Assignment_2: Unsupervised Data Mining

Q1. 30 Points

Q2. 30 Points

Q3. 20 Points

Q4. 20 Points

Q5. 10 Bonus Points

In [1]:

```
%matplotlib inline

import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib as mpl
import matplotlib.pyplot as plt
import sklearn
import pickle

from sklearn.preprocessing import StandardScaler
from sklearn.utils import check_random_state
from sklearn.decomposition import PCA

from nose.tools import assert_equal, assert_is_instance, assert_is_not
from numpy.testing import assert_array_equal, assert_array_almost_equal, assert_almost_equal
from pandas.util.testing import assert_frame_equal

import warnings
warnings.filterwarnings("ignore")
```

```
<ipython-input-1-1650308bd6ff>:17: FutureWarning: pandas.util.testing is deprecated. Use the functions in the publi
c API at pandas.testing instead.
  from pandas.util.testing import assert_frame_equal
```

The things you should pay attention:

Make sure you fill in any place that says YOUR CODE HERE. Do not write your answer in anywhere else other than where it says YOUR CODE HERE. Anything you write anywhere else will be removed or overwritten by the autograder.

Before you submit your assignment, make sure everything runs as expected. If you have sufficient time, please go to menubar, select Kernel, and restart the kernel and run all cells (Restart & Run all).

Make sure that you save your work (in the menubar, select File → Save and CheckPoint)

Good Luck!

UP

Problem_1: Dimension Reduction

With Problem_1, we aim to have a better understanding of dimension reduction with PCA. We will use Delta Airline data. Delta and other major airlines have data on all of their aircrafts on their website. [e.g. (https://www.delta.com/content/www/en_US/traveling-with-us/airports-and-aircraft/Aircraft.html)](https://www.delta.com/content/www/en_US/traveling-with-us/airports-and-aircraft/Aircraft.html)

We will use delta.csv uploaded on Canvas Module for this assignment.

This data set has 34 columns (including the names of the aircrafts) on 44 aircrafts. It inclues both quantitative measurements such as cruising speed, accommodation and range in miles, as well as categorical data, such as whether a particular aircraft has Wi-Fi or video. These binary are assigned values of either 1 or 0, for yes or no respectively.

In [2]:
```
df = pd.read_csv('delta.csv', index_col='Aircraft')
```

In [3]:
```
df.head()
```

Out[3]:

Aircraft	Seat Width (Club)	Seat Pitch (Club)	Seat (Club)	Seat Width (First Class)	Seat Pitch (First Class)	Seats (First Class)	Seat Width (Business)	Seat Pitch (Business)	Seats (Business)	Seat Width (Eco Comfort)	...	Video	Power	Satellite	Flat-bed	Sleeper
Airbus A319	0.0	0	0	21.0	36.0	12	0.0	0.0	0	17.2	...	0	0	0	0	0
Airbus A319 VIP	19.4	44	12	19.4	40.0	28	21.0	59.0	14	0.0	...	1	0	0	0	0
Airbus A320	0.0	0	0	21.0	36.0	12	0.0	0.0	0	17.2	...	0	0	0	0	0
Airbus A320 32-R	0.0	0	0	21.0	36.0	12	0.0	0.0	0	17.2	...	0	0	0	0	0
Airbus A330-200	0.0	0	0	0.0	0.0	0	21.0	60.0	32	18.0	...	1	1	0	1	0

5 rows × 33 columns

First, let's look at the attributes related to the aircraft physical characteristics:

Cruising Speed (mph) Range (miles) Engines Wingspan (ft) Tail Height (ft) Length (ft) These six variables are about in the middle of the data frame (and it's part of your task to figure out where they are located).

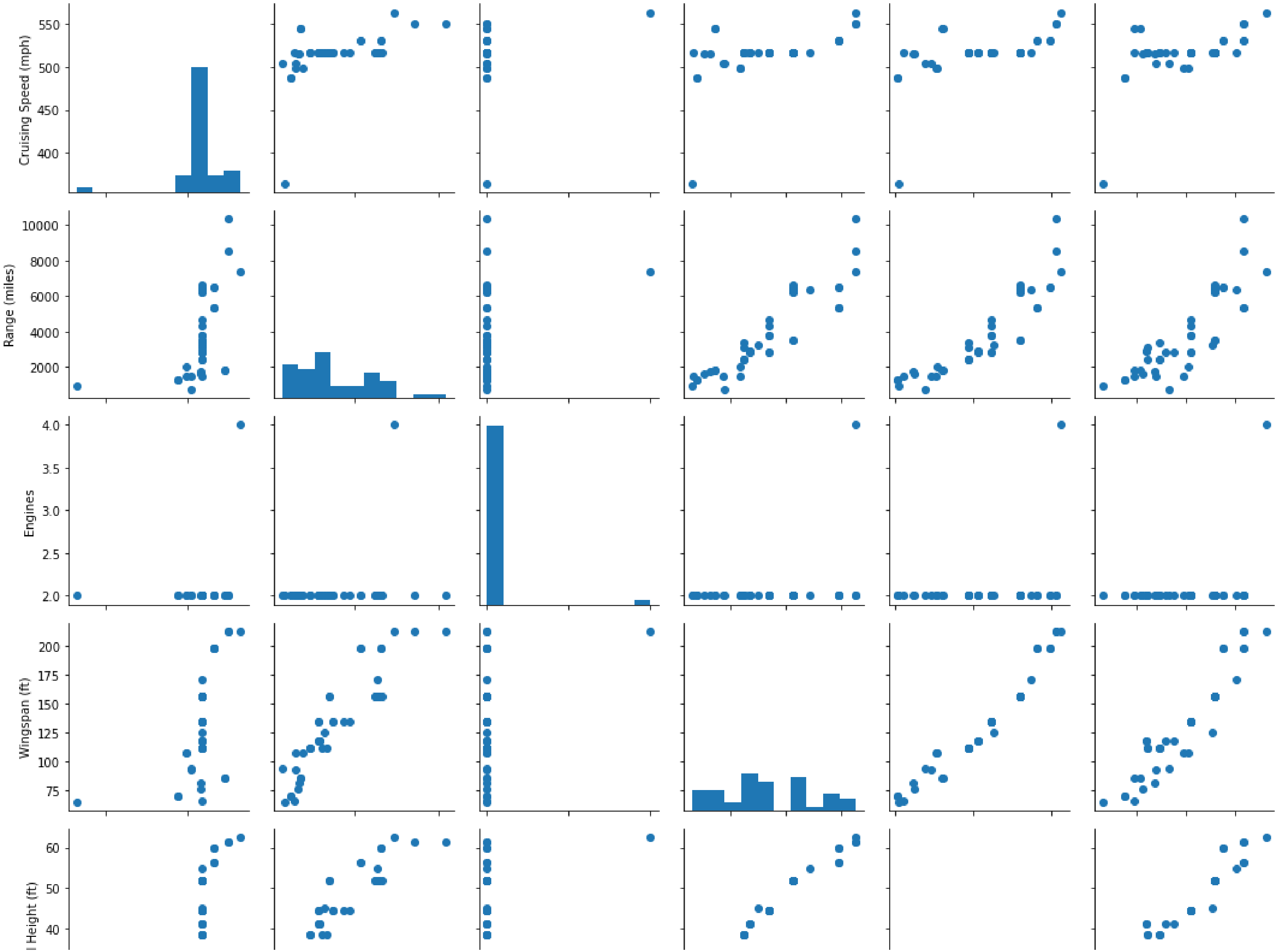
Write a function named plot_pairgrid() that takes a pandas.DataFrame and uses seaborn.PairGrid to visualize the attributes related to the six physical characteristics listed above. The plots on the diagonal should be histograms of corresponding attributes, and the off-diagonal should be scatter plots.

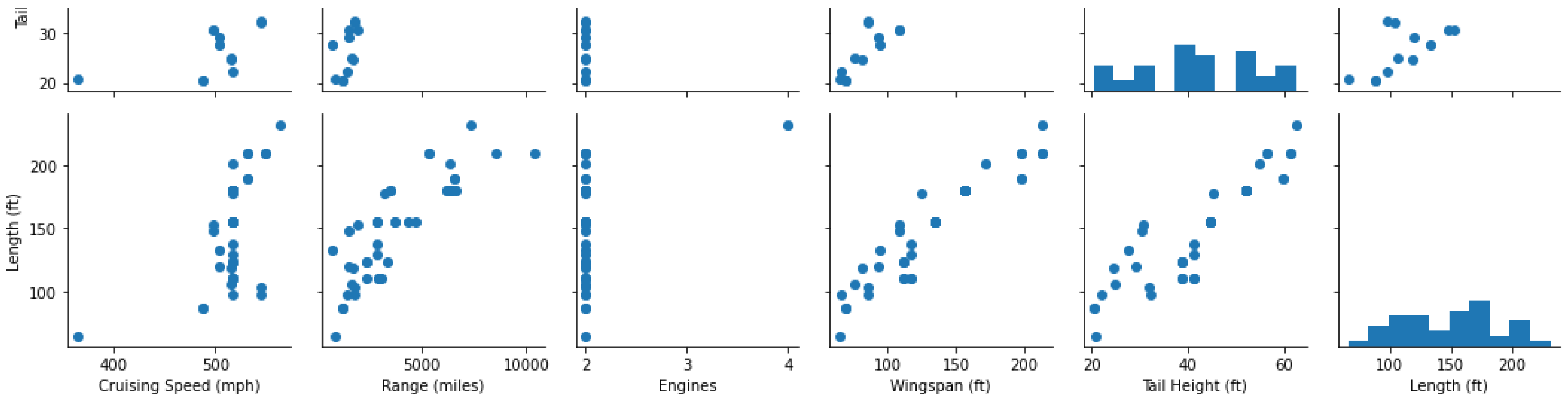
In [4]:

```python
def plot_pairgrid(df):
    '''
    Uses seaborn.PairGrid to visualize the attributes related to the six physical characteristics.
    Diagonal plots are histograms. The off-diagonal plots are scatter plots.

    Parameters
    ----------
    df: A pandas.DataFrame. Comes from importing delta.csv.

    Returns
    -------
    A seaborn.axisgrid.PairGrid instance.
    '''
    ax = sns.PairGrid(data = df[['Cruising Speed (mph)', 'Range (miles)', 'Engines',
        'Wingspan (ft)', 'Tail Height (ft)', 'Length (ft)']])

    # YOUR CODE HERE


    ax.map_diag(plt.hist)
    ax.map_offdiag(plt.scatter)

    return ax
```

In [5]:

```
pg = plot_pairgrid(df); #your answer should look like this
```





We observe that pretty strong positive correlations between all these variables, as most of them are related to the aircraft's overall size. Remarkably there is an almost perfectly linear relationship between wingspan and tail height.

The exception here is engines. There is one outlier which has four engines, while all the other aircraft have two. In this way the engines variable is really more like a categorical variable, but we shall as the analysis progresses that this is not really important, as there are other variables which more strongly discern the aircraft from one another than this.

In [6]:
```python
### This is the unittest cell, please just run this cell without any modification once you generated "pg" above

cols = ['Cruising Speed (mph)', 'Range (miles)', 'Engines',
        'Wingspan (ft)', 'Tail Height (ft)', 'Length (ft)']

assert_is_instance(pg.fig, plt.Figure)
assert_equal(set(pg.data.columns), set(cols))

for ax in pg.diag_axes:
    assert_equal(len(ax.patches), 10)

for i, j in zip(*np.triu_indices_from(pg.axes, 1)):
    ax = pg.axes[i, j]
    x_in = df[cols[j]]
    y_in = df[cols[i]]
    x_out, y_out = ax.collections[0].get_offsets().T
    assert_array_equal(x_in, x_out)
    assert_array_equal(y_in, y_out)

for i, j in zip(*np.tril_indices_from(pg.axes, -1)):
    ax = pg.axes[i, j]
    x_in = df[cols[j]]
    y_in = df[cols[i]]
    x_out, y_out = ax.collections[0].get_offsets().T
    assert_array_equal(x_in, x_out)
    assert_array_equal(y_in, y_out)

for i, j in zip(*np.diag_indices_from(pg.axes)):
    ax = pg.axes[i, j]
    assert_equal(len(ax.collections), 0)
```

Apply PCA

I assume we dont know anything about dimensionality reduction techniques and just naively apply principle components to the data.

Write a function named fit_pca() that takes a pandas.DataFrame and uses [sklearn.decomposition.PCA (http://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html)](http://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html) to fit a PCA model on all values of df.

In [7]:

```python
def fit_pca(df, n_components):
    '''
    Uses sklearn.decomposition.PCA to fit a PCA model on "df".

    Parameters
    ----------
    df: A pandas.DataFrame. Comes from delta.csv.
    n_components: An int. Number of principal components to keep.

    Returns
    -------
    An sklearn.decomposition.pca.PCA instance.
    '''

    # YOUR CODE HERE
    pca = PCA(n_components)
    pca.fit(df)


    return pca
```

In [8]:

```python
# we keep all components by setting n_components = no of cols in df. FYI df.shape[0] returns # of rows, len(df.columr
pca_naive = fit_pca(df, n_components=df.shape[1])
```

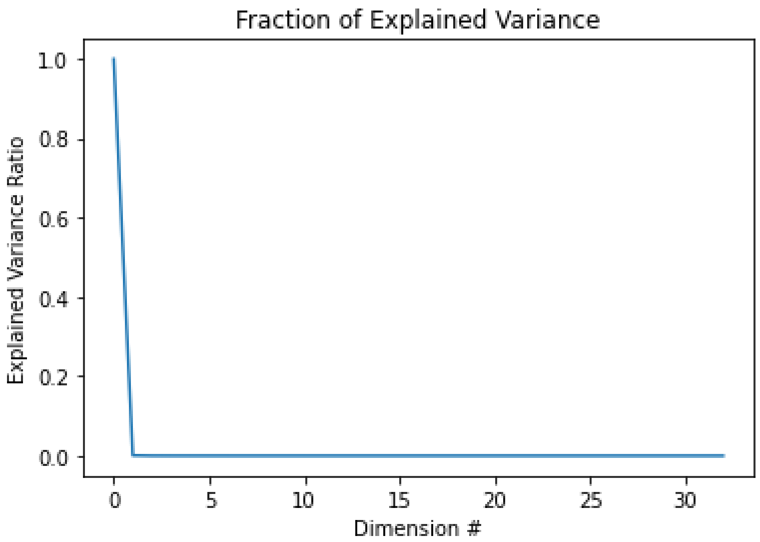
In [9]:

```python
assert_is_instance(pca_naive, PCA)
assert_almost_equal(pca_naive.explained_variance_ratio_.sum(), 1.0, 3)
assert_equal(pca_naive.n_components_, df.shape[1])
assert_equal(pca_naive.whiten, False)
```

In [10]:

```python
def plot_naive_variance(pca):
    '''
    Plots the variance explained by each of the principal components.
    Attributes are not scaled, hence a naive approach.

    Parameters
    ----------
    pca: An sklearn.decomposition.pca.PCA instance.

    Returns
    -------
    A matplotlib.Axes instance.
    '''

    # YOUR CODE HERE
    ax = plt.axes()
    plt.plot(pca.explained_variance_ratio_)

    ax.set_xlabel("Dimension #")
    ax.set_ylabel("Explained Variance Ratio")
    ax.set_title("Fraction of Explained Variance")


    return ax
```

In [11]:
```
naive_var = plot_naive_variance(pca_naive)
```



In [12]:

```
assert_is_instance(naive_var, mpl.axes.Axes)
assert_equal(len(naive_var.lines), 1)

assert_is_not(len(naive_var.title.get_text()), 0,
    msg="Your plot doesn't have a title.")
assert_is_not(naive_var.xaxis.get_label_text(), '',
    msg="Change the x-axis label to something more descriptive.")
assert_is_not(naive_var.yaxis.get_label_text(), '',
    msg="Change the y-axis label to something more descriptive.")

xdata, ydata = naive_var.lines[0].get_xydata().T
assert_array_equal(xdata, list(range(df.shape[1])))
assert_array_almost_equal(ydata, pca_naive.explained_variance_ratio_)
```

In [13]:

```
abs_val = np.abs(pca_naive.components_[0])
max_pos = abs_val.argmax()
max_val = abs_val.max()

print('"{0}" accounts for {1:0.3f} % of the variance.'.format(df.columns[max_pos], max_val))
```

```
"Range (miles)" accounts for 0.999 % of the variance.
```

Taking this naive approach, we can see that the first principal component accounts for 99.9% of the variance in the data. (Note the y-axis is on a log scale.) Looking more closely, can we see that the first principle component is just the range in miles? This is because the scale of the different variables in the data set is quite variable.

PCA is a scale-dependent method. For example, if the range of one column is [-100, 100], while the that of another column is [-0.1, 0.1], PCA will place more weight on the feature with larger values. One way to avoid this is to standardize a data set by scaling each feature so that the individual features all look like Gausssian distributions with zero mean and unit variance.

Please write a function named standardize() where StandardScaler function of sklearn will be used to scale each feature so that they have zero mean and unit variance.

In [14]:

```python
def standardize(df):
    '''
    Uses sklearn.preprocessing.StandardScaler to make each features look like
    a Gaussian with zero mean and unit variance.

    Parameters
    ----------
    df: A pandas.DataFrame

    Returns
    -------
    A numpy array.
    '''

    # YOUR CODE HERE
    scaler = StandardScaler().fit(df)
    scaled = scaler.transform(df)

    return scaled
```

In [15]:

```python
scaled = standardize(df)
```

In [16]:
```
rng = np.random.RandomState(0)
n_samples, n_features = 4, 5

df_t1 = pd.DataFrame(
    rng.randn(n_samples, n_features),
    index=[i for i in 'abcd'],
    columns=[c for c  in 'abcde']
    )
df_t1.loc[:, 'a'] = 0.0  # make first feature zero

scaled_t1 = standardize(df_t1)

assert_is_not(df_t1, scaled_t1)
assert_is_instance(scaled_t1, np.ndarray)
assert_array_almost_equal(
    scaled_t1.mean(axis=0),
    n_features * [0.0] # scaled data should have mean zero
    )
assert_array_almost_equal(
    scaled_t1.std(axis=0),
    [0., 1., 1., 1., 1.] # unit variance except for 1st feature
    )
```

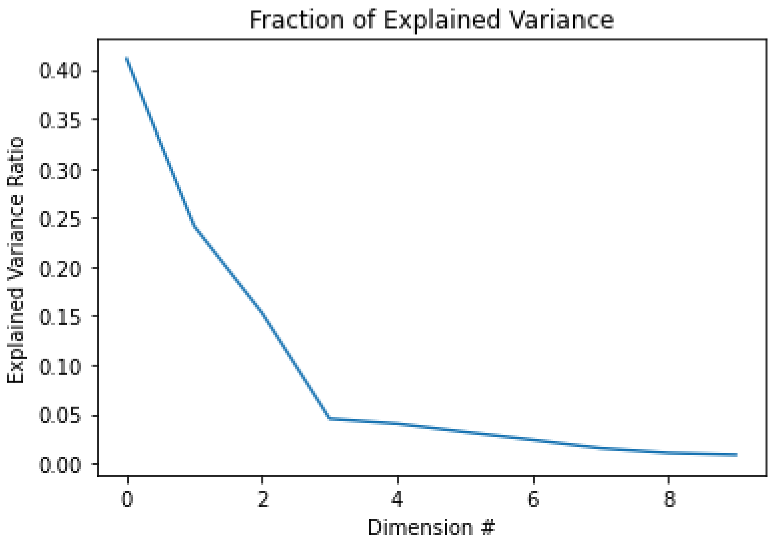
In [17]:
```
# we keep only 10 components
n_components = 10
pca = fit_pca(scaled, n_components=n_components)
```

Let's take another look to the explained variance of the first 10 principal components from the scaled data.

In [18]:

```python
def plot_scaled_variance(pca):
    '''
    Plots the variance explained by each of the principal components.
    Features are scaled with sklearn.StandardScaler.

    Parameters
    ----------
    pca: An sklearn.decomposition.pca.PCA instance.

    Returns
    -------
    A matplotlib.Axes instance.
    '''

    # YOUR CODE HERE
    ax = plt.axes()
    plt.plot(pca.explained_variance_ratio_)

    ax.set_xlabel("Dimension #")
    ax.set_ylabel("Explained Variance Ratio")
    ax.set_title("Fraction of Explained Variance")



    return ax
```

In [19]:
```
ax = plot_scaled_variance(pca)
```



In [20]:
```
assert_is_instance(ax, mpl.axes.Axes)
assert_equal(len(ax.lines), 1)

assert_is_not(len(ax.title.get_text()), 0, msg="Your plot doesn't have a title.")
assert_is_not(ax.xaxis.get_label_text(), '', msg="Change the x-axis label to something more descriptive.")
assert_is_not(ax.yaxis.get_label_text(), '', msg="Change the y-axis label to something more descriptive.")

xdata, ydata = ax.lines[0].get_xydata().T
assert_array_equal(xdata, list(range(n_components)))
assert_array_almost_equal(ydata, pca.explained_variance_ratio_)
```

Nice, it looks good to go. There are various rules of thumb for selecting the number of principal components to retain in an analysis of this type, one of which I've experienced about is:

Pick the number of components which explain 85% or greater of the variation. So, we will keep the first 4 principal components (remember that we are counting from zero, so we are keeping 0th, 1st, 2nd, and 3rd components—four components). Later in this assignment, we will use these four components to fit a k-means model. Before we move on to the next problem, let's apply the dimensional reduction on the scaled data. (In the previous sections, we didn't actually have to apply transform(). This step is to make sure that the scaled data is actually "transformed".)

Write a function named reduce() that takes a PCA model (that is already trained on array) and a Numpy array, and applies dimensional reduction on the array.

In [21]:
```python
def reduce(pca, array):
    '''
    Applies the `pca` model on array.

    Parameters
    ----------
    pca: An sklearn.decomposition.PCA instance.

    Returns
    -------
    A Numpy array
    '''

    # YOUR CODE HERE
    pca = PCA(n_components=10)
    reduced = pca.fit_transform(array)


    return reduced
```

In [22]:
```python
reduced = reduce(pca, scaled)
```

In [23]:
```python
assert_is_instance(reduced, np.ndarray)
assert_array_almost_equal(reduced, pca.fit_transform(scaled))
```

In [24]:
```python
# Save the reduced data to the same directory of your notebook  as 'delta_reeuced.npy' that we will use later on
np.save('delta_reduced.npy', reduced)
```

Problem 2. Clustering

We will use the first 10 principal components of the Delta Airline data set that we created in the first step.

In [25]:

```
##Standard imports just in case

%matplotlib inline

import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib as mpl
import matplotlib.pyplot as plt
import sklearn

from sklearn.utils import check_random_state
from sklearn.cluster import KMeans

from nose.tools import assert_equal, assert_is_instance, assert_true, assert_is_not
from numpy.testing import assert_array_equal, assert_array_almost_equal, assert_almost_equal
```

In [98]:

```
## Reload the the first 10 components of delta dataset
reduced = np.load('delta_reduced.npy')
```

Write a function named cluster() that fits a k-means clustering algorithm, and returns a tuple (sklearn.cluster.k_means_.KMeans, np.array). The second element of the tuple is a 1-d array that contains the predictions of k-means clustering, i.e. which cluster each data point belongs to. Please remember how we were generating and using the labels for seeds, movements, iris etc.

Use default values for all parameters in KMeans() execept for n_clusters and random_state.

In [27]:

```
len(reduced)
```

Out[27]: 44

In [28]:

```python
def cluster(array, random_state, n_clusters=4):
    '''
    Fits and predicts k-means clustering on "array"

    Parameters
    ----------
    array: A numpy array
    random_state: Random seed, e.g. check_random_state(0)
    n_clusters: The number of clusters. Default: 4

    Returns
    -------
    A tuple (sklearn.KMeans, np.ndarray)
    '''
    model = KMeans(n_clusters = n_clusters, init = 'k-means++', random_state = 0)

    clusters= model.fit_predict(array)


    return model, clusters

#here we return fitted (model) and predicted (clusters) arrays as a tuple
```

In [29]:

```python
k_means_t, cluster_t = cluster(reduced, random_state=check_random_state(1), n_clusters=5)

assert_is_instance(k_means_t, sklearn.cluster._kmeans.KMeans)
assert_is_instance(cluster_t, np.ndarray)
assert_equal(k_means_t.n_init, 10)
assert_equal(k_means_t.n_clusters, 5)
assert_equal(len(cluster_t), len(reduced))
assert_true((cluster_t < 5).all()) # n_cluster = 5 so labels should be between 0 and 5
assert_true((cluster_t >= 0).all())
labels_gold = -1. * np.ones(len(reduced), dtype=int)
mindist = np.empty(len(reduced))
mindist.fill(np.infty)
for i in range(5):
    dist = np.sum((reduced - k_means_t.cluster_centers_[i])**2., axis=1)
    labels_gold[dist < mindist] = i
    mindist = np.minimum(dist, mindist)
assert_true((mindist >= 0.0).all())
assert_true((labels_gold != -1).all())
assert_array_equal(labels_gold, cluster_t)
```

The scikit-learn documentation on sklearn.cluster.KMeans says that [Kmeans cluster (http://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html)](http://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html) has the inertia value in the inertia_ attribute. So we can vary the number of clusters in KMeans, plot KMeans.inertia_ as a function of the number of clusters, and pick the "elbow" in the plot.

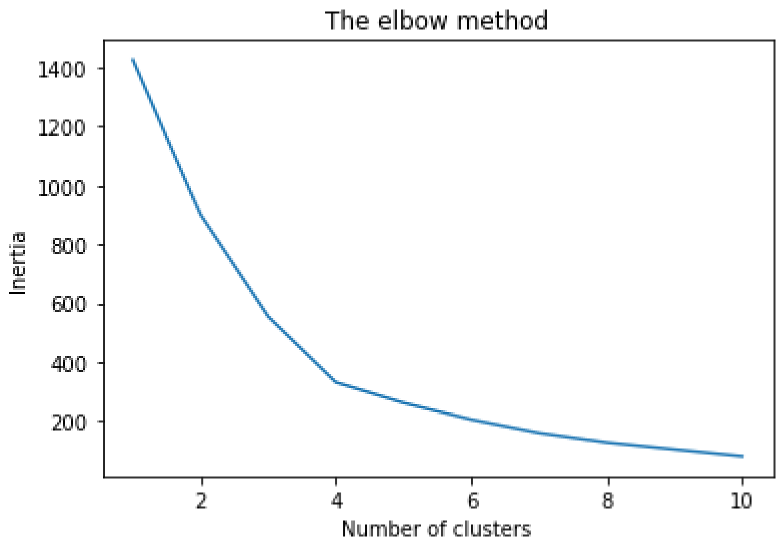
Always use check_random_state(0) to seed the random number generator.

In [30]:

```python
def plot_inertia(array, start=1, end=10):
    '''
    Increase the number of clusters from "start" to "end" (inclusive).
    Finds the inertia of k-means clustering for different k.
    Plots inertia as a function of the number of clusters.


    Parameters
    ----------
    array: A numpy array.
    start: An int. Default: 1
    end: An int. Default: 10

    Returns
    -------
    A matplotlib.Axes instance.
    '''
    #Your code is here
    x_axis = range(1, 11)
    inertia = []

    for i in x_axis:

        # Create a KMeans instance with k clusters: model
        model = KMeans(n_clusters=i,init = 'k-means++', random_state = 0)

    # Fit model to samples
        model.fit(array)

    # Append the inertia to the list of inertias
        inertia.append(model.inertia_)


    fig, ax = plt.subplots(figsize=(6,4))
    ax.set_title('The elbow method')
    ax.set_ylabel('Inertia')
    ax.set_xlabel('Number of clusters')
    plt.plot(x_axis, inertia)

    return ax
```

In [31]:

```
inertia = plot_inertia(reduced)
```



In [32]:

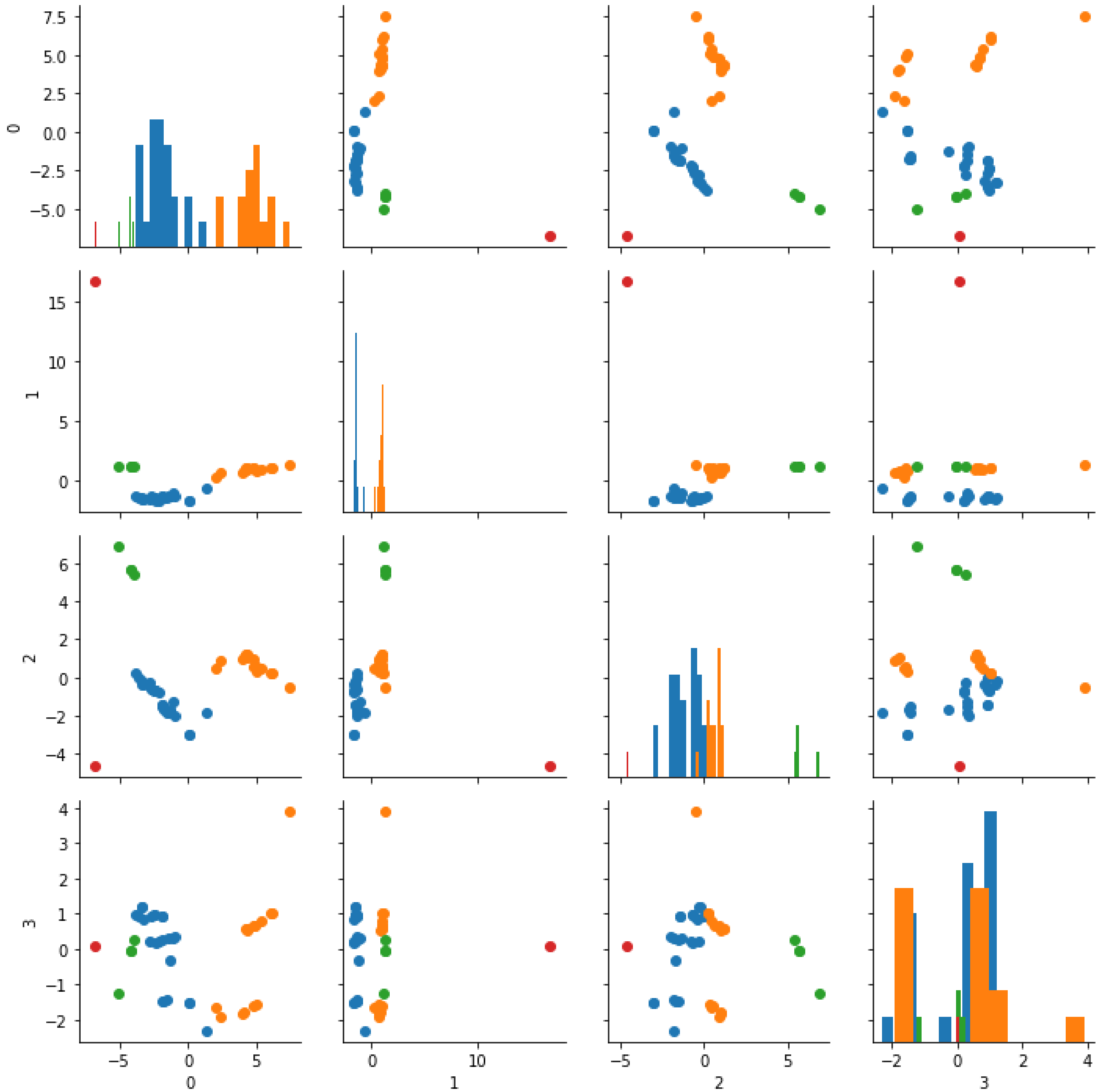
```
assert_is_instance(inertia, mpl.axes.Axes)
assert_true(len(inertia.lines) >= 1)

xdata, ydata = inertia.lines[0].get_xydata().T

for i in range(1, 11):
    k_means_t, cluster_t = cluster(reduced, random_state=check_random_state(0), n_clusters=i)
    assert_array_equal(xdata[i - 1], i)
    assert_almost_equal(ydata[i - 1], k_means_t.inertia_)

assert_is_not(len(inertia.title.get_text()), 0,
    msg="Your plot doesn't have a title.")
assert_is_not(inertia.xaxis.get_label_text(), '',
    msg="Change the x-axis label to something more descriptive.")
assert_is_not(inertia.yaxis.get_label_text(), '',
    msg="Change the y-axis label to something more descriptive.")
```

In [33]:

```python
def plot_pair(reduced, clusters):
    '''
    Uses seaborn.PairGrid to visualize the data distribution
    when axes are the first four principal components.
    Diagonal plots are histograms. The off-diagonal plots are scatter plots.

    Parameters
    ----------
    reduced: A numpy array. Comes from importing delta_reduced.npy

    Returns
    -------
    A seaborn.axisgrid.PairGrid instance.
    '''

    df = pd.DataFrame(reduced)
    df['c'] = clusters
    subset = [0,1,2,3, 'c']
    columns = [0,1,2,3]

    ax = sns.PairGrid(df[subset], vars = columns, hue = 'c')
    ax = ax.map_diag(plt.hist)
    ax = ax.map_offdiag(plt.scatter)

    return ax
```

In [34]:

```
k_means, clusters = cluster(reduced, random_state=check_random_state(0), n_clusters=4)
pg = plot_pair(reduced, clusters)
```



0
1
2
3

We observe that the one outlier is in its own cluster, there's 3 or 4 points in the other clusters and the remainder are split into two clusters of greater size.

In [35]:
```
assert_is_instance(pg.fig, plt.Figure)
assert_true(len(pg.data.columns) >= 4)

for ax in pg.diag_axes:
    assert_equal(len(ax.patches), 4 * 10) # 4 clusters with 10 patches in each histogram

for i, j in zip(*np.triu_indices_from(pg.axes, 1)):
    ax = pg.axes[i, j]
    x_out, y_out = ax.collections[0].get_offsets().T
    x_in = reduced[clusters == 0, j] # we only check the first cluster
    y_in = reduced[clusters == 0, i]
    assert_array_equal(x_in, x_out)
    assert_array_equal(y_in, y_out)

for i, j in zip(*np.tril_indices_from(pg.axes, -1)):
    ax = pg.axes[i, j]
    x_in = reduced[clusters == 0, j]
    y_in = reduced[clusters == 0, i]
    x_out, y_out = ax.collections[0].get_offsets().T
    assert_array_equal(x_in, x_out)
    assert_array_equal(y_in, y_out)

for i, j in zip(*np.diag_indices_from(pg.axes)):
    ax = pg.axes[i, j]
    assert_equal(len(ax.collections), 0)
```

Let's Continue our Analysis and brainstorm

You don't have to write any code in this section, but here's one interpretaion of what we have done.

Let's take a closer look at each cluster.

In [36]:

```python
df = pd.read_csv('delta.csv', index_col='Aircraft')
df['Clusters'] = clusters
df['Aircraft'] = df.index
df_grouped = df.groupby('Clusters').mean()
print(df_grouped.Accommodation)
```

```
Clusters
0    153.625000
1    244.733333
2     44.500000
3     54.000000
Name: Accommodation, dtype: float64
```

In [37]:

```python
print(df_grouped['Length (ft)'])
```

```
Clusters
0    137.048083
1    190.538400
2     84.810750
3    111.000000
Name: Length (ft), dtype: float64
```

Cluster 3 has only one aircraft:

In [38]:

```python
clust3 = df[df.Clusters == 3]
print(clust3.Aircraft)
```

```
Aircraft
Airbus A319 VIP    Airbus A319 VIP
Name: Aircraft, dtype: object
```

Airbus A319 VIP is not one of Delta Airline's regular fleet and is one of Airbus corporate jets.

Cluster 2 has four aircrafts.

In [39]:

```python
clust2 = df[df.Clusters == 2]
print(clust2.Aircraft)
```

```
Aircraft
CRJ 100/200 Pinnacle/SkyWest    CRJ 100/200 Pinnacle/SkyWest
CRJ 100/200 ExpressJet                CRJ 100/200 ExpressJet
E120                                                    E120
ERJ-145                                              ERJ-145
Name: Aircraft, dtype: object
```

These are small aircrafts and only have economy seats.

In [40]:

```python
cols_seat = ['First Class', 'Business', 'Eco Comfort', 'Economy']
print(df.loc[clust2.index, cols_seat])
```

```
                              First Class  Business  Eco Comfort  Economy
Aircraft
CRJ 100/200 Pinnacle/SkyWest            0         0            0        1
CRJ 100/200 ExpressJet                  0         0            0        1
E120                                    0         0            0        1
ERJ-145                                 0         0            0        1
```

In [41]:
```python
clust1 = df[df.Clusters == 1]
print(clust1.Aircraft)
```

```
Aircraft
Airbus A330-200                            Airbus A330-200
Airbus A330-200 (3L2)                Airbus A330-200 (3L2)
Airbus A330-200 (3L3)                Airbus A330-200 (3L3)
Airbus A330-300                            Airbus A330-300
Boeing 747-400 (74S)                  Boeing 747-400 (74S)
Boeing 757-200 (75E)                  Boeing 757-200 (75E)
Boeing 757-200 (75X)                  Boeing 757-200 (75X)
Boeing 767-300 (76G)                  Boeing 767-300 (76G)
Boeing 767-300 (76L)                  Boeing 767-300 (76L)
Boeing 767-300 (76T)                  Boeing 767-300 (76T)
Boeing 767-300 (76Z V.1)          Boeing 767-300 (76Z V.1)
Boeing 767-300 (76Z V.2)          Boeing 767-300 (76Z V.2)
Boeing 767-400 (76D)                  Boeing 767-400 (76D)
Boeing 777-200ER                          Boeing 777-200ER
Boeing 777-200LR                          Boeing 777-200LR
Name: Aircraft, dtype: object
```

Interesting, Cluster 1 aircrafts do not have first class seating.

In [42]:

```python
print(df.loc[clust1.index, cols_seat])
```

```
                          First Class  Business  Eco Comfort  Economy
Aircraft
Airbus A330-200                     0         1            1        1
Airbus A330-200 (3L2)               0         1            1        1
Airbus A330-200 (3L3)               0         1            1        1
Airbus A330-300                     0         1            1        1
Boeing 747-400 (74S)                0         1            1        1
Boeing 757-200 (75E)                0         1            1        1
Boeing 757-200 (75X)                0         1            1        1
Boeing 767-300 (76G)                0         1            1        1
Boeing 767-300 (76L)                0         1            1        1
Boeing 767-300 (76T)                0         1            1        1
Boeing 767-300 (76Z V.1)            0         1            1        1
Boeing 767-300 (76Z V.2)            0         1            1        1
Boeing 767-400 (76D)                0         1            1        1
Boeing 777-200ER                    0         1            1        1
Boeing 777-200LR                    0         1            1        1
```

In [43]:

```python
clust0 = df[df.Clusters == 0]
print(clust0.Aircraft)
```

```
Aircraft
Airbus A319                               Airbus A319
Airbus A320                               Airbus A320
Airbus A320 32-R                     Airbus A320 32-R
Boeing 717                                 Boeing 717
Boeing 737-700 (73W)            Boeing 737-700 (73W)
Boeing 737-800 (738)            Boeing 737-800 (738)
Boeing 737-800 (73H)            Boeing 737-800 (73H)
Boeing 737-900ER (739)        Boeing 737-900ER (739)
Boeing 757-200 (75A)            Boeing 757-200 (75A)
Boeing 757-200 (75M)            Boeing 757-200 (75M)
Boeing 757-200 (75N)            Boeing 757-200 (75N)
Boeing 757-200 (757)            Boeing 757-200 (757)
Boeing 757-200 (75V)            Boeing 757-200 (75V)
Boeing 757-300                         Boeing 757-300
Boeing 767-300 (76P)            Boeing 767-300 (76P)
Boeing 767-300 (76Q)            Boeing 767-300 (76Q)
Boeing 767-300 (76U)            Boeing 767-300 (76U)
CRJ 700                                       CRJ 700
CRJ 900                                       CRJ 900
E170                                             E170
E175                                             E175
MD-88                                           MD-88
MD-90                                           MD-90
MD-DC9-50                                   MD-DC9-50
Name: Aircraft, dtype: object
```

The aircrafts in cluster 0 (except for one aircraft) have first class seating but no business class.

In [44]:
```
print(df.loc[clust0.index, cols_seat])
```

```
                        First Class  Business  Eco Comfort  Economy
Aircraft
Airbus A319                       1         0            1        1
Airbus A320                       1         0            1        1
Airbus A320 32-R                  1         0            1        1
Boeing 717                        1         0            1        1
Boeing 737-700 (73W)              1         0            1        1
Boeing 737-800 (738)              1         0            1        1
Boeing 737-800 (73H)              1         0            1        1
Boeing 737-900ER (739)            1         0            1        1
Boeing 757-200 (75A)              1         0            1        1
Boeing 757-200 (75M)              1         0            1        1
Boeing 757-200 (75N)              1         0            1        1
Boeing 757-200 (757)              1         0            1        1
Boeing 757-200 (75V)              1         0            1        1
Boeing 757-300                    1         0            1        1
Boeing 767-300 (76P)              1         0            1        1
Boeing 767-300 (76Q)              1         0            1        1
Boeing 767-300 (76U)              0         1            1        1
CRJ 700                           1         0            1        1
CRJ 900                           1         0            1        1
E170                              1         0            1        1
E175                              1         0            1        1
MD-88                             1         0            1        1
MD-90                             1         0            1        1
MD-DC9-50                         1         0            1        1
```

Problem 3

(No Unit Tests in this portion)

Run DBSCAN on Iris.csv and compare/discuss the results with K-Means. Please submit your code and output, and write down 3-4 sentences that you observed from the results.

Run DBSCAN on Reduced_Delta dataset and compare/discuss the results with K-Means. Please submit your code and output, and write down 3-4 sentences that you observed from the results.

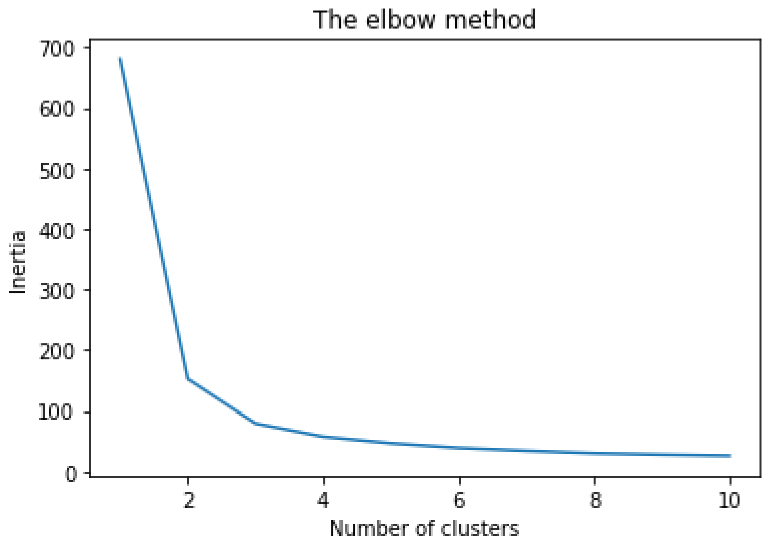
Run KMeans on movements.csv compare/discuss the results with DBSCAN and Hierarchical Clustering (Agglomerative). Please submit your code and output, and write down 3-4 sentences that you observed from the results

In [87]:
```
from sklearn.neighbors import NearestNeighbors
from sklearn.cluster import DBSCAN
from sklearn.preprocessing import Normalizer
from sklearn.cluster import AgglomerativeClustering
from scipy.cluster.hierarchy import linkage, dendrogram
from scipy.cluster.hierarchy import fcluster
```

k means - iris

In [46]:
```
iris = pd.read_csv("iris.csv").iloc[:,1:5]
iris_array = np.array(iris)
```

In [47]:
```
plot_inertia(iris_array, start=1, end=10);
```

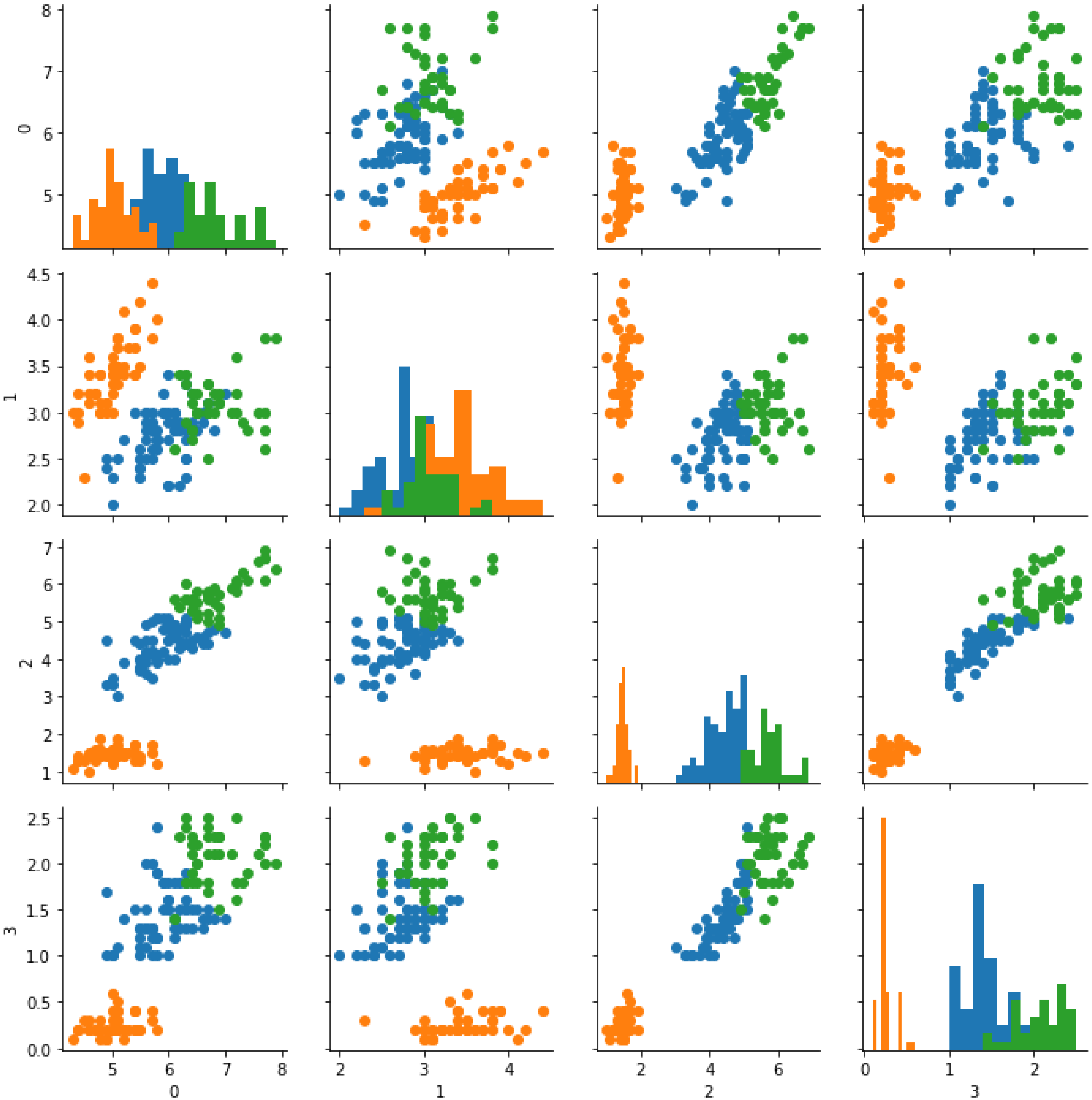


In [48]:

```
model,clusters  = cluster((iris_array), random_state=0, n_clusters=3)
```

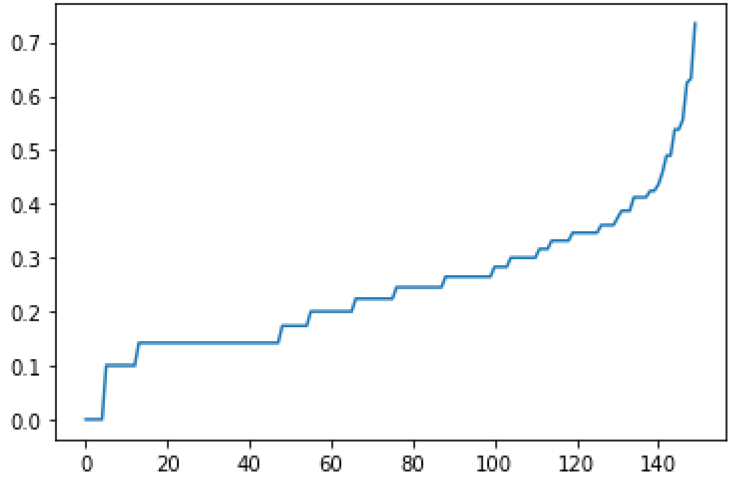
In [49]:

```
plot_pair(iris_array, clusters);
```


0
1
2
3

dbscan - iris

In [50]:
```
nbrs = NearestNeighbors(n_neighbors=len(iris_array)).fit(iris_array)
distances, indices = nbrs.kneighbors(iris_array)

distances = np.sort(distances, axis=0)
distances = distances[:,1]
plt.figure(figsize=(6,4));
plt.plot(distances);
```

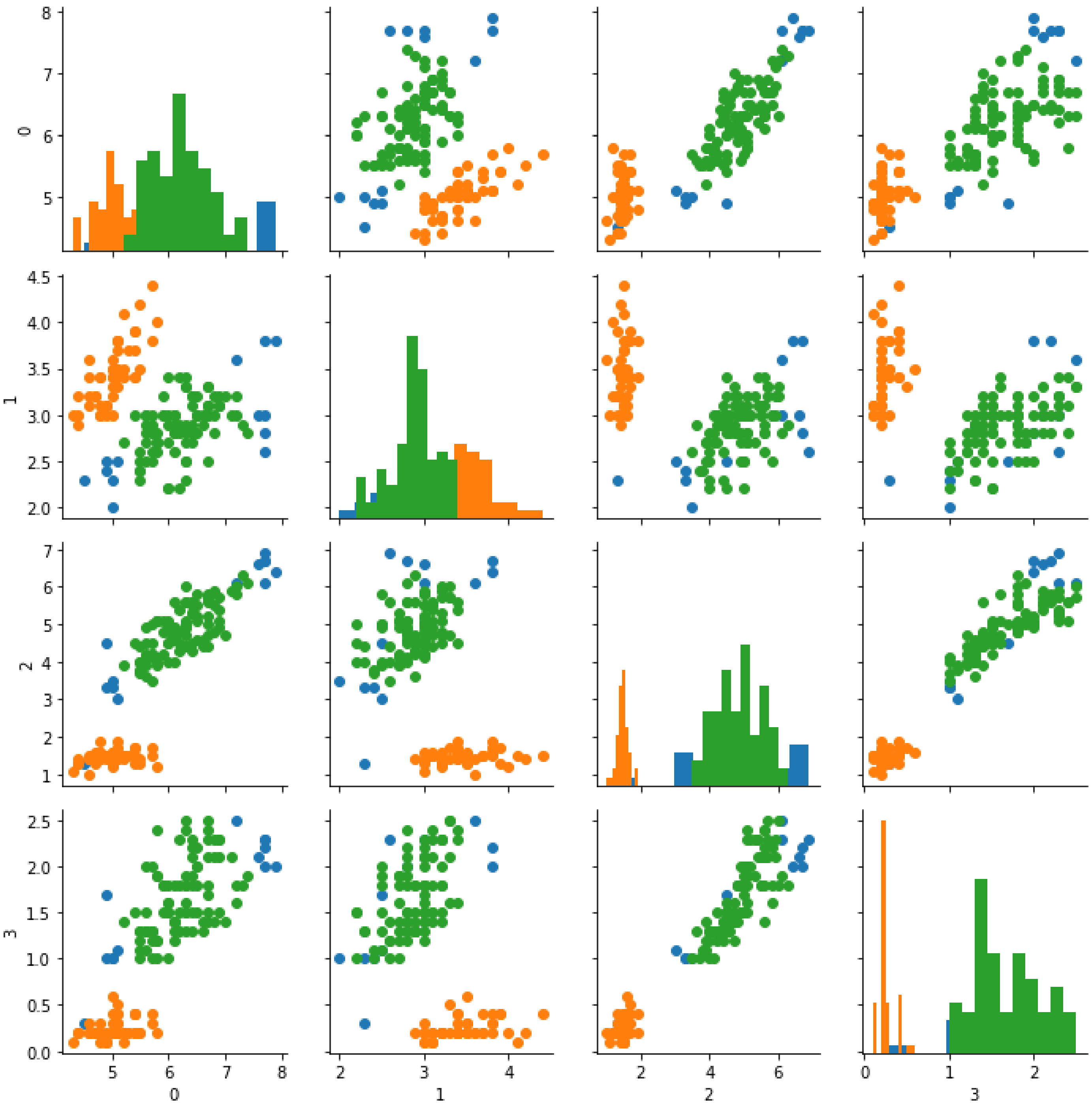


In [51]:
```python
# first lock eps=5 based on KNN dist crook, then adjust min_samples
# thumb of rule of min_sample is > dim, close 2 * dim.
dbsc = DBSCAN(eps = 0.6, metric='euclidean', min_samples=8).fit(iris_array)
clusters = dbsc.labels_
clusters
```

Out[51]:
```
array([ 0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
        0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
        0,  0,  0,  0,  0,  0,  0, -1,  0,  0,  0,  0,  0,  0,  0,  0,  1,
        1,  1,  1,  1,  1,  1, -1,  1,  1, -1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1, -1,  1,  1,  1,  1, -1,  1,  1,  1,
        1,  1,  1, -1, -1,  1,  1, -1,  1,  1,  1,  1,  1,  1,  1, -1, -1,
        1,  1,  1, -1,  1,  1,  1,  1,  1,  1,  1,  1, -1,  1,  1,  1, -1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1],
      dtype=int64)
```

In [52]:

```
plot_pair(iris_array, clusters);
```



0
1
2
3

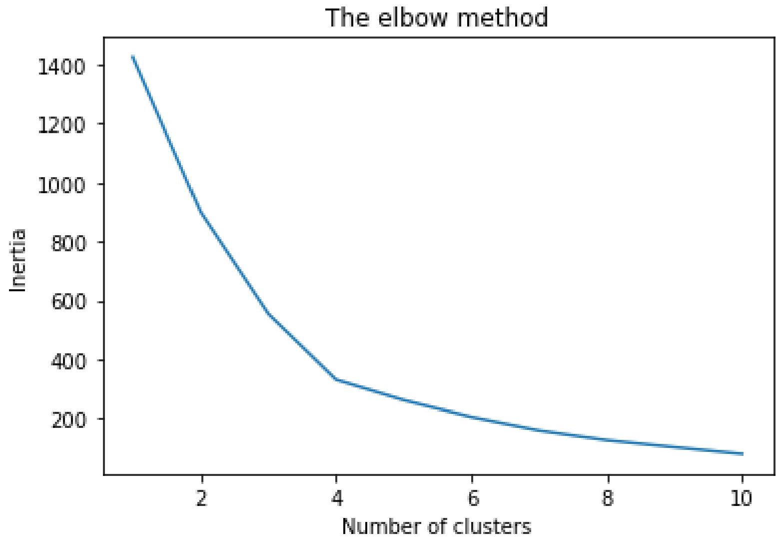
Observation for iris

In this iris dataset, two models show a different way of clustering. We will find K-means breaks the big cloud into two parts, while DBSCAN considered it as a whole except for some outliers. The pattern differences are due to their algorithm differences. DBSCAN is a density-based algorithm using the concept of reachability i.e. how many neighbors has a point within a radius, and it seems to correspond more to human intuitions of clustering in this case, however, by checking and comparing with the actual label, the K-means are actually closer to the actual.

kmeans - Reduced_Delta

In [53]:
```
reduced = np.load('delta_reduced.npy')
```

In [54]:
```
plot_inertia(reduced, start=1, end=10);
```

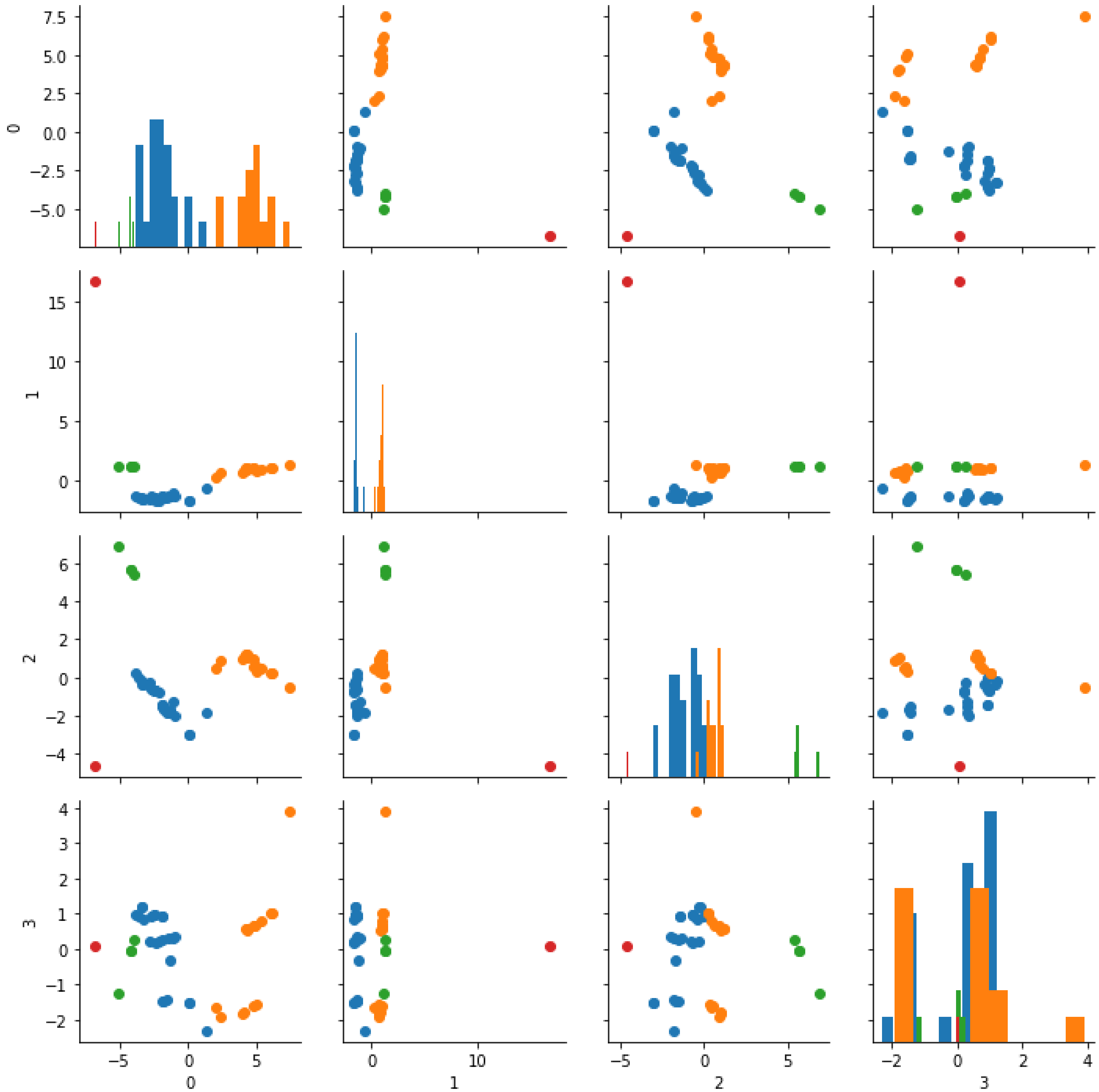


In [55]:
```python
model,clusters = cluster(np.array(reduced), random_state=0, n_clusters=4)
clusters
```

Out[55]:
```
array([0, 3, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0,
       1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 2, 2, 0, 0, 2, 0, 0, 2, 0, 0, 0])
```

In [56]:

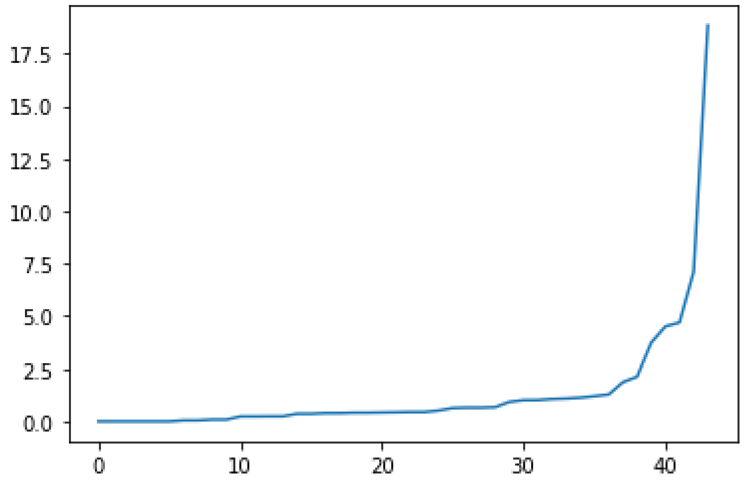
```
plot_pair(reduced, clusters);
```



0
1
2
3
0
1
2
3

dbscan - Reduced_Delta

In [57]:
```
nbrs = NearestNeighbors(n_neighbors=len(reduced)).fit(reduced)
distances, indices = nbrs.kneighbors(reduced)

distances = np.sort(distances, axis=0)
distances = distances[:,1]
plt.figure(figsize=(6,4));
plt.plot(distances);
```

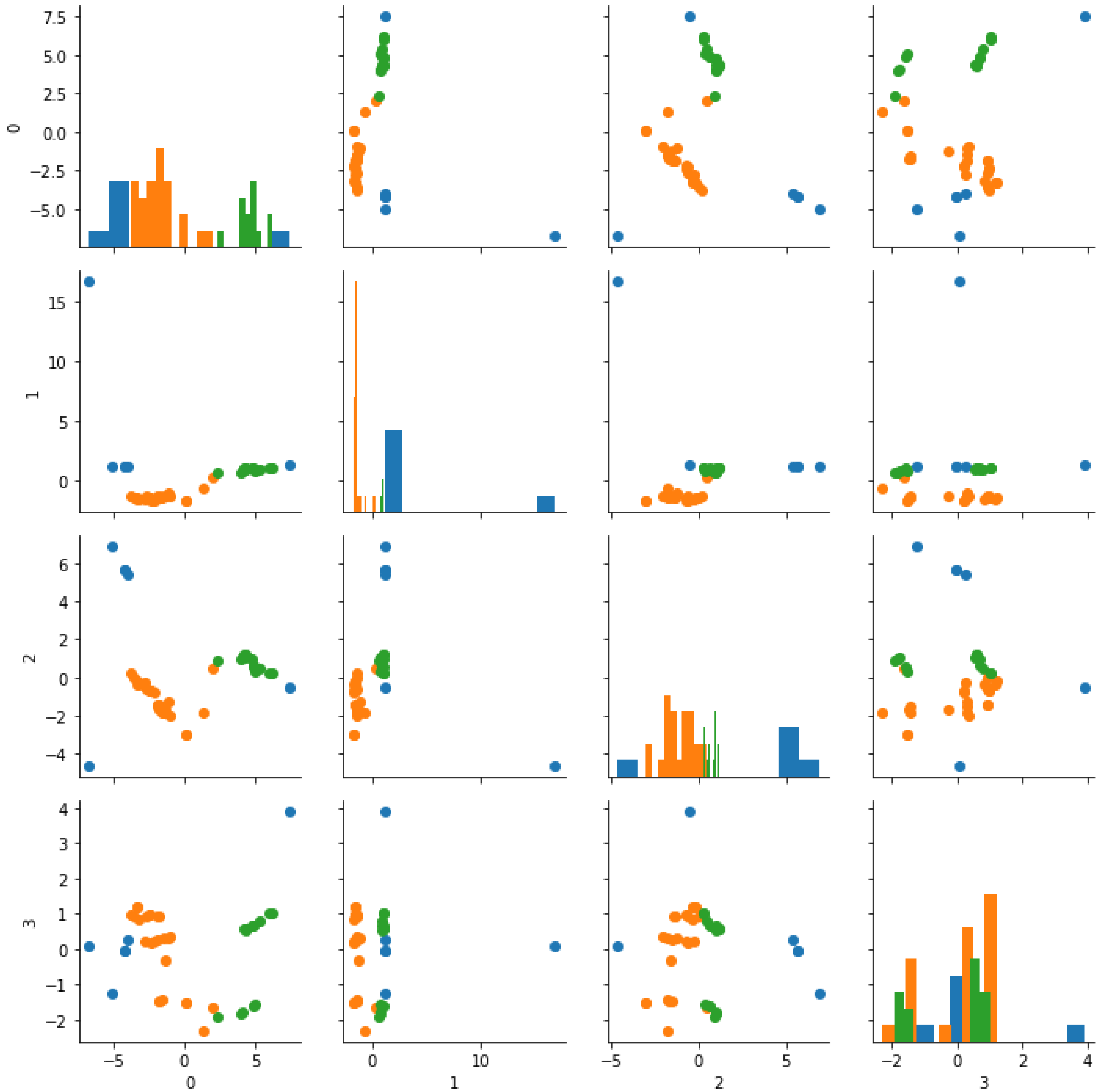


In [58]:
```
# first lock eps=5 based on KNN dist crook, then adjust min_samples
# thumb of rule of min_sample is > dim, close 2 * dim.

dbsc = DBSCAN(eps = 5, metric='euclidean', min_samples=14).fit(reduced)
clusters = dbsc.labels_
clusters
```

Out[58]:
```
array([ 0, -1,  0,  0,  1,  1,  1,  1,  0,  0,  0,  0,  0, -1,  0,  1,  0,
        0,  0,  0,  0,  0,  1,  1,  0,  0,  1,  0,  1,  1,  1,  1,  1, -1,
       -1,  0,  0, -1,  0,  0, -1,  0,  0,  0], dtype=int64)
```

In [59]:

```
plot_pair(reduced, clusters);
```



0
1
2
3

Observation for Reduced_Delta

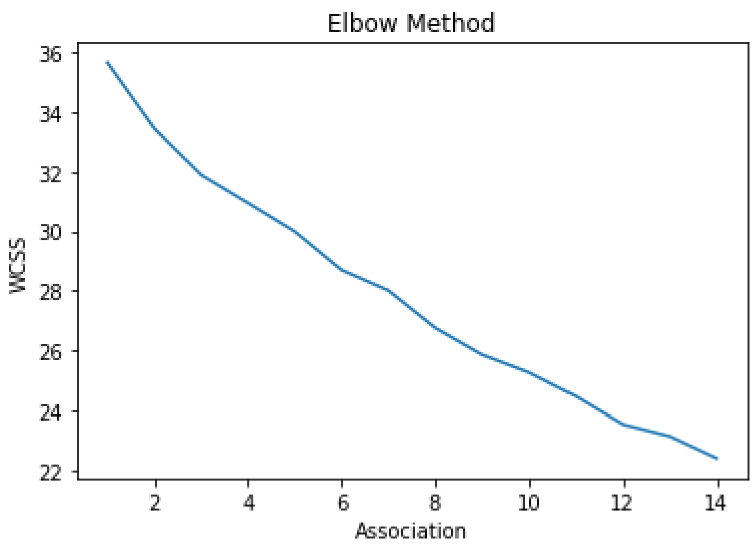
In this reduced_delta dataset, we find two models' results have very close similarity. I guess it's because these clusters are far apart. The diameter (the largest distance between any two points) is far smaller than the shortest distance separating the sets (the smallest distance). However, difference is that K means treats the minorities as another cluster(red point), but DBSCAN treat as outliers.

kmeans - movements

In [60]:
```
movements = pd.read_csv("movements.csv", header=None, index_col=False, skiprows=1, usecols=range(1,964))
movements_array = np.array(movements)
normalized_move = Normalizer().fit_transform(movements_array)
```

In [61]:

```
from sklearn.cluster import KMeans
wcss = []

for i in range(1, 15):
    kmeans = KMeans(n_clusters = i, init = 'k-means++',
                    max_iter = 400, n_init = 10, random_state = 0)
    kmeans.fit(normalized_move)
    wcss.append(kmeans.inertia_)

plt.plot(range(1, 15), wcss)
plt.title('Elbow Method')
plt.xlabel('Association')
plt.ylabel('WCSS')
plt.show()

# I think we should choose cluster = 12, but reference notebook gives 10 cluster.
```



In [62]:

```
companies=pd.read_csv("movements.csv", header=None, index_col=False, skiprows=1, usecols=[0])
companies=companies[0].values
```

In [63]:

```
# I think we should choose cluster = 12, but reference notebook gives 10 cluster.
model,clusters = cluster(normalized_move, random_state=0, n_clusters=10)
clusters
```

Out[63]:

```
array([9, 8, 9, 8, 2, 8, 7, 5, 0, 6, 0, 5, 0, 0, 1, 5, 3, 9, 8, 7, 3, 5,
       5, 3, 1, 6, 8, 6, 4, 2, 0, 7, 3, 5, 5, 0, 2, 7, 4, 7, 6, 6, 7, 7,
       0, 5, 7, 1, 5, 7, 1, 1, 7, 0, 2, 8, 6, 0, 3, 1])
```

In [64]:
```
df = pd.DataFrame({'labels': clusters, 'companies': companies})

# Display df sorted by cluster label
print(df.sort_values('labels'))
```

```
    labels                           companies
8        0                         Caterpillar
57       0                               Exxon
53       0                       Valero Energy
13       0                  DuPont de Nemours
12       0                             Chevron
35       0                            Navistar
44       0                        Schlumberger
30       0                          MasterCard
10       0                      ConocoPhillips
14       1                                Dell
59       1                               Yahoo
47       1                            Symantec
50       1  Taiwan Semiconductor Manufacturing
51       1                   Texas instruments
24       1                               Intel
54       2                            Walgreen
36       2                    Northrop Grumman
29       2                     Lookheed Martin
4        2                              Boeing
20       3                          Home Depot
23       3                                 IBM
16       3                   General Electrics
58       3                               Xerox
32       3                                  3M
38       4                               Pepsi
28       4                           Coca Cola
48       5                              Toyota
45       5                                Sony
7        5                               Canon
34       5                          Mitsubishi
33       5                           Microsoft
11       5                               Cisco
15       5                                Ford
21       5                               Honda
```

22	5	HP
27	6	Kimberly-Clark
25	6	Johnson & Johnson
9	6	Colgate-Palmolive
56	6	Wal-Mart
40	6	Procter Gamble
41	6	Philip Morris
49	7	Total
19	7	GlaxoSmithKline
6	7	British American Tobacco
46	7	Sanofi-Aventis
42	7	Royal Dutch Shell
31	7	McDonalds
39	7	Pfizer
37	7	Novartis
43	7	SAP
52	7	Unilever
1	8	AIG
55	8	Wells Fargo
18	8	Goldman Sachs
5	8	Bank of America
26	8	JPMorgan Chase
3	8	American express
2	9	Amazon
17	9	Google/Alphabet
0	9	Apple

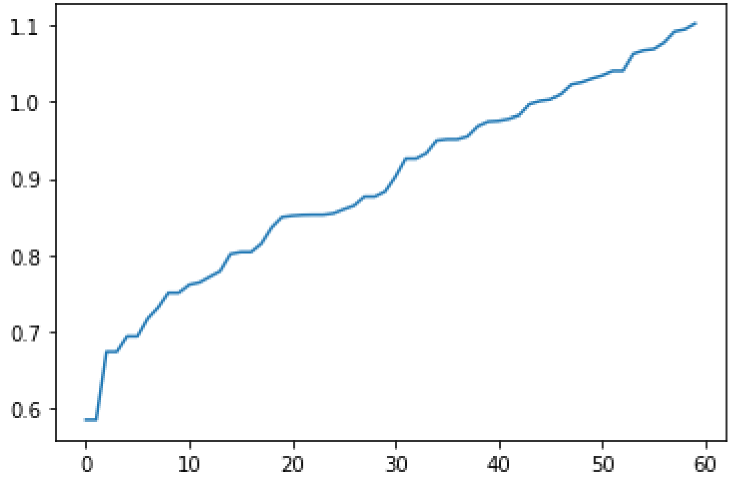
In [65]:

```
plot_pair(movements_array, clusters);
```

dbscan - movements

In [66]:

```
from sklearn.neighbors import NearestNeighbors
nbrs = NearestNeighbors(n_neighbors=len(normalized_move)).fit(normalized_move)
distances, indices = nbrs.kneighbors(normalized_move)

distances = np.sort(distances, axis=0)
distances = distances[:,1]
plt.figure(figsize=(6,4));
plt.plot(distances);
```



In [67]:

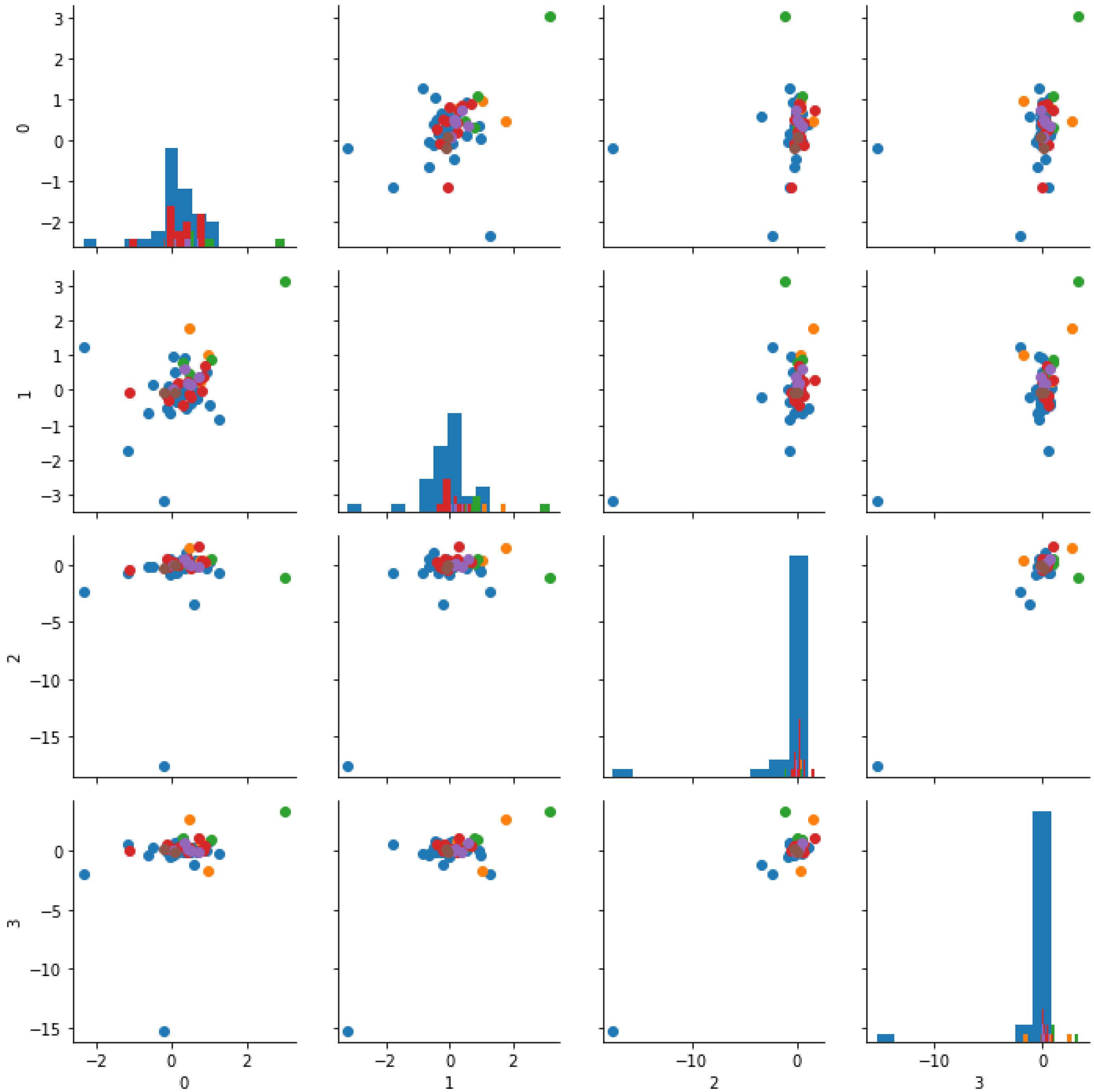
```
# DBSCAN doesn't work well for time series data, it's sensitive to parameter, difficult to get a 10 cluster I want.

dbsc = DBSCAN(eps = 0.88, metric='euclidean', min_samples=2).fit(normalized_move)
clusters = dbsc.labels_
clusters
```

Out[67]:

```
array([-1, -1, -1, -1,  0,  1,  2,  3,  2, -1,  2, -1,  2,  2, -1, -1, -1,
       -1,  1,  2, -1,  3, -1, -1,  4, -1,  1, -1, -1,  0, -1, -1,  2, -1,
        3, -1,  0,  2, -1, -1, -1, -1,  2,  2,  2,  3,  2, -1,  3,  2, -1,
        4,  2, -1, -1,  1, -1,  2, -1, -1], dtype=int64)
```

In [68]:

```
plot_pair(movements_array, clusters);
```



0
1
2
3
0
1
2
3

Agglomerative - movements

In [89]:
```python
Agglomerative = AgglomerativeClustering(n_clusters=10, affinity='euclidean', linkage='ward').fit(normalized_move)
clusters = Agglomerative.labels_
clusters
```

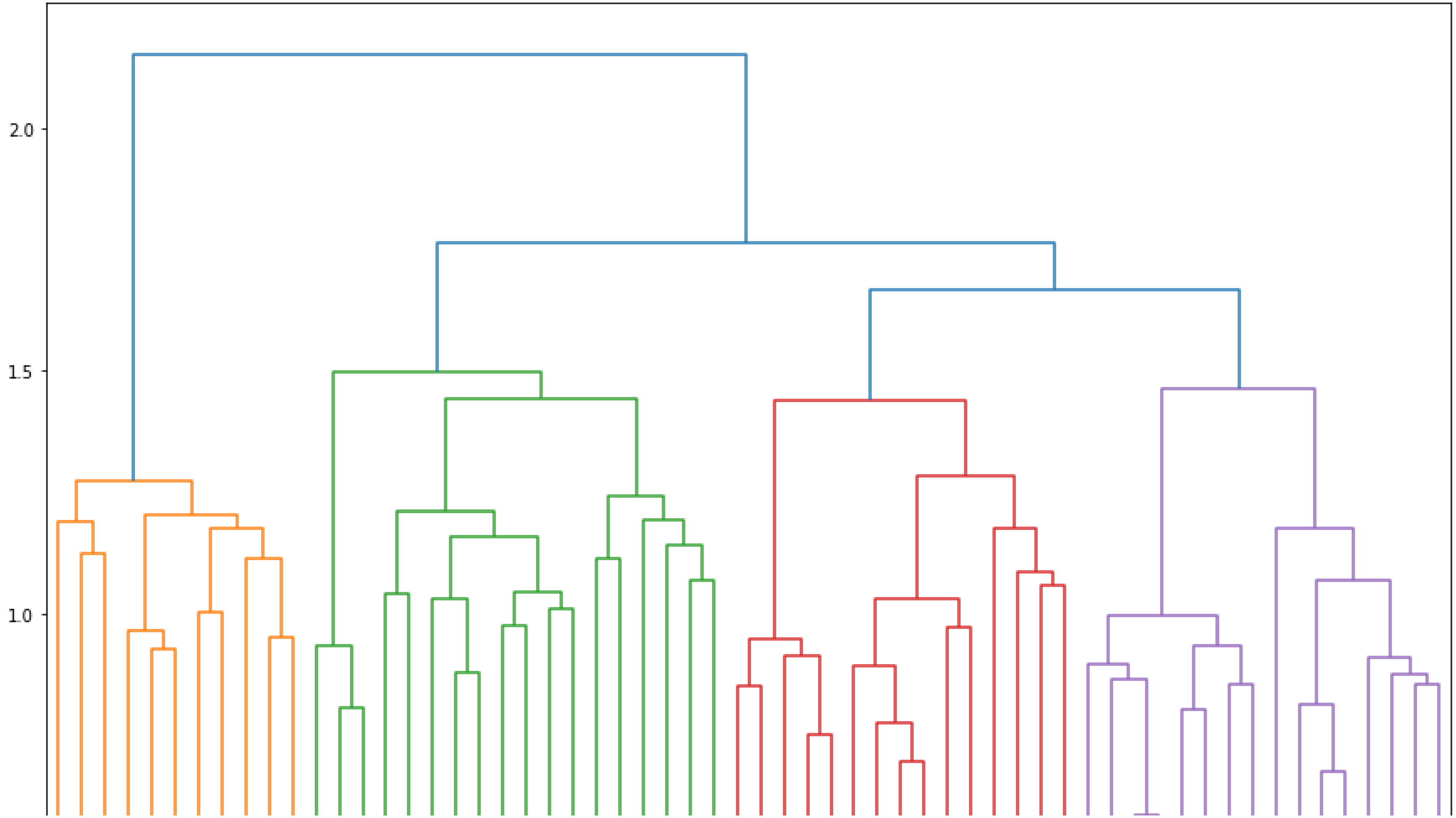
Out[89]:
```
array([0, 3, 0, 8, 9, 8, 5, 7, 6, 1, 6, 2, 6, 6, 2, 3, 8, 0, 8, 5, 4, 7,
       2, 2, 2, 1, 8, 1, 1, 9, 0, 0, 6, 2, 7, 3, 9, 5, 1, 1, 1, 1, 5, 5,
       6, 7, 5, 2, 7, 5, 2, 2, 5, 6, 4, 8, 4, 6, 3, 0], dtype=int64)
```

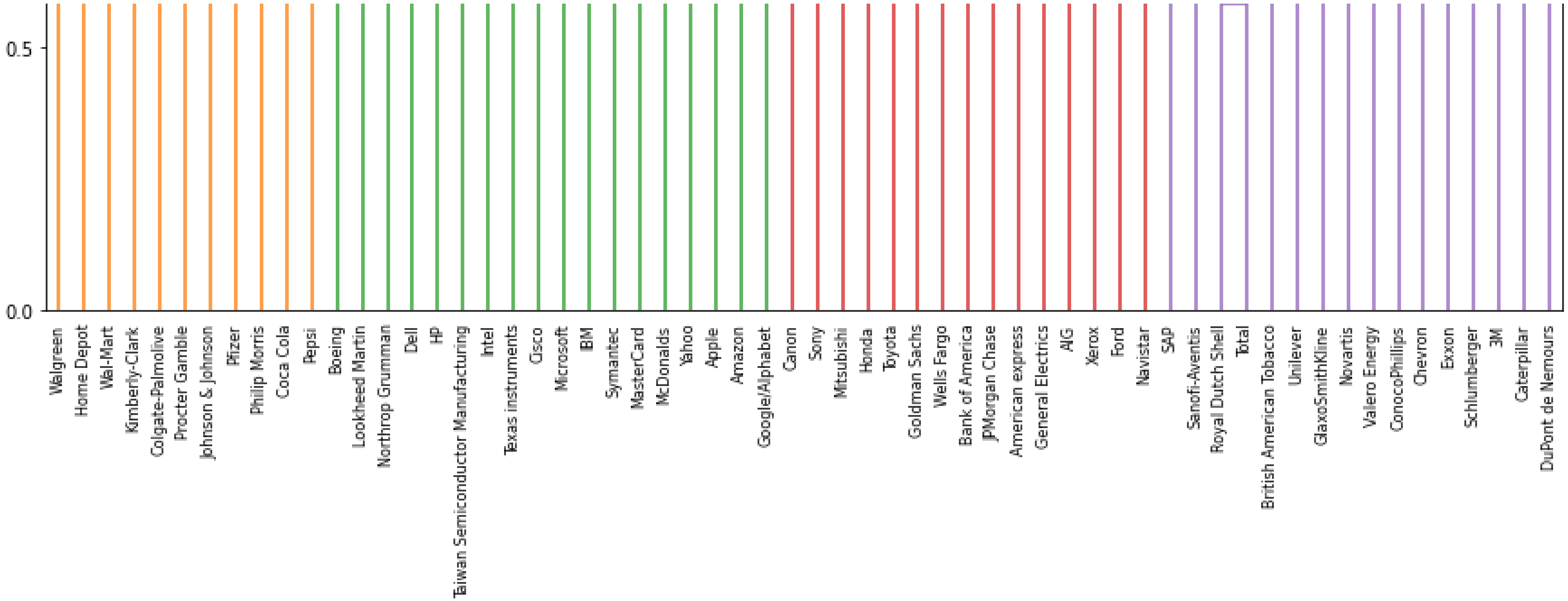
In [90]:
```python
df = pd.DataFrame({'labels': clusters, 'companies': companies})

# Display df sorted by cluster label
print(df.sort_values('labels'))
```

```
    labels                           companies
0        0                               Apple
31       0                           McDonalds
30       0                          MasterCard
17       0                      Google/Alphabet
59       0                               Yahoo
2        0                              Amazon
28       1                           Coca Cola
41       1                      Philip Morris
40       1                      Procter Gamble
9        1                   Colgate-Palmolive
39       1                              Pfizer
38       1                               Pepsi
27       1                      Kimberly-Clark
25       1                   Johnson & Johnson
47       2                            Symantec
33       2                           Microsoft
50       2  Taiwan Semiconductor Manufacturing
24       2                               Intel
23       2                                 IBM
22       2                                  HP
51       2                   Texas instruments
14       2                                Dell
11       2                               Cisco
35       3                            Navistar
15       3                                Ford
58       3                               Xerox
1        3                                 AIG
20       4                          Home Depot
56       4                            Wal-Mart
54       4                            Walgreen
52       5                            Unilever
37       5                            Novartis
42       5                   Royal Dutch Shell
46       5                      Sanofi-Aventis
```

6	5	British American Tobacco
49	5	Total
19	5	GlaxoSmithKline
43	5	SAP
44	6	Schlumberger
8	6	Caterpillar
32	6	3M
12	6	Chevron
53	6	Valero Energy
13	6	DuPont de Nemours
57	6	Exxon
10	6	ConocoPhillips
7	7	Canon
34	7	Mitsubishi
45	7	Sony
48	7	Toyota
21	7	Honda
55	8	Wells Fargo
5	8	Bank of America
26	8	JPMorgan Chase
18	8	Goldman Sachs
3	8	American express
16	8	General Electrics
4	9	Boeing
36	9	Northrop Grumman
29	9	Lookheed Martin

In [91]:
```
plt.figure(figsize=(15,12))

mergings = linkage(normalized_move, method = 'ward')
dendrogram(
    mergings,
    labels=companies,
    leaf_rotation=90.,
    leaf_font_size=8
)

plt.show()
```

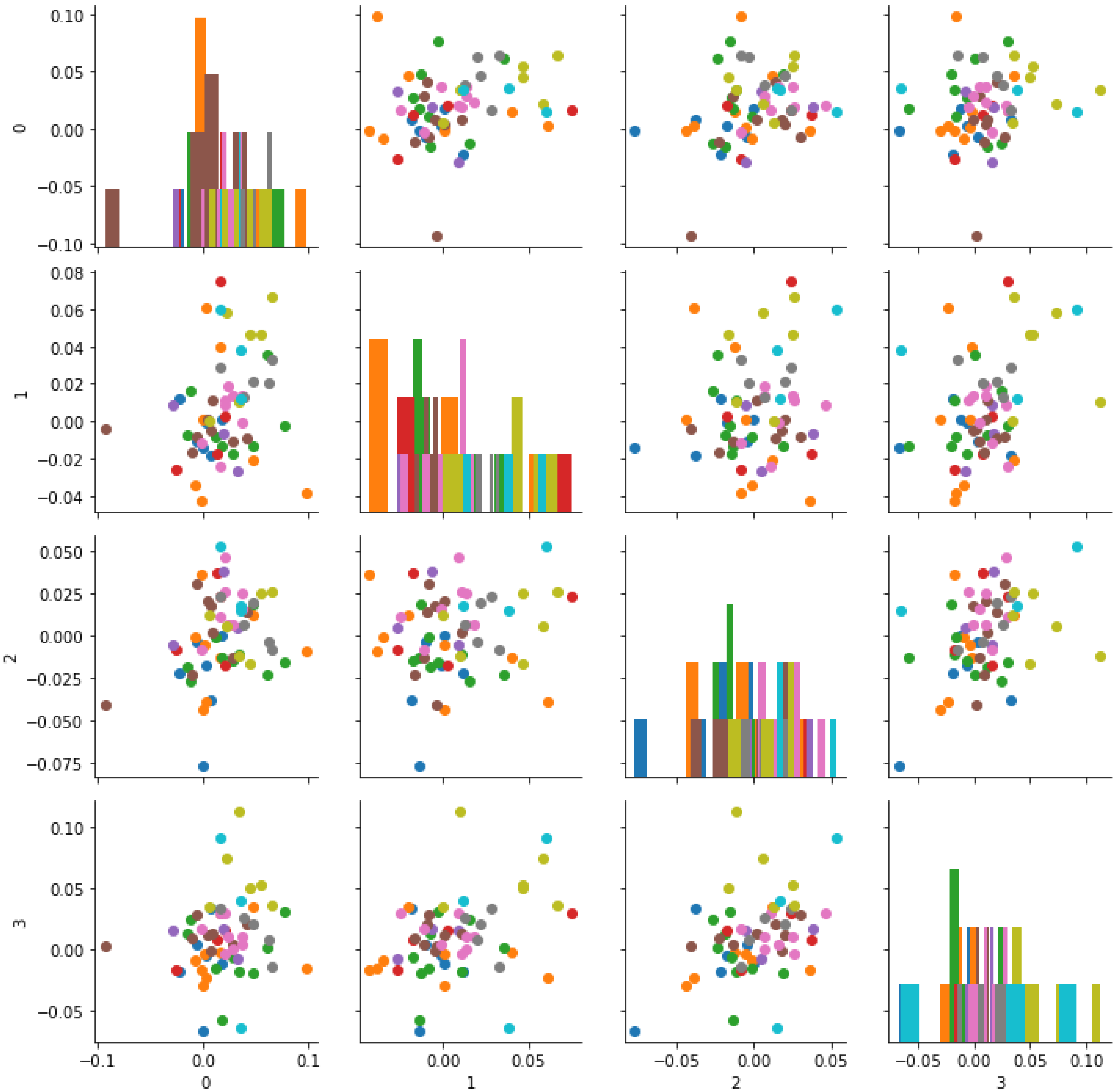




0.5
0.0
Walgreen
Home Depot
Wal-Mart
Kimberly-Clark
Colgate-Palmolive
Procter Gamble
Johnson & Johnson
Pfizer
Philip Morris
Coca Cola
Pepsi
Boeing
Lookheed Martin
Northrop Grumman
Dell
HP
Taiwan Semiconductor Manufacturing
Intel
Texas instruments
Cisco
Microsoft
IBM
Symantec
MasterCard
McDonalds
Yahoo
Apple
Amazon
Google/Alphabet
Canon
Sony
Mitsubishi
Honda
Toyota
Goldman Sachs
Wells Fargo
Bank of America
JPMorgan Chase
American express
General Electrics
AIG
Xerox
Ford
Navistar
SAP
Sanofi-Aventis
Royal Dutch Shell
Total
British American Tobacco
Unilever
GlaxoSmithKline
Novartis
Valero Energy
ConocoPhillips
Chevron
Exxon
Schlumberger
3M
Caterpillar
DuPont de Nemours

In [93]:

```
plot_pair(normalized_move, clusters);
```



0
1
2
3
0.10
0.05
0.00
−0.05
−0.10
0.08
0.06
0.04
0.02
0.00
−0.02
−0.04
0.050
0.025
0.000
−0.025
−0.050
−0.075
0.10
0.05
0.00
−0.05
−0.1
0.0
0.1
0.00
0.05
−0.05
0.00
0.05
−0.05
0.00
0.05
0.10
0
1
2
3

Observation for Reduced_Delta

For such a high dimension dataset, it's hardly possible for K-Means and DBSCAN to visualize the cluster pattern in a 2d scatterplot. The data points will be in a mess and overlapping together. However, in Agglomerative clustering, the dendrogram makes it possible for us to visualize. The other observation is that K-Means works well for breaking into 10+ categories in high dimensional time series dataset, while DBSCAN will be very sensitive to parameter in this case (easy zero clusters, all noise data), so it's very difficult for us to adjust the eplison and minpoints to get a 10+ categoreis if we want. It would be a big problem if we have many features but few data points.

Problem 4

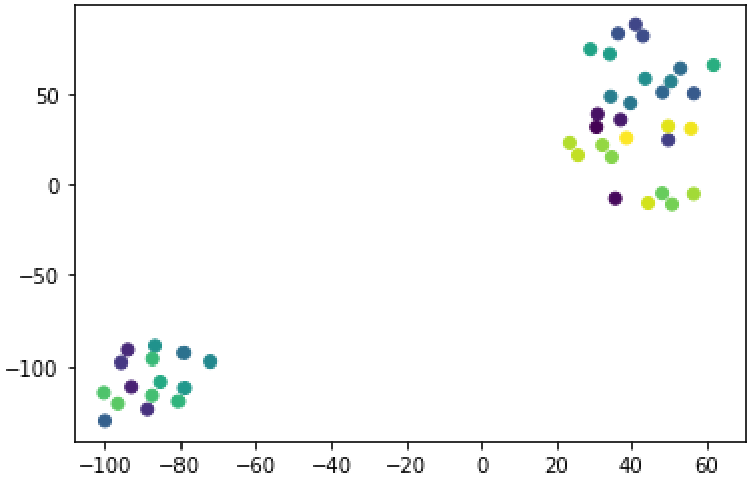
Apply t-SNE reduction to delta.csv file and compare/discuss the results with PCA. Please submit your code and output, and write down 3-4 sentences that you observed from the results.

In [81]:

```
from sklearn.manifold import TSNE
delta = pd.read_csv('delta.csv', index_col='Aircraft')
aircraft_ = delta.reset_index().iloc[:,[0]]['Aircraft'].astype('category').cat.codes
craft = list(delta.reset_index()["Aircraft"].astype('category'))
```

In [274]:

```
model_dt = TSNE(learning_rate=280,perplexity = 10,random_state=0)
Tsne_transformed = model_dt.fit_transform(scaled)

xs = Tsne_transformed[:,0]
ys = Tsne_transformed[:,1]
plt.scatter(xs, ys, c = aircraft_)
plt.show()
```

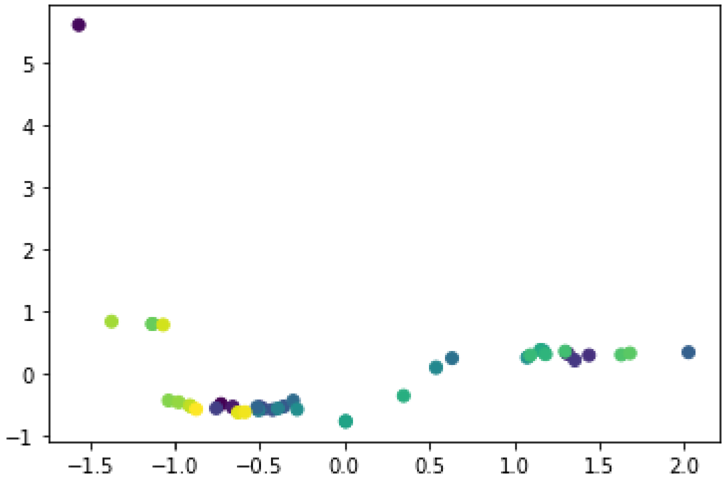


In [283]:

```
# Import TSNE
from sklearn.manifold import TSNE


plt.figure(figsize=(28,23))


# Create a TSNE instance: model
model = TSNE(learning_rate=280,perplexity = 10,random_state=0)

# Apply fit_transform to normalized_movements: tsne_features
tsne_features = model.fit_transform(scaled)

# Select the 0th feature: xs
xs = tsne_features[:,0]

# Select the 1th feature: ys
ys = tsne_features[:,1]

# Scatter plot
plt.scatter(xs, ys, alpha=0.5)
# Annotate the points
for x, y, company in zip(xs, ys, craft):
    plt.annotate(company, (x, y), fontsize=15, alpha=0.75)

plt.show()
```

In [198]:

```
## PCA 2-dim
X = scaled[:,:-1]
y = scaled[:,32]
model_PCA = PCA(n_components=2, whiten=True,random_state=0)
model_PCA_transformed= model_PCA.fit_transform(X)
xs = model_PCA_transformed[:,0]
ys = model_PCA_transformed[:,1]
plt.scatter(xs, ys,c = aircraft_);
```



Observation PCA vs tSNE

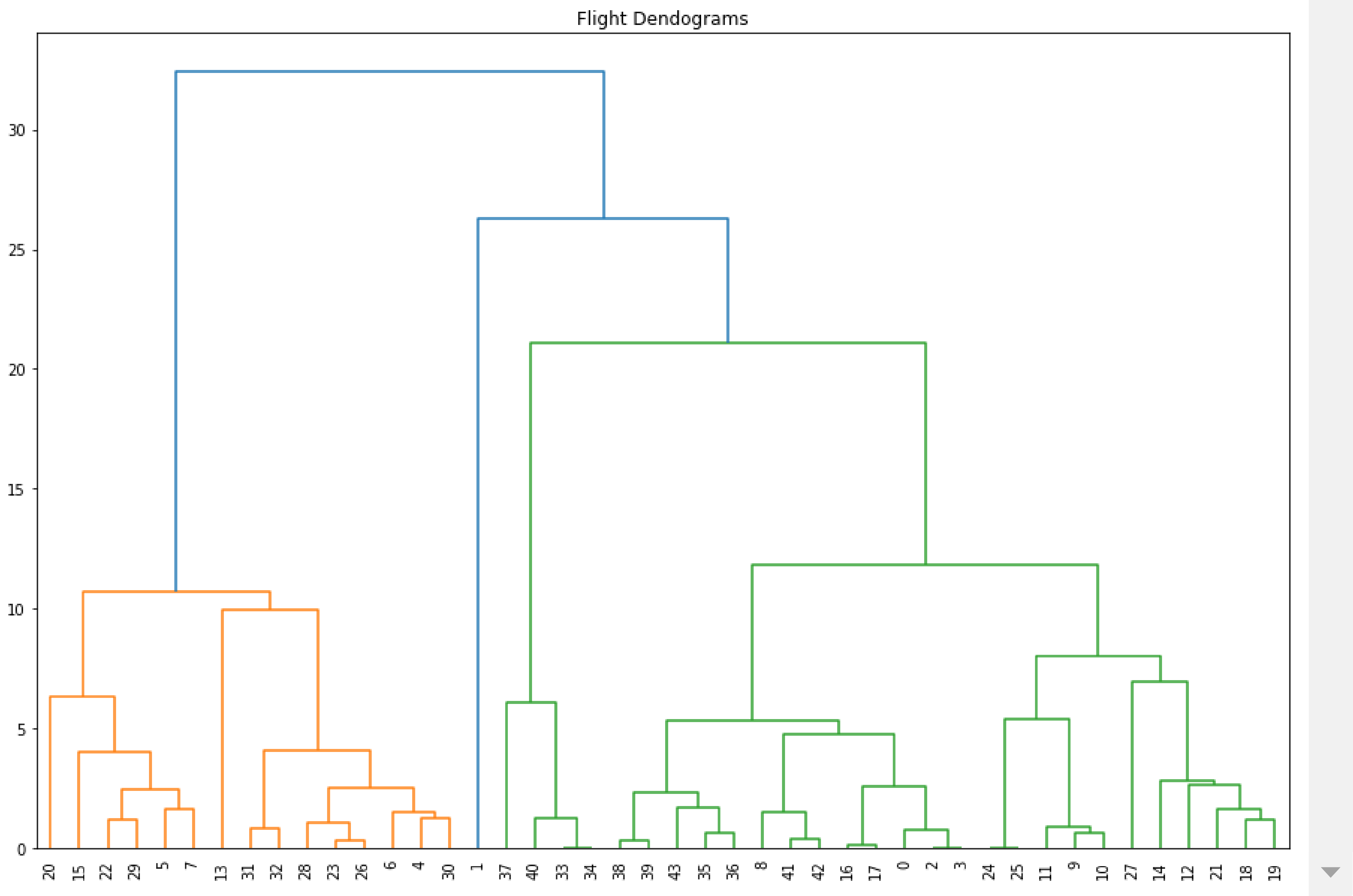
From comparing the scatter plot, we find tSNE's performance of clustering is much better than PCA for this 33 dim dataset. For PCA, it's not very intuitive to visualize two clusters in 2d scatter plot.

Problem 5 (Bonus)

Apply Hiearchical Clustering to delta.csv and observe how physical features are being clustered in ealry leaves at the bottom. Please submit your code and dendrogram graph along with 1-2 sentences interpretation.

In [280]:

```
plt.figure(figsize=(15, 10))
plt.title("Flight Dendograms")
mergings = linkage(scaled, method = 'ward')

labels = fcluster(mergings, 70, criterion = 'distance')
dendrogram(mergings,
           leaf_rotation=90,
           leaf_font_size=10,
)
plt.show()
```



We can see it has been divided to three clusters. and the green cluster occupies a large proportion.