Main Goals

For part 1: The main goal is to gain some experience with data processing issues as related to similarity assessment

For part 2: The main goal is to gain some experience with clustering of data. Explore algorithmic and search issues in both subject areas.

1. Description of Available Data

Attached to this announcement please find the following files:

1.	PIMA_Diabetes_2015	in ARFF format
2.	Training_Examples_TEXT_File1	in text format
3.	Available_Distances_of_Pairs_of_Records1	in text format

The first file was retrieved from UCI's machine Learning Data Set Repository. It has all the data records available on that site. The second file is a random subset of the first data set (approximately of size $\frac{2}{3}$ of the original size). This data set (i.e. the second file) was used to form a number of random pairs of records (150 out of all possibilities). Some may be duplicates.

It is assumed that a panel of experts has examined each such pair and came up with a value that assesses in their collective opinion how close the two records are of each other. The closer they are, the more similar the corresponding records are assumed to be. For this study, such assessments have been ****simulated**** as follows.

A Euclidean distance was assumed to exist and then it was used on pairs of vectors to come up with these distance values. This Euclidean distance expresses a weighted sum of the squared differences of the attribute values of pairs of records.

This data set is defined on 9 attributes. The last one is the class attribute. For the purpose of this study please ignore the class attribute (which assumes binary value). Thus we have 8 attributes (all of which are numerical and ordinal). These values have been normalized by dividing the values of each attribute by the maximum value for that attribute. The values normalized as described above were next used in the "hidden" weighted Euclidean distance formula.

In [1]:
```
%matplotlib inline
```

In [2]:
```
import re
import numpy as np
from time import time

import matplotlib.pyplot as plt
from scipy.cluster import hierarchy
from sklearn import tree
import pydotplus

from IPython.display import clear_output, Image, display, Math, Latex
```

Read file #1: `PIMA_Diabetes_2015.ARFF`, the Pima Diabetes data set

In [3]:
```
with open('dm/PIMA_Diabetes_2015.ARFF') as pima_file:
    contents = pima_file.readlines()
```

In [4]:
```
attributes = [line.split()[1] for line in contents if line.startswith('@ATTRIB
UTE')]
attributes.pop(-1)

data = [line for line in contents if line and not line.startswith(('%', '@', '
, '\n'))]
```

In [5]:
```
Pima_Diabetes_Data = []
Pima_Diabetes_Target = []

for record in data:
    p = record.split(',')
    Pima_Diabetes_Data.append((
            int(p[0]),
            int(p[1]),
            int(p[2]),
            int(p[3]),
            int(p[4]),
            float(p[5]),
            float(p[6]),
            int(p[7]),
    ))
    Pima_Diabetes_Target.append(int(p[8]))

Pima_Diabetes_Data = np.array(Pima_Diabetes_Data)
Pima_Diabetes_Target = np.array(Pima_Diabetes_Target)
Pima_Feature_Names = [
    'Number of times pregnant',
    'Plasma glucose concentration (glucose tolerance test)',
    'Diastolic blood pressure (mm Hg)',
    'Triceps skin fold thickness (mm)',
    '2-Hour serum insulin (mu U/ml)',
    'Body mass index (weight in kg/(height in m)\^2)',
    'Diabetes pedigree function',
    'Age (years)'
]
Pima_Target_Names = np.array(['positive', 'negative'],
      dtype='|S8')
```

Read file #2: `Training_Examples_TEXT_File1.txt`, the training examples file

In [6]:
```
Training_Examples = []
```

In [7]:
```
with open('dm/Training_Examples_TEXT_File1.txt') as training_file:
    data = training_file.readlines()
```

In [8]:
```
for record in data:
    p = record.split(',')
    Training_Examples.append((
            int(p[0]),
            int(p[1]),
            int(p[2]),
            int(p[3]),
            int(p[4]),
            float(p[5]),
            float(p[6]),
            int(p[7]),
    ))
```

Read file #3: `Available_Distances_of_Pairs_of_Records1.txt`, the distance pairs file

```
In [9]:  with open('dm/Available_Distances_of_Pairs_of_Records1.txt') as
         distances_file:
             contents = distances_file.readlines()
```

```
In [10]: Available_Distances = []
```

```
In [11]: pattern = re.compile(r' Pair No = ([-+]?\d+) First Record = ([-+]?\d+) Second
          Record = ([-+]?\d+) Distance = ([-+]?\d*\.\d+|\d+)\s*')
```

```
In [12]: for line in contents:
             p = pattern.match(line)
             p = p.groups()
             Available_Distances.append((
                     int(p[0]),
                     int(p[1]),
                     int(p[2]),
                     float(p[3]),
             ))
```

```
In [13]: Available_Distances[0]

Out[13]: (1, 153, 362, 0.195109293281901)
```

```
In [14]: Training_Examples[0]

Out[14]: (9, 164, 78, 0, 0, 32.8, 0.148, 45)
```

2. Tasks for Part 1 (On similarities)

Part 1 focuses on determining similarity values. For that you will have to first estimate the form of the Euclidean formula used to derive the values presented in the third file. In other words, you will have to use some type of search to estimate the weight values used in that formula. You may want to use a search method based on **steepest descent** as described in class. Once the "hidden" weighted Euclidean formula has been estimated, use it to determine similarity values between any pair of records from the second file.

In your report for part 1 you will have to present what the Euclidean formula you have derived (along with the weights used) and the values of the estimated distances of pairs described in the input file. Determine the total difference (say, expressed as the sum of all squared differences divided by the number ($= 150$) of pairs)

The general form of a weighted Euclidean distance function is:

$$d_{x_1x_2} = \left(\sum_{i=1}^{n} w_i(x_{i_1} - x_{i_2})^2\right)^{\frac{1}{2}}, \quad 0 \leq w_i \leq 1$$

We know the training data was normalized by dividing its attribute value by the maximum value of that attribute to derive the training data but we do not have that information.

The Euclidean distance measure used must have looked like:

$$d_{x_1x_2} = \left(\sum_{i=1}^{8} w_i(x_{i_1} - x_{i_2})^2\right)^{\frac{1}{2}}$$

Let's normalize the values in Training_Examples

```
In [15]: Training_Examples = np.array(Training_Examples)
```

```
In [16]: Normalized_Examples = np.divide(Training_Examples, Training_Examples.max(axis
         = 0))
```

We randomly choose our starting point. To do so we generate 8 random numbers and normalize the results to get weights that cumulatively yield 1.

```
In [17]: def random_starting_point():
             w = np.random.rand(8)
             w /= w.sum()
             return w
```

```
In [18]: random_starting_point()

Out[18]: array([ 0.04671395,  0.06652313,  0.20963805,  0.08954534,  0.17261056,
                 0.22420806,  0.02467463,  0.16608628])
```

```
In [19]: def weighted_euclidean_distance(x_1, x_2, w):
             res = .0
             for i in range(8):
                 res+= w[i] * (x_1[i] - x_2[i]) ** 2.0
             return np.sqrt(res)
```

Unit test for `weighted_euclidean_distance()`

```
In [20]: Normalized_Examples[8]

Out[20]: array([ 0.11764706,  0.45728643,  0.50819672,  0.        ,  0.        ,
                 0.40685544,  0.21694215,  0.27160494])
```

```
In [21]: data = np.array([
                 [   0.,     119.,       0.,       0.,       0.,      32.4,      0.141,
            24.,   ],
                 [  10.,     111.,      70.,      27.,       0.,      27.5,      0.141,
            40.,   ],
                 [   2.,      92.,      52.,       0.,       0.,      30.1,      0.141,
            22.,   ],
             ])
         weighted_euclidean_distance(

         x_1=np.array([0.52941176,0.8241206,0.63934426,0.,0.,0.48882265,0.06115702,0.55
         555556]),
             x_2=np.array([0.11764706,0.45728643,0.50819672,0.,0.,0.40685544,0.21694215
         27160494]),
             w=np.array([0.04020281,0.08841431,0.08442112,0.06732131,0.01573263,0.51261
         337,0.10934525,0.08194919]),
         )

Out[21]: 0.18130426730743721
```

Let's test our distance function:

```
In [22]: weighted_euclidean_distance(Normalized_Examples[0], Normalized_Examples[1], ra
         ndom_starting_point())

Out[22]: 0.140837699757144
```

Let's see how far we are from values in Available_Distances

```
In [23]: def calc_ms_error(weights, distances):
             ms_error = 0.0
             for record in distances:
                 x_1 = Normalized_Examples[record[1]]
                 x_2 = Normalized_Examples[record[2]]
                 dist = weighted_euclidean_distance(x_1, x_2, weights)
                 ms_error += np.power(dist - record[3], 2)
             return np.divide(ms_error, 150)
```

```
In [24]: calc_ms_error(random_starting_point(), Available_Distances)

Out[24]: 0.009693059896241565
```

To choose the next group of weights we generate 4 random set of $\delta = \langle \delta_1, \delta_2, \ldots, \delta_8 \rangle,\ \sum_{i=1}^{8} \delta_i = 0$ sets of weights and add them to current weights. If after 10,000 iterations no further better candidate is found then the search will be stopped.

```
In [25]: def get_candidates(criteron, current_weights, distances, num_candidates = 4, s
         top_at = 100):
             start_time = time()
             res = []
             iterations_control = 0
             while len(res) <= num_candidates:
                 iterations_control += 1
                 # Generate 8 numbers (one per each attribute) between 0 and 1
                 d = np.random.rand(8)
                 # Normalize the numbers to add up to zero
                 d = d / d.sum() - .125

                 w_candidate = d + current_weights

                 # We want positive weights
                 if np.all(w_candidate > 0):
                     # Ensure the error decreases with these weights
                     e = calc_ms_error(w_candidate, distances)
                     if e < criteron:
                         res += [(w_candidate, e)]
                 if time() - start_time > stop_at:
                     if res:
                         return res
                     raise ValueError
             return res
```

With all these pieces together let's perform steepest ascent search for 100 iterations and try to minimize the minimum squared error at each iteration, and its progress is which is depicted as a plot.

In [26]:
```
def plot_calc(mse, weights):
    def plot_mse(mse):
        plt.plot(mse)
        plt.xlabel('Number of iterations')
        plt.ylabel('MSE')
        plt.title('Minimum Squared Error')
        plt.grid()

    def plot_weights(weights):
        for a, h in weights.iteritems():
            plt.plot(h, label = a)
        plt.xlabel('Number of iterations')
        plt.ylabel('Attribute Weight')
        plt.title('Estimated Eclidean Distance Weights')
        plt.legend(loc='upper right', bbox_to_anchor=(1, 0.5))
        plt.grid()

    plt.figure(figsize = (10, 10))
    plt.suptitle('Statistics for {} iterations'.format(len(mse)), fontsize = 1
4)

    plt.subplot(2, 1, 1)
    plot_mse(mse)

    plt.subplot(2, 1, 2)
    plot_weights(weights)

    plt.tight_layout()
    plt.subplots_adjust(top = 0.85)
    plt.show()
```

Putting it altogether

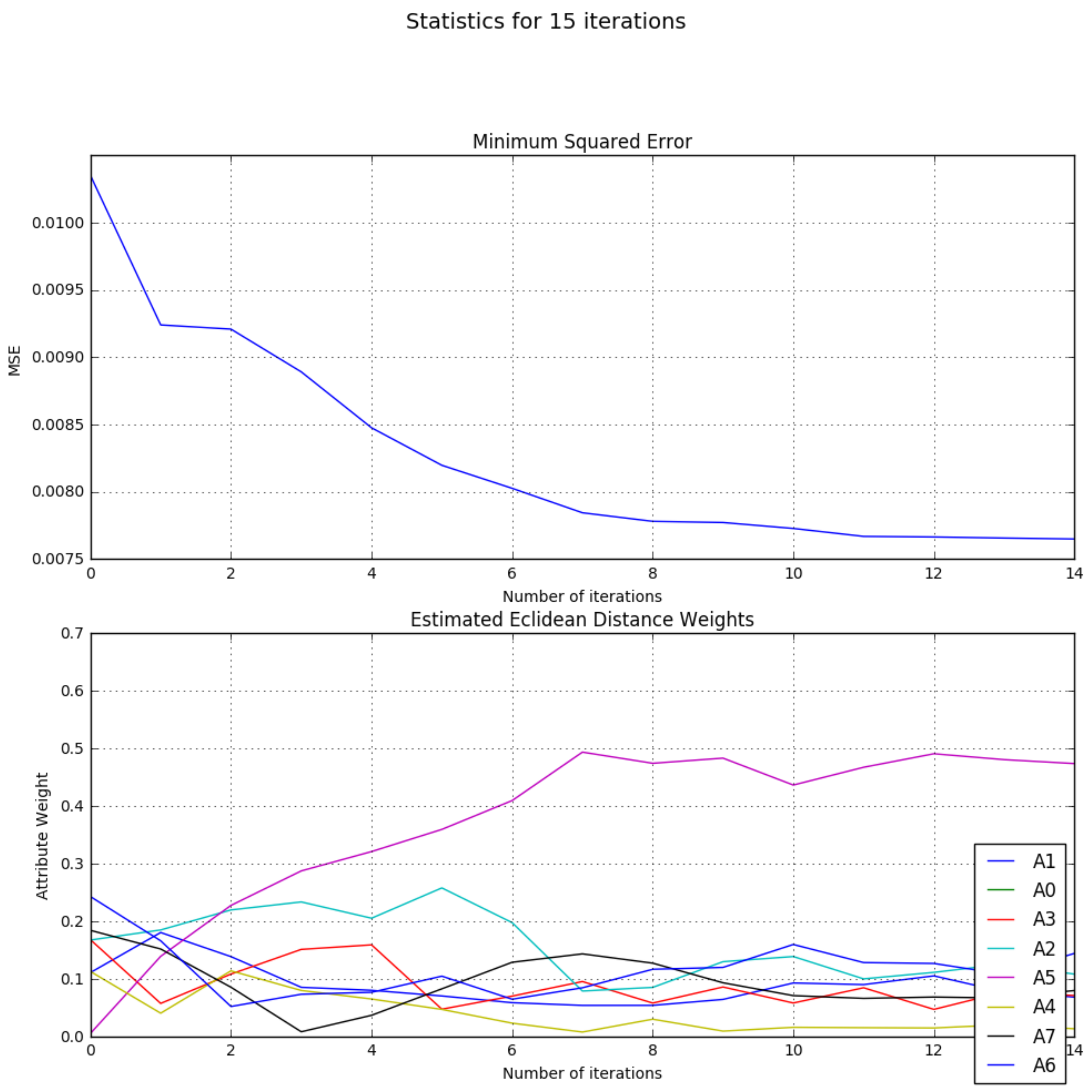
Let's try running the search algorithm for a couple of iterations

In [27]:
```
def GRASP(weights, max_refinements = 40, stop_at = 100, mse = np.infty, mse_hi
story = None, weight_history = None):
    # "max_iterations" iterations at most
    for i in range(max_refinements):
        # Get the list of candidates
        try:
            candidates = get_candidates(mse, weights, Available_Distances, sto
p_at = stop_at)
        except ValueError:
            return weights, i, mse_history, weight_history
        # Randomly choose a candidate
        weights, mse = candidates[np.random.randint(len(candidates))]

        # Add MSE and weights to the history
        if mse_history:
            mse_history += [mse]
        else:
            mse_history = [mse]
        for i, attr in enumerate(attributes):
            if weight_history:
                if weight_history.has_key(attr):
                    weight_history[attr] += [weights[i]]
                else:
                    weight_history[attr] = [weights[i]]
            else:
                weight_history = {
                    attr: weights[i]
                }

        # Plot the updated values
        clear_output(wait = True)
        plot_calc(mse_history, weight_history)
    return weights, i, mse_history, weight_history
```

In [28]:
```
def display_weights(w):
    display(Math(r'''
        d_{{x_1 x_2}} \approx
        \left(
            \begin{{array}}{{ll}}
                & ({0}) x_1 \\
              + & ({1}) x_2 \\
              + & ({2}) x_3 \\
              + & ({3}) x_4 \\
              + & ({4}) x_5 \\
              + & ({5}) x_6 \\
              + & ({6}) x_7 \\
              + & ({7}) x_8 \\
            \end{{array}}
        \right)^{{\tfrac{{1}}{{2}}}}
        '''.format(*w)))
```

In [29]:
```
estimations = []
```

In [30]:
```
max_refinements = 40
estimate, num_refinements, e_history, w_history = GRASP(
    random_starting_point(),
    max_refinements = max_refinements,
    stop_at = 60
)

if num_refinements < max_refinements:
    print 'GRASP terminated after {} iterations.'.format(num_refinements)

display_weights(estimate)
estimations += [estimate]
```



```
GRASP terminated after 15 iterations.
```

$$d_{x_1 x_2} \approx \left(\begin{array}{ll} & (0.0398861877803)x_1 \\ + & (0.0686722749895)x_2 \\ + & (0.108411529045)x_3 \\ + & (0.0713242995259)x_4 \\ + & (0.0136763273414)x_5 \\ + & (0.473512493829)x_6 \\ + & (0.144425564348)x_7 \\ + & (0.0800913231401)x_8 \end{array} \right)^{\frac{1}{2}}$$

```
In [31]: for w in estimations:
             display_weights(w)
```

$$d_{x_1x_2} \approx \begin{pmatrix} & (0.0398861877803)x_1 \\ + & (0.0686722749895)x_2 \\ + & (0.108411529045)x_3 \\ + & (0.0713242995259)x_4 \\ + & (0.0136763273414)x_5 \\ + & (0.473512493829)x_6 \\ + & (0.144425564348)x_7 \\ + & (0.0800913231401)x_8 \end{pmatrix}^{\frac{1}{2}}$$

3. Tasks for Part 2 (On Clustering)

Part 2 focuses on clustering. Thus, once you have estimated the "hidden" weighted Euclidean formula used in Part 1 to determine the distance values, use it to cluster the data described in the second file. The number of clusters of the proposed clustering scheme is up to you to decide.

In your report for Part 2 you will have to describe what objective function you tried to optimize in your clustering scheme. You may have to present and analyze a number of alternative clustering schemes and then describe and explain which one is the best one. For clustering data you can use any one of the methods we used in class.

Partition Purity Measures

If $p(i|t)$ denotes the fraction of items belonging to class i at node t and c denotes the number of classes then we can use the following measures of to observe purity of partitions:

$$\begin{aligned} \text{Entropy(t)} &= -\sum_{i=0}^{c-1} p(i|t)\log_2 p(i|t) \\ \text{Gini Index} &= 1 - \sum_{i=0}^{c-1} [p(i|t)]^2 \\ \text{Classification Error} &= 1 - \max_i p(i|t) \end{aligned}$$

Entropy can be implemented as follows:

In [32]:
```
def entropy(data, target_attr):
    data_entropy = 0.

    # Frequency of each of value in target_attr
    val_freq = np.array(np.unique(data[:, target_attr], return_counts=True)
[1])

    # Entropy of data for target_attr
    p_it = val_freq.astype(float) / data.shape[0]
    data_entropy = np.nansum(-p_it * np.log2(p_it))

    return data_entropy
```

To Determine the quality of test conditions based on one of the impurity measures at child node v_j, denoted as $I(\cdot)$, we would like to maximize Δ_{info}, information gain, which is defined as:

$$\Delta = I(\text{parent}) - \sum_{j=1}^{k} \frac{N(v_j)}{N} I(v_j)$$

Where k is the number of attributes and $N(\cdot)$ represents the number of records associated with a node.

In [33]:
```
def gain(data, attr, target_attr):
    subset_entropy = 0.

    # Frequency of each of value in target_attr
    vals, val_freq = np.unique(data[:, attr], return_counts=True)

    # Calculate the sum of the entropy for each subset of records weighted
    # by their probability of occuring in the training set.
    sum_records = float(sum(val_freq))
    for i in range(len(vals)):
        val, freq = vals[i], val_freq[i]
        val_prob = freq / sum_records
        data_subset = data[data[:, attr] == val]
        i_ent = entropy(data_subset, target_attr)
        subset_entropy += val_prob * i_ent

    # Subtract the entropy of the chosen attribute from the entropy of the
    # whole data set with respect to the target attribute (and return it)
    return (entropy(data, target_attr) - subset_entropy)
```

Unit test for gain

```
In [34]: data = np.array([
                 [   0.,     119.,       0.,       0.,       0.,      32.4,      0.141,
            24.,   ],
                 [  10.,     111.,      70.,      27.,       0.,      27.5,      0.141,
            40.,   ],
                 [   2.,      92.,      52.,       0.,       0.,      30.1,      0.141,
            22.,   ],
              ])
         gain(data, attr=2, target_attr=7)

Out[34]: 1.5849625007211561
```

However, instead of using gain, Δ, that is typically used in decision trees, based on the idea proposed by [Liu et all] (http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=5497736), I decided to use our weighted Euclidean distance function and use *average weighted Euclidean distance* for purity measurement.

Assuming D is the dataser A is an attribute, i is an iterator for each record in D, j is an iterator for each class:

$$\mathrm{AED}(A) = \frac{\sqrt{\sum_{i=1}^{n} (\, x_{i1} - x_{i2}\,)^2}}{n}$$

Reference:

Liu, Quan, Daojing Hu, and Qicui Yan. "Decision tree algorithm based on average Euclidean distance." *Future Computer and Communication (ICFCC), 2010 2nd International Conference on. Vol. 1. IEEE, 2010.*

```
In [35]: def ae_distance(data, attr, target_attr):
             def _ae_distance(data, attr, w):
                 ws = data.sum(axis=0)
                 t = (w * data ** 2.0)
                 tt = t[:, attr] / ws[attr]
                 return tt.sum()

             averaging_weights = estimations[0]

             subset_distance = _ae_distance(data, attr, averaging_weights)
             target_distance = _ae_distance(data, target_attr, averaging_weights)

             return (subset_distance - target_distance)
```

Unit test for ae_distance

```
In [36]: data = np.array([
                 [   0.,     119.,       0.,       0.,       0.,      32.4,      0.141,
            24.,   ],
                 [  10.,     111.,      70.,      27.,       0.,      27.5,      0.141,
            40.,   ],
                 [   2.,      92.,      52.,       0.,       0.,      30.1,      0.141,
            22.,   ],
              ])
         ae_distance(data, attr=2, target_attr=7)

Out[36]: 4.2798163138276522
```

```
# Example weights:
w =
array([ 0.03884355,  0.09437358,  0.0874599 ,  0.05884017,  0.0067347 ,
        0.50965664,  0.13201275,  0.07207871])
```

We choose the next attribute, which is the attribute with the highest information gain (lowest entropy).

```
In [37]: def choose_attribute(data, attributes, target_attr, fit_fx):
             best_gain = -np.inf
             best_attr = None

             for attr in attributes:
                 gain = fit_fx(data, attr, target_attr)

                 if (gain >= best_gain and attr != target_attr):
                     best_gain = gain
                     best_attr = attr

             if best_attr is None:
                 print 'data', data.shape, 'attributes', repr(attributes), 'target_att
         r', repr(target_attr)
                 print 'data\n', repr(data)
                 raise ValueError
             return best_attr
```

Unit test for `choose_attribute()`

```
In [38]: #data (2, 8)
         choose_attribute(
             data=np.array([
                 [  13.   ,  106.   ,   70.   ,    0.   ,    0.   ,   34.2  ,    0.251,
            52.   ],
                 [   1.   ,  106.   ,   70.   ,   28.   ,  135.   ,   34.2  ,    0.142,
            22.   ]
             ]),
             attributes=[0, 3, 4, 6, 7],
             target_attr=7,
             fit_fx=ae_distance,
         )

Out[38]: 3
```

Decision Tree Implementation

The decision tree algorithm can be implemented as follows:

In [39]:
```
def make_decision_tree(data, attrs, target, fit_fx):
    # If dataset's empty or no attributes are left
    if np.all(data == 0) or len(attrs) <= 1:
        return vals[0]

    data = data.copy()
    vals = data[:, target]

    unique_vals, val_freq = np.unique(vals, return_counts = True)

    # If all data belongs to the same attribute
    if np.sum(vals == vals[0]) == vals.shape[0]:
        default = unique_vals[np.argmax(val_freq)]
        return default


    # Choose the next best attribute to best classify our data
    best = choose_attribute(
        data, attrs, target, fit_fx
    )

    tree = { best: {} }

    other_attrs = attrs[:]

    other_attrs.remove(best)

    possible_values = np.unique(data[:, best])

    np.delete(data, best, axis = 1)
    # For each of the values in best attribute field
    for val in possible_values:
        subtree = make_decision_tree(
            data[data[:, best] == val],
            other_attrs,
            target,
            fit_fx)

        # Add the new subtree to the empty dictionary object in our new tree/n
ode we just created.
        tree[best][val] = subtree
    return tree
```

In [40]:
```
def print_tree(tree, str=''):
    """
    This function recursively crawls through the d-tree and prints it out in a
    more readable format than a straight print of the Python dict object.
    """
    if type(tree) == dict:
        print "%s%s" % (str, tree.keys()[0])
        for item in tree.values()[0].keys():
            print "%s\t%s" % (str, item)
            print_tree(tree.values()[0][item], str + "\t")
    else:
        print "%s\t->\t%s" % (str, tree)
```

In [41]:
```
Pima_Tree_Gain = make_decision_tree(Training_Examples, list(range(len(attribut
es))), len(attributes) - 1, gain)
```

In [42]:
```
Pima_Tree_AE = make_decision_tree(Training_Examples,
list(range(len(attributes))), len(attributes) - 1, ae_distance)
```

```
/home/pamini/.local/lib/python2.7/site-packages/ipykernel/__main__.py:5: Runt
imeWarning: invalid value encountered in divide
```

```
In [43]: print_tree(Pima_Tree_AE)
```

```
5
        0.0
        2
                0.0
                1
                        114.0
                                ->      26.0
                        115.0
                                ->      30.0
                        84.0
                                ->      21.0
                        94.0
                                ->      25.0
                64.0
                        ->      21.0
                82.0
                        ->      69.0
        19.5
        2
                74.0
                        ->      28.0
                62.0
                        ->      25.0
        46.2
                ->      46.0
        22.5
        2
                56.0
                        ->      22.0
                96.0
                        ->      21.0
        19.4
                ->      22.0
        42.4
                ->      29.0
        45.8
                ->      31.0
        32.4
        1
                128.0
                        ->      27.0
                132.0
                        ->      21.0
                133.0
                        ->      37.0
                108.0
                        ->      42.0
                141.0
                        ->      22.0
                110.0
                        ->      27.0
                142.0
                        ->      63.0
                119.0
                        ->      24.0
                154.0
                        ->      45.0
```

```
          91.0
                    ->      27.0
22.4
2
          122.0
                    ->      27.0
          44.0
                    ->      22.0
21.0
1
          101.0
                    ->      21.0
          150.0
                    ->      37.0
34.1
1
          112.0
                    ->      26.0
          123.0
                    ->      28.0
          181.0
                    ->      38.0
23.0
          ->      40.0
24.0
1
          111.0
                    ->      23.0
          108.0
                    ->      35.0
          103.0
                    ->      33.0
25.0
2
          78.0
                    ->      64.0
          60.0
                    ->      27.0
          70.0
                    ->      25.0
          62.0
                    ->      36.0
26.0
1
          136.0
                    ->      51.0
          110.0
                    ->      30.0
27.0
          ->      36.0
28.0
1
          129.0
                    ->      27.0
          146.0
                    ->      29.0
          132.0
```

```
                ->      63.0
29.0
1
        106.0
                ->      22.0
        119.0
                ->      29.0
30.0
1
        80.0
                ->      22.0
        184.0
                ->      49.0
        100.0
                ->      46.0
        125.0
                ->      32.0
31.0
2
        80.0
                ->      21.0
        50.0
                ->      26.0
32.0
1
        0.0
                ->      22.0
        65.0
                ->      42.0
        100.0
                ->      42.0
        137.0
                ->      39.0
        128.0
                ->      33.0
        79.0
                ->      22.0
        144.0
                ->      58.0
        188.0
                ->      22.0
21.8
1
        99.0
                ->      26.0
        87.0
                ->      21.0
        124.0
                ->      21.0
        101.0
                ->      22.0
        71.0
                ->      26.0
34.0
1
        180.0
                ->      26.0
```

```
          124.0
                    ->       38.0
          99.0
                    ->       30.0
          76.0
                    ->       25.0
          155.0
                    ->       46.0
35.0
1
          136.0
                    ->       35.0
          168.0
                    ->       52.0
          122.0
                    ->       29.0
          95.0
                    ->       43.0
36.0
          ->        25.0
29.5
1
          144.0
                    ->       37.0
          113.0
                    ->       25.0
          147.0
                    ->       50.0
39.0
1
          0.0
                    ->       41.0
          121.0
                    ->       28.0
          163.0
                    ->       33.0
          127.0
                    ->       51.0
40.0
1
          128.0
                    ->       24.0
          181.0
                    ->       22.0
18.2
          ->        27.0
42.0
          ->        25.0
27.9
2
          84.0
                    ->       62.0
          76.0
                    ->       31.0
44.0
          ->        43.0
45.0
```

```
        ->      47.0
43.4
2
        90.0
                ->      42.0
        100.0
                ->      35.0
30.8
1
        105.0
                ->      37.0
        164.0
                ->      32.0
        100.0
                ->      21.0
        102.0
                ->      36.0
        108.0
                ->      21.0
45.5
        ->      23.0
35.1
1
        129.0
                ->      23.0
        122.0
                ->      30.0
        179.0
                ->      37.0
55.0
        ->      26.0
34.5
1
        186.0
                ->      37.0
        117.0
                ->      40.0
        102.0
                ->      24.0
        127.0
                ->      28.0
38.2
        ->      29.0
31.9
2
        66.0
                ->      29.0
        68.0
                ->      25.0
22.3
        ->      24.0
28.4
2
        90.0
                ->      29.0
        86.0
                ->      22.0
        78.0
                ->      34.0
27.6
2
        72.0
        1
                124.0
                        ->      29.0
                103.0
                        ->      27.0
        74.0
                ->      40.0
        78.0
        1
                88.0
                        ->      37.0
                122.0
                        ->      45.0
        62.0
                ->      44.0
33.6
1
        96.0
                ->      43.0
        165.0
                ->      49.0
        104.0
                ->      22.0
        175.0
                ->      38.0
        113.0
                ->      21.0
        148.0
                ->      50.0
45.3
2
        88.0
                ->      26.0
        76.0
        1
                107.0
                        ->      24.0
                67.0
                        ->      46.0
36.1
1
        138.0
                ->      50.0
        116.0
                ->      25.0
28.7
1
        139.0
                ->      22.0
        93.0
                ->      23.0
        126.0
```

```
                ->      21.0
        117.0
                ->      30.0
35.5
1
        184.0
                ->      41.0
        158.0
                ->      35.0
49.6
        ->      26.0
48.3
        ->      32.0
24.1
        ->      23.0
32.5
2
        80.0
                ->      26.0
        82.0
                ->      70.0
        52.0
                ->      22.0
        70.0
                ->      39.0
33.2
1
        137.0
                ->      22.0
        117.0
                ->      24.0
26.6
1
        81.0
                ->      24.0
        155.0
                ->      27.0
        85.0
                ->      31.0
21.1
2
        0.0
                ->      25.0
        68.0
                ->      26.0
26.3
        ->      24.0
28.9
1
        114.0
                ->      24.0
        115.0
                ->      46.0
        146.0
                ->      27.0
        134.0
                ->      23.0
```

```
45.4
        ->      54.0
48.8
        ->      37.0
33.7
1
        176.0
                ->      58.0
        105.0
                ->      29.0
        115.0
                ->      40.0
        78.0
                ->      25.0
37.1
1
        136.0
                ->      43.0
        111.0
                ->      56.0
36.5
1
        196.0
                ->      29.0
        180.0
                ->      35.0
        95.0
                ->      26.0
29.6
2
        74.0
                ->      31.0
        90.0
                ->      46.0
        78.0
                ->      40.0
33.3
1
        113.0
                ->      23.0
        122.0
                ->      33.0
        75.0
                ->      38.0
        171.0
                ->      24.0
        118.0
                ->      23.0
23.3
1
        105.0
                ->      53.0
        183.0
                ->      32.0
44.6
        ->      22.0
34.2
```

```
1
        80.0
                ->      27.0
        112.0
                ->      36.0
        106.0
        2
                70.0
                3
                        0.0
                                ->      52.0
                        28.0
                                ->      22.0
        179.0
                ->      60.0
        130.0
                ->      45.0
37.6
        ->      51.0
33.8
1
        117.0
                ->      44.0
        125.0
        2
                76.0
                        ->      54.0
                60.0
                        ->      31.0
34.7
1
        98.0
                ->      22.0
        197.0
                ->      62.0
        150.0
                ->      42.0
27.4
1
        124.0
                ->      36.0
        114.0
                ->      34.0
        116.0
                ->      21.0
        126.0
                ->      21.0
        159.0
                ->      40.0
38.1
1
        97.0
                ->      30.0
        114.0
                ->      21.0
        115.0
                ->      28.0
```

```
22.1
        ->      21.0
34.3
1
        128.0
                ->      24.0
        121.0
                ->      33.0
        83.0
                ->      25.0
        189.0
                ->      41.0
23.8
        ->      60.0
35.2
        ->      29.0
23.5
        ->      25.0
34.6
2
        80.0
                ->      45.0
        78.0
                ->      22.0
        70.0
                ->      32.0
33.5
        ->      42.0
29.9
1
        88.0
                ->      23.0
        104.0
                ->      41.0
        136.0
                ->      50.0
        147.0
                ->      28.0
        169.0
                ->      31.0
46.7
        ->      42.0
34.8
2
        64.0
                ->      26.0
        80.0
                ->      24.0
35.7
1
        112.0
                ->      21.0
        101.0
                ->      26.0
        173.0
                ->      22.0
32.8
```

```
1
        164.0
        2
                82.0
                        ->      50.0
                78.0
                        ->      45.0
        114.0
                ->      42.0
        76.0
                ->      41.0
        133.0
                ->      45.0
        154.0
                ->      23.0
39.1
2
        86.0
                ->      42.0
        70.0
                ->      23.0
38.5
1
        144.0
                ->      37.0
        129.0
                ->      41.0
        146.0
                ->      67.0
        100.0
                ->      26.0
47.9
        ->      26.0
30.5
1
        128.0
                ->      25.0
        160.0
                ->      39.0
        106.0
                ->      34.0
        108.0
                ->      33.0
        182.0
                ->      29.0
        120.0
                ->      26.0
35.3
2
        0.0
                ->      29.0
        60.0
                ->      25.0
        70.0
                ->      39.0
36.2
        ->      28.0
```

```
27.7
1
        0.0
                ->      21.0
        81.0
                ->      25.0
        162.0
                ->      54.0
        127.0
                ->      25.0
30.4
1
        89.0
                ->      38.0
        84.0
                ->      21.0
        93.0
                ->      23.0
        142.0
                ->      43.0
        165.0
                ->      49.0
23.2
2
        80.0
                ->      32.0
        76.0
                ->      21.0
28.6
1
        130.0
                ->      21.0
        139.0
                ->      26.0
35.8
        ->      35.0
49.7
        ->      31.0
36.7
        ->      31.0
39.5
2
        74.0
                ->      42.0
        90.0
                ->      25.0
32.9
1
        132.0
                ->      23.0
        101.0
                ->      63.0
        102.0
                ->      46.0
        174.0
                ->      36.0
        124.0
```

```
                ->      30.0
        133.0
                ->      39.0
23.1
1
        118.0
                ->      46.0
        109.0
                ->      26.0
        94.0
                ->      56.0
36.3
1
        138.0
                ->      25.0
        123.0
                ->      52.0
        127.0
                ->      23.0
24.8
1
        88.0
                ->      22.0
        156.0
                ->      53.0
37.2
1
        162.0
                ->      24.0
        84.0
                ->      28.0
38.0
        ->      34.0
40.6
1
        153.0
        2
                88.0
                        ->      39.0
                82.0
                        ->      23.0
        102.0
                ->      21.0
        135.0
                ->      26.0
40.7
        ->      21.0
21.2
        ->      23.0
30.9
1
        106.0
                ->      24.0
        195.0
                ->      31.0
        191.0
                ->      34.0
```

```
29.8
1
        122.0
                ->      22.0
        91.0
                ->      31.0
        158.0
                ->      63.0
27.3
2
        68.0
                ->      32.0
        62.0
                ->      22.0
36.6
2
        94.0
                ->      51.0
        60.0
                ->      21.0
        78.0
                ->      46.0
        62.0
                ->      25.0
37.7
1
        90.0
                ->      29.0
        187.0
                ->      41.0
        162.0
                ->      52.0
        95.0
                ->      27.0
42.9
        ->      22.0
29.2
        ->      21.0
36.8
1
        97.0
                ->      25.0
        122.0
                ->      27.0
        107.0
                ->      31.0
        173.0
                ->      38.0
        95.0
                ->      57.0
33.9
1
        144.0
                ->      40.0
        187.0
                ->      34.0
23.6
```

```
2
        72.0
                ->      58.0
        66.0
                ->      26.0
37.3
        ->      40.0
26.2
1
        80.0
                ->      41.0
        121.0
                ->      30.0
        143.0
                ->      21.0
        111.0
                ->      23.0
32.7
        ->      36.0
29.3
2
        68.0
                ->      42.0
        86.0
                ->      34.0
        78.0
                ->      36.0
26.1
1
        194.0
                ->      67.0
        151.0
                ->      22.0
29.7
1
        74.0
                ->      23.0
        75.0
                ->      33.0
        84.0
                ->      46.0
        117.0
                ->      30.0
        146.0
                ->      29.0
28.3
        ->      29.0
34.4
2
        72.0
                ->      25.0
        80.0
                ->      44.0
        82.0
                ->      46.0
37.8
1
```

```
        112.0
                ->      41.0
        100.0
                ->      24.0
42.1
        ->      26.0
41.5
1
        152.0
                ->      27.0
        105.0
                ->      22.0
34.9
1
        105.0
                ->      25.0
        90.0
                ->      56.0
        143.0
                ->      41.0
        147.0
                ->      30.0
        119.0
                ->      23.0
20.4
        ->      27.0
25.8
1
        106.0
                ->      27.0
        166.0
                ->      51.0
39.2
        ->      65.0
25.5
        ->      37.0
32.2
        ->      22.0
23.9
        ->      27.0
35.4
1
        99.0
                ->      50.0
        124.0
                ->      34.0
        134.0
                ->      29.0
38.8
        ->      22.0
24.3
1
        105.0
                ->      21.0
        125.0
                ->      25.0
        103.0
```

```
                                ->      29.0
39.7
                ->      29.0
43.1
                ->      33.0
30.1
1
                92.0
                        ->      22.0
                85.0
                        ->      35.0
                181.0
                        ->      60.0
35.9
1
                194.0
                        ->      41.0
                122.0
                        ->      26.0
                109.0
                        ->      43.0
                181.0
                        ->      51.0
24.6
2
                80.0
                        ->      34.0
                65.0
                        ->      31.0
                52.0
                        ->      21.0
18.4
                ->      27.0
40.2
                ->      37.0
25.3
                ->      28.0
43.6
1
                171.0
                        ->      26.0
                187.0
                        ->      53.0
30.7
                ->      24.0
24.4
2
                80.0
                        ->      24.0
                55.0
                        ->      42.0
36.4
                ->      36.0
39.8
1
                80.0
                        ->      28.0
```

```
        196.0
                ->      41.0
40.5
        ->      44.0
31.6
1
        163.0
                ->      28.0
        139.0
                ->      25.0
        112.0
                ->      25.0
        131.0
                ->      32.0
        94.0
                ->      23.0
        158.0
        2
                90.0
                        ->      66.0
                76.0
                        ->      28.0
59.4
        ->      25.0
44.1
        ->      23.0
43.5
2
        60.0
                ->      23.0
        70.0
                ->      21.0
36.9
2
        64.0
                ->      28.0
        72.0
                ->      28.0
        88.0
                ->      21.0
25.1
1
        90.0
                ->      25.0
        102.0
                ->      21.0
26.8
1
        152.0
                ->      43.0
        73.0
                ->      27.0
        132.0
                ->      69.0
41.2
        ->      38.0
26.5
```

```
1
        121.0
                ->      62.0
        106.0
                ->      29.0
31.2
1
        129.0
                ->      29.0
        104.0
                ->      38.0
        137.0
                ->      30.0
        106.0
                ->      42.0
        136.0
                ->      22.0
        115.0
                ->      44.0
        89.0
                ->      23.0
        122.0
                ->      41.0
        189.0
                ->      29.0
        158.0
                ->      24.0
24.9
        ->      57.0
37.4
2
        74.0
                ->      24.0
        85.0
        1
                140.0
                        ->      41.0
                95.0
                        ->      24.0
28.2
2
        82.0
                ->      50.0
        60.0
                ->      22.0
44.5
        ->      24.0
46.8
        ->      45.0
37.9
1
        145.0
                ->      40.0
        146.0
                ->      28.0
25.6
1
```

```
          99.0
                    ->        28.0
          139.0
          2
                    75.0
                              ->        29.0
                    54.0
                              ->        22.0
          157.0
                    ->        24.0
41.3
1
          144.0
                    ->        28.0
          86.0
                    ->        29.0
19.3
          ->        30.0
24.2
1
          56.0
                    ->        22.0
          137.0
                    ->        55.0
          92.0
                    ->        28.0
          101.0
                    ->        26.0
45.6
          ->        38.0
23.7
1
          96.0
                    ->        28.0
          131.0
                    ->        21.0
38.4
1
          112.0
                    ->        28.0
          173.0
                    ->        25.0
41.8
          ->        38.0
42.7
          ->        21.0
25.4
2
          80.0
                    ->        22.0
          60.0
                    ->        21.0
23.4
          ->        33.0
19.1
          ->        21.0
20.8
```

```
1
        96.0
                ->      26.0
        120.0
                ->      48.0
42.3
2
        80.0
                ->      48.0
        114.0
                ->      44.0
        68.0
                ->      24.0
27.8
2
        64.0
                ->      23.0
        74.0
                ->      30.0
        76.0
                ->      58.0
        88.0
                ->      66.0
        52.0
                ->      22.0
        24.0
                ->      21.0
        58.0
                ->      28.0
32.1
        ->      24.0
27.5
1
        128.0
                ->      22.0
        81.0
                ->      22.0
        146.0
                ->      28.0
        129.0
                ->      31.0
        111.0
                ->      40.0
67.1
        ->      26.0
52.3
        ->      23.0
25.9
1
        130.0
                ->      22.0
        92.0
                ->      31.0
        197.0
                ->      39.0
        134.0
                ->      81.0
```

```
39.4
1
        106.0
                ->      22.0
        147.0
                ->      43.0
        100.0
                ->      43.0
        157.0
                ->      30.0
        158.0
                ->      29.0
42.8
        ->      24.0
39.9
1
        129.0
                ->      44.0
        178.0
                ->      41.0
        91.0
                ->      25.0
43.3
1
        128.0
                ->      31.0
        105.0
                ->      45.0
        180.0
                ->      41.0
        103.0
                ->      33.0
24.5
        ->      36.0
24.7
1
        0.0
                ->      22.0
        96.0
                ->      39.0
        125.0
                ->      21.0
        94.0
                ->      21.0
26.4
1
        129.0
                ->      28.0
        107.0
                ->      23.0
        134.0
                ->      21.0
42.6
        ->      24.0
42.2
        ->      29.0
21.9
```

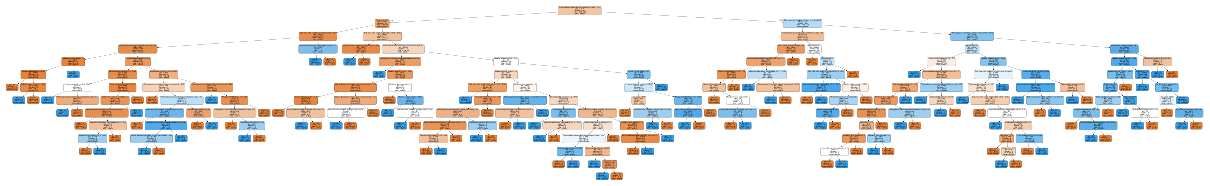
```
        ->      25.0
40.9
        ->      47.0
27.1
1
        139.0
                ->      57.0
        119.0
                ->      33.0
57.3
        ->      22.0
30.3
        ->      53.0
45.2
        ->      24.0
28.5
1
        82.0
                ->      25.0
        109.0
                ->      22.0
25.2
2
        0.0
                ->      37.0
        82.0
                ->      22.0
        54.0
                ->      23.0
        56.0
                ->      23.0
        90.0
                ->      21.0
        62.0
                ->      21.0
38.3
        ->      39.0
33.1
1
        91.0
                ->      22.0
        131.0
                ->      28.0
32.3
1
        116.0
                ->      35.0
        167.0
                ->      30.0
19.6
2
        66.0
                ->      25.0
        90.0
                ->      60.0
```

In [44]:
```
clf = tree.DecisionTreeClassifier()
cf = clf.fit(Pima_Diabetes_Data, Pima_Diabetes_Target)
```

In [55]:
```
dot_data = tree.export_graphviz(
    clf,
    out_file=None,
    feature_names=Pima_Feature_Names,
    class_names=Pima_Target_Names,
    filled=True,
    rounded=True,
    special_characters=True
)
graph = pydotplus.graph_from_dot_data(dot_data)
```

In [56]:
```
#Image(graph.create_svg())
Image(graph.create_png())
```

Out[56]:



Note

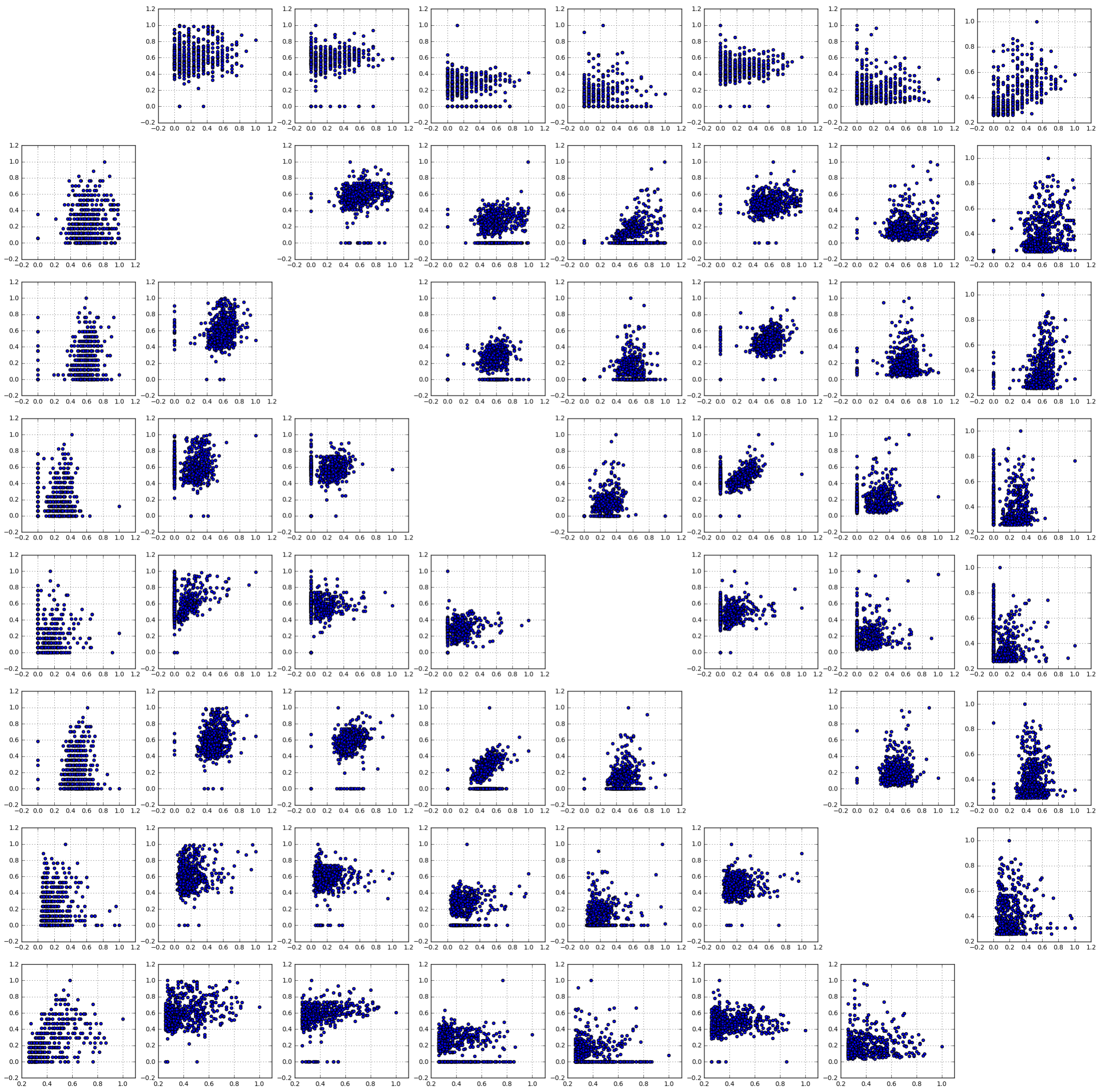
I dropped the K-means solution, since I could not come up with a meaningful interpretations of the results. Futher progress and refactoring in this project may have broken the K-means clustering code.

In [46]:
```
# Calculates euclidean distance between
# a data point and all the available cluster
# centroids.
def euclidean_dist(data, centroids, clusters):
    for instance in data:
        # Find which centroid is the closest
        # to the given data point.
        mu_index = min(
            [
                (
                    i[0],
                    np.linalg.norm(instance - centroids[i[0]])
                ) for i in enumerate(centroids)
            ],
            key = lambda t:t[1]
        )[0]
        try:
            clusters[mu_index] += [instance]
        except KeyError:
            clusters[mu_index] = [instance]

    # If any cluster is empty then assign one point
    # from data set randomly so as to not have empty
    # clusters and 0 means.
    for cluster in clusters:
        if not cluster:
            cluster.append(
                data[
                    np.random.randint(0, len(data), size = 1)
                ].flatten().tolist()
            )
    return clusters
```

In [47]:
```
# randomize initial centroids
def randomize_centroids(data, centroids, k):
    for cluster in range(0, k):
        centroids.append(data[np.random.randint(0, len(data), size=1)].flatten
tolist())
    return centroids
```

In [48]:
```
# check if clusters have converged
def has_converged(centroids, old_centroids, iterations):
    MAX_ITERATIONS = 1000
    if iterations > MAX_ITERATIONS:
        return True
    return old_centroids == centroids
```

```
In [49]: plt.figure(figsize = (30, 30))
         for i in range(8):
             for j in range(8):
                 if i != j:
                     plot_id = i * 8 + j + 1
                     plt.subplot(8, 8, plot_id)
                     plt.scatter(Normalized_Examples[:, i], Normalized_Examples[:, j])
                     plt.grid()

         plt.show()
```



```
In [50]: # kmeans clustering algorithm
         # data = set of data points
         # k = number of clusters
         def k_means(data, k):
             centroids = []

             centroids = randomize_centroids(data, centroids, k)

             old_centroids = [[] for i in range(k)]

             iterations = 0
             while not (has_converged(centroids, old_centroids, iterations)):
                 iterations += 1

                 clusters = [[] for i in range(k)]

                 # Assign data points to clusters
                 clusters = euclidean_dist(data, centroids, clusters)

                 # Recalculate centroids
                 index = 0
                 for cluster in clusters:
                     old_centroids[index] = centroids[index]
                     centroids[index] = np.mean(cluster, axis=0).tolist()
                     index += 1


             print("The total number of data instances is: " + str(len(data)))
             print("The total number of iterations necessary is: " + str(iterations))
             print("The means of each cluster are: " + str(centroids))
             print("The clusters are as follows:")
             for cluster in clusters:
                 print("Cluster with a size of " + str(len(cluster)) + " starts here:")
                 print(np.array(cluster).tolist())
                 print("Cluster ends here.")

             return np.array(cluster).tolist()
```

```
In [51]: a = k_means(Training_Examples, 5)
```

```
The total number of data instances is: 514
The total number of iterations necessary is: 24
The means of each cluster are: [[3.9887005649717513, 95.46327683615819, 64.57
062146892656, 14.338983050847459, 9.305084745762711, 29.47683615819209З, 0.40
992090395480246, 31.35593220338983], [2.962406015037594, 117.05263157894737,
 69.83458646616542, 28.18045112781955, 105.66165413533835, 32.38721804511278,
 0.4849473684210527, 29.24812030075188], [4.235294117647059, 156.764705882352
93, 72.23529411764706, 33.35294117647059, 465.05882352941177, 35.964705882352
93, 0.6399411764705885, 32.588235294117645], [3.5921052631578947, 138.0789473
6842104, 73.21052631578948, 30.026315789473685, 217.46052631578948, 34.596052
631578964, 0.586342105263158, 33.48684210526316], [4.837837837837838, 149.360
36036036037, 74.57657657657657, 11.36936936936937, 0.5855855855855856, 33.310
810810810814, 0.473801801801802, 40.0990990990991]]
The clusters are as follows:
Cluster with a size of 177 starts here:
[[1.0, 79.0, 60.0, 42.0, 48.0, 43.5, 0.678, 23.0], [1.0, 71.0, 62.0, 0.0, 0.
0, 21.8, 0.416, 26.0], [2.0, 91.0, 62.0, 0.0, 0.0, 27.3, 0.525, 22.0], [9.0,
 119.0, 80.0, 35.0, 0.0, 29.0, 0.263, 29.0], [6.0, 87.0, 80.0, 0.0, 0.0, 23.
2, 0.084, 32.0], [4.0, 76.0, 62.0, 0.0, 0.0, 34.0, 0.391, 25.0], [8.0, 95.0,
 72.0, 0.0, 0.0, 36.8, 0.485, 57.0], [2.0, 111.0, 60.0, 0.0, 0.0, 26.2, 0.34
3, 23.0], [9.0, 112.0, 82.0, 24.0, 0.0, 28.2, 1.282, 50.0], [6.0, 80.0, 66.0,
 30.0, 0.0, 26.2, 0.313, 41.0], [10.0, 90.0, 85.0, 32.0, 0.0, 34.9, 0.825, 5
6.0], [1.0, 79.0, 80.0, 25.0, 37.0, 25.4, 0.583, 22.0], [9.0, 102.0, 76.0, 3
7.0, 0.0, 32.9, 0.665, 46.0], [1.0, 0.0, 74.0, 20.0, 23.0, 27.7, 0.299, 21.
0], [4.0, 114.0, 64.0, 0.0, 0.0, 28.9, 0.126, 24.0], [2.0, 56.0, 56.0, 28.0,
 45.0, 24.2, 0.332, 22.0], [8.0, 74.0, 70.0, 40.0, 49.0, 35.3, 0.705, 39.0],
 [4.0, 92.0, 80.0, 0.0, 0.0, 42.2, 0.237, 29.0], [5.0, 78.0, 48.0, 0.0, 0.0,
 33.7, 0.654, 25.0], [5.0, 116.0, 74.0, 29.0, 0.0, 32.3, 0.66, 35.0], [0.0, 1
19.0, 0.0, 0.0, 0.0, 32.4, 0.141, 24.0], [1.0, 90.0, 62.0, 18.0, 59.0, 25.1,
 1.268, 25.0], [10.0, 92.0, 62.0, 0.0, 0.0, 25.9, 0.167, 31.0], [7.0, 109.0,
 80.0, 31.0, 0.0, 35.9, 1.127, 43.0], [0.0, 104.0, 76.0, 0.0, 0.0, 18.4, 0.58
2, 27.0], [10.0, 75.0, 82.0, 0.0, 0.0, 33.3, 0.263, 38.0], [3.0, 88.0, 58.0,
 11.0, 54.0, 24.8, 0.267, 22.0], [13.0, 129.0, 0.0, 30.0, 0.0, 39.9, 0.569, 4
4.0], [5.0, 111.0, 72.0, 28.0, 0.0, 23.9, 0.407, 27.0], [2.0, 92.0, 76.0, 20.
0, 0.0, 24.2, 1.698, 28.0], [2.0, 96.0, 68.0, 13.0, 49.0, 21.1, 0.647, 26.0],
 [8.0, 107.0, 80.0, 0.0, 0.0, 24.6, 0.856, 34.0], [1.0, 85.0, 66.0, 29.0, 0.
0, 26.6, 0.351, 31.0], [0.0, 113.0, 80.0, 16.0, 0.0, 31.0, 0.874, 21.0], [1.
0, 88.0, 62.0, 24.0, 44.0, 29.9, 0.422, 23.0], [3.0, 87.0, 60.0, 18.0, 0.0, 2
1.8, 0.444, 21.0], [8.0, 84.0, 74.0, 31.0, 0.0, 38.3, 0.457, 39.0], [1.0, 11
3.0, 64.0, 35.0, 0.0, 33.6, 0.543, 21.0], [13.0, 106.0, 70.0, 0.0, 0.0, 34.2,
 0.251, 52.0], [2.0, 108.0, 64.0, 0.0, 0.0, 30.8, 0.158, 21.0], [1.0, 97.0, 7
0.0, 40.0, 0.0, 38.1, 0.218, 30.0], [7.0, 114.0, 64.0, 0.0, 0.0, 27.4, 0.732,
 34.0], [8.0, 110.0, 76.0, 0.0, 0.0, 27.8, 0.237, 58.0], [8.0, 105.0, 100.0,
 36.0, 0.0, 43.3, 0.239, 45.0], [0.0, 91.0, 80.0, 0.0, 0.0, 32.4, 0.601, 27.
0], [6.0, 91.0, 0.0, 0.0, 0.0, 29.8, 0.501, 31.0], [0.0, 105.0, 84.0, 0.0, 0.
0, 27.9, 0.741, 62.0], [5.0, 44.0, 62.0, 0.0, 0.0, 25.0, 0.587, 36.0], [7.0,
 106.0, 60.0, 24.0, 0.0, 26.5, 0.296, 29.0], [1.0, 93.0, 70.0, 31.0, 0.0, 30.
4, 0.315, 23.0], [0.0, 78.0, 88.0, 29.0, 40.0, 36.9, 0.434, 21.0], [8.0, 112.
0, 72.0, 0.0, 0.0, 23.6, 0.84, 58.0], [2.0, 75.0, 64.0, 24.0, 55.0, 29.7, 0.3
7, 33.0], [7.0, 107.0, 74.0, 0.0, 0.0, 29.6, 0.254, 31.0], [8.0, 85.0, 55.0,
 20.0, 0.0, 24.4, 0.136, 42.0], [1.0, 116.0, 70.0, 28.0, 0.0, 27.4, 0.204, 2
1.0], [10.0, 115.0, 0.0, 0.0, 0.0, 0.0, 0.261, 30.0], [6.0, 96.0, 0.0, 0.0,
 0.0, 23.7, 0.19, 28.0], [12.0, 84.0, 72.0, 31.0, 0.0, 29.7, 0.297, 46.0],
 [1.0, 95.0, 66.0, 13.0, 38.0, 19.6, 0.334, 25.0], [4.0, 117.0, 62.0, 12.0,
 0.0, 29.7, 0.38, 30.0], [0.0, 74.0, 52.0, 10.0, 36.0, 27.8, 0.269, 22.0],
 [3.0, 61.0, 82.0, 28.0, 0.0, 34.4, 0.243, 46.0], [1.0, 119.0, 54.0, 13.0, 5
0.0, 22.3, 0.205, 24.0], [0.0, 95.0, 85.0, 25.0, 36.0, 37.4, 0.247, 24.0],
 [0.0, 111.0, 65.0, 0.0, 0.0, 24.6, 0.66, 31.0], [13.0, 104.0, 72.0, 0.0, 0.
0, 31.2, 0.465, 38.0], [0.0, 107.0, 76.0, 0.0, 0.0, 45.3, 0.686, 24.0], [3.0,
 112.0, 74.0, 30.0, 0.0, 31.6, 0.197, 25.0], [1.0, 96.0, 122.0, 0.0, 0.0, 22.
4, 0.207, 27.0], [4.0, 115.0, 72.0, 0.0, 0.0, 28.9, 0.376, 46.0], [3.0, 74.0,
 68.0, 28.0, 45.0, 29.7, 0.293, 23.0], [1.0, 90.0, 68.0, 8.0, 0.0, 24.5, 1.13
8, 36.0], [4.0, 83.0, 86.0, 19.0, 0.0, 29.3, 0.317, 34.0], [5.0, 114.0, 74.0,
 0.0, 0.0, 24.9, 0.744, 57.0], [5.0, 88.0, 78.0, 30.0, 0.0, 27.6, 0.258, 37.
0], [0.0, 67.0, 76.0, 0.0, 0.0, 45.3, 0.194, 46.0], [4.0, 90.0, 88.0, 47.0, 5
4.0, 37.7, 0.362, 29.0], [6.0, 80.0, 80.0, 36.0, 0.0, 39.8, 0.177, 28.0], [3.
0, 106.0, 72.0, 0.0, 0.0, 25.8, 0.207, 27.0], [2.0, 90.0, 80.0, 14.0, 55.0, 2
4.4, 0.249, 24.0], [1.0, 100.0, 74.0, 12.0, 46.0, 19.5, 0.149, 28.0], [8.0, 1
18.0, 72.0, 19.0, 0.0, 23.1, 1.476, 46.0], [4.0, 99.0, 76.0, 15.0, 51.0, 23.
2, 0.223, 21.0], [6.0, 105.0, 80.0, 28.0, 0.0, 32.5, 0.878, 26.0], [3.0, 78.
0, 70.0, 0.0, 0.0, 32.5, 0.27, 39.0], [8.0, 120.0, 86.0, 0.0, 0.0, 28.4, 0.25
9, 22.0], [1.0, 83.0, 68.0, 0.0, 0.0, 18.2, 0.624, 27.0], [7.0, 81.0, 78.0, 4
0.0, 48.0, 46.7, 0.261, 42.0], [2.0, 83.0, 66.0, 23.0, 50.0, 32.2, 0.497, 22.
0], [2.0, 112.0, 75.0, 32.0, 0.0, 35.7, 0.148, 21.0], [1.0, 100.0, 66.0, 15.
0, 56.0, 23.6, 0.666, 26.0], [1.0, 91.0, 64.0, 24.0, 0.0, 29.2, 0.192, 21.0],
 [1.0, 92.0, 62.0, 25.0, 41.0, 19.5, 0.482, 25.0], [1.0, 87.0, 78.0, 27.0, 3
2.0, 34.6, 0.101, 22.0], [6.0, 114.0, 0.0, 0.0, 0.0, 0.0, 0.189, 26.0], [11.
0, 111.0, 84.0, 40.0, 0.0, 46.8, 0.925, 45.0], [1.0, 80.0, 74.0, 11.0, 60.0,
 30.0, 0.527, 22.0], [1.0, 93.0, 56.0, 11.0, 0.0, 22.5, 0.417, 22.0], [2.0, 8
9.0, 90.0, 30.0, 0.0, 33.5, 0.292, 42.0], [5.0, 73.0, 60.0, 0.0, 0.0, 26.8,
 0.268, 27.0], [0.0, 108.0, 68.0, 20.0, 0.0, 27.3, 0.787, 32.0], [2.0, 99.0,
 70.0, 16.0, 44.0, 20.4, 0.235, 27.0], [4.0, 99.0, 72.0, 17.0, 0.0, 25.6, 0.2
94, 28.0], [5.0, 95.0, 72.0, 33.0, 0.0, 37.7, 0.37, 27.0], [13.0, 76.0, 60.0,
 0.0, 0.0, 32.8, 0.18, 41.0], [5.0, 77.0, 82.0, 41.0, 42.0, 35.8, 0.156, 35.
0], [0.0, 101.0, 62.0, 0.0, 0.0, 21.9, 0.336, 25.0], [0.0, 102.0, 52.0, 0.0,
 0.0, 25.1, 0.078, 21.0], [4.0, 132.0, 0.0, 0.0, 0.0, 32.9, 0.302, 23.0], [3.
0, 121.0, 52.0, 0.0, 0.0, 36.0, 0.127, 25.0], [4.0, 97.0, 60.0, 23.0, 0.0, 2
8.2, 0.443, 22.0], [1.0, 81.0, 72.0, 18.0, 40.0, 26.6, 0.283, 24.0], [0.0, 10
1.0, 76.0, 0.0, 0.0, 35.7, 0.198, 26.0], [5.0, 112.0, 66.0, 0.0, 0.0, 37.8,
 0.261, 41.0], [3.0, 84.0, 72.0, 32.0, 0.0, 37.2, 0.267, 28.0], [2.0, 105.0,
 75.0, 0.0, 0.0, 23.3, 0.56, 53.0], [4.0, 122.0, 68.0, 0.0, 0.0, 35.0, 0.394,
 29.0], [4.0, 96.0, 56.0, 17.0, 49.0, 20.8, 0.34, 26.0], [10.0, 115.0, 0.0,
 0.0, 0.0, 35.3, 0.134, 29.0], [0.0, 107.0, 60.0, 25.0, 0.0, 26.4, 0.133, 23.
0], [11.0, 85.0, 74.0, 0.0, 0.0, 30.1, 0.3, 35.0], [7.0, 114.0, 66.0, 0.0, 0.
0, 32.8, 0.258, 42.0], [8.0, 108.0, 70.0, 0.0, 0.0, 30.5, 0.955, 33.0], [0.0,
 117.0, 0.0, 0.0, 0.0, 33.8, 0.932, 44.0], [10.0, 111.0, 70.0, 27.0, 0.0, 27.
5, 0.141, 40.0], [2.0, 92.0, 52.0, 0.0, 0.0, 30.1, 0.141, 22.0], [1.0, 101.0,
 50.0, 15.0, 36.0, 24.2, 0.526, 26.0], [0.0, 113.0, 76.0, 0.0, 0.0, 33.3, 0.2
78, 23.0], [4.0, 84.0, 90.0, 23.0, 56.0, 39.5, 0.159, 25.0], [5.0, 103.0, 10
8.0, 37.0, 0.0, 39.2, 0.305, 65.0], [10.0, 101.0, 86.0, 37.0, 0.0, 45.6, 1.13
6, 38.0], [5.0, 110.0, 68.0, 0.0, 0.0, 26.0, 0.292, 30.0], [0.0, 94.0, 0.0,
 0.0, 0.0, 0.0, 0.256, 25.0], [6.0, 115.0, 60.0, 39.0, 0.0, 33.7, 0.245, 40.
0], [5.0, 115.0, 76.0, 0.0, 0.0, 31.2, 0.343, 44.0], [1.0, 0.0, 68.0, 35.0,
 0.0, 32.0, 0.389, 22.0], [6.0, 109.0, 60.0, 27.0, 0.0, 25.0, 0.206, 27.0],
 [8.0, 65.0, 72.0, 23.0, 0.0, 32.0, 0.6, 42.0], [2.0, 122.0, 70.0, 27.0, 0.0,
 36.8, 0.34, 27.0], [1.0, 80.0, 55.0, 0.0, 0.0, 19.1, 0.258, 21.0], [0.0, 73.
0, 0.0, 0.0, 0.0, 21.1, 0.342, 25.0], [1.0, 89.0, 24.0, 19.0, 25.0, 27.8, 0.5
59, 21.0], [1.0, 105.0, 58.0, 0.0, 0.0, 24.3, 0.187, 21.0], [1.0, 102.0, 74.
0, 0.0, 0.0, 39.5, 0.293, 42.0], [4.0, 110.0, 66.0, 0.0, 0.0, 31.9, 0.471, 2
9.0], [2.0, 129.0, 0.0, 0.0, 0.0, 38.5, 0.304, 41.0], [10.0, 94.0, 72.0, 18.
0, 0.0, 23.1, 0.595, 56.0], [3.0, 90.0, 78.0, 0.0, 0.0, 42.7, 0.559, 21.0],
 [9.0, 106.0, 52.0, 0.0, 0.0, 31.2, 0.38, 42.0], [2.0, 84.0, 0.0, 0.0, 0.0,
 0.0, 0.304, 21.0], [6.0, 107.0, 88.0, 0.0, 0.0, 36.8, 0.727, 31.0], [6.0, 10
3.0, 66.0, 0.0, 0.0, 24.3, 0.249, 29.0], [4.0, 95.0, 70.0, 32.0, 0.0, 32.1,
 0.612, 24.0], [3.0, 96.0, 78.0, 39.0, 0.0, 37.3, 0.238, 40.0], [0.0, 93.0, 6
0.0, 0.0, 0.0, 35.3, 0.263, 25.0], [2.0, 90.0, 60.0, 0.0, 0.0, 23.5, 0.191, 2
5.0], [0.0, 105.0, 90.0, 0.0, 0.0, 29.6, 0.197, 46.0], [6.0, 117.0, 96.0, 0.
0, 0.0, 28.7, 0.157, 30.0], [8.0, 99.0, 84.0, 0.0, 0.0, 35.4, 0.388, 50.0],
 [6.0, 0.0, 68.0, 41.0, 0.0, 39.0, 0.727, 41.0], [0.0, 119.0, 66.0, 27.0, 0.
0, 38.8, 0.259, 22.0], [4.0, 85.0, 58.0, 22.0, 49.0, 27.8, 0.306, 28.0], [9.
0, 122.0, 56.0, 0.0, 0.0, 33.3, 1.114, 33.0], [10.0, 108.0, 66.0, 0.0, 0.0, 3
2.4, 0.272, 42.0], [3.0, 83.0, 58.0, 31.0, 18.0, 34.3, 0.336, 25.0], [3.0, 11
3.0, 44.0, 13.0, 0.0, 22.4, 0.14, 22.0], [6.0, 102.0, 82.0, 0.0, 0.0, 30.8,
 0.18, 36.0], [0.0, 101.0, 64.0, 17.0, 0.0, 21.0, 0.252, 21.0], [7.0, 119.0,
 0.0, 0.0, 0.0, 25.2, 0.209, 37.0], [1.0, 0.0, 48.0, 20.0, 0.0, 24.7, 0.14, 2
2.0], [1.0, 107.0, 50.0, 19.0, 0.0, 28.3, 0.181, 29.0], [0.0, 100.0, 70.0, 2
6.0, 50.0, 30.8, 0.597, 21.0], [1.0, 89.0, 76.0, 34.0, 37.0, 31.2, 0.192, 23.
0], [1.0, 79.0, 75.0, 30.0, 0.0, 32.0, 0.396, 22.0], [4.0, 94.0, 65.0, 22.0,
 0.0, 24.7, 0.148, 21.0], [2.0, 81.0, 60.0, 22.0, 0.0, 27.7, 0.29, 25.0]]
Cluster ends here.
Cluster with a size of 133 starts here:
[[3.0, 78.0, 50.0, 32.0, 88.0, 31.0, 0.248, 26.0], [4.0, 146.0, 85.0, 27.0, 1
00.0, 28.9, 0.189, 27.0], [2.0, 101.0, 58.0, 35.0, 90.0, 21.8, 0.155, 22.0],
 [6.0, 105.0, 70.0, 32.0, 68.0, 30.8, 0.122, 37.0], [1.0, 112.0, 80.0, 45.0,
 132.0, 34.8, 0.217, 24.0], [11.0, 120.0, 80.0, 37.0, 150.0, 42.3, 0.785, 48.
0], [1.0, 115.0, 70.0, 30.0, 96.0, 34.6, 0.529, 32.0], [0.0, 104.0, 64.0, 37.
0, 64.0, 33.6, 0.51, 22.0], [0.0, 120.0, 74.0, 18.0, 63.0, 30.5, 0.285, 26.
0], [9.0, 145.0, 80.0, 46.0, 130.0, 37.9, 0.637, 40.0], [0.0, 102.0, 64.0, 4
6.0, 78.0, 40.6, 0.496, 21.0], [0.0, 180.0, 90.0, 26.0, 90.0, 36.5, 0.314, 3
5.0], [0.0, 106.0, 70.0, 37.0, 148.0, 39.4, 0.605, 22.0], [0.0, 105.0, 64.0,
 41.0, 142.0, 41.5, 0.173, 22.0], [1.0, 125.0, 70.0, 24.0, 110.0, 24.3, 0.22
1, 25.0], [2.0, 100.0, 54.0, 28.0, 105.0, 37.8, 0.498, 24.0], [0.0, 129.0, 11
0.0, 46.0, 130.0, 67.1, 0.319, 26.0], [3.0, 191.0, 68.0, 15.0, 130.0, 30.9,
 0.299, 34.0], [3.0, 96.0, 56.0, 34.0, 115.0, 24.7, 0.944, 39.0], [11.0, 143.
0, 94.0, 33.0, 146.0, 36.6, 0.254, 51.0], [1.0, 88.0, 30.0, 42.0, 99.0, 55.0,
 0.496, 26.0], [2.0, 100.0, 68.0, 25.0, 71.0, 38.5, 0.324, 26.0], [2.0, 99.0,
 60.0, 17.0, 160.0, 36.6, 0.453, 21.0], [5.0, 136.0, 84.0, 41.0, 88.0, 35.0,
 0.286, 35.0], [0.0, 95.0, 80.0, 45.0, 92.0, 36.5, 0.33, 26.0], [2.0, 68.0, 7
0.0, 32.0, 66.0, 25.0, 0.187, 25.0], [2.0, 155.0, 74.0, 17.0, 96.0, 26.6, 0.4
33, 27.0], [0.0, 140.0, 65.0, 26.0, 130.0, 42.6, 0.431, 24.0], [1.0, 139.0, 4
6.0, 19.0, 83.0, 28.7, 0.654, 22.0], [3.0, 89.0, 74.0, 16.0, 85.0, 30.4, 0.55
1, 38.0], [3.0, 99.0, 80.0, 11.0, 64.0, 19.3, 0.284, 30.0], [0.0, 124.0, 56.
0, 13.0, 105.0, 21.8, 0.452, 21.0], [0.0, 126.0, 86.0, 27.0, 120.0, 27.4, 0.5
15, 21.0], [2.0, 105.0, 58.0, 40.0, 94.0, 34.9, 0.225, 25.0], [7.0, 133.0, 8
8.0, 15.0, 155.0, 32.4, 0.262, 37.0], [0.0, 117.0, 80.0, 31.0, 53.0, 45.2, 0.
089, 24.0], [0.0, 107.0, 62.0, 30.0, 74.0, 36.6, 0.757, 25.0], [1.0, 128.0, 9
8.0, 41.0, 58.0, 32.0, 1.321, 33.0], [4.0, 144.0, 58.0, 28.0, 140.0, 29.5, 0.
287, 37.0], [0.0, 118.0, 64.0, 23.0, 89.0, 0.0, 1.731, 21.0], [0.0, 98.0, 82.
0, 15.0, 84.0, 25.2, 0.299, 22.0], [6.0, 134.0, 70.0, 23.0, 130.0, 35.4, 0.54
2, 29.0], [2.0, 99.0, 52.0, 15.0, 94.0, 24.6, 0.637, 21.0], [11.0, 138.0, 74.
0, 26.0, 144.0, 36.1, 0.557, 50.0], [4.0, 117.0, 64.0, 27.0, 120.0, 33.2, 0.2
3, 24.0], [5.0, 117.0, 86.0, 30.0, 105.0, 39.1, 0.251, 42.0], [3.0, 115.0, 6
6.0, 39.0, 140.0, 38.1, 0.15, 28.0], [2.0, 106.0, 56.0, 27.0, 165.0, 29.0, 0.
426, 22.0], [3.0, 130.0, 78.0, 23.0, 79.0, 28.4, 0.323, 34.0], [0.0, 97.0, 6
4.0, 36.0, 100.0, 36.8, 0.6, 25.0], [4.0, 127.0, 88.0, 11.0, 155.0, 34.5, 0.5
98, 28.0], [5.0, 96.0, 74.0, 18.0, 67.0, 33.6, 0.997, 43.0], [5.0, 139.0, 64.
0, 35.0, 140.0, 28.6, 0.411, 26.0], [6.0, 93.0, 50.0, 30.0, 64.0, 28.7, 0.35
6, 23.0], [1.0, 122.0, 64.0, 32.0, 156.0, 35.1, 0.692, 30.0], [1.0, 84.0, 64.
0, 23.0, 115.0, 36.9, 0.471, 28.0], [17.0, 163.0, 72.0, 41.0, 114.0, 40.9, 0.
817, 47.0], [2.0, 121.0, 70.0, 32.0, 95.0, 39.1, 0.886, 23.0], [1.0, 82.0, 6
4.0, 13.0, 95.0, 21.2, 0.415, 23.0], [1.0, 167.0, 74.0, 17.0, 144.0, 23.4, 0.
```

```
447, 33.0], [0.0, 94.0, 70.0, 27.0, 115.0, 43.5, 0.347, 21.0], [15.0, 136.0,
 70.0, 32.0, 110.0, 37.1, 0.153, 43.0], [5.0, 99.0, 54.0, 28.0, 83.0, 34.0,
 0.499, 30.0], [3.0, 113.0, 50.0, 10.0, 85.0, 29.5, 0.626, 25.0], [2.0, 108.
0, 62.0, 32.0, 56.0, 25.2, 0.128, 21.0], [1.0, 103.0, 30.0, 38.0, 83.0, 43.3,
 0.183, 33.0], [3.0, 180.0, 64.0, 25.0, 70.0, 34.0, 0.271, 26.0], [3.0, 99.0,
 62.0, 19.0, 74.0, 21.8, 0.279, 26.0], [2.0, 84.0, 50.0, 23.0, 76.0, 30.4, 0.
968, 21.0], [1.0, 164.0, 82.0, 43.0, 67.0, 32.8, 0.341, 50.0], [2.0, 112.0, 8
6.0, 42.0, 160.0, 38.4, 0.246, 28.0], [9.0, 134.0, 74.0, 33.0, 60.0, 25.9, 0.
46, 81.0], [3.0, 103.0, 72.0, 30.0, 152.0, 27.6, 0.73, 27.0], [2.0, 112.0, 6
8.0, 22.0, 94.0, 34.1, 0.315, 26.0], [5.0, 121.0, 72.0, 23.0, 112.0, 26.2, 0.
245, 30.0], [2.0, 82.0, 52.0, 22.0, 115.0, 28.5, 1.699, 25.0], [3.0, 81.0, 8
6.0, 16.0, 66.0, 27.5, 0.306, 22.0], [6.0, 125.0, 68.0, 30.0, 120.0, 30.0, 0.
464, 32.0], [2.0, 122.0, 60.0, 18.0, 106.0, 29.8, 0.717, 22.0], [2.0, 110.0,
 74.0, 29.0, 125.0, 32.4, 0.698, 27.0], [1.0, 86.0, 66.0, 52.0, 65.0, 41.3,
 0.917, 29.0], [6.0, 108.0, 44.0, 20.0, 130.0, 24.0, 0.813, 35.0], [7.0, 83.
0, 78.0, 26.0, 71.0, 29.3, 0.767, 36.0], [7.0, 136.0, 74.0, 26.0, 135.0, 26.
0, 0.647, 51.0], [2.0, 108.0, 52.0, 26.0, 63.0, 32.5, 0.318, 22.0], [1.0, 12
1.0, 78.0, 39.0, 74.0, 39.0, 0.261, 28.0], [3.0, 80.0, 82.0, 31.0, 70.0, 34.
2, 1.292, 27.0], [3.0, 106.0, 54.0, 21.0, 158.0, 30.9, 0.292, 24.0], [1.0, 10
9.0, 38.0, 18.0, 120.0, 23.1, 0.407, 26.0], [1.0, 117.0, 88.0, 24.0, 145.0, 3
4.5, 0.403, 40.0], [2.0, 120.0, 76.0, 37.0, 105.0, 39.7, 0.215, 29.0], [2.0,
 125.0, 60.0, 20.0, 140.0, 33.8, 0.088, 31.0], [8.0, 179.0, 72.0, 42.0, 130.
0, 32.7, 0.719, 36.0], [1.0, 130.0, 70.0, 13.0, 105.0, 25.9, 0.472, 22.0],
 [1.0, 91.0, 54.0, 25.0, 100.0, 25.2, 0.234, 23.0], [1.0, 133.0, 102.0, 28.0,
 140.0, 32.8, 0.234, 45.0], [2.0, 174.0, 88.0, 37.0, 120.0, 44.5, 0.646, 24.
0], [2.0, 102.0, 86.0, 36.0, 120.0, 45.5, 0.127, 23.0], [1.0, 103.0, 80.0, 1
1.0, 82.0, 19.4, 0.491, 22.0], [0.0, 95.0, 64.0, 39.0, 105.0, 44.6, 0.366, 2
2.0], [0.0, 102.0, 78.0, 40.0, 90.0, 34.5, 0.238, 24.0], [13.0, 153.0, 88.0,
 37.0, 140.0, 40.6, 1.174, 39.0], [1.0, 109.0, 58.0, 18.0, 116.0, 28.5, 0.21
9, 22.0], [3.0, 171.0, 72.0, 33.0, 135.0, 33.3, 0.199, 24.0], [5.0, 123.0, 7
4.0, 40.0, 77.0, 34.1, 0.269, 28.0], [3.0, 169.0, 74.0, 19.0, 125.0, 29.9, 0.
268, 31.0], [8.0, 109.0, 76.0, 39.0, 114.0, 27.9, 0.64, 31.0], [3.0, 116.0, 7
4.0, 15.0, 105.0, 26.3, 0.107, 24.0], [1.0, 149.0, 68.0, 29.0, 127.0, 29.3,
 0.349, 42.0], [1.0, 143.0, 74.0, 22.0, 61.0, 26.2, 0.256, 21.0], [1.0, 106.
0, 70.0, 28.0, 135.0, 34.2, 0.142, 22.0], [0.0, 135.0, 94.0, 46.0, 145.0, 40.
6, 0.284, 26.0], [4.0, 91.0, 70.0, 32.0, 88.0, 33.1, 0.446, 22.0], [2.0, 122.
0, 52.0, 43.0, 158.0, 36.2, 0.816, 28.0], [1.0, 109.0, 56.0, 21.0, 135.0, 25.
2, 0.833, 23.0], [3.0, 111.0, 90.0, 12.0, 78.0, 28.4, 0.495, 29.0], [12.0, 10
0.0, 84.0, 33.0, 105.0, 30.0, 0.488, 46.0], [2.0, 106.0, 64.0, 35.0, 119.0, 3
0.5, 1.4, 34.0], [2.0, 98.0, 60.0, 17.0, 120.0, 34.7, 0.198, 22.0], [0.0, 10
4.0, 64.0, 23.0, 116.0, 27.8, 0.454, 23.0], [9.0, 120.0, 72.0, 22.0, 56.0, 2
0.8, 0.733, 48.0], [3.0, 129.0, 64.0, 29.0, 115.0, 26.4, 0.219, 28.0], [1.0,
 100.0, 72.0, 12.0, 70.0, 25.3, 0.658, 28.0], [1.0, 126.0, 56.0, 29.0, 152.0,
 28.7, 0.801, 21.0], [3.0, 84.0, 68.0, 30.0, 106.0, 31.9, 0.591, 25.0], [2.0,
 94.0, 76.0, 18.0, 66.0, 31.6, 0.649, 23.0], [3.0, 163.0, 70.0, 18.0, 105.0,
 31.6, 0.268, 28.0], [1.0, 118.0, 58.0, 36.0, 94.0, 33.3, 0.261, 23.0], [0.0,
 93.0, 100.0, 39.0, 72.0, 43.4, 1.021, 35.0], [4.0, 109.0, 64.0, 44.0, 99.0,
 34.8, 0.905, 26.0], [2.0, 117.0, 90.0, 19.0, 71.0, 25.2, 0.313, 21.0], [0.0,
 119.0, 64.0, 18.0, 92.0, 34.9, 0.725, 23.0], [6.0, 104.0, 74.0, 18.0, 156.0,
 29.9, 0.722, 41.0]]
Cluster ends here.
Cluster with a size of 17 starts here:
[[0.0, 165.0, 90.0, 33.0, 680.0, 52.3, 0.427, 23.0], [1.0, 131.0, 64.0, 14.0,
 415.0, 23.7, 0.389, 21.0], [3.0, 158.0, 64.0, 13.0, 387.0, 31.2, 0.295, 24.
0], [1.0, 139.0, 62.0, 41.0, 480.0, 40.7, 0.536, 21.0], [1.0, 153.0, 82.0, 4
2.0, 485.0, 40.6, 0.687, 23.0], [2.0, 146.0, 70.0, 38.0, 360.0, 28.0, 0.337,
 29.0], [8.0, 155.0, 62.0, 26.0, 495.0, 34.0, 0.543, 46.0], [4.0, 197.0, 70.
```

```
0, 39.0, 744.0, 36.7, 2.329, 31.0], [3.0, 173.0, 84.0, 33.0, 474.0, 35.7, 0.2
58, 22.0], [6.0, 134.0, 80.0, 37.0, 370.0, 46.2, 0.238, 46.0], [3.0, 173.0, 8
2.0, 48.0, 465.0, 38.4, 2.137, 25.0], [7.0, 142.0, 90.0, 24.0, 480.0, 30.4,
 0.128, 43.0], [8.0, 181.0, 68.0, 36.0, 495.0, 30.1, 0.615, 60.0], [7.0, 187.
0, 50.0, 33.0, 392.0, 33.9, 0.826, 34.0], [7.0, 150.0, 66.0, 42.0, 342.0, 34.
7, 0.718, 42.0], [9.0, 124.0, 70.0, 33.0, 402.0, 35.4, 0.282, 34.0], [2.0, 15
7.0, 74.0, 35.0, 440.0, 39.4, 0.134, 30.0]]
Cluster ends here.
Cluster with a size of 76 starts here:
[[5.0, 166.0, 72.0, 19.0, 175.0, 25.8, 0.587, 51.0], [12.0, 92.0, 62.0, 7.0,
 258.0, 27.6, 0.926, 44.0], [1.0, 119.0, 88.0, 41.0, 170.0, 45.3, 0.507, 26.
0], [1.0, 130.0, 60.0, 23.0, 170.0, 28.6, 0.692, 21.0], [5.0, 158.0, 84.0, 4
1.0, 210.0, 39.4, 0.395, 29.0], [1.0, 109.0, 60.0, 8.0, 182.0, 25.4, 0.947, 2
1.0], [1.0, 100.0, 66.0, 29.0, 196.0, 32.0, 0.444, 42.0], [6.0, 119.0, 50.0,
 22.0, 176.0, 27.1, 1.318, 33.0], [1.0, 140.0, 74.0, 26.0, 180.0, 24.1, 0.82
8, 23.0], [4.0, 103.0, 60.0, 33.0, 192.0, 24.0, 0.966, 33.0], [1.0, 196.0, 7
6.0, 36.0, 249.0, 36.5, 0.875, 29.0], [14.0, 100.0, 78.0, 25.0, 184.0, 36.6,
 0.412, 46.0], [0.0, 127.0, 80.0, 37.0, 210.0, 36.3, 0.804, 23.0], [2.0, 146.
0, 76.0, 35.0, 194.0, 38.2, 0.329, 29.0], [12.0, 151.0, 70.0, 40.0, 271.0, 4
1.8, 0.742, 38.0], [2.0, 127.0, 58.0, 24.0, 275.0, 27.7, 1.6, 25.0], [3.0, 15
8.0, 76.0, 36.0, 245.0, 31.6, 0.851, 28.0], [0.0, 134.0, 58.0, 20.0, 291.0, 2
6.4, 0.352, 21.0], [0.0, 137.0, 40.0, 35.0, 168.0, 43.1, 2.288, 33.0], [1.0,
 157.0, 72.0, 21.0, 168.0, 25.6, 0.123, 24.0], [5.0, 139.0, 80.0, 35.0, 160.
0, 31.6, 0.361, 25.0], [4.0, 129.0, 86.0, 20.0, 270.0, 35.1, 0.231, 23.0],
 [1.0, 111.0, 62.0, 13.0, 182.0, 24.0, 0.138, 23.0], [1.0, 116.0, 78.0, 29.0,
 180.0, 36.1, 0.496, 25.0], [5.0, 187.0, 76.0, 27.0, 207.0, 43.6, 1.034, 53.
0], [3.0, 174.0, 58.0, 22.0, 194.0, 32.9, 0.593, 36.0], [2.0, 122.0, 76.0, 2
7.0, 200.0, 35.9, 0.483, 26.0], [2.0, 123.0, 48.0, 32.0, 165.0, 42.1, 0.52, 2
6.0], [4.0, 147.0, 74.0, 25.0, 293.0, 34.9, 0.385, 30.0], [5.0, 105.0, 72.0,
 29.0, 325.0, 36.9, 0.159, 28.0], [0.0, 139.0, 62.0, 17.0, 210.0, 22.1, 0.20
7, 21.0], [0.0, 121.0, 66.0, 30.0, 165.0, 34.3, 0.203, 33.0], [2.0, 105.0, 8
0.0, 45.0, 191.0, 33.7, 0.711, 29.0], [4.0, 154.0, 62.0, 31.0, 284.0, 32.8,
 0.237, 23.0], [1.0, 112.0, 72.0, 30.0, 176.0, 34.4, 0.528, 25.0], [1.0, 95.
0, 82.0, 25.0, 180.0, 35.0, 0.233, 43.0], [5.0, 189.0, 64.0, 33.0, 325.0, 31.
2, 0.583, 29.0], [9.0, 171.0, 110.0, 24.0, 240.0, 45.4, 0.721, 54.0], [0.0, 1
88.0, 82.0, 14.0, 185.0, 32.0, 0.682, 22.0], [7.0, 187.0, 68.0, 39.0, 304.0,
 37.7, 0.254, 41.0], [3.0, 187.0, 70.0, 22.0, 200.0, 36.4, 0.408, 36.0], [6.
0, 129.0, 90.0, 7.0, 326.0, 19.6, 0.582, 60.0], [3.0, 123.0, 100.0, 35.0, 24
0.0, 57.3, 0.88, 22.0], [0.0, 91.0, 68.0, 32.0, 210.0, 39.9, 0.381, 25.0],
 [0.0, 152.0, 82.0, 39.0, 272.0, 41.5, 0.27, 27.0], [9.0, 112.0, 82.0, 32.0,
 175.0, 34.2, 0.26, 36.0], [7.0, 181.0, 84.0, 21.0, 192.0, 35.9, 0.586, 51.
0], [8.0, 186.0, 90.0, 35.0, 225.0, 34.5, 0.423, 37.0], [6.0, 165.0, 68.0, 2
6.0, 168.0, 33.6, 0.631, 49.0], [1.0, 181.0, 64.0, 30.0, 180.0, 34.1, 0.328,
 38.0], [1.0, 136.0, 74.0, 50.0, 204.0, 37.4, 0.399, 24.0], [3.0, 128.0, 72.
0, 25.0, 190.0, 32.4, 0.549, 27.0], [2.0, 124.0, 68.0, 28.0, 205.0, 32.9, 0.8
75, 30.0], [2.0, 128.0, 78.0, 37.0, 182.0, 43.3, 1.224, 31.0], [6.0, 144.0, 7
2.0, 27.0, 228.0, 33.9, 0.255, 40.0], [7.0, 160.0, 54.0, 32.0, 175.0, 30.5,
 0.588, 39.0], [0.0, 126.0, 84.0, 29.0, 215.0, 30.7, 0.52, 24.0], [3.0, 158.
0, 70.0, 30.0, 328.0, 35.5, 0.344, 35.0], [4.0, 111.0, 72.0, 47.0, 207.0, 37.
1, 1.39, 56.0], [8.0, 176.0, 90.0, 34.0, 300.0, 33.7, 0.467, 58.0], [5.0, 14
4.0, 82.0, 26.0, 285.0, 32.0, 0.452, 58.0], [10.0, 101.0, 76.0, 48.0, 180.0,
 32.9, 0.171, 63.0], [0.0, 135.0, 68.0, 42.0, 250.0, 42.3, 0.365, 24.0], [1.
0, 114.0, 66.0, 36.0, 200.0, 38.1, 0.289, 21.0], [7.0, 124.0, 70.0, 33.0, 21
5.0, 25.5, 0.161, 37.0], [8.0, 100.0, 74.0, 40.0, 215.0, 39.4, 0.661, 43.0],
 [10.0, 148.0, 84.0, 48.0, 237.0, 37.6, 1.001, 51.0], [4.0, 131.0, 68.0, 21.
0, 166.0, 33.1, 0.16, 28.0], [0.0, 118.0, 84.0, 47.0, 230.0, 45.8, 0.551, 31.
0], [4.0, 129.0, 60.0, 12.0, 231.0, 27.5, 0.527, 31.0], [0.0, 128.0, 68.0, 1
```

```
9.0, 180.0, 30.5, 1.391, 25.0], [1.0, 181.0, 78.0, 42.0, 293.0, 40.0, 1.258,
 22.0], [1.0, 128.0, 82.0, 17.0, 183.0, 27.5, 0.115, 22.0], [9.0, 145.0, 88.
0, 34.0, 165.0, 30.3, 0.771, 53.0], [1.0, 122.0, 90.0, 51.0, 220.0, 49.7, 0.3
25, 31.0], [0.0, 165.0, 76.0, 43.0, 255.0, 47.9, 0.259, 26.0]]
Cluster ends here.
Cluster with a size of 111 starts here:
[[9.0, 164.0, 78.0, 0.0, 0.0, 32.8, 0.148, 45.0], [4.0, 134.0, 72.0, 0.0, 0.
0, 23.8, 0.277, 60.0], [7.0, 147.0, 76.0, 0.0, 0.0, 39.4, 0.257, 43.0], [9.0,
 164.0, 84.0, 21.0, 0.0, 30.8, 0.831, 32.0], [4.0, 136.0, 70.0, 0.0, 0.0, 31.
2, 1.182, 22.0], [5.0, 128.0, 80.0, 0.0, 0.0, 34.6, 0.144, 45.0], [1.0, 199.
0, 76.0, 43.0, 0.0, 42.9, 1.394, 22.0], [0.0, 147.0, 85.0, 54.0, 0.0, 42.8,
 0.375, 24.0], [2.0, 197.0, 70.0, 99.0, 0.0, 34.7, 0.575, 62.0], [4.0, 145.0,
 82.0, 18.0, 0.0, 32.5, 0.235, 70.0], [2.0, 128.0, 64.0, 42.0, 0.0, 40.0, 1.1
01, 24.0], [0.0, 180.0, 66.0, 39.0, 0.0, 42.0, 1.893, 25.0], [6.0, 148.0, 72.
0, 35.0, 0.0, 33.6, 0.627, 50.0], [0.0, 123.0, 72.0, 0.0, 0.0, 36.3, 0.258, 5
2.0], [5.0, 147.0, 75.0, 0.0, 0.0, 29.9, 0.434, 28.0], [7.0, 178.0, 84.0, 0.
0, 0.0, 39.9, 0.331, 41.0], [10.0, 122.0, 78.0, 31.0, 0.0, 27.6, 0.512, 45.
0], [6.0, 124.0, 72.0, 0.0, 0.0, 27.6, 0.368, 29.0], [3.0, 150.0, 76.0, 0.0,
 0.0, 21.0, 0.207, 37.0], [1.0, 124.0, 74.0, 36.0, 0.0, 27.8, 0.1, 30.0], [9.
0, 184.0, 85.0, 15.0, 0.0, 30.0, 1.213, 49.0], [7.0, 136.0, 90.0, 0.0, 0.0, 2
9.9, 0.21, 50.0], [8.0, 194.0, 80.0, 0.0, 0.0, 26.1, 0.551, 67.0], [2.0, 158.
0, 90.0, 0.0, 0.0, 31.6, 0.805, 66.0], [7.0, 137.0, 90.0, 41.0, 0.0, 32.0, 0.
391, 39.0], [6.0, 137.0, 61.0, 0.0, 0.0, 24.2, 0.151, 55.0], [10.0, 122.0, 6
8.0, 0.0, 0.0, 31.2, 0.258, 41.0], [13.0, 126.0, 90.0, 0.0, 0.0, 43.4, 0.583,
 42.0], [5.0, 126.0, 78.0, 27.0, 22.0, 29.6, 0.439, 40.0], [0.0, 141.0, 84.0,
 26.0, 0.0, 32.4, 0.433, 22.0], [3.0, 139.0, 54.0, 0.0, 0.0, 25.6, 0.402, 22.
0], [4.0, 156.0, 75.0, 0.0, 0.0, 48.3, 0.238, 32.0], [7.0, 196.0, 90.0, 0.0,
 0.0, 39.8, 0.451, 41.0], [6.0, 195.0, 70.0, 0.0, 0.0, 30.9, 0.328, 31.0],
 [1.0, 151.0, 60.0, 0.0, 0.0, 26.1, 0.179, 22.0], [6.0, 114.0, 88.0, 0.0, 0.
0, 27.8, 0.247, 66.0], [5.0, 158.0, 70.0, 0.0, 0.0, 29.8, 0.207, 63.0], [7.0,
 133.0, 84.0, 0.0, 0.0, 40.2, 0.696, 37.0], [1.0, 144.0, 82.0, 40.0, 0.0, 41.
3, 0.607, 28.0], [4.0, 141.0, 74.0, 0.0, 0.0, 27.6, 0.244, 40.0], [4.0, 128.
0, 70.0, 0.0, 0.0, 34.3, 0.303, 24.0], [0.0, 146.0, 70.0, 0.0, 0.0, 37.9, 0.3
34, 28.0], [9.0, 170.0, 74.0, 31.0, 0.0, 44.0, 0.403, 43.0], [0.0, 179.0, 90.
0, 27.0, 0.0, 44.1, 0.686, 23.0], [4.0, 132.0, 86.0, 31.0, 0.0, 28.0, 0.419,
 63.0], [4.0, 144.0, 82.0, 32.0, 0.0, 38.5, 0.554, 37.0], [2.0, 134.0, 70.0,
 0.0, 0.0, 28.9, 0.542, 23.0], [0.0, 125.0, 68.0, 0.0, 0.0, 24.7, 0.206, 21.
0], [12.0, 121.0, 78.0, 17.0, 0.0, 26.5, 0.259, 62.0], [5.0, 137.0, 108.0, 0.
0, 0.0, 48.8, 0.227, 37.0], [4.0, 171.0, 72.0, 0.0, 0.0, 43.6, 0.479, 26.0],
 [8.0, 183.0, 64.0, 0.0, 0.0, 23.3, 0.672, 32.0], [5.0, 136.0, 82.0, 0.0, 0.
0, 0.0, 0.64, 69.0], [7.0, 184.0, 84.0, 33.0, 0.0, 35.5, 0.355, 41.0], [0.0,
 146.0, 82.0, 0.0, 0.0, 40.5, 1.781, 44.0], [9.0, 156.0, 86.0, 0.0, 0.0, 24.
8, 0.23, 53.0], [4.0, 137.0, 84.0, 0.0, 0.0, 31.2, 0.252, 30.0], [3.0, 132.0,
 80.0, 0.0, 0.0, 34.4, 0.402, 44.0], [0.0, 180.0, 78.0, 63.0, 14.0, 59.4, 2.4
2, 25.0], [9.0, 130.0, 70.0, 0.0, 0.0, 34.2, 0.652, 45.0], [8.0, 143.0, 66.0,
 0.0, 0.0, 34.9, 0.129, 41.0], [5.0, 124.0, 74.0, 0.0, 0.0, 34.0, 0.22, 38.
0], [0.0, 189.0, 104.0, 25.0, 0.0, 34.3, 0.435, 41.0], [10.0, 139.0, 80.0, 0.
0, 0.0, 27.1, 1.441, 57.0], [0.0, 125.0, 96.0, 0.0, 0.0, 22.5, 0.262, 21.0],
 [5.0, 132.0, 80.0, 0.0, 0.0, 26.8, 0.186, 69.0], [5.0, 162.0, 104.0, 0.0, 0.
0, 37.7, 0.151, 52.0], [7.0, 194.0, 68.0, 28.0, 0.0, 35.9, 0.745, 41.0], [5.
0, 143.0, 78.0, 0.0, 0.0, 45.0, 0.19, 47.0], [3.0, 182.0, 74.0, 0.0, 0.0, 30.
5, 0.345, 29.0], [8.0, 154.0, 78.0, 32.0, 0.0, 32.4, 0.443, 45.0], [2.0, 139.
0, 75.0, 0.0, 0.0, 25.6, 0.167, 29.0], [0.0, 167.0, 0.0, 0.0, 0.0, 32.3, 0.83
9, 30.0], [0.0, 131.0, 88.0, 0.0, 0.0, 31.6, 0.743, 32.0], [0.0, 129.0, 80.0,
 0.0, 0.0, 31.2, 0.703, 29.0], [3.0, 162.0, 52.0, 38.0, 0.0, 37.2, 0.652, 24.
0], [10.0, 133.0, 68.0, 0.0, 0.0, 27.0, 0.245, 36.0], [7.0, 159.0, 64.0, 0.0,
 0.0, 27.4, 0.294, 40.0], [0.0, 141.0, 0.0, 0.0, 0.0, 42.4, 0.205, 29.0], [1.
```

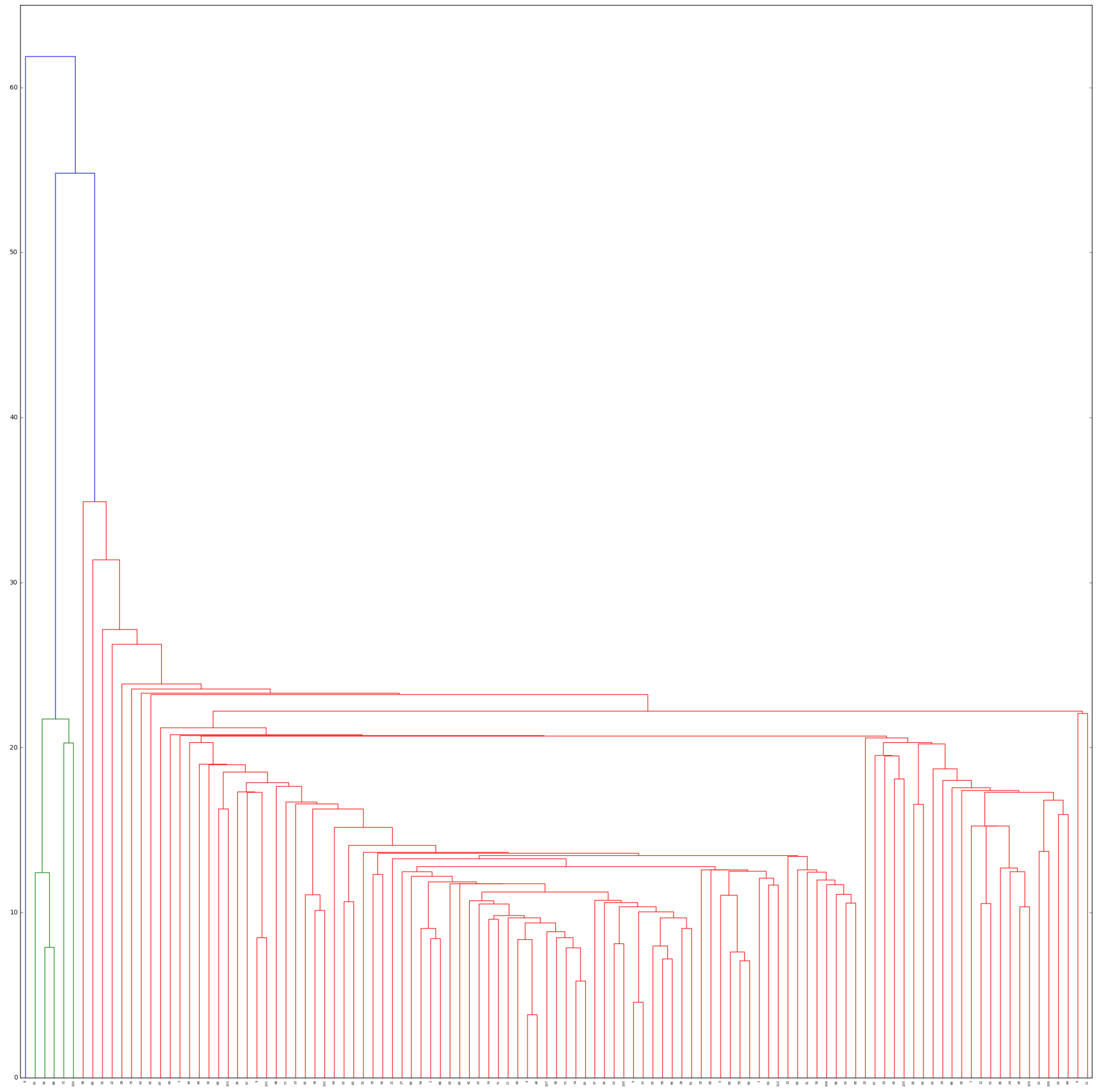
```
0, 163.0, 72.0, 0.0, 0.0, 39.0, 1.222, 33.0], [13.0, 152.0, 90.0, 33.0, 29.0,
 26.8, 0.731, 43.0], [2.0, 129.0, 84.0, 0.0, 0.0, 28.0, 0.284, 27.0], [7.0, 1
79.0, 95.0, 31.0, 0.0, 34.2, 0.164, 60.0], [8.0, 197.0, 74.0, 0.0, 0.0, 25.9,
 1.191, 39.0], [0.0, 123.0, 88.0, 37.0, 0.0, 35.2, 0.197, 29.0], [10.0, 168.
0, 74.0, 0.0, 0.0, 38.0, 0.537, 34.0], [0.0, 124.0, 70.0, 20.0, 0.0, 27.4, 0.
254, 36.0], [0.0, 162.0, 76.0, 36.0, 0.0, 49.6, 0.364, 26.0], [0.0, 138.0, 0.
0, 0.0, 0.0, 36.3, 0.933, 25.0], [3.0, 122.0, 78.0, 0.0, 0.0, 23.0, 0.254, 4
0.0], [10.0, 162.0, 84.0, 0.0, 0.0, 27.7, 0.182, 54.0], [2.0, 146.0, 0.0, 0.
0, 0.0, 27.5, 0.24, 28.0], [10.0, 129.0, 62.0, 36.0, 0.0, 41.2, 0.441, 38.0],
 [1.0, 168.0, 88.0, 29.0, 0.0, 35.0, 0.905, 52.0], [11.0, 127.0, 106.0, 0.0,
 0.0, 39.0, 0.19, 51.0], [8.0, 120.0, 78.0, 0.0, 0.0, 25.0, 0.409, 64.0], [8.
0, 133.0, 72.0, 0.0, 0.0, 32.9, 0.27, 39.0], [4.0, 146.0, 78.0, 0.0, 0.0, 38.
5, 0.52, 67.0], [1.0, 173.0, 74.0, 0.0, 0.0, 36.8, 0.088, 38.0], [9.0, 165.0,
 88.0, 0.0, 0.0, 30.4, 0.302, 49.0], [6.0, 125.0, 76.0, 0.0, 0.0, 33.8, 0.12
1, 54.0], [3.0, 142.0, 80.0, 15.0, 0.0, 32.4, 0.2, 63.0], [1.0, 146.0, 56.0,
 0.0, 0.0, 29.7, 0.564, 29.0], [13.0, 158.0, 114.0, 0.0, 0.0, 42.3, 0.257, 4
4.0], [1.0, 180.0, 0.0, 0.0, 0.0, 43.3, 0.282, 41.0], [14.0, 175.0, 62.0, 30.
0, 0.0, 33.6, 0.212, 38.0], [0.0, 137.0, 70.0, 38.0, 0.0, 33.2, 0.17, 22.0],
 [0.0, 132.0, 78.0, 0.0, 0.0, 32.4, 0.393, 21.0], [10.0, 179.0, 70.0, 0.0, 0.
0, 35.1, 0.2, 37.0], [12.0, 140.0, 85.0, 33.0, 0.0, 37.4, 0.244, 41.0], [6.0,
 147.0, 80.0, 0.0, 0.0, 29.5, 0.178, 50.0]]
Cluster ends here.
```

```
In [52]: for i in a:
             print i
```

```
[9.0, 164.0, 78.0, 0.0, 0.0, 32.8, 0.148, 45.0]
[4.0, 134.0, 72.0, 0.0, 0.0, 23.8, 0.277, 60.0]
[7.0, 147.0, 76.0, 0.0, 0.0, 39.4, 0.257, 43.0]
[9.0, 164.0, 84.0, 21.0, 0.0, 30.8, 0.831, 32.0]
[4.0, 136.0, 70.0, 0.0, 0.0, 31.2, 1.182, 22.0]
[5.0, 128.0, 80.0, 0.0, 0.0, 34.6, 0.144, 45.0]
[1.0, 199.0, 76.0, 43.0, 0.0, 42.9, 1.394, 22.0]
[0.0, 147.0, 85.0, 54.0, 0.0, 42.8, 0.375, 24.0]
[2.0, 197.0, 70.0, 99.0, 0.0, 34.7, 0.575, 62.0]
[4.0, 145.0, 82.0, 18.0, 0.0, 32.5, 0.235, 70.0]
[2.0, 128.0, 64.0, 42.0, 0.0, 40.0, 1.101, 24.0]
[0.0, 180.0, 66.0, 39.0, 0.0, 42.0, 1.893, 25.0]
[6.0, 148.0, 72.0, 35.0, 0.0, 33.6, 0.627, 50.0]
[0.0, 123.0, 72.0, 0.0, 0.0, 36.3, 0.258, 52.0]
[5.0, 147.0, 75.0, 0.0, 0.0, 29.9, 0.434, 28.0]
[7.0, 178.0, 84.0, 0.0, 0.0, 39.9, 0.331, 41.0]
[10.0, 122.0, 78.0, 31.0, 0.0, 27.6, 0.512, 45.0]
[6.0, 124.0, 72.0, 0.0, 0.0, 27.6, 0.368, 29.0]
[3.0, 150.0, 76.0, 0.0, 0.0, 21.0, 0.207, 37.0]
[1.0, 124.0, 74.0, 36.0, 0.0, 27.8, 0.1, 30.0]
[9.0, 184.0, 85.0, 15.0, 0.0, 30.0, 1.213, 49.0]
[7.0, 136.0, 90.0, 0.0, 0.0, 29.9, 0.21, 50.0]
[8.0, 194.0, 80.0, 0.0, 0.0, 26.1, 0.551, 67.0]
[2.0, 158.0, 90.0, 0.0, 0.0, 31.6, 0.805, 66.0]
[7.0, 137.0, 90.0, 41.0, 0.0, 32.0, 0.391, 39.0]
[6.0, 137.0, 61.0, 0.0, 0.0, 24.2, 0.151, 55.0]
[10.0, 122.0, 68.0, 0.0, 0.0, 31.2, 0.258, 41.0]
[13.0, 126.0, 90.0, 0.0, 0.0, 43.4, 0.583, 42.0]
[5.0, 126.0, 78.0, 27.0, 22.0, 29.6, 0.439, 40.0]
[0.0, 141.0, 84.0, 26.0, 0.0, 32.4, 0.433, 22.0]
[3.0, 139.0, 54.0, 0.0, 0.0, 25.6, 0.402, 22.0]
[4.0, 156.0, 75.0, 0.0, 0.0, 48.3, 0.238, 32.0]
[7.0, 196.0, 90.0, 0.0, 0.0, 39.8, 0.451, 41.0]
[6.0, 195.0, 70.0, 0.0, 0.0, 30.9, 0.328, 31.0]
[1.0, 151.0, 60.0, 0.0, 0.0, 26.1, 0.179, 22.0]
[6.0, 114.0, 88.0, 0.0, 0.0, 27.8, 0.247, 66.0]
[5.0, 158.0, 70.0, 0.0, 0.0, 29.8, 0.207, 63.0]
[7.0, 133.0, 84.0, 0.0, 0.0, 40.2, 0.696, 37.0]
[1.0, 144.0, 82.0, 40.0, 0.0, 41.3, 0.607, 28.0]
[4.0, 141.0, 74.0, 0.0, 0.0, 27.6, 0.244, 40.0]
[4.0, 128.0, 70.0, 0.0, 0.0, 34.3, 0.303, 24.0]
[0.0, 146.0, 70.0, 0.0, 0.0, 37.9, 0.334, 28.0]
[9.0, 170.0, 74.0, 31.0, 0.0, 44.0, 0.403, 43.0]
[0.0, 179.0, 90.0, 27.0, 0.0, 44.1, 0.686, 23.0]
[4.0, 132.0, 86.0, 31.0, 0.0, 28.0, 0.419, 63.0]
[4.0, 144.0, 82.0, 32.0, 0.0, 38.5, 0.554, 37.0]
[2.0, 134.0, 70.0, 0.0, 0.0, 28.9, 0.542, 23.0]
[0.0, 125.0, 68.0, 0.0, 0.0, 24.7, 0.206, 21.0]
[12.0, 121.0, 78.0, 17.0, 0.0, 26.5, 0.259, 62.0]
[5.0, 137.0, 108.0, 0.0, 0.0, 48.8, 0.227, 37.0]
[4.0, 171.0, 72.0, 0.0, 0.0, 43.6, 0.479, 26.0]
[8.0, 183.0, 64.0, 0.0, 0.0, 23.3, 0.672, 32.0]
[5.0, 136.0, 82.0, 0.0, 0.0, 0.0, 0.64, 69.0]
[7.0, 184.0, 84.0, 33.0, 0.0, 35.5, 0.355, 41.0]
[0.0, 146.0, 82.0, 0.0, 0.0, 40.5, 1.781, 44.0]
[9.0, 156.0, 86.0, 0.0, 0.0, 24.8, 0.23, 53.0]
[4.0, 137.0, 84.0, 0.0, 0.0, 31.2, 0.252, 30.0]
```

```
[3.0, 132.0, 80.0, 0.0, 0.0, 34.4, 0.402, 44.0]
[0.0, 180.0, 78.0, 63.0, 14.0, 59.4, 2.42, 25.0]
[9.0, 130.0, 70.0, 0.0, 0.0, 34.2, 0.652, 45.0]
[8.0, 143.0, 66.0, 0.0, 0.0, 34.9, 0.129, 41.0]
[5.0, 124.0, 74.0, 0.0, 0.0, 34.0, 0.22, 38.0]
[0.0, 189.0, 104.0, 25.0, 0.0, 34.3, 0.435, 41.0]
[10.0, 139.0, 80.0, 0.0, 0.0, 27.1, 1.441, 57.0]
[0.0, 125.0, 96.0, 0.0, 0.0, 22.5, 0.262, 21.0]
[5.0, 132.0, 80.0, 0.0, 0.0, 26.8, 0.186, 69.0]
[5.0, 162.0, 104.0, 0.0, 0.0, 37.7, 0.151, 52.0]
[7.0, 194.0, 68.0, 28.0, 0.0, 35.9, 0.745, 41.0]
[5.0, 143.0, 78.0, 0.0, 0.0, 45.0, 0.19, 47.0]
[3.0, 182.0, 74.0, 0.0, 0.0, 30.5, 0.345, 29.0]
[8.0, 154.0, 78.0, 32.0, 0.0, 32.4, 0.443, 45.0]
[2.0, 139.0, 75.0, 0.0, 0.0, 25.6, 0.167, 29.0]
[0.0, 167.0, 0.0, 0.0, 0.0, 32.3, 0.839, 30.0]
[0.0, 131.0, 88.0, 0.0, 0.0, 31.6, 0.743, 32.0]
[0.0, 129.0, 80.0, 0.0, 0.0, 31.2, 0.703, 29.0]
[3.0, 162.0, 52.0, 38.0, 0.0, 37.2, 0.652, 24.0]
[10.0, 133.0, 68.0, 0.0, 0.0, 27.0, 0.245, 36.0]
[7.0, 159.0, 64.0, 0.0, 0.0, 27.4, 0.294, 40.0]
[0.0, 141.0, 0.0, 0.0, 0.0, 42.4, 0.205, 29.0]
[1.0, 163.0, 72.0, 0.0, 0.0, 39.0, 1.222, 33.0]
[13.0, 152.0, 90.0, 33.0, 29.0, 26.8, 0.731, 43.0]
[2.0, 129.0, 84.0, 0.0, 0.0, 28.0, 0.284, 27.0]
[7.0, 179.0, 95.0, 31.0, 0.0, 34.2, 0.164, 60.0]
[8.0, 197.0, 74.0, 0.0, 0.0, 25.9, 1.191, 39.0]
[0.0, 123.0, 88.0, 37.0, 0.0, 35.2, 0.197, 29.0]
[10.0, 168.0, 74.0, 0.0, 0.0, 38.0, 0.537, 34.0]
[0.0, 124.0, 70.0, 20.0, 0.0, 27.4, 0.254, 36.0]
[0.0, 162.0, 76.0, 36.0, 0.0, 49.6, 0.364, 26.0]
[0.0, 138.0, 0.0, 0.0, 0.0, 36.3, 0.933, 25.0]
[3.0, 122.0, 78.0, 0.0, 0.0, 23.0, 0.254, 40.0]
[10.0, 162.0, 84.0, 0.0, 0.0, 27.7, 0.182, 54.0]
[2.0, 146.0, 0.0, 0.0, 0.0, 27.5, 0.24, 28.0]
[10.0, 129.0, 62.0, 36.0, 0.0, 41.2, 0.441, 38.0]
[1.0, 168.0, 88.0, 29.0, 0.0, 35.0, 0.905, 52.0]
[11.0, 127.0, 106.0, 0.0, 0.0, 39.0, 0.19, 51.0]
[8.0, 120.0, 78.0, 0.0, 0.0, 25.0, 0.409, 64.0]
[8.0, 133.0, 72.0, 0.0, 0.0, 32.9, 0.27, 39.0]
[4.0, 146.0, 78.0, 0.0, 0.0, 38.5, 0.52, 67.0]
[1.0, 173.0, 74.0, 0.0, 0.0, 36.8, 0.088, 38.0]
[9.0, 165.0, 88.0, 0.0, 0.0, 30.4, 0.302, 49.0]
[6.0, 125.0, 76.0, 0.0, 0.0, 33.8, 0.121, 54.0]
[3.0, 142.0, 80.0, 15.0, 0.0, 32.4, 0.2, 63.0]
[1.0, 146.0, 56.0, 0.0, 0.0, 29.7, 0.564, 29.0]
[13.0, 158.0, 114.0, 0.0, 0.0, 42.3, 0.257, 44.0]
[1.0, 180.0, 0.0, 0.0, 0.0, 43.3, 0.282, 41.0]
[14.0, 175.0, 62.0, 30.0, 0.0, 33.6, 0.212, 38.0]
[0.0, 137.0, 70.0, 38.0, 0.0, 33.2, 0.17, 22.0]
[0.0, 132.0, 78.0, 0.0, 0.0, 32.4, 0.393, 21.0]
[10.0, 179.0, 70.0, 0.0, 0.0, 35.1, 0.2, 37.0]
[12.0, 140.0, 85.0, 33.0, 0.0, 37.4, 0.244, 41.0]
[6.0, 147.0, 80.0, 0.0, 0.0, 29.5, 0.178, 50.0]
```

In [53]:
```
plt.figure(figsize = (30, 30))
cc = hierarchy.linkage(a)
hierarchy.dendrogram(cc)
plt.show()
```



Let's perform K-Means clustering on this data.

The objective function in K-Means clustering scheme is:

$$J = \sum_{j=1}^{k} \sum_{i=1}^{n} ||x_i^{(j)} - c_j||^2$$

The procedure for finding K-Means consists of:

1. Choose K random points as the initial group centroids.
2. Each object is assigned to the group with the closest centroid.
3. Recalculate the positions of the K centroids after all objects are assigned.
4. If the stop condition is met, which is when centroids no longer move, then we have finished. Otherwise goto step 2.

Note: K-Means clustering is very sensitive to the initial randomly selected cluster centers. Performing the algorithm several times may alleviate this issue.

```
def cluster_points(X, mu): clusters = {} for x in X: bestmukey = min([(i[0], np.linalg.norm(x-mu[i[0]])) \ for i in
enumerate(mu)], key=lambda t:t[1])[0] try: clusters[bestmukey].append(x) except KeyError: clusters[bestmukey] =
[x] return clusters def reevaluate_centers(mu, clusters): newmu = [] keys = sorted(clusters.keys()) for k in keys:
newmu.append(np.mean(clusters[k], axis = 0)) return newmu def has_converged(mu, oldmu): return (set([tuple(a)
for a in mu]) == set([tuple(a) for a in oldmu]) def find_centers(X, K): # Initialize to K random centers oldmu =
random.sample(X, K) mu = random.sample(X, K) while not has_converged(mu, oldmu): oldmu = mu # Assign all
points in X to clusters clusters = cluster_points(X, mu) # Reevaluate centers mu = reevaluate_centers(oldmu,
clusters) return(mu, clusters)
```

Attach this form on front of your answers.