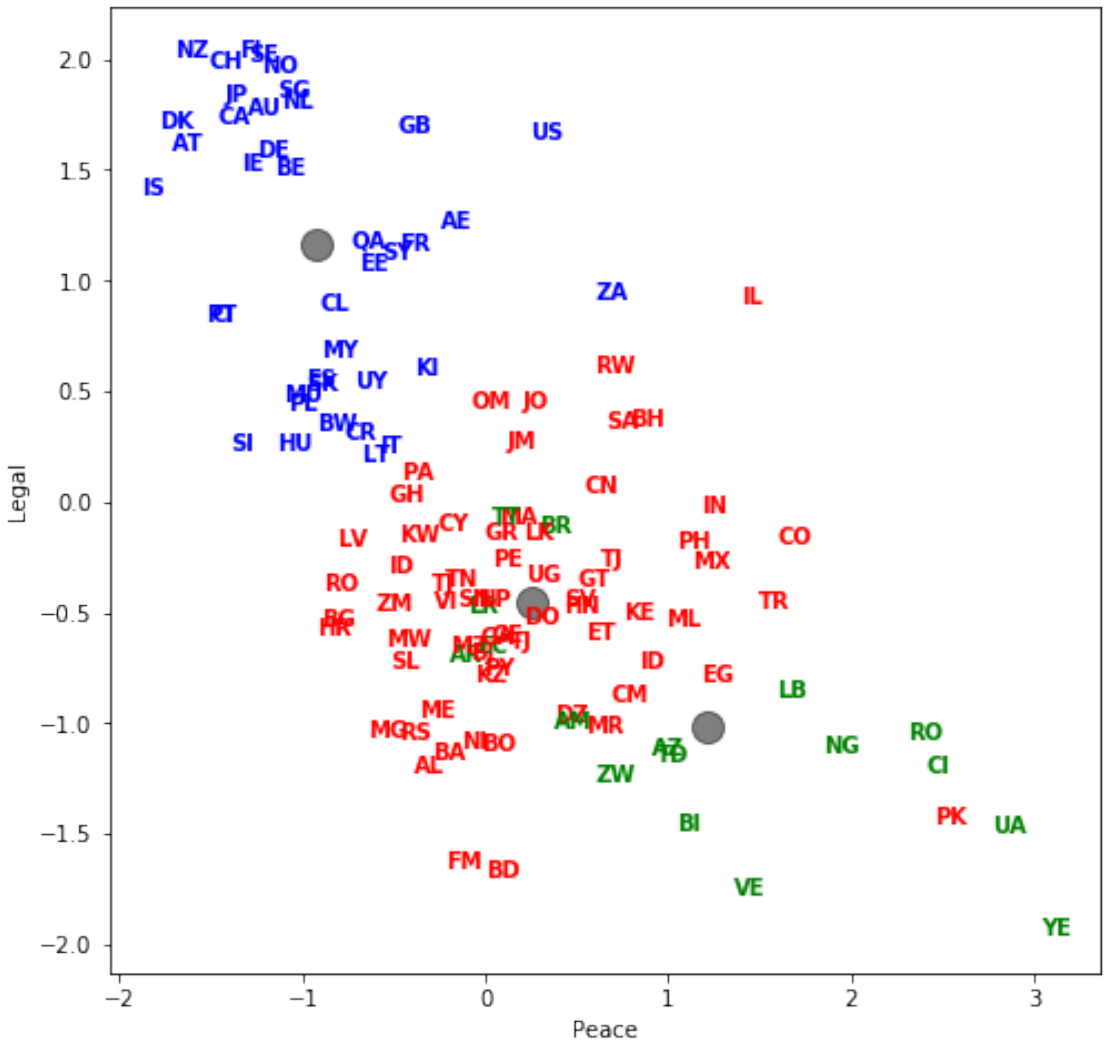

GDP Growth
Peace
2
1
0
−1
−2
−3
−4
−5
−6
−2
−1
0
1
2
3

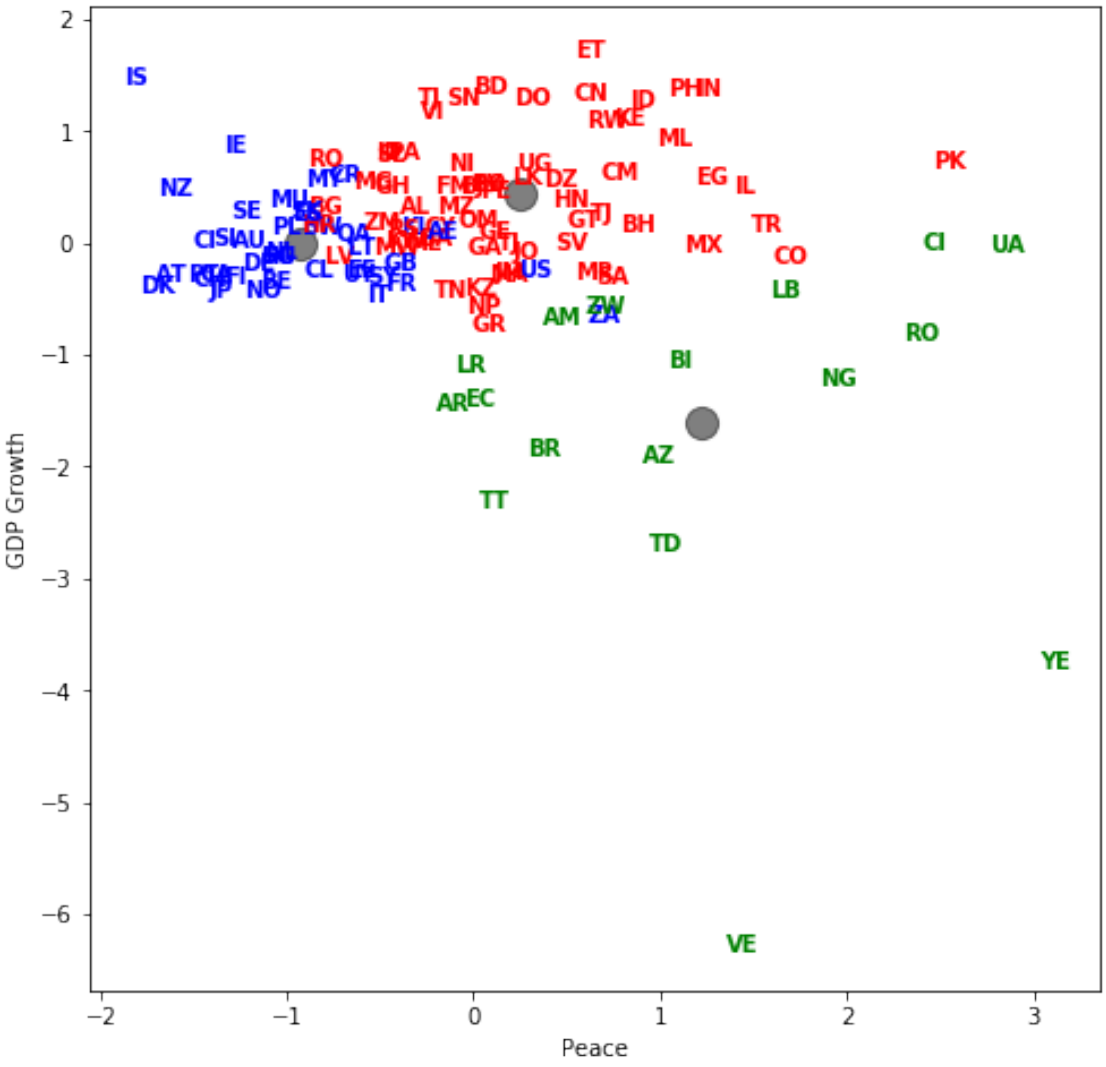


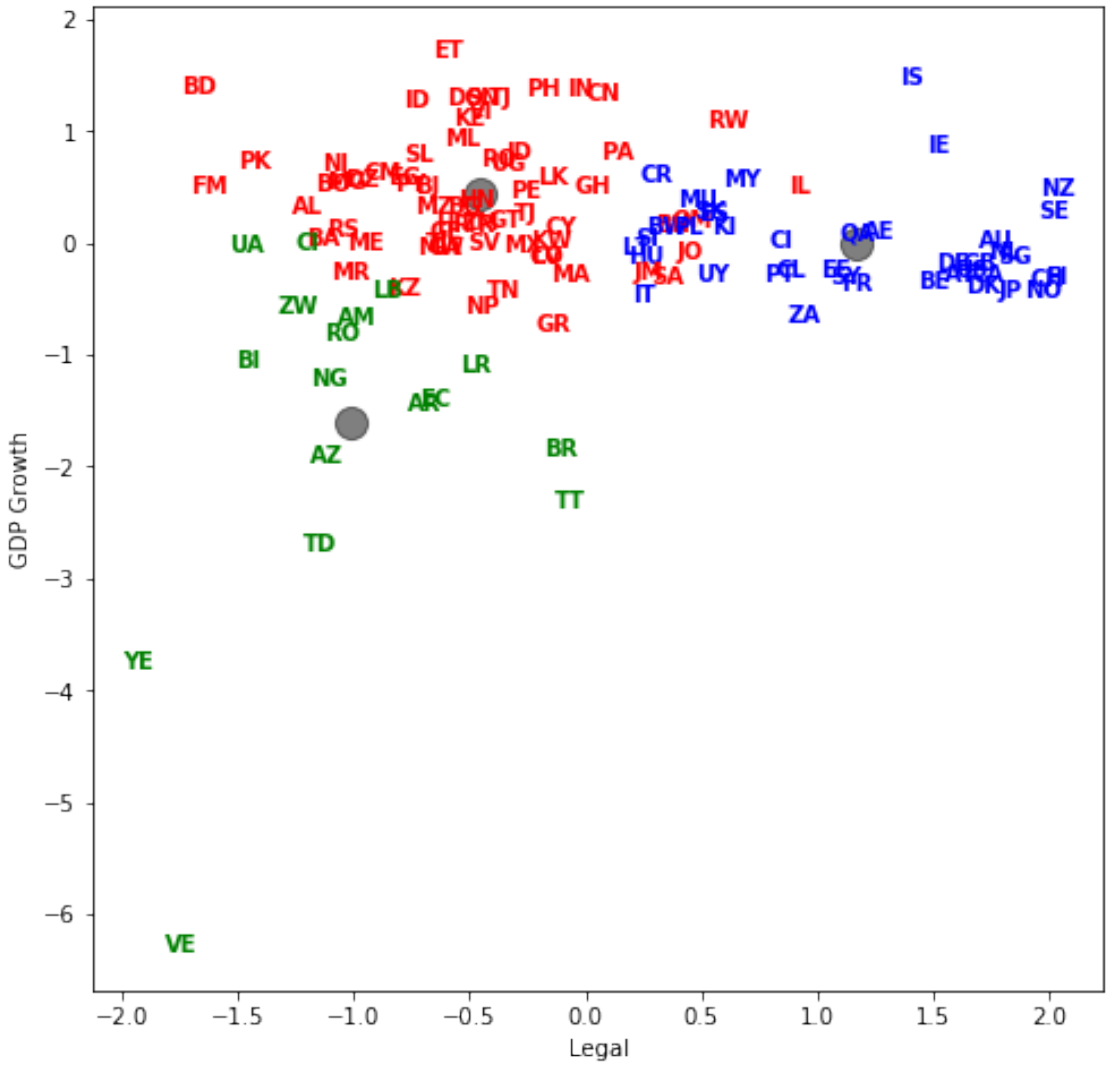
Visualize the result (3 2D plots)

plot country abbreviations instead of dots.

[45]:
```
%matplotlib inline
import matplotlib.pyplot as plt

figs = [(0, 1), (0, 2), (1, 2)]
labels = ['Peace', 'Legal', 'GDP Growth']
colors = ['blue','green', 'red']

for i in range(3):
    fig = plt.figure(i, figsize=(8, 8))
    x_1 = figs[i][0]
    x_2 = figs[i][1]
    plt.scatter(X.iloc[:, x_1], X.iloc[:, x_2], c=y, s=0, alpha=0)
    plt.scatter(centers[:, x_1], centers[:, x_2], c='black', s=200, alpha=0.5)
    for j in range(X.shape[0]):
        plt.text(X.iloc[j, x_1], X.iloc[j, x_2], raw['Abbrev'].iloc[j],
                 color=colors[y[j]], weight='semibold', horizontalalignment =␣
 ↪'center', verticalalignment = 'center')
    plt.xlabel(labels[x_1])
    plt.ylabel(labels[x_2])

plt.show()
```


Legal
Peace
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−2
−1
0
1
2
3
NZ
CH
NO
JP
SG
CA
AU
NL
DK
AT
GB
US
IE
DE
BE
IS
AE
QA
FR
EE
SY
ZA
IL
CL
MY
UY
KI
RW
OM
JO
SA
BH
SI
HU
BW
CR
LT
JM
PA
GH
CN
IN
CY
GR
LK
BR
LV
KW
PH
CO
PE
TJ
MX
ID
UG
GT
RO
ZM
TR
KE
ML
DO
ET
MW
SL
ID
EG
LB
CM
ME
MR
NG
RO
BO
BA
AL
ZW
CI
PK
UA
BI
FM
BD
VE
YE


GDP Growth
Peace
2
1
0
−1
−2
−3
−4
−5
−6
−2
−1
0
1
2
3
ET
IS
BD
DO
CN
PHIN
ML
IE
PK
NZ
EG
IL
SE
CM
DZ
HN
BH
TR
SV
MX
CO
CI
UA
LB
RO
AM
LR
BI
NG
AR
EC
BR
AZ
TT
TD
YE
VE



0.1.6 List the result

```
[ ]: result = pd.DataFrame({'Country':raw['Country'], 'Abbrev':raw['Abbrev'], 'Label':
      ↪y})
     with pd.option_context('display.max_rows', None, 'display.max_columns', 3):
         print(result.sort_values('Label'))
```

0.1.7 Silhouette Analysis

For each observation i calculate $a(i)$, the average distance from other observations in its cluster, and $b(i)$, the average distance from observations in the closest other cluster. The silhouette score

for observation i, $s(i)$, is defined as

$$s(i) = \frac{b(i) - a(i)}{\max[a(i), b(i)]} \tag{1}$$

Choose the number of clusters that maximizes the average silhouette score across all observations

[ ]:
```
# Silhouette Analysis
range_n_clusters = [2,3,4,5,6,7,8,9,10]
silhouette       = []
for n_clusters in range_n_clusters:
    clusterer=KMeans(n_clusters=n_clusters, random_state=0)
    cluster_labels=clusterer.fit_predict(X)
    silhouette_avg=silhouette_score(X,cluster_labels)
    silhouette.append(silhouette_avg)
    #print("For n_clusters=", n_clusters,
    #      "The average silhouette_score is :", silhouette_avg)
```

[ ]:
```
plt.plot(range_n_clusters, silhouette)
```

0.2 References

John C. Hull, **Machine Learning in Business: An Introduction to the World of Data Science**, Amazon, 2019.

Paul Wilmott, **Machine Learning: An Applied Mathematics Introduction**, Panda Ohana Publishing, 2019.