02-supervised-and-unsupervised_models

October 19, 2021

Run in Google Colab

1 Supervised and Unsupervised Models

1.1 What is Supervised Learning

1.1.1 Linear Regression

A linear model makes a prediction by simply computing a weighted sum of the input features, plus a constant called the *bias* term (also called the *intercept* term):

$$\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n \tag{1}$$

where:

- $\hat{y}$ is the predicted value;
- n is the number of features;
- x_i is the i^{th} feature value;
- θ_j is the j^{th} model parameter (including the bias term θ_0 and the feature weights $\theta_1, \theta_2, \ldots, \theta_n$

Training a model means setting its parameters so that the model best fits the training set. For this purpose, we first need a measure of how well (or poorly) the model fits the training data. The most common performance measure of a regression model is the Root Mean Square Error (RMSE), therefore, to train a Linear Regression model, you need to find the value of θ that minimizes the RMSE. In practice, it is simpler to minimize the Mean Square Error (MSE) than the RMSE, and it leads to the same result.

1.1.2 Example 1 - Predicting Iowa House Prices (from Kaggle)

[2]:
```
# 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 Problem The objective is to predict the prices of house in Iowa from features. We have 800 observations in training set, 600 in validation set, and 508 in test set

Categorical Features Categorical features are features where there are a number of non-numerical alternatives. We can define a dummy variable for each alternative. The variable equals 1 if the alternative is true and zero otherwise. This is known as **one-hot encoding**. But sometimes we do not have to do this because there is a natural ordering of variables. For example in this problem one of the categorical features is concerned with the basement quality as indicated by the ceiling height. The categories are:

- *Excellent (< 100 inches)*
- *Good (90-99 inches)*
- *Typical (80-89 inches)*
- *Fair (70-79 inches)*
- *Poor (< 70 inches)*
- *No Basement*

This is an example of a categorical variable where *there is* a natural ordering. We created a new variable that had a values of 5, 4, 3, 2, 1 and 0 for the above six categories respectively.

The other categorical features specifies the location of the house as in one of 25 neighborhoods. We introduce 25 dummy variables with a one-hot encoding. The dummy variable equals one for an observation if the neighborhood is that in which the house is located and zero otherwise.

Loading data (J. C. Hull, 2019, Chapter 3) To illustrate the regression techniques discussed in this chapter we will use a total of 48 feature. 21 are numerical and two are categorical and to this we had, as discussed above, 25 categorical variables for the neighborhoods.

[3]:
```
if 'google.colab' in str(get_ipython()):
    from google.colab import files
    uploaded = files.upload()
    path = ''
else:
    path = './data/'
```

[4]:
```
# Both features and target have already been scaled: mean = 0; SD = 1
data = pd.read_csv(path + 'Houseprice_data_scaled.csv')
```

First of all check how many records we have

[5]:
```
print("Number of available data = "  + str(len(data.index)))
```

```
Number of available data = 2908
```

Before starting we emphasize the need to divide all available data into three parts: a **training set**, a **validation set** and a **test set**. The training set is used to determine parameters for trial models. The validation set is used to determine the extent to chich the models created from the training set generalize to new data. Finally, the test set is used as a final estimate of the accuracy of the chosen model.

We had 2908 observations. We split this as follows: 1800 in the training set, 600 in the validation set and 508 in the test set.

[6]:
```
# First 1800 data items are training set; the next 600 are the validation set
train = data.iloc[:1800]
val = data.iloc[1800:2400]
```

We now procede to create **labels** and **features**. As we have already said, the labels are the values of the target that is to be predicted, in this case the 'Sale Price', and we indicate that whit 'y':

[7]:
```
y_train, y_val = train[['Sale Price']], val[['Sale Price']]
```

The features and dummy variables were scaled using the Z-score method. Also the target values (i.e. the house prices) have been scaled with the Z-score method. The features are the variables from which the predictions are to be made and, in this case, can be obtained simply dropping the column 'Sale Price' from our dataset:

[8]:
```
X_train, X_val = train.drop('Sale Price', axis=1), val.drop('Sale Price', axis=1)
```

[9]:
```
X_train.columns
```

[9]:
```
Index(['LotArea', 'OverallQual', 'OverallCond', 'YearBuilt', 'YearRemodAdd',
       'BsmtFinSF1', 'BsmtUnfSF', 'TotalBsmtSF', '1stFlrSF', '2ndFlrSF',
       'GrLivArea', 'FullBath', 'HalfBath', 'BedroomAbvGr', 'TotRmsAbvGrd',
       'Fireplaces', 'GarageCars', 'GarageArea', 'WoodDeckSF', 'OpenPorchSF',
       'EnclosedPorch', 'Blmngtn', 'Blueste', 'BrDale', 'BrkSide', 'ClearCr',
       'CollgCr', 'Crawfor', 'Edwards', 'Gilbert', 'IDOTRR', 'MeadowV',
       'Mitchel', 'Names', 'NoRidge', 'NPkVill', 'NriddgHt', 'NWAmes',
       'OLDTown', 'SWISU', 'Sawyer', 'SawyerW', 'Somerst', 'StoneBr', 'Timber',
       'Veenker', 'Bsmt Qual'],
      dtype='object')
```

Linear Regression with sklearn

[10]:
```
# Importing models
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error as mse
```

[11]:
```
lr = LinearRegression()
lr.fit(X_train,y_train)
```

[11]:
```
LinearRegression()
```

[12]:
```
lr.intercept_
```

[12]:
```
array([-3.06335941e-11])
```

[13]:
```
coeffs = pd.DataFrame(
    [
        ['intercept'] + list(X_train.columns),
        list(lr.intercept_) + list(lr.coef_[0])
    ]
)
coeffs
```

[13]:
```
             0          1            2            3          4             5   \
0    intercept    LotArea  OverallQual  OverallCond  YearBuilt  YearRemodAdd
1 -3.06336e-11  0.0789996     0.214395    0.0964787   0.160799     0.0253524

            6          7            8         9   ...         38        39  \
0  BsmtFinSF1  BsmtUnfSF  TotalBsmtSF  1stFlrSF  ...     NWAmes   OLDTown
1   0.0914664 -0.0330798     0.138199  0.152786  ... -0.0517591 -0.026499

            40         41         42         43         44          45  \
0       SWISU     Sawyer    SawyerW    Somerst    StoneBr      Timber
1 -0.00414298 -0.0181341 -0.0282754  0.0275063  0.0630586 -0.00276173

            46         47
0     Veenker  Bsmt Qual
1  0.00240311  0.0113115

[2 rows x 48 columns]
```

[14]:
```
# Create dataFrame with corresponding feature and its respective coefficients
coeffs = pd.DataFrame(
    [
        ['intercept'] + list(X_train.columns),
        list(lr.intercept_) + list(lr.coef_[0])
    ]
).transpose().set_index(0)
coeffs
```

[14]:
```
                          1
0
intercept     -3.06336e-11
```

```
LotArea          0.0789996
OverallQual       0.214395
OverallCond      0.0964787
YearBuilt         0.160799
YearRemodAdd     0.0253524
BsmtFinSF1       0.0914664
BsmtUnfSF       -0.0330798
TotalBsmtSF       0.138199
1stFlrSF          0.152786
2ndFlrSF          0.132765
GrLivArea         0.161303
FullBath        -0.0208076
HalfBath          0.0171941
BedroomAbvGr    -0.0835202
TotRmsAbvGrd     0.0832203
Fireplaces       0.0282578
GarageCars       0.0379971
GarageArea       0.0518093
WoodDeckSF       0.0208337
OpenPorchSF      0.0340982
EnclosedPorch   0.00682223
Blmngtn         -0.0184305
Blueste         -0.0129214
BrDale          -0.0246262
BrkSide          0.0207618
ClearCr        -0.00737828
CollgCr        -0.00675362
Crawfor          0.0363235
Edwards       -0.000690065
Gilbert        -0.00834022
IDOTRR         -0.00153683
MeadowV          -0.016418
Mitchel         -0.0284821
Names           -0.0385057
NoRidge          0.0515626
NPkVill         -0.0219519
NriddgHt           0.12399
NWAmes          -0.0517591
OLDTown          -0.026499
SWISU          -0.00414298
Sawyer          -0.0181341
SawyerW         -0.0282754
Somerst          0.0275063
StoneBr          0.0630586
Timber         -0.00276173
Veenker         0.00240311
Bsmt Qual        0.0113115
```

[15]:
```
len(coeffs.index)
```

[15]: 48

[16]:
```
pred_t=lr.predict(X_train)
mse(y_train,pred_t)
```

[16]: 0.11401526431246334

[17]:
```
pred_v=lr.predict(X_val)
mse(y_val,pred_v)
```

[17]: 0.11702499460121657

For the data we are considering it turns out that this regression model generalizes well. The mean squared error for the validation set was only a little higher than that for the training set. However linear regression with no regularization leads to some strange results because of the correlation between features. For example it makes no sense that the weights for number of full bathrooms and number of bedrooms are negative!

[18]:
```
x1 = X_train['GrLivArea']
x2 = X_train['BedroomAbvGr']
x1.corr(x2)
```

[18]: 0.5347396038733939

Ridge Regression

[19]:
```
# Importing Ridge
from sklearn.linear_model import Ridge
```

We try using Ridge regression with different values of the hyperparameter λ. The following code shows the effect of this parameter on the prediction error.

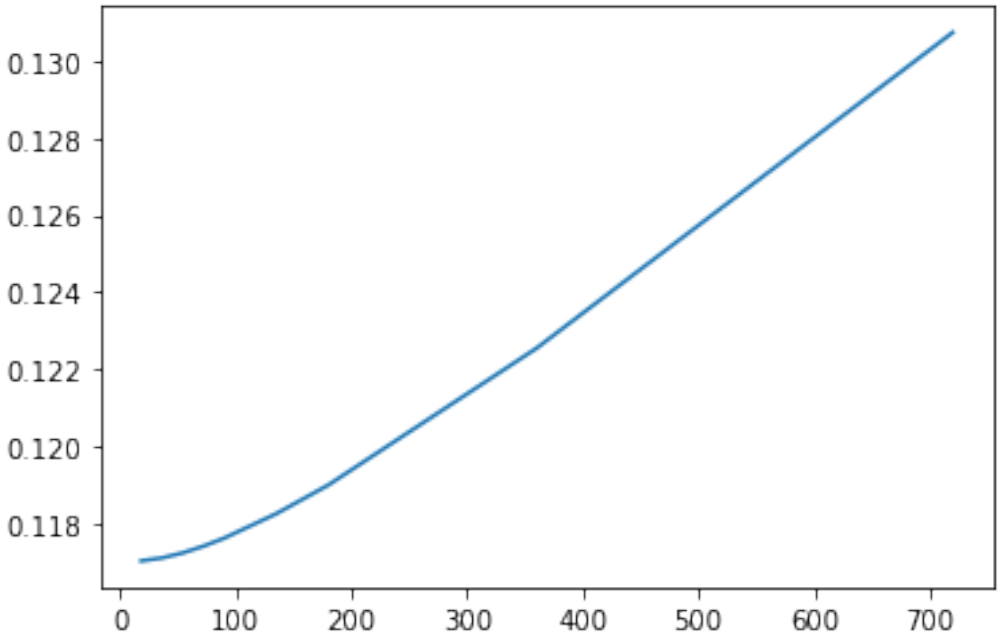
[20]:
```
# The alpha used by Python's ridge should be the lambda in Hull's book times the␣
 ↪number of observations
alphas=[0.01*1800, 0.02*1800, 0.03*1800, 0.04*1800, 0.05*1800, 0.075*1800,0.
 ↪1*1800,0.2*1800, 0.4*1800]
mses=[]
for alpha in alphas:
    ridge=Ridge(alpha=alpha)
    ridge.fit(X_train,y_train)
    pred=ridge.predict(X_val)
    mses.append(mse(y_val,pred))
    print(mse(y_val,pred))
```

```
0.11703284346091342
0.11710797319752984
```

```
0.11723952924901117
0.11741457158889518
0.1176238406871145
0.11825709631198021
0.11900057469147927
0.1225464999629295
0.13073599680747128
```

[21]:
```
plt.plot(alphas, mses)
```

[21]:
```
[<matplotlib.lines.Line2D at 0x2407109ca08>]
```



As expected the prediction error increases as λ increases. Values of λ in the range 0 to 0.1 might be reasonably be considered because prediction errors increases only slightly when λ is in this range. However it turns out that the improvement in the model is quite small for these values of λ.

Lasso

[22]:
```
# Import Lasso
from sklearn.linear_model import Lasso
```

[23]:
```
# Here we produce results for alpha=0.05 which corresponds to lambda=0.1 in␣
 ↪Hull's book
lasso = Lasso(alpha=0.05)
lasso.fit(X_train, y_train)
```

[23]:
```
Lasso(alpha=0.05)
```

[24]:
```
# DataFrame with corresponding feature and its respective coefficients
coeffs = pd.DataFrame(
    [
        ['intercept'] + list(X_train.columns),
        list(lasso.intercept_) + list(lasso.coef_)
    ]
).transpose().set_index(0)
coeffs
```

[24]:

0	1
intercept	-1.25303e-11
LotArea	0.0443042
OverallQual	0.298079
OverallCond	0
YearBuilt	0.0520907
YearRemodAdd	0.0644712
BsmtFinSF1	0.115875
BsmtUnfSF	-0
TotalBsmtSF	0.10312
1stFlrSF	0.0322946
2ndFlrSF	0
GrLivArea	0.297065
FullBath	0
HalfBath	0
BedroomAbvGr	-0
TotRmsAbvGrd	0
Fireplaces	0.0204043
GarageCars	0.027512
GarageArea	0.0664096
WoodDeckSF	0.00102883
OpenPorchSF	0.00215018
EnclosedPorch	-0
Blmngtn	-0
Blueste	-0
BrDale	-0
BrkSide	0
ClearCr	0
CollgCr	-0
Crawfor	0
Edwards	-0
Gilbert	0
IDOTRR	-0
MeadowV	-0
Mitchel	-0
Names	-0
NoRidge	0.013209

```
NPkVill                -0
NriddgHt        0.0842993
NWAmes                 -0
OLDTown                -0
SWISU                  -0
Sawyer                 -0
SawyerW                -0
Somerst                 0
StoneBr         0.0168153
Timber                  0
Veenker                 0
Bsmt Qual       0.0202754
```

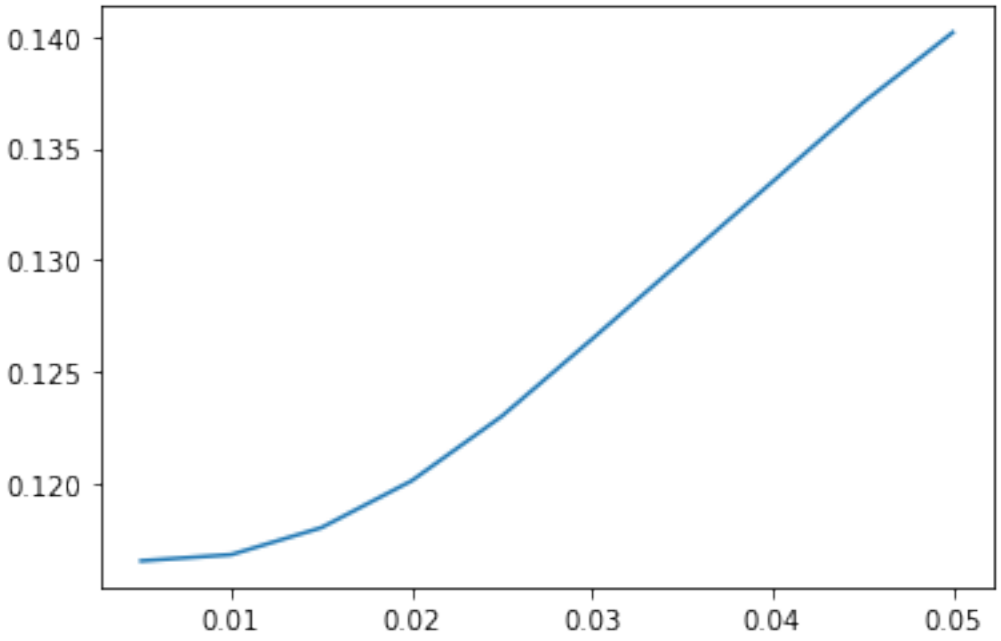
Lasso with different levels of alpha and its mse

[25]:
```
# We now consider different lambda values. The alphas are half the lambdas
alphas=[0.01/2, 0.02/2, 0.03/2, 0.04/2, 0.05/2, 0.06/2, 0.08/2, 0.09/2, 0.1/2]
mses=[]
for alpha in alphas:
    lasso=Lasso(alpha=alpha)
    lasso.fit(X_train,y_train)
    pred=lasso.predict(X_val)
    mses.append(mse(y_val,pred))
    print("lambda = " + '{:<05}'.format(alpha) + " - mse = " +␣
 ↪str(round(mse(y_val, pred),6)))
```

```
lambda = 0.005 - mse = 0.116548
lambda = 0.010 - mse = 0.116827
lambda = 0.015 - mse = 0.118033
lambda = 0.020 - mse = 0.120128
lambda = 0.025 - mse = 0.123015
lambda = 0.030 - mse = 0.126462
lambda = 0.040 - mse = 0.133492
lambda = 0.045 - mse = 0.137016
lambda = 0.050 - mse = 0.140172
```

[26]:
```
plt.plot(alphas, mses)
```

[26]:
```
[<matplotlib.lines.Line2D at 0x240751c6bc8>]
```



Lasso regression leads to more interesting results. In the plot above you can see how the error in the validation set changes as tha value of the lasso λ increases. For small values of λ the error is actually less than when $\lambda = 0$ but as λ increases beyond about 0.03 the error starts to increase. A value of $\lambda = 0.04$ could be chosen.

1.2 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.

1.2.1 *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.

By Chire - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=59409335

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:

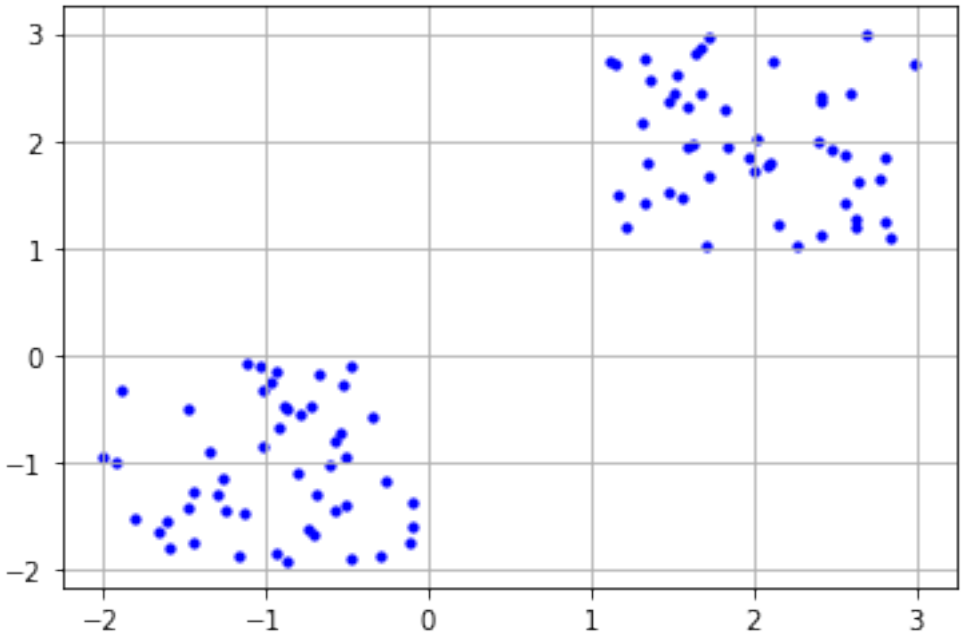
$$Distance = \sqrt{(x_B - x_B)^2 + (y_B - y_A)^2}$$

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.

[27]:
```
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:

[28]:
```
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.

[29]:
```
from sklearn.cluster import KMeans
Kmean = KMeans(n_clusters=2)
Kmean.fit(X)
```

[29]:
```
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:

[30]:
```
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)
```

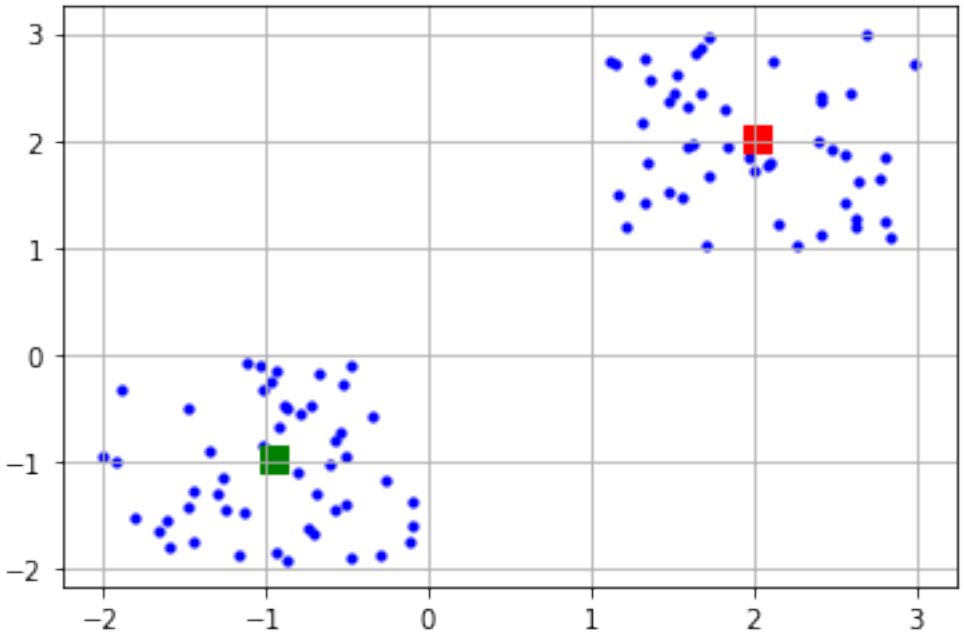
[30]:
```
KMeans(n_clusters=2, n_jobs=1, precompute_distances='auto')
```

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

[31]:
```
array([[ 1.97987956,  1.97644875],
       [-0.94247201, -1.05421376]])
```

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, 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, 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])
```

1.2.2 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).

1.2.3 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())
```

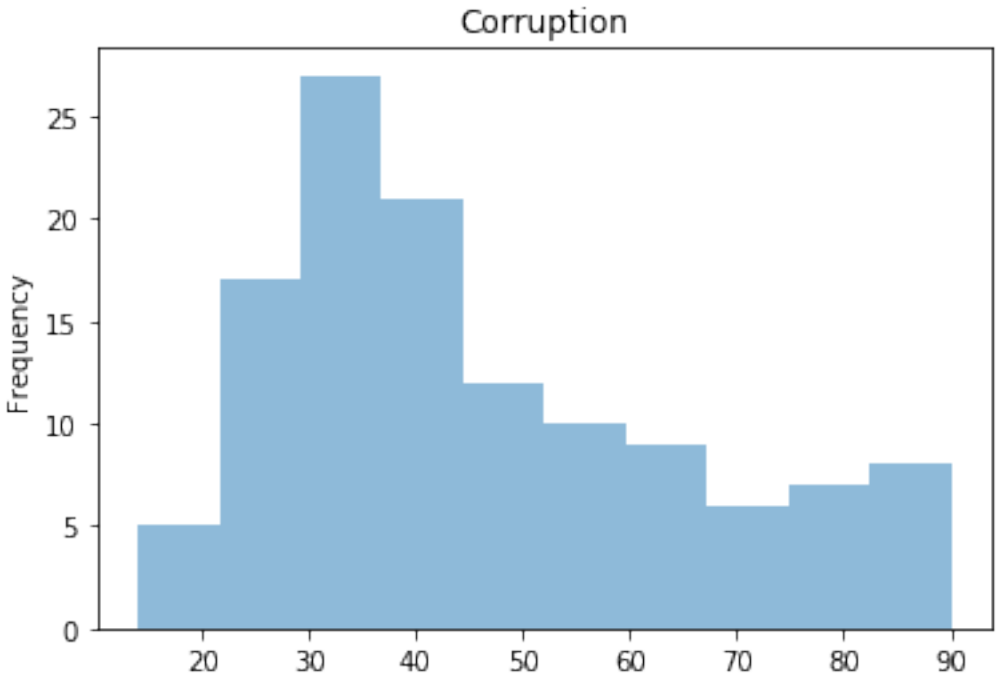
```
Summary statistics
        Corruption       Peace       Legal  GDP Growth
count  122.000000  122.000000  122.000000  122.000000
mean    46.237705    2.003730    5.598861    2.372566
std     19.126397    0.447826    1.487328    3.241424
min     14.000000    1.192000    2.728000  -18.000000
25%     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
```

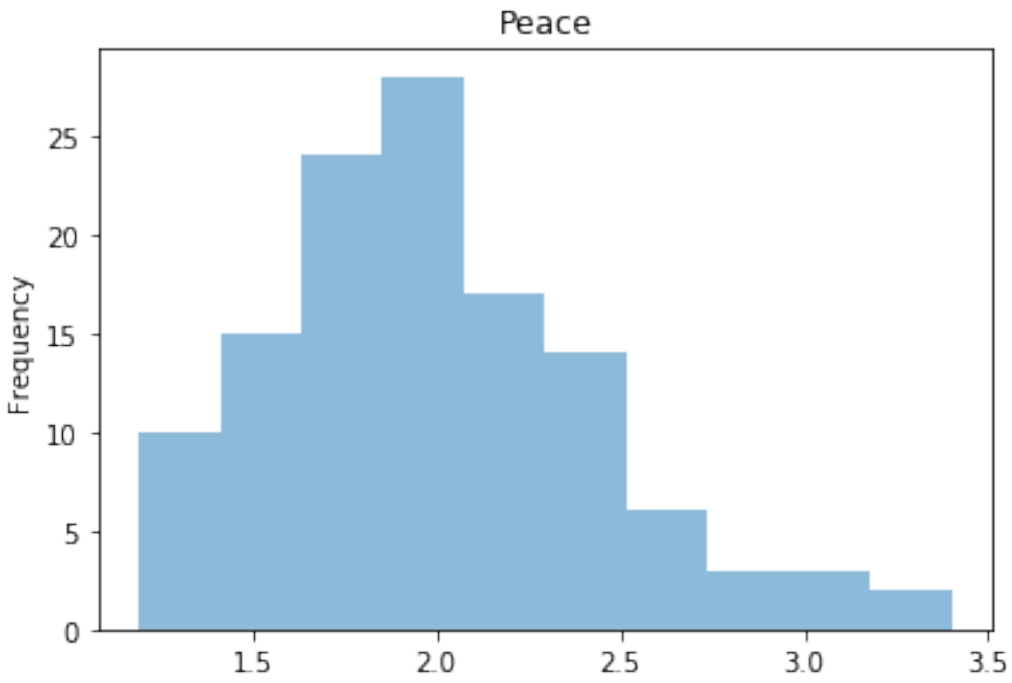
Plot histogram

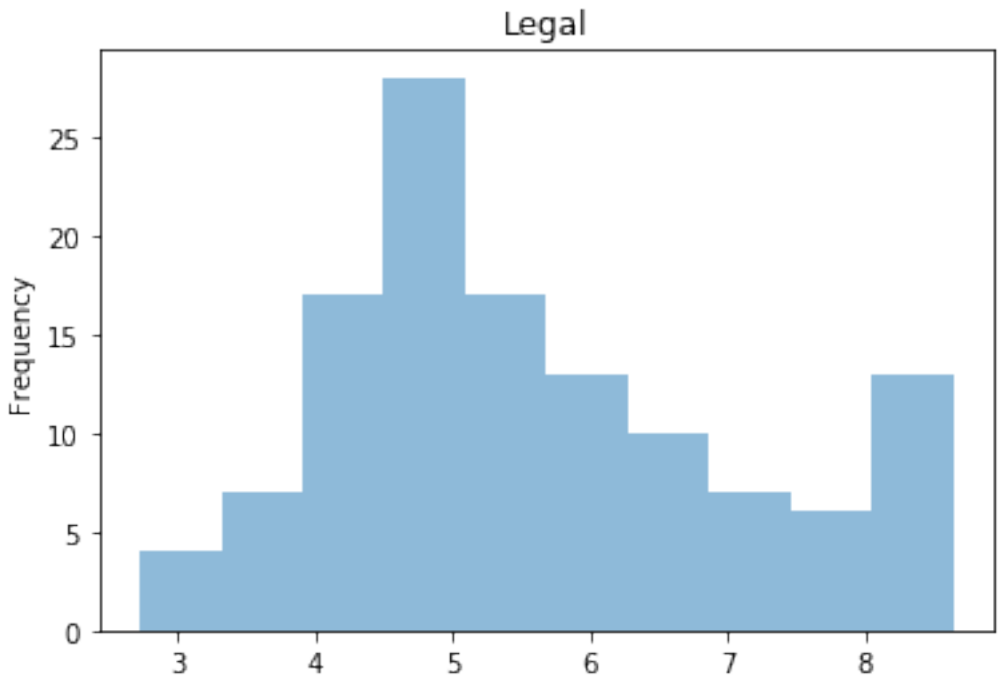
Note that distributions for GDP Growth is quite skewed.

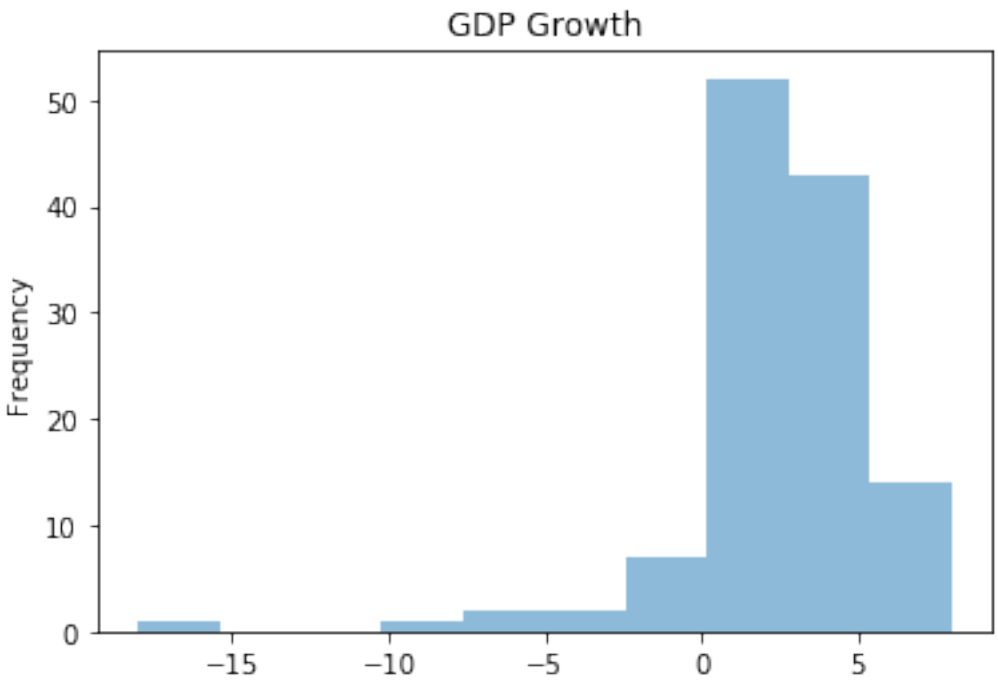
[39]:
```
# plot histograms
plt.figure(1)
raw['Corruption'].plot(kind = 'hist', title = 'Corruption', alpha = 0.5)
```

```
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



1.2.4 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
```

1.2.5 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

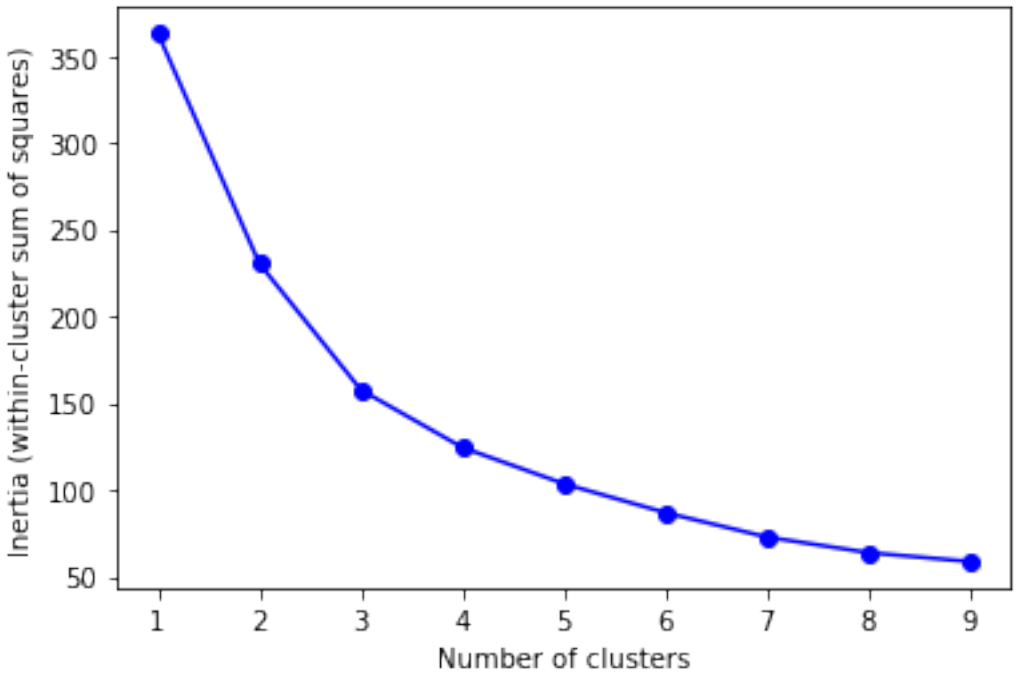
can be used to choose the number of parameters in other data-driven models, such as the number of principal components to describe a data set.

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**

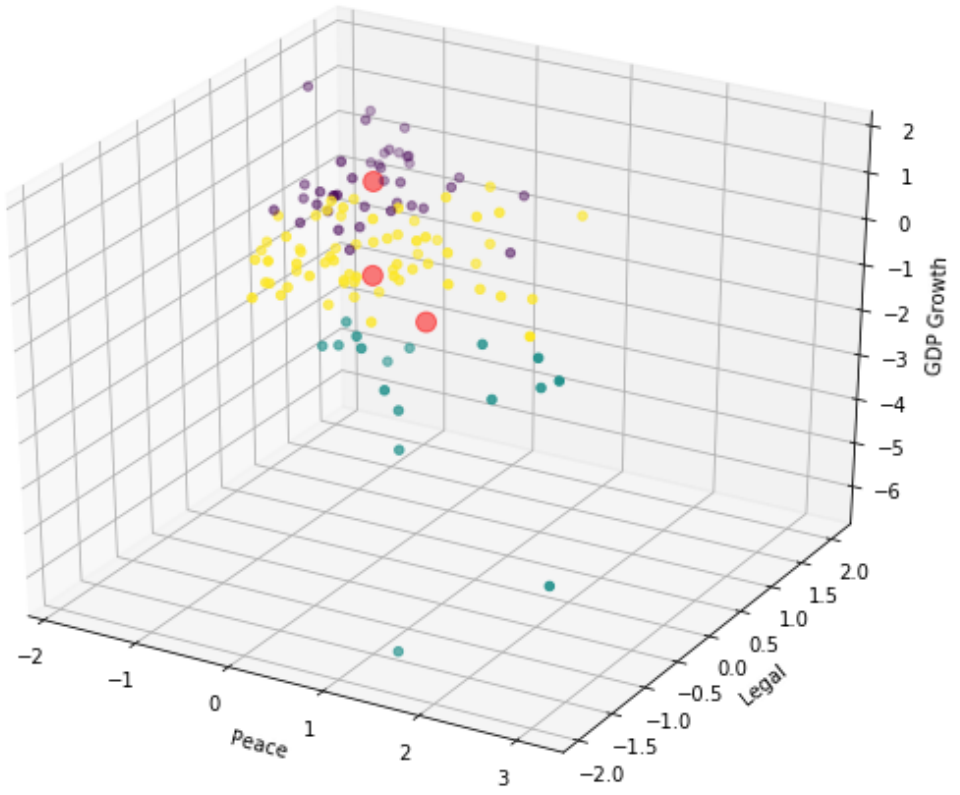
[42]:
```
k = 3
kmeans = KMeans(n_clusters=k, random_state=0)
kmeans.fit(X)
```

```
# 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]
```

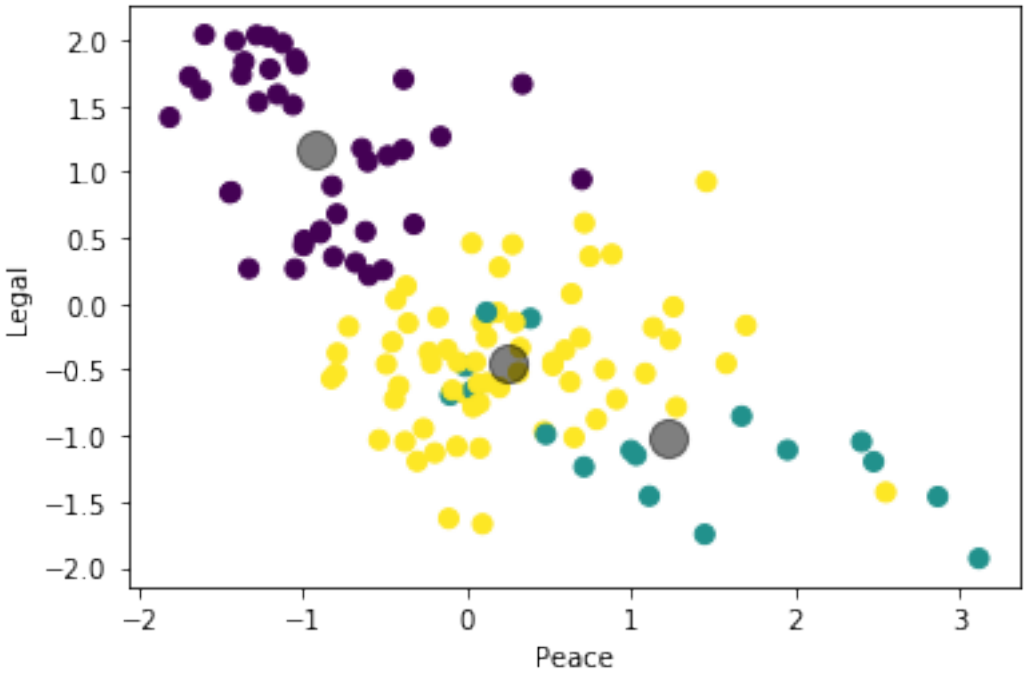
Visualize the result (3D plot)

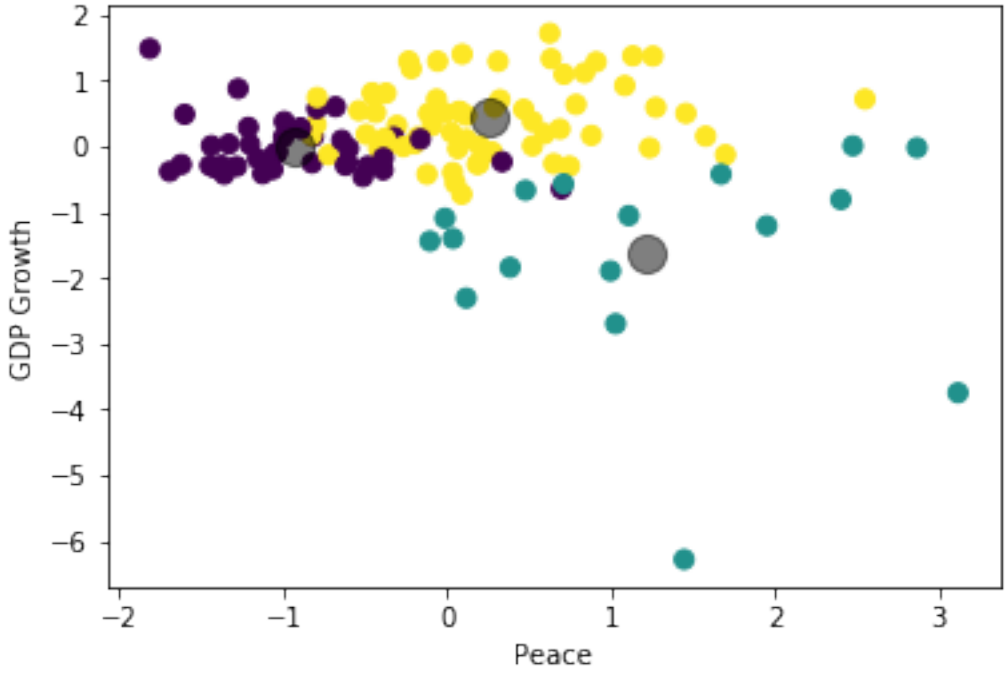
[43]:
```
# set up the color
norm = clrs.Normalize(vmin=0.,vmax=y.max())
cmap = cm.viridis

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()
```

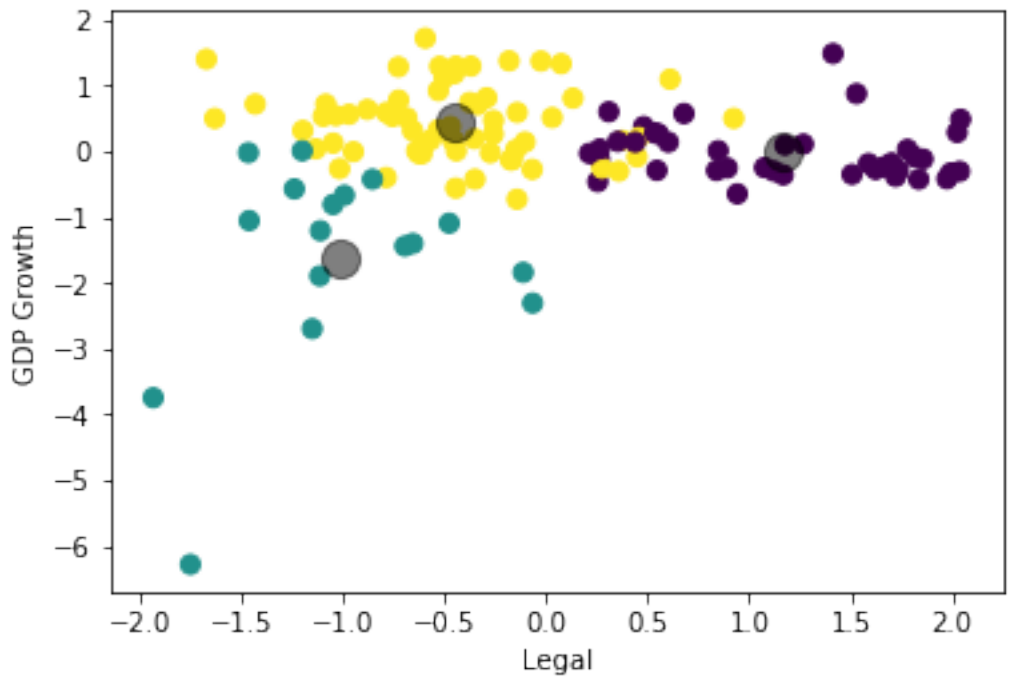


Visualize the result (3 2D plots)

[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

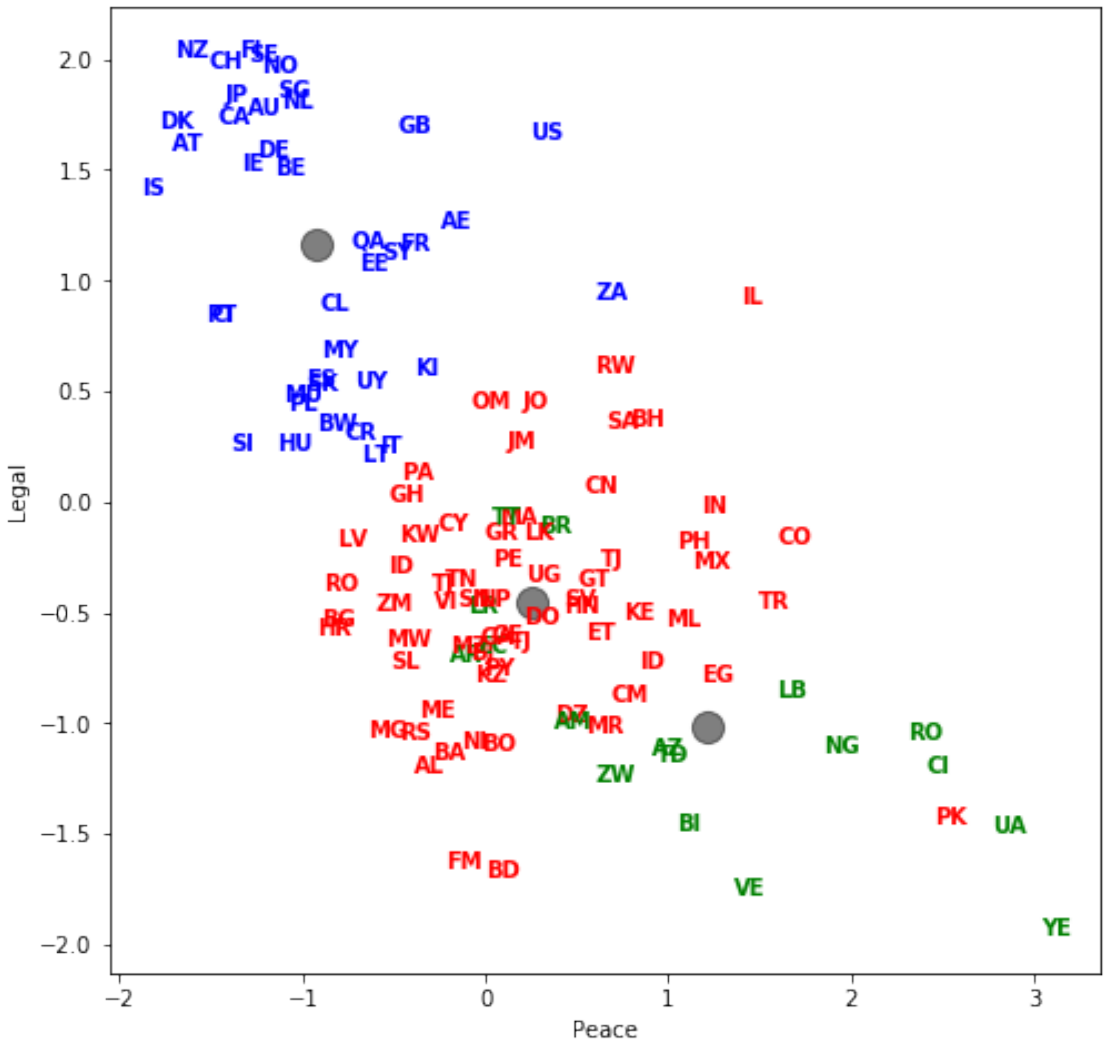

GDP Growth
Peace
2
1
0
−1
−2
−3
−4
−5
−6
−2
−1
0
1
2
3

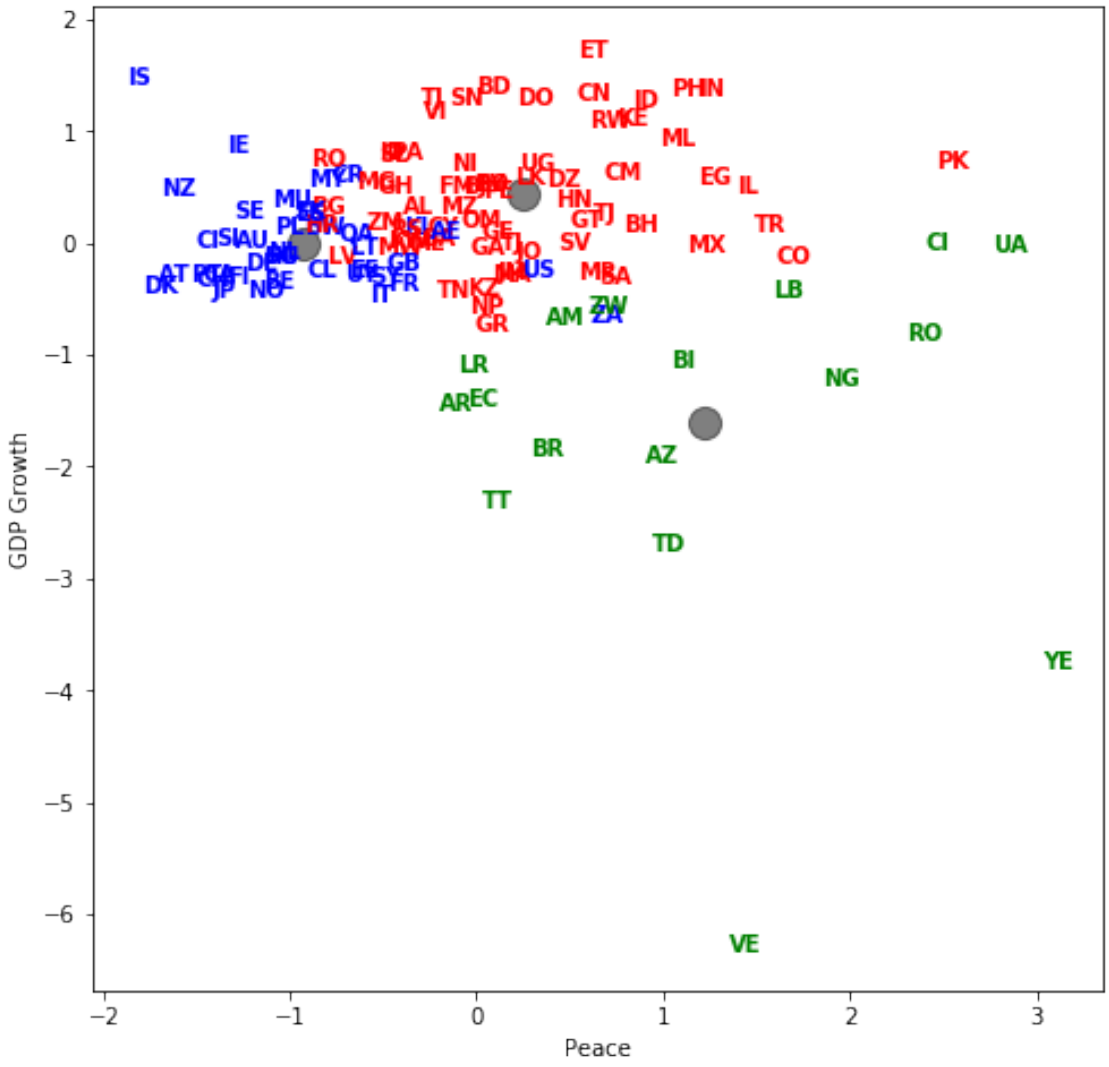


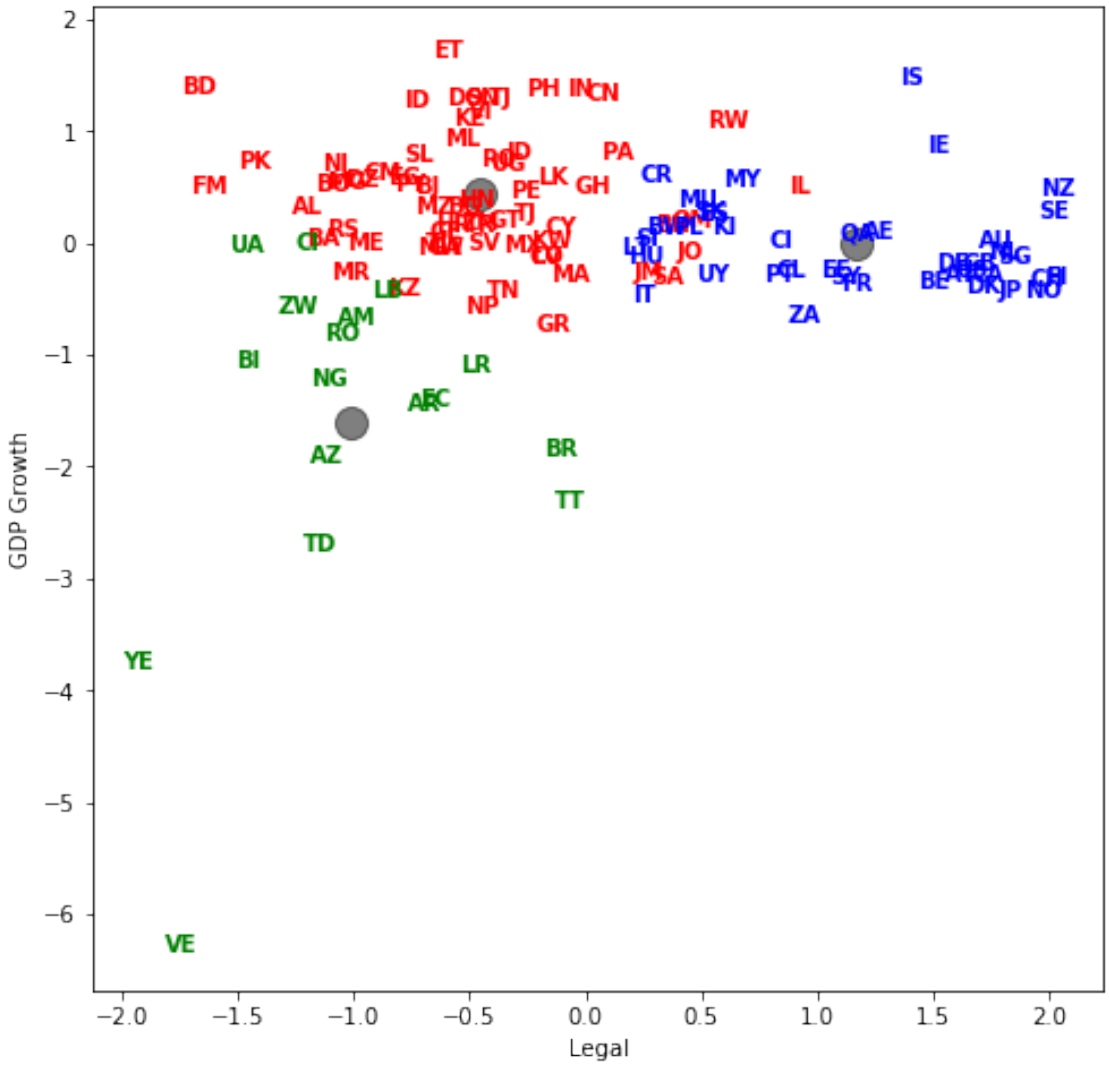
Visualize the result (3 2D plots)

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
AU
DK
CA
GB
US
AT
IE
DE
BE
IS
AE
QA
FR
EE
ZA
IL
CL
MY
UY
KI
RW
OM
JO
SA
BH
SI
HU
BW
CR
LT
JM
PA
GH
CN
IN
CY
KW
GR
BR
LV
PH
CO
ID
PE
TJ
MX
RO
UG
GT
ZM
TR
KE
ML
DO
ET
MW
SL
ID
EG
LB
CM
ME
DZ
AM
MR
RS
BA
BO
AL
NG
RO
CI
ZW
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
PHIN
ML
IE
PK
NZ
EG
IL
SE
CM
DZ
HN
BH
TR
MX
CO
CI
UA
LB
RO
LR
BI
NG
EC
BR
AZ
TT
TD
YE
VE



1.2.6 List the result

[46]:
```
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'))
```

```
                    Country Abbrev  Label
23                Costa Rica     CR      0
88                     Qatar     QA      0
26           Czech Republic     CI      0
87                 Portugal     PT      0
28                  Denmark     DK      0
86                   Poland     PL      0
```

79	Norway	NO	0
76	New Zealand	NZ	0
33	Estonia	EE	0
75	Netherlands	NL	0
35	Finland	FI	0
36	France	FR	0
68	Mauritius	MU	0
65	Malaysia	MY	0
39	Germany	DE	0
62	Lithuania	LT	0
57	Korea (South)	KI	0
53	Japan	JP	0
44	Hungary	HU	0
45	Iceland	IS	0
51	Italy	IT	0
96	Singapore	SG	0
49	Ireland	IE	0
20	Chile	CL	0
115	United States	US	0
114	United Kingdom	GB	0
113	United Arab Emirates	AE	0
9	Belgium	BE	0
97	Slovakia	SK	0
116	Uruguay	UY	0
104	Taiwan	SY	0
5	Austria	AT	0
13	Botswana	BW	0
103	Switzerland	CH	0
102	Sweden	SE	0
100	Spain	ES	0
99	South Africa	ZA	0
18	Canada	CA	0
98	Slovenia	SI	0
4	Australia	AU	0
119	Yemen	YE	1
117	Venezuela	VE	1
112	Ukraine	UA	1
108	Trinidad and Tobago	TT	1
78	Nigeria	NG	1
90	Russia	RO	1
61	Liberia	LR	1
60	Lebanon	LB	1
121	Zimbabwe	ZW	1
14	Brazil	BR	1
19	Chad	TD	1
27	Democratic Republic of Congo	CI	1
30	Ecuador	EC	1
6	Azerbaijan	AZ	1

16	Burundi	BI	1
2	Argentina	AR	1
3	Armenia	AM	1
24	Croatia	HR	2
91	Rwanda	RW	2
89	Romania	RO	2
93	Senegal	SN	2
94	Serbia	RS	2
95	Sierra Leone	SL	2
22	Colombia	CO	2
21	China	CN	2
25	Cyprus	CY	2
17	Cameroon	CM	2
92	Saudi Arabia	SA	2
47	Indonesia	ID	2
15	Bulgaria	BG	2
12	Bosnia and Herzegovina	BA	2
105	Tanzania	TJ	2
106	Thailand	TJ	2
107	The FYR of Macedonia	TJ	2
11	Bolivia	BO	2
109	Tunisia	TN	2
110	Turkey	TR	2
111	Uganda	UG	2
10	Benin	BJ	2
8	Bangladesh	BD	2
7	Bahrain	BH	2
118	Vietnam	VI	2
1	Algeria	DZ	2
101	Sri Lanka	LK	2
29	Dominican Republic	DO	2
85	Philippines	PH	2
84	Peru	PE	2
63	Madagascar	MG	2
40	Ghana	GH	2
41	Greece	GR	2
120	Zambia	ZM	2
59	Latvia	LV	2
58	Kuwait	KW	2
64	Malawi	MW	2
42	Guatemala	GT	2
55	Kazakhstan	KZ	2
54	Jordan	JO	2
43	Honduras	HN	2
52	Jamaica	JM	2
46	India	IN	2
50	Israel	IL	2
56	Kenya	KE	2

38	Georgia	GE	2
66	Mali	ML	2
67	Mauritania	MR	2
83	Paraguay	PY	2
82	Panama	PA	2
81	Pakistan	PK	2
80	Oman	OM	2
31	Egypt	EG	2
77	Nicaragua	NI	2
32	El Salvador	SV	2
34	Ethiopia	ET	2
74	Nepal	NP	2
73	Mozambique	MZ	2
72	Morocco	MA	2
71	Montenegro	ME	2
70	Moldova	FM	2
69	Mexico	MX	2
37	Gabon	GA	2
48	Iran	ID	2
0	Albania	AL	2

1.2.7 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

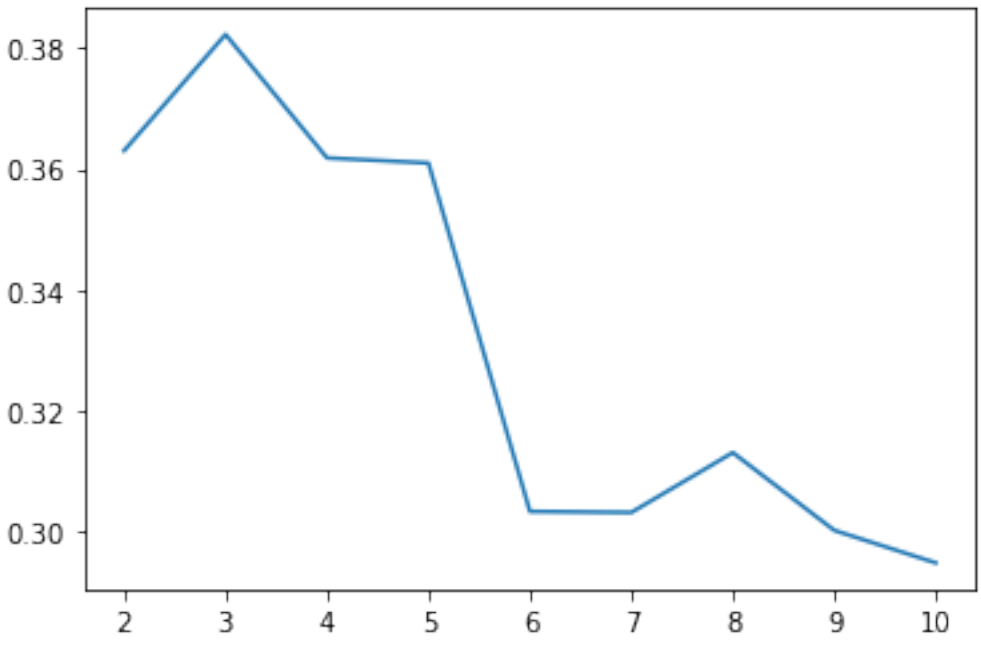
$$s(i) = \frac{b(i) - a(i)}{\max[a(i), b(i)]} \tag{2}$$

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

[47]:
```
# 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)
```

[48]:
```
plt.plot(range_n_clusters, silhouette)
```

[48]:
```
[<matplotlib.lines.Line2D at 0x24075aa1188>]
```



1.3 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.