Importation of:

reader, sqrt and numpy

In [192]:

```
# Make Predictions with k-nearest neighbors on the Iris Flowers Dataset
from csv import reader
from math import sqrt
import numpy as np
```

Load csv File

We will load_csv() function to open the file in read mode. It will loop each row and if it is not a row we shall join them.

In [193]:

```
# Load a CSV file
def load_csv(filename):
 dataset = list()
 with open(filename, 'r') as file:
  csv_reader = reader(file)
  for row in csv_reader:
   if not row:
    continue
   dataset.append(row)
 return dataset
```

Assigning a name to the file

We assign filename to the file 'African_Crises_Project1.1_Reduced.csv

In [194]:

```
# Assign the file a name
filename = 'African_Crises_Project1.1_Reduced.csv'
dataset = load_csv(filename)
```

Convert String To Integer

We shall convert each column name to floating point values and remove any whitespaces.

In [195]:

```
# Convert string column to float
def str_column_to_float(dataset, column):
 for row in dataset:
  row[column] = float(row[column].strip())
```

Convert string to integer

We shall use str_column_to_int() function to print the mapping of string names to integers.

In [196]:

```
# Convert string column to integer
def str_column_to_int(dataset, column):
 class_values = [row[column] for row in dataset]
 unique = set(class_values)
 lookup = dict()
 for i, value in enumerate(unique):
  lookup[value] = i
  print('[%s] => %d' % (value, i))
 for row in dataset:
  row[column] = lookup[row[column]]
 return lookup
```

Convert String To Integer

We Loop each row and convert values from string to integer

In [197]:

```
for i in range(len(dataset[0])-1):
 str_column_to_float(dataset, i)
# convert class column to integers
str_column_to_int(dataset, len(dataset[0])-1)
```

```
[sovereign_external_debt_default] => 0
[independence] => 1
[banking_crisis] => 2
[domestic_debt_in_default] => 3
[inflation_crises] => 4
[exch_usd] => 5
[gdp_weighted_default] => 6
[systemic_crisis] => 7
[currency_crises] => 8
[inflation_annual_cpi] => 9
```

Out[197]:

```
{'sovereign_external_debt_default': 0,
 'independence': 1,
 'banking_crisis': 2,
 'domestic_debt_in_default': 3,
 'inflation_crises': 4,
 'exch_usd': 5,
 'gdp_weighted_default': 6,
 'systemic_crisis': 7,
 'currency_crises': 8,
 'inflation_annual_cpi': 9}
```

Getting The distance between clusters

We shall define distance() function. We will return a numpy array taking the differnce between one row dataset and another. For example we will find the distance between systemic_crisis cluster and the rest of the clusters.

In [198]:

```
def distance(instance1, instance2):
    instance1 = np.array(instance1)
    instance2 = np.array(instance2)
    return np.linalg.norm(instance1 - instance2)
#print(distance([3, 5], [1, 1]))
print('exch_usd:')
print(distance(dataset[0], dataset[1]))
print('domestic_debt_in_default:')
print(distance(dataset[0], dataset[2]))
print('sovereign_external_debt_default:')
print(distance(dataset[0], dataset[3]))
print('gdp_weighted_Default:')
print(distance(dataset[0], dataset[4]))
print('inflation_annual_cpi:')
print(distance(dataset[0], dataset[5]))
print('independence:')
print(distance(dataset[0], dataset[6]))
print('currency_crises:')
print(distance(dataset[0], dataset[7]))
print('inflation_crises:')
print(distance(dataset[0], dataset[8]))
print('banking_crisis:')
print(distance(dataset[0], dataset[9]))
```

```
exch_usd:
3885.7015967354496
domestic_debt_in_default:
10.954451150103322
sovereign_external_debt_default:
14.730919862656235
gdp_weighted_Default:
9.158165755215396
inflation_annual_cpi:
21989794.792916346
independence:
27.892651361962706
currency_crises:
13.674794331177344
inflation_crises:
13.19090595827292
banking_crisis:
7.0
```

In this case we see that most related factors to systemic_crisis are:

1) banking_crisis 2) gdp_weighted_default 3) domestic_debt_in_default

Now let us predict the cluster in which a new dataset belongs to in relation with the dataset we have.

Finding Min and Max Values

We shall for loop to wrap the first dataset to the column names. We then assign the min values and max values to min(col_values) and max(col_values)

In [199]:

```
# Find the min and max values for each column
def dataset_minmax(dataset):
 minmax = list()
 for i in range(len(dataset[0])):
  col_values = [row[i] for row in dataset]
  value_min = min(col_values)
  value_max = max(col_values)
  minmax.append([value_min, value_max])
 return minmax
```

Normalizing the dataset

We shall define normalize_dataset() function, then we use a for loop, to take values in each row and subtract the minimum value and divide it by maximum value minus the minimum value.

In [200]:

```
# Rescale dataset columns to the range 0-1
def normalize_dataset(dataset, minmax):
 for row in dataset:
  for i in range(len(row)):
   row[i] = (row[i] - minmax[i][0]) / (minmax[i][1] - minmax[i][0])
```

Define Euclidean Distance

We shall define euclidean_distance() function. We initialise the distanced to 0.0. We ignore the last column which is taken as the attributes. We use for loop to wrap each dataset and square the distance between two datasets to avoid negative values.

In [201]:

```
# Calculate the Euclidean distance between two vectors
def euclidean_distance(row1, row2):
 distance = 0.0
 for i in range(len(row1)-1):
  distance += (row1[i] - row2[i])**2
 return sqrt(distance)
```

Locating most similar neighbors

We define get_neighbors() function. The train_row distance is sorted ensuring the second tuple, tup[1] is used in the sorting operation. List of similar neighbors to the test_row is returned.

In [202]:

```
# Locate the most similar neighbors
def get_neighbors(train, test_row, num_neighbors):
 distances = list()
 for train_row in train:
  dist = euclidean_distance(test_row, train_row)
  distances.append((train_row, dist))
 distances.sort(key=lambda tup: tup[1])
 neighbors = list()
 for i in range(num_neighbors):
  neighbors.append(distances[i][0])
 return neighbors
```

Predicting class

To return the most represented class in the classification we use the max() function. The max() function, returns takes a class unique values and calls the count on the list of class values for each class in the set.

In [203]:

```
# Make a prediction with neighbors
def predict_classification(train, test_row, num_neighbors):
 neighbors = get_neighbors(train, test_row, num_neighbors)
 output_values = [row[-1] for row in neighbors]
 prediction = max(set(output_values), key=output_values.count)
 return prediction
```

Assigning file a name

We shall assign our African_Crises_Project1.1_Reduced.csv file the name filename.

In [204]:

```
# Make a prediction with KNN on Iris Dataset
filename = 'African_Crises_Project1.1_Reduced.csv'
dataset = load_csv(filename)
```

Printing the predicted class

We shall use for loop to wrap each dataset apart from the last column which is attributes. We then convert string columns to floating point and string columns to integers. Having the predict_classification() function, we will use a row with a new observation to predict the class label.

In [205]:

```
for i in range(len(dataset[0])-1):
 str_column_to_float(dataset, i)
# convert class column to integers
str_column_to_int(dataset, len(dataset[0])-1)
# define model parameter
num_neighbors = 4
# define a new record
row = [
 [3.441455696, 14.14913958, -3.718592965, 11.20389701, -3.848560701, -20.92417833, -1.76
9547325, 29.11604525, -1.492537313, -16.83135705, 3.881188119, 12.61913839, -12.35612729
, -1.390498262, -15.94202899, 9.756097561, 22.22222222, 16.36363636, 28.125, 46.34146341
, 41.66666667, 29.41176471, 18.18181818, 69.23076923, 63.63636364, 23.61111111, 0, 6.741
573034, 6.315789474, -0.99009901, 1, 0, 1.98019802, 2.912621359, 12.26415094, 9.24369747
9, 5.384615385, 2.919708029, 6.599987838, 2.626631862, 3.656320975, 6.172816266, 4.73422
8715, 8.961134756, 8.87917187, 12.09893382, 17.17351914, 11.45037883, 9.668492611, 14.60
957788, 9.668, 14.61, 6.593, 7.835, 6.31, 10.432, 14.007, 5.857, 5.938, 9.172, 9.272, 25
.9, 31.7, 20.5, 29, 29.8, 18.7, 5.7, 4.95, 2.6, 0.3, 4.2, 1.43, 4.259, 3.972, 1.382, 2.3
15, 3.674, 4.855, 5.743, 3.913, 4.522, 8.916, 3.255, 2.917, 29.62962963, 45.71428571, 68
.62745098, 126.744186, -27.17948718, -7.042253521, -17.42424242, -9.174311927, -5.050505
051, -5.319148936, 0, -2.247191011, 2.298850575, 3.370786517, 5.434782609, -1.030927835,
2.083333333, 2.040816327, 1, 13.86138614, 16.52173913, 17.91044776, 13.92405063, 7.77777
7778, 5.154639175, 4.901960784, 13.55140187, -16.87242798, -0.99009901, 4, -7.692307692,
3.125, 0, -4.04040404, 0, 3.157894737, 1.020408163, 2.02020202, 0, -0.99009901, 2, 5.882
352941, 7.965434021, 5.777758599, 15.79827455, 15.67485015, 27.41525989, 29.00043489, 80
.69990445, 69.00935079, 48.46106867, 101.3013176, 46.70755462, 85.265, 299.097, 1379.476
, 949.771, 2672.23, 4146.01, 221.492, 107.429, 248.248, 325.029, 152.586, 108.893, 98.34
2, 43.559, 22.961, 13.305, 12.249, 12.465, 13.721, 14.48, 13.484, 10.285, 8.782, 7.296,
6.818181818, 12.76595745, 3.773584906, 10.90909091, 6.557377049, 7.692307692, 3.71428571
4, 10.05509642, 9.386733417, 2.745995423, 1.893095768, 4.808743169, 0.521376434, 3.73443
9834, 27.69953052, -8.823529412, 5.64516129, 9.541984733, 16.02787456, 10.51051051, 11.1
4130435, 11.49144254, 9.210526316, 13.3, 14.66, 13.243, 14.558, 2.604, 10.457, 2.411, -6
.986, -3.934, 0.645, -0.203, -2.848, -0.757, -2.909, 24.55, 19.2, 3.723, 1.597, -1.871,
-1.415, 3.202, 3.843, 2.298, 4.353, -2.244, 2.885, 6.694, 0.935, 9.262, 3.522, 1.491, 1.
195, 5.874, 6.552, 11.584, 16.21609969, 2.326185353, -2.273304086, 2.326185353, 4.544795
33, 13.04340287, 21.15476063, 6.348281887, -2.152483452, 14.43033046, -0.177568901, 8.71
9549217, -3.169044947, 1.757528693, 4.665098294, 2.766186537, 7.747595621, 2.502927383,
8.075116891, -1.07225635, 1.194615424, 12.9325741, 18.22418299, 10.61769304, 12.90885269
, 24.48243736, 16.03173523, 21.75954473, 8.81, 8.682, 7.389, 5.879, 4.281, 1.753, 6.837,
6.978, 6.935, 0.986, -0.658, 1.575, 4.218, 2.13, 25.956, 14.1, 2.7, 6.301, 5.219, 0.92,
-0.381, 4.355, 3.08, 3.297, 1.459, 3.884, 2.466, 1.896, 6.315, 1.009, 1.8, 4.448, 1.3, 2
.584, 0.449, 5.944477972, 5.952720023, 5.940860215, 5.95026643, 5.951383068, 5.944846293
, 5.941967143, 5.95106233, 5.94943927, 5.938284894, 5.952582557, 12.97850236, -4.2017401
15, -4.171896921, 1.556480197, -8.528072838, -10.87425348, -0.43741275, -0.514114788, 10
.55153622, 1.784803672, -12.40814963, 4.232602479, -2.231571246, -0.477081384, -6.438575
054, 3.616636528, -2.840798914, -6.216944417, -12.33241115, 7.949993931, 10.17539915, 1.
959383611, -13.92253028, -4.953488372, -5.578664057, 9.70458668, 2.846344632, 14.2168121
3, -6.374422378, 7.388316151, 0.11, -0.549395665, -7.533145842, -2.639582881, 15.3631596
6, 9.448742747, 1.307767076, 8.87047536, 1.185707419, -11.7418844, 1.291827398, 6.243911
08, 11.40380127, -1.137384017, -16.00060551, 19.326921, 35.4800225, 18.04671609, 12.1743
099, 20.60262757, -28.50213688, -11.8067676, -10.90447887, 7.201266764, 5.682103785, -11
.35479535, -8.356955722, 4.5057024, -3.339660342, -6.922294859, -5.020231154, -10.848239
95, -9.7931274, 23.96925479, 5.648570427, -7.249727068, -1.022813275, 11.43314281, -1.43
5148115, 10.38362042, 24.18818835, 40.76638415, 17.48643148, 11.37938502, 0, 2.119179547
, -3.112803427, 0.713958211, 0, 8.510015609, 7.599868141, -10.25483916, -0.952735624, -2
.885699962, 0, 3.961929343, 3.249231915, -1.340547039, 0.694613697, 0.726128525, 1.17746
005, -4.203776274, 2.060865254, 7.824338769, 13.12090131, 6.693697414, 0.119617553, 0.45
5818017, 3.676736584, 3.36871659, 2.942979233, 2.424102664, 5.740312478, 10.64040499, 9.
899265451, 10.29192345, 11.61462645, 11.27087249, 9.90279235, 20.5, 10.4, 14.9, 15.982,
17.06, 12.108, 23.9, 25.185, 15.185, 20.129, 21.219, 14.737, 21.142, 11.042, 9.046, 9.36
1, 7.095, 6.167, 5.041, 3.745, 2.849, 2.431, 3.21, 0, 8.106, 8.826, 4.202, 10.959, 11.69
8, 16.24, 11.69, 11.09, 8.65, 6.914, 10.099, 9.339830607, 3.522920204, 8.553710537, 10.8
7767339, 6.450926123, 3.032, 4.410280301, 6.369450435, 1.786276086, 2.630675184, 0, 0.85
6645697, 0.846051758, 2.520151341, 2.740605244, 0, -0.285715557, 5.762329909, 3.76191313
4, 0.537398087, 0.661865236, -0.131090559, 1.832593812, 5.614773537, 4.52521913, 12.6628
9775, 16.41376144, 19.7347384, 9.527607326, 18.01088176, 15.35621451, 8.080283232, 13.86
6, 7.895, 13.821, 11.603, 20.667, 11.398, 10.284, 13.007, 4.804, 7.617, 11.2, 19.104, 27
.332, 45.979, 28.814, 1.554, 8.862, 11.924, 6.716, 5.753, 9.955, 5.824, 2.156, 5.983, 8.
381, 7.823, 6.041, 4.265, 15.101, 10.552, 4.309, 14.022, 9.378, 5.717, 6.878, -9.0909090
91, 31.99785465, 1.023932388, 4.904878735, 11.15115499, 1.941981994, -0.634431887, 0.956
872627, -0.315372291, 2.857481221, -0.926032535, 0.622571969, -0.618719992, 1.8693761, -
0.612777053, 0.798570117, 1.753672463, 1.8269885, 1.893322561, 2.982014118, 2.937830401,
5.329676052, 2.978677387, 0.011228728, 1.886196037, 5.551930333, 20.82296472, 23.7980168
3, 15.20534955, 10.75945144, 9.794534493, 9.063188827, 23.21616753, 33.04, 26.45, 13.384
, 7.452, 5.525, 8.321, 4.336, 0.719, 3.6, 13.61, 10.705, 12.817, 2.857, 15.288, 7.335, 6
.026, 6.553, 6.831, 6.808, 6.872, 4.23, 5.391, 6.414, 3.931, 4.703, 4.921, 8.93, 8.827,
9.731, 2.516, 2.929, 6.526, 4.902, 4.131, 3.772, 25, 32, 38.1935295, 28.50635593, 34.811
6396, 18.98618075, 56.75522894, 57.47303544, 40.14656285, 22.99979203, 1.625584232, 23.2
010648, 5.194366741, -1.234239604, 2.49932689, 5.86226574, 5.714750199, 2.317242942, 4.3
83973421, -2.171161818, 2.219347442, 2.27574134, 5.408269117, 5.738828475, 4.233330742,
2.481368989, -0.234959675, -1.126519599, 1.138535071, 1.691647635, 1.915840245, 4.397441
072, 3.211368966, 7.193348119, 15.97890765, 7.00696628, 10.97328486, 10.81415052, 9.7200
54887, 8.651393052, 9.408, 12.493, 10.528, 6.208, 12.448, 7.729, 8.734, 2.699, 2.369, 3.
138, 6.026, 8.991, 5.74, 5.183, 5.142, 6.124, 2.987, 1.041, 2.745, 0.69, 1.923, 0.613, 2
.779, 1.163, 1.493, 0.983, 3.285, 2.036, 3.891, 0.972, 0.994, 0.907, 1.287, 1.881, 0.443
, 3.10291645, 6.173562277, 6.862909067, 1.546396847, -3.399990138, 4.707596881, 4.325727
898, 6.217386806, 2.911602856, -1.028492383, 0.938030761, 4.601492449, 8.140460297, -4.5
5304656, 2.733415482, 11.00320434, 13.41347233, 15.47842436, -0.008248214, 11.67346946,
11.25943742, 38.50391131, 18.33643443, 22.3551061, 15.78133117, 10.06822352, 9.97, 20.55
5, 5.882, 22.222, 40.909, 3.226, 6.25, 11.765, 34.211, 49.02, 7.895, 44.565, 57.143, 57.
416, 72.729, 29.292, 10.673, 7.862, 6.618, 6.938, 18.869, 12.883, 14.033, 15.001, 17.856
, 8.218, 5.413, 11.581, 12.543, 13.72, 10.841, 12.225, 8.495, 8.048, 4.109589041, 9.2043
68175, 2.414285714, -8.236853118, -5.130348864, -6.753725365, -7.251482086, -1.556276054
, -1.590288887, 4.838401224, 15.38672018, 3.999683819, 2.561374173, 1.252408478, 1.23691
7222, 4.87275882, 5.81828209, 9.889250814, 7.001422812, 0.93079949, 35.18691332, -18.073
65899, -3.95519429, 0.758592218, 0, -1.505761844, -1.523581717, 2.323371, -0.758592218,
-1.523581717, -2.328651389, -3.968211061, -5.432640883, 2.39909513, -1.453403872, -0.448
35113, 0.906666667, 5.63777308, -0.32243718, 0.858895706, 3.926122539, 6.02851974, 8.671
651528, 4.964211498, 3.884733832, 0.762291958, 6.060606061, 4.287695661, 6.847285025, 2.
564102564, 6.875866852, 7.653375726, 6.609004882, 0.777385159, 3.8457223, 1.699035682, 1
.888446215, 3.263731394, 3.160578598, 0.785513275, 1.583062206, 2.079449304, 1.325278636
, 1.005222535, 4.141937863, 3.489437844, 3.638360819, 2.151507653, 3.045535464, 3.812391
596, 4.882526907, 6.705448811, 8.431511329, 10.76861601, 13.81223876, 11.25107466, 11.31
771632, 10.21066965, 12.23442663, 13.89287442, 14.236, 15.741, 14.4, 12.587, 11.18, 16.2
01, 18.75, 16.194, 12.892, 14.506, 14.286, 15.566, 13.673, 9.874, 8.824, 8.709, 7.32, 8.
623, 6.872, 5.211, 5.374, 5.7, 9.177, 5.806, 1.392, 3.393, 4.688, 7.09, 11.536, 7.13, 4.
257, 5, 5.654, 5.752, 22.01300157, 24.58203197, 24.99631323, 72.10948561, 37.61310666, 1
3.77833126, 49.61253886, 67.09975712, 69.14172635, 4.156960191, 3.502982107, 15.03591871
, 7.006177993, -0.595293824, 4.192011553, 0, 4.807532957, 5.504844321, 5.216941093, -8.2
64216667, 1.801884462, -1.769991264, 5.78395737, 0.970776621, 5.939573494, 5.549985502,
4.536102074, 2.843700217, 2.498763969, 2.804694188, 1.562639485, 4.253194018, 2.71596692
4, 5.784875818, 4.685641394, 10.05684308, 3.293128476, 8.19487405, 4.63026082, 8.5571198
49, 10.011, 8.903, 13.673, 8.972, 8.596, 7.551, 6.159, 8.224, 7.156, 7.722, 6.502, 7.693
, 5.518, 4.04, 5.423, 6.232, 3.733, 3.599, 3.102, 2.768, 2.77, 1.987, 2.706, 2.724, 3.62
1, 1.444, 4.143, 3.438, 4.913, 3.53, 4.409, 3.541, 5.139, 5.805, 4.924, 2.857142857, 2.7
77777778, 2.702702703, 7.894736842, 2.43902439, 4.761904762, 2.272727273, 4.444444444, 6
.382978723, 4, 3.846153846, 3.703703704, 3.571428571, 3.103448276, 3.177257525, 1.134521
88, 2.083333333, 3.453689168, 0.151745068, 1.363636364, -0.448430493, 3.003003003, 8.163
265306, 10.2425876, 5.012224939, 10.71012806, 2.523659306, 2.564102564, 6.52173913, 4.76
1904762, 6.493506494, 7.926829268, 10.16949153, 18.97435897, 19.82758621, 16.18705036, 9
.59752322, 11.729, 13.997, 12.495, 19.692, 20.019, 37.43, 54.8, 47.028, 54.042, 128.294,
109.558, 97.701, 165.725, 183.263, 54.614, 34.905, 43.095, 24.41, 24.456, 26.79, 26.1, 2
1.357, 22.239, 21.4, 17.969, 18.325, 9.017, 10.655, 12.449, 13.392, 8.5, 8.658, 6.575, 6
.978, 7.811, -17.24137931, -13.88888889, -4.032258065, 0.847457627, 0.840336134, 0, -5,
-7.01754386, -3.773584906, -2.941176471, -1.01010101, 0, -1.020408163, 6.18556701, -2.91
2621359, 0.906344411, 2.994011976, 3.779069767, 5.602240896, 6.100795756, 3, 2.669902913
, 5.200945626, 2.921348315, 9.388646288, 3.79241517, 7.692307692, 6.964285714, 8.1803005
01, 2.777777778, 0, 1.651651652, 4.431314623, 2.97029703, 3.571428571, 2.652519894, 2.45
4780362, 2.90037831, 2.083333333, 1.080432173, 2.500022554, 2.500003432, 3.121953195, 2.
365179967, 1.38631904, 0.364644862, 2.088991186, 3.024884047, 2.84975748, 3.106641376, 6
.596092646, 10.00764002, 11.0416805, 10.25640067, 8.229915112, 12.55279758, 7.2, 13.2, 1
0.4, 23.2, 20.3, 8.3, 14.5, 12.5, 7.4, 12.8, 17.4, 24, 41.6, 28.2, 21.11354311, 25.80873
955, 16.40028408, 20.0650384, 46.61127497, 56.92320336, 55.20369616, 112.1184112, 198.92
8606, 598.7447505, 132.7467739, 585.8443656, 1281.113605, 66279.89237, 21989695.22, -7.6
7, 3.217, 4.92, 3.72, 1.632]
]
# predict the label
label = predict_classification(dataset, row, num_neighbors)


#print('row: %s' %(row))
```

```
print('Predicted: %s' % (label))
```

```
[sovereign_external_debt_default] => 0
[independence] => 1
[banking_crisis] => 2
[domestic_debt_in_default] => 3
[inflation_crises] => 4
[exch_usd] => 5
[gdp_weighted_default] => 6
[systemic_crisis] => 7
[currency_crises] => 8
[inflation_annual_cpi] => 9
Predicted: 0
```

In [ ]: