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_e
from pandas.testing import assert_frame_equal

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

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.

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]:

	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)	...
Aircraft											
Airbus A319	0.0	0	0	21.0	36.0	12	0.0	0.0	0	17.2	...
Airbus A319 VIP	19.4	44	12	19.4	40.0	28	21.0	59.0	14	0.0	...
Airbus A320	0.0	0	0	21.0	36.0	12	0.0	0.0	0	17.2	...
Airbus A320 32-R	0.0	0	0	21.0	36.0	12	0.0	0.0	0	17.2	...
Airbus A330-200	0.0	0	0	0.0	0.0	0	21.0	60.0	32	18.0	...

5 rows × 33 columns

In [4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 44 entries, Airbus A319 to MD-DC9-50
Data columns (total 33 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   Seat Width (Club)          44 non-null     float64
 1   Seat Pitch (Club)          44 non-null     int64
 2   Seat (Club)                44 non-null     int64
 3   Seat Width (First Class)   44 non-null     float64
 4   Seat Pitch (First Class)   44 non-null     float64
 5   Seats (First Class)        44 non-null     int64
 6   Seat Width (Business)      44 non-null     float64
 7   Seat Pitch (Business)      44 non-null     float64
 8   Seats (Business)           44 non-null     int64
 9   Seat Width (Eco Comfort)   44 non-null     float64
 10  Seat Pitch (Eco Comfort)   44 non-null     float64
 11  Seats (Eco Comfort)        44 non-null     int64
 12  Seat Width (Economy)       44 non-null     float64
 13  Seat Pitch (Economy)       44 non-null     float64
 14  Seats (Economy)            44 non-null     int64
 15  Accommodation              44 non-null     int64
 16  Cruising Speed (mph)       44 non-null     int64
 17  Range (miles)              44 non-null     int64
 18  Engines                    44 non-null     int64
 19  Wingspan (ft)              44 non-null     float64
 20  Tail Height (ft)           44 non-null     float64
```

```
 21   Length (ft)              44 non-null     float64
 22   Wifi                     44 non-null     int64
 23   Video                    44 non-null     int64
 24   Power                    44 non-null     int64
 25   Satellite                44 non-null     int64
 26   Flat-bed                 44 non-null     int64
 27   Sleeper                  44 non-null     int64
 28   Club                     44 non-null     int64
 29   First Class              44 non-null     int64
 30   Business                 44 non-null     int64
 31   Eco Comfort              44 non-null     int64
 32   Economy                  44 non-null     int64
dtypes: float64(12), int64(21)
memory usage: 11.7+ KB
```

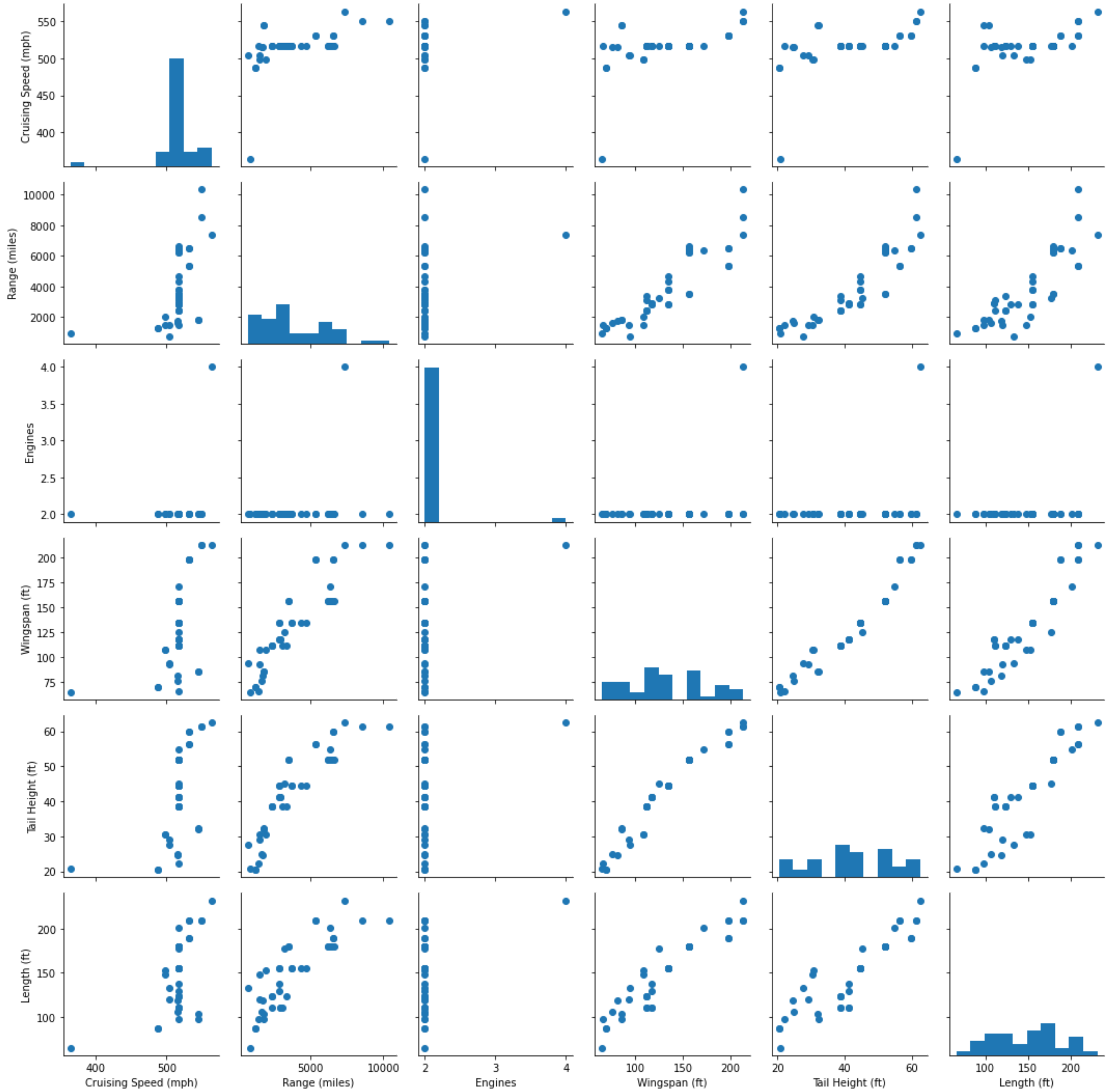
First, let's look at the attributes related to the aircraft physical characteristics:

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 [5]:
```python
def plot_pairgrid(df):
    '''
    Uses seaborn.PairGrid to visualize the attributes related to the six physical charac
    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.
    '''

    # YOUR CODE HERE
    # cols = ['Cruising Speed (mph)', 'Range (miles)', 'Engines', 'Wingspan (ft)', 'Tail

    ax = sns.PairGrid(df.iloc[:, df.columns.get_loc("Cruising Speed (mph)") : df.columns
    ax.map_diag(plt.hist, bins=10)
    ax.map_offdiag(plt.scatter)

    return ax
```

In [6]:
```python
pg = plot_pairgrid(df)
```



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 [7]:

```
### This is the unittest cell, please just run this cell without any modification once y

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 to fit a PCA model on all values of df.

In [8]:
```
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=n_components)
    pca.fit(df)


    return pca
```

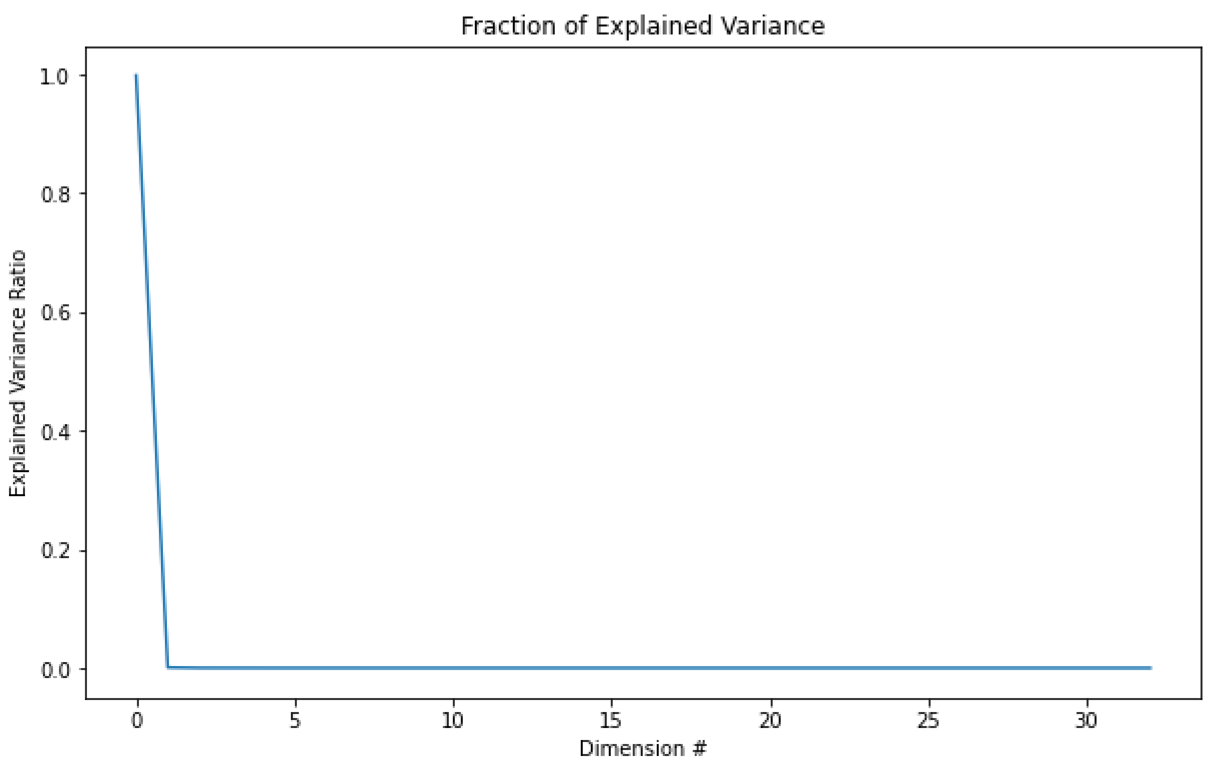
In [9]:
```
# we keep all components by setting n_components = no of cols in df. FYI df.shape[0] ret
pca_naive = fit_pca(df, n_components=df.shape[1])
```

In [10]:
```
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 [11]:
```
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
    fig, ax = plt.subplots(figsize=(10, 6))
    ax.plot(range(0, len(pca.explained_variance_ratio_)), pca.explained_variance_ratio_)
    ax.set_xlabel('Dimension #')
    ax.set_ylabel('Explained Variance Ratio')
    ax.set_title('Fraction of Explained Variance')

    return ax
```

In [12]:
```
naive_var = plot_naive_variance(pca_naive)
```



In [13]:
```
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 [14]:
```
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_v
```

```
"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 [15]:

```
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()
    scaled = scaler.fit_transform(df)


    return scaled
```

In [16]:

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

In [17]:

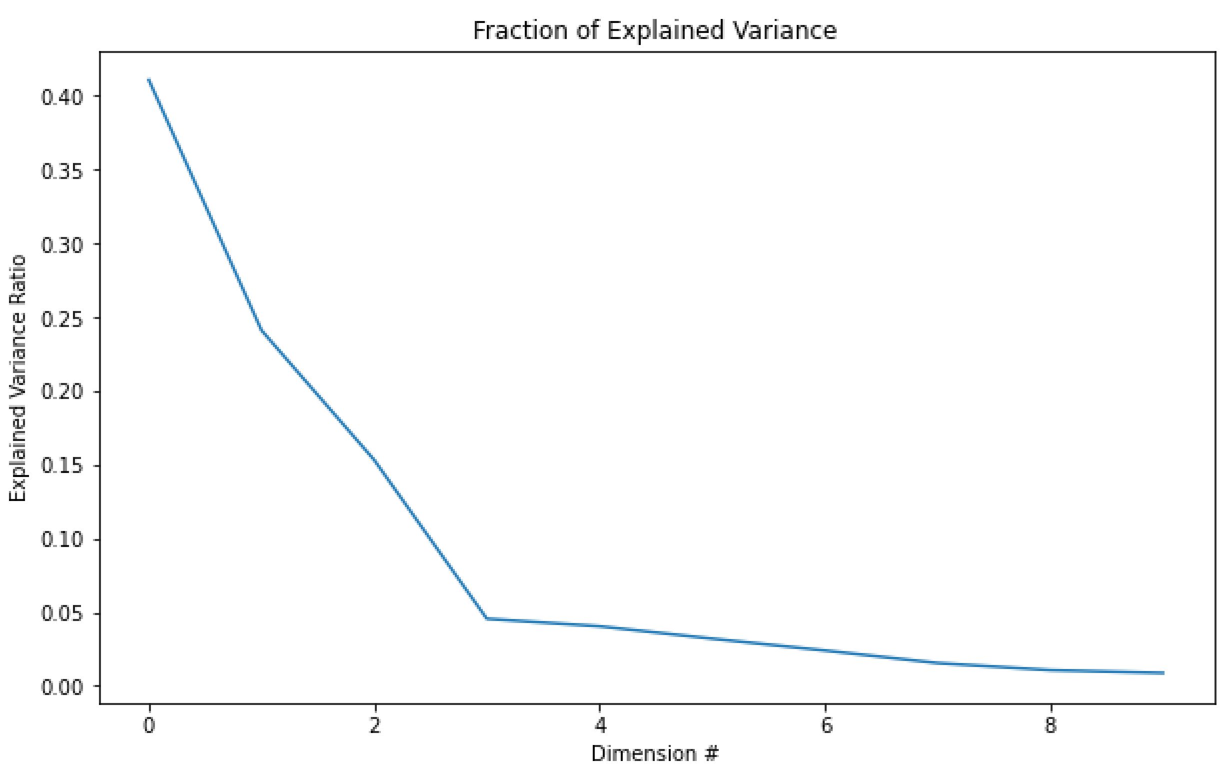
```
rng = np.random.RandomState(0)
n_samples, n_features = 4, 5

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 [18]:
```
# 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 [19]:
```
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
    fig, ax = plt.subplots(figsize=(10, 6))
    ax.plot(range(0, len(pca.explained_variance_ratio_)), pca.explained_variance_ratio_)
    ax.set_xlabel('Dimension #')
    ax.set_ylabel('Explained Variance Ratio')
    ax.set_title('Fraction of Explained Variance')

    return ax
```

In [20]:
```
ax = plot_scaled_variance(pca)
```



In [21]:
```
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 m
assert_is_not(ax.yaxis.get_label_text(), '', msg="Change the y-axis label to something m
```

```
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 [22]:
```
def reduce(pca, array):
    '''
    Applies the `pca` model on array.

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

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

    # YOUR CODE HERE
    reduced = pca.transform(array)

    return reduced
```

In [23]:
```
reduced = reduce(pca, scaled)
```

In [24]:
```
assert_is_instance(reduced, np.ndarray)
assert_array_almost_equal(reduced, pca.fit_transform(scaled))
```

In [25]:
```
# Save the reduced data to the same directory of your notebook  as 'delta_reduced.npy' t
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 [26]:
```
##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_e
```

In [27]:
```
## 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 [28]:
```
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, random_state=random_state, n_init=10).fit(arra
    clusters = model.predict(array)

    return model, clusters

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

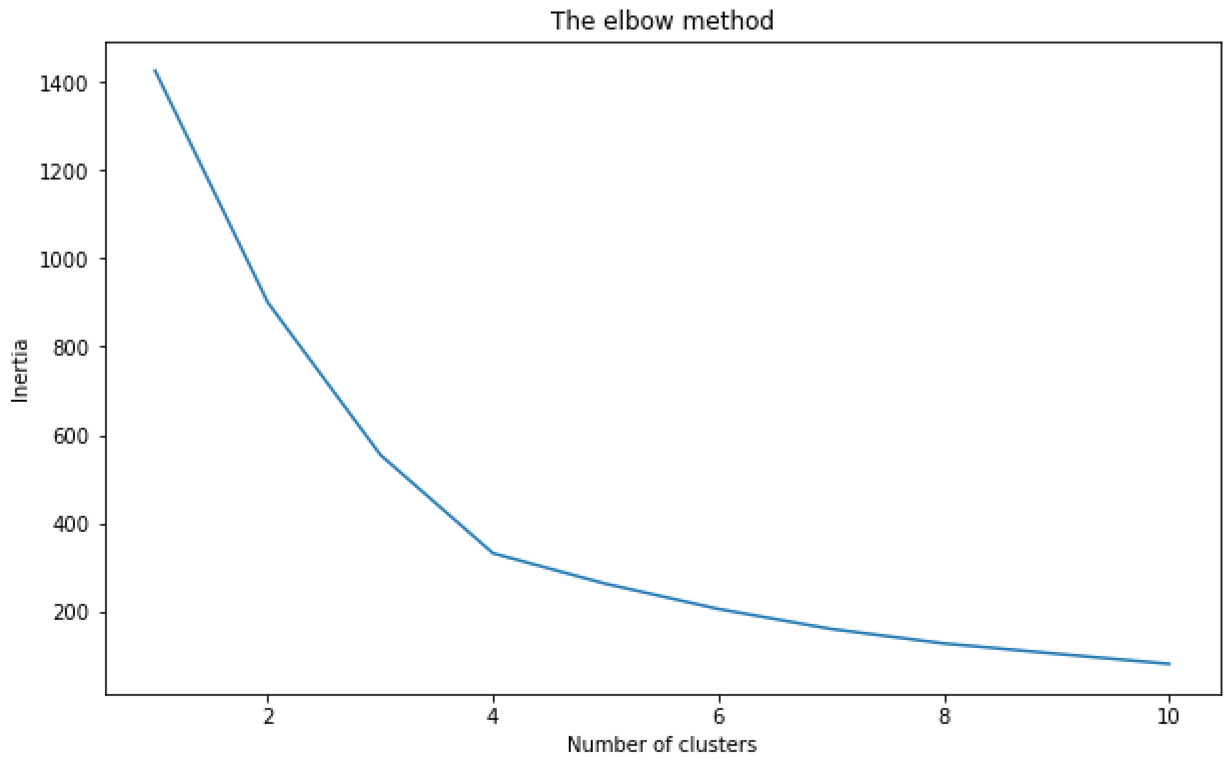
In [29]:
```
k_means_t, cluster_t = cluster(reduced, random_state=check_random_state(1), n_clusters=5

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=np.int64)
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 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]:
```
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(start, end+1)
    inertia = []

    for k in x_axis:
        # Create a KMeans instance with k clusters: model
        model = KMeans(n_clusters=k, random_state=check_random_state(0), n_init=10)

        # Fit model to array
        model.fit(array)

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


    fig, ax = plt.subplots(figsize=(10,6))

    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)
```

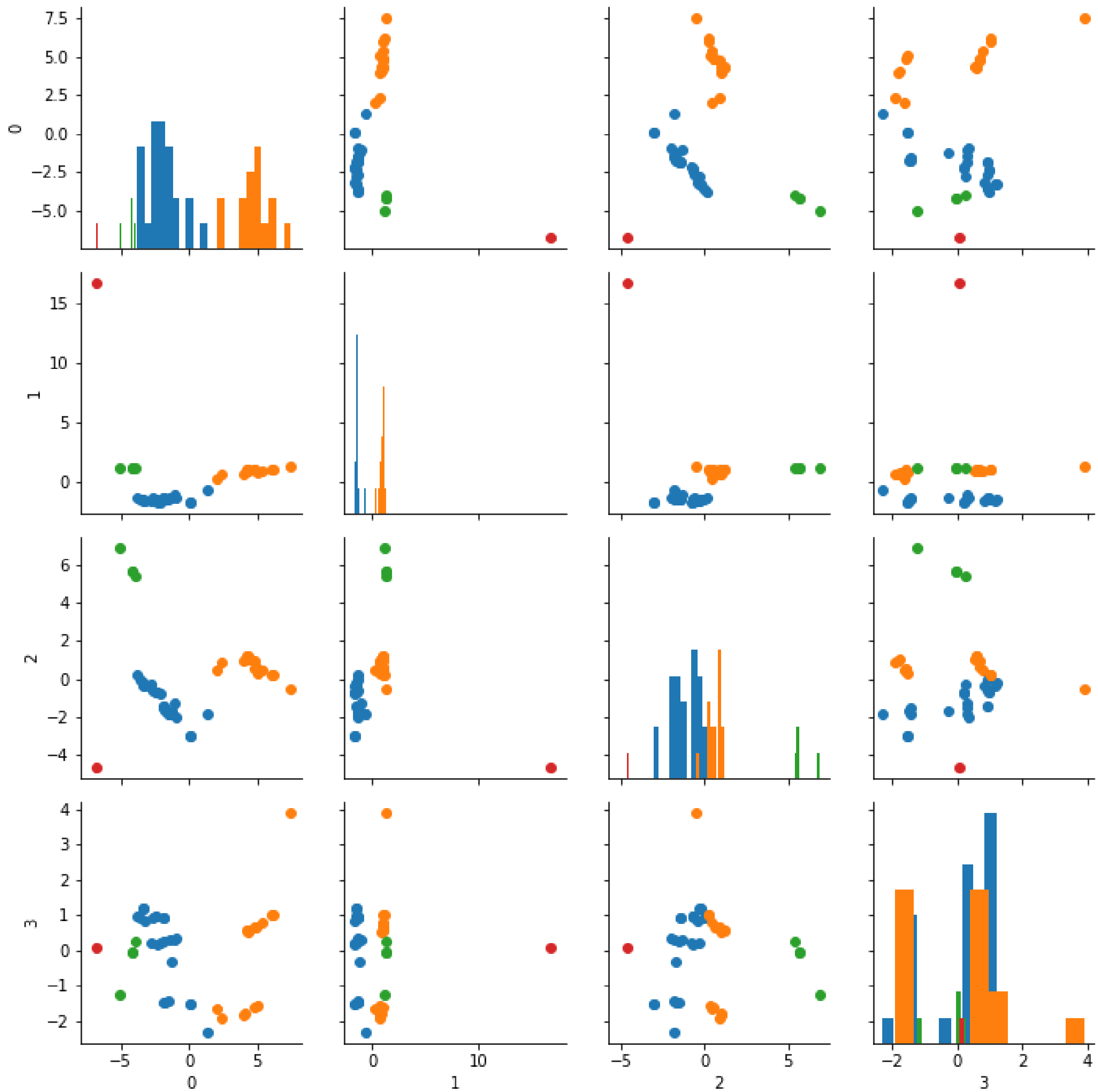


In [32]:
```
assert_is_instance(inertia, mpl.axes.Axes)
assert_true(len(inertia.lines) >= 1)

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_cluste
    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]:
```
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)
```



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]:
```
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]:
```
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]:
```
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]:
```
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]:
```
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]:
```
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]:
```
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]:
```
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]:

```python
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)

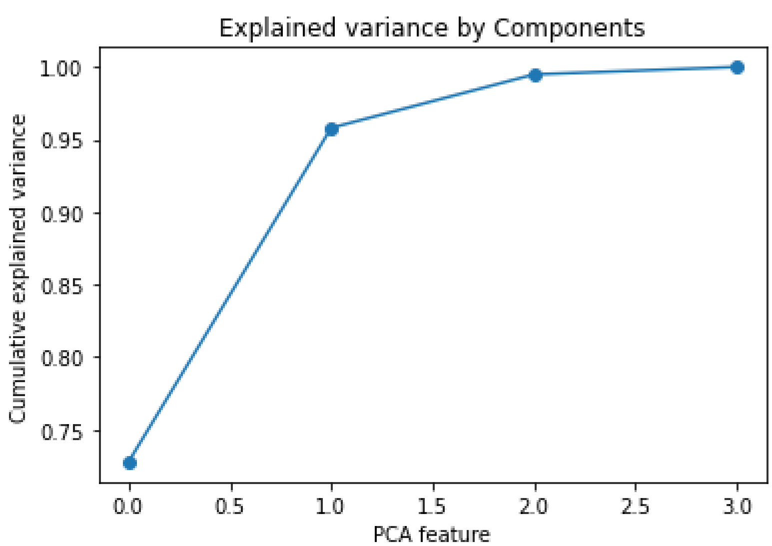
Run DBSCAN on Iris.csv and compare/discuss the results with K-Means. Please submit your code and output, and write down 3-4 sentences that you observed from the results.

In [45]:

```python
import pandas as pd
from sklearn.cluster import DBSCAN, KMeans
```

```
from sklearn import metrics
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt

# Load Iris dataset
iris = pd.read_csv("Iris.csv")
iris = iris.drop(["Id"], axis=1)

X,Y = iris.iloc[:, :-1].values, iris.iloc[:, -1].values
iris_species = iris['Species'].values

# Standardise the data
scaler = StandardScaler()
X_norm = scaler.fit_transform(X)
```

In [46]:
```
# Create a PCA instance: pca
pca = PCA()

# Fit pca to 'X'
pca.fit(X_norm)

# Plot the explained variances
plt.plot(range(0,pca.n_components_), pca.explained_variance_ratio_.cumsum(), marker='o')
plt.title('Explained variance by Components')
plt.xlabel('PCA feature')
plt.ylabel('Cumulative explained variance')
plt.show()
```



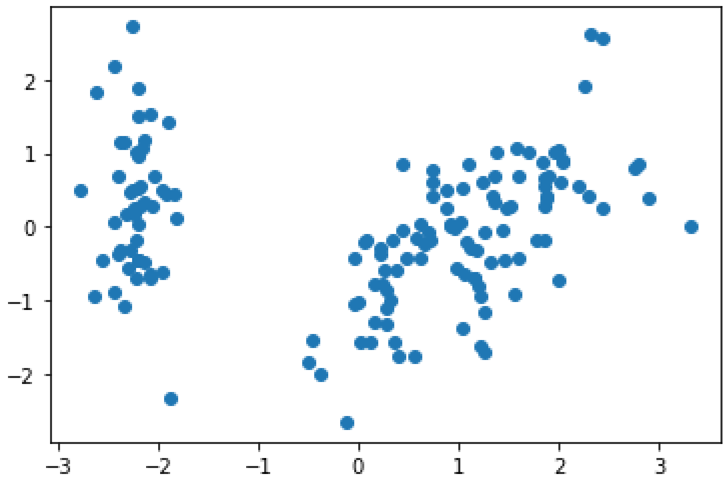
We can see that 2 PCA features explain about 95% of the variance. We can now use PCA for dimensionality reduction of the iris dataset, retaining only the 2 most important components.

In [47]:
```
# Create a PCA model with 2 components: pca
pca = PCA(n_components=2)

# Fit the PCA instance to the scaled samples
pca.fit(X_norm)

# Transform the scaled samples: pca_features
pca_features = pca.transform(X_norm)

# Print the shape of pca_features
print(pca_features.shape)
print("Variance explained by each of the n_components: ",pca.explained_variance_ratio_)
print("Total variance explained by the n_components: ",sum(pca.explained_variance_ratio_
```

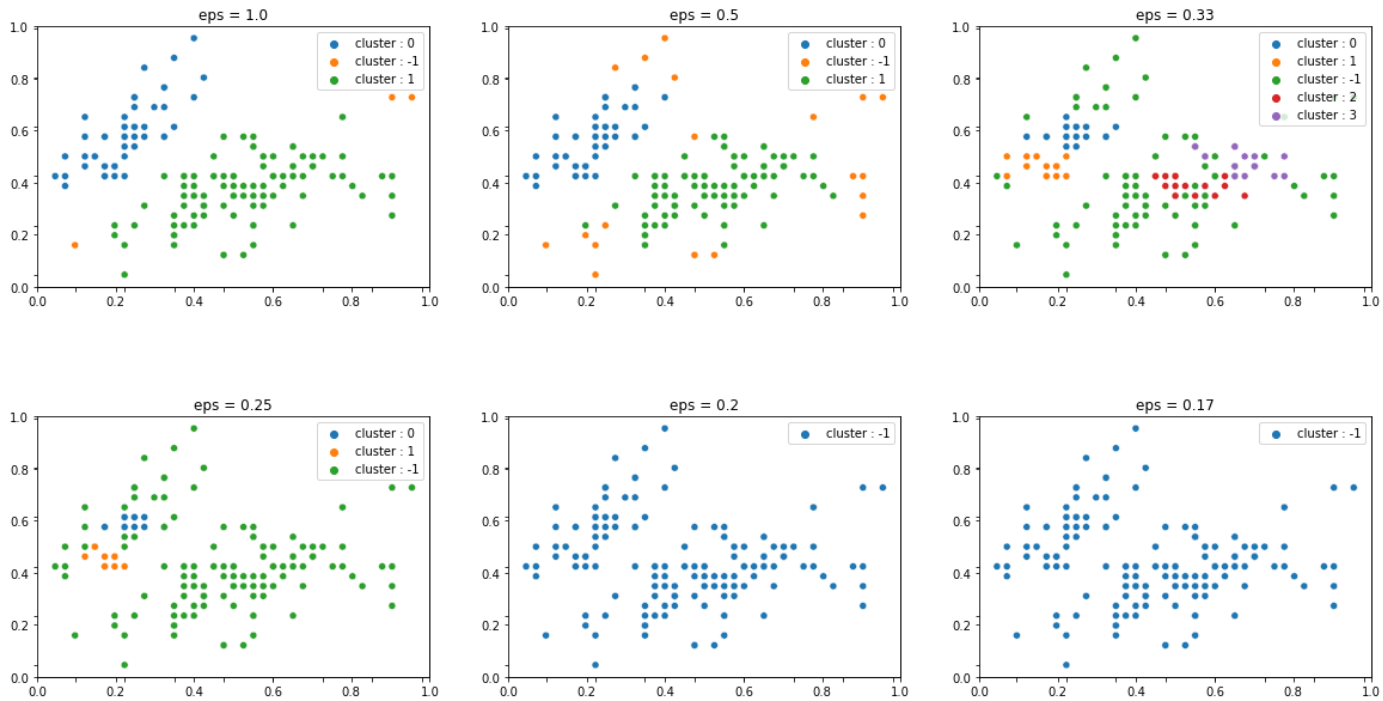
```
(150, 2)
Variance explained by each of the n_components:  [0.72770452 0.23030523]
Total variance explained by the n_components:  0.9580097536148198
```

In [48]:
```python
# Visualising the selected features
plt.scatter(pca_features[:, 0], pca_features[:, 1])
```

Out[48]:
```
<matplotlib.collections.PathCollection at 0x7f7cf95c8e50>
```



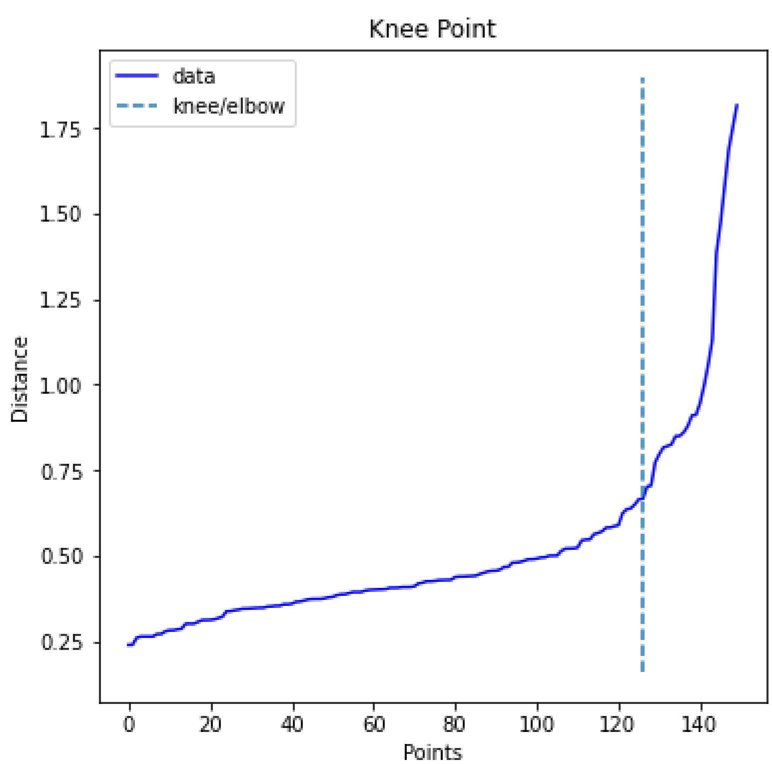
In [49]:
```python
# Apply DBSCAN

i = 1
fig, ax = plt.subplots(2,3,figsize=(20, 10))
fig.subplots_adjust(hspace=.5, wspace=.2)


for x in range(6, 0, -1):
    eps = 1/(7-x)
    db = DBSCAN(eps=eps, min_samples=10).fit(pca_features)
    core_samples_mask = np.zeros_like(db.labels_, dtype=bool)
    core_samples_mask[db.core_sample_indices_] = True
    labels = db.labels_

    ax = fig.add_subplot(2, 3, i)
    ax.set_title("eps = {}".format(round(eps, 2)))
    ax.set_yticklabels([])
    ax.set_xticklabels([])

    sns.scatterplot(X[:,0], X[:,1], hue=["cluster : {}".format(x) for x in labels])

    i += 1
```



In [50]:

```
from sklearn.neighbors import NearestNeighbors
nearest_neighbors = NearestNeighbors(n_neighbors=10)
neighbors = nearest_neighbors.fit(pca_features)
distances, indices = neighbors.kneighbors(pca_features)
distances = np.sort(distances[:,9], axis=0)

from kneed import KneeLocator
i = np.arange(len(distances))
knee = KneeLocator(i, distances, S=1, curve='convex', direction='increasing', interp_met
fig = plt.figure(figsize=(10, 10))
knee.plot_knee()
plt.xlabel("Points")
plt.ylabel("Distance")

print(distances[knee.knee])
```

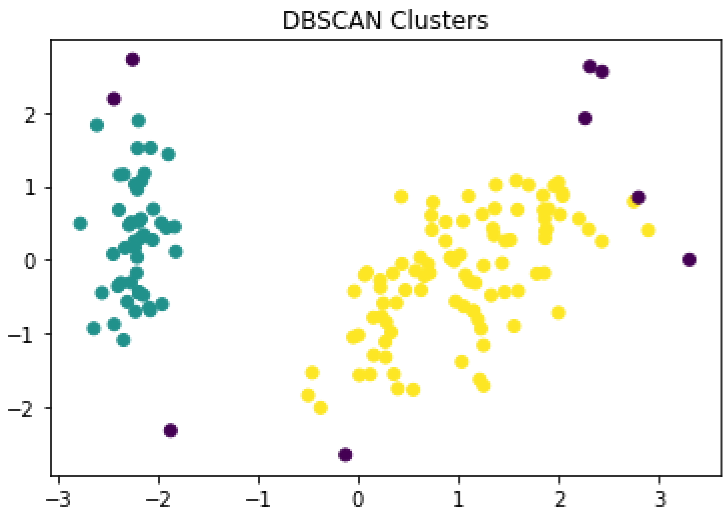
```
0.6657308263923705
<Figure size 720x720 with 0 Axes>
```



We apply DBSCAN with min_samples = 10 and eps = 0.6 and visualise the clusters

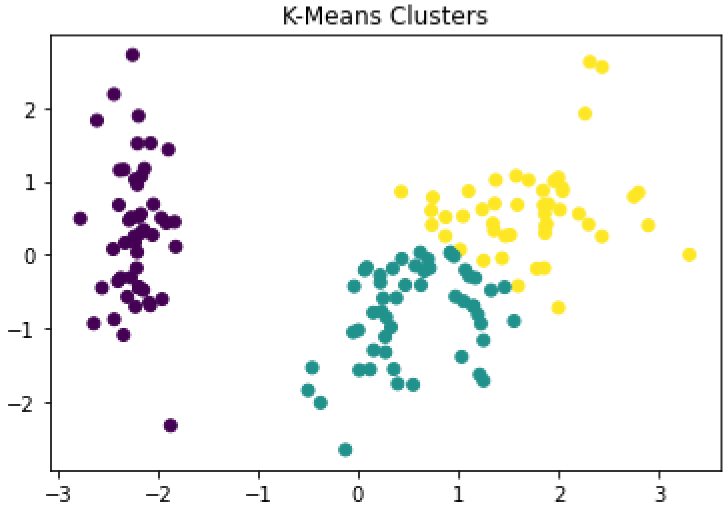
In [51]:
```
db_model = DBSCAN(eps=0.6, min_samples=10)
db_labels = db_model.fit_predict(pca_features)

db_df = pd.DataFrame({'labels': db_labels, 'species': iris_species})

# Create crosstab: ct
db_ct = pd.crosstab(db_df['labels'], db_df['species'])
print(db_ct)

# Visualise the clusters
plt.scatter(pca_features[:, 0], pca_features[:, 1], c=db_labels)
plt.title("DBSCAN Clusters")
plt.show()
```

```
species   Iris-setosa  Iris-versicolor  Iris-virginica
labels
-1                  3                1               5
 0                 47                0               0
 1                  0               49              45
```



In [52]:
```
# Apply KMeans

# Create a KMeans model with 3 clusters: model
km_model = KMeans(n_clusters=3, random_state=check_random_state(0), n_init=10)

# Use fit_predict to fit model and obtain cluster labels: labels
km_labels = km_model.fit_predict(pca_features)

km_df = pd.DataFrame({'labels': km_labels, 'species': iris_species})

# Create crosstab: ct
km_ct = pd.crosstab(km_df['labels'], km_df['species'])
print(km_ct)

# Visualise the clusters
plt.scatter(pca_features[:, 0], pca_features[:, 1], c=km_labels)
plt.title("K-Means Clusters")
plt.show()
```

```
species   Iris-setosa  Iris-versicolor  Iris-virginica
labels
0                  50                0               0
1                   0               39              14
2                   0               11              36
```



In [53]:
```
from sklearn.metrics.cluster import adjusted_rand_score

print ( "Measuring performance of the 2 methods : \n")
#k-means performance:
print("K-Means =", int(adjusted_rand_score(iris_species, km_labels)*100),"%")

#DBSCAN performance:
print("DBSCAN =", int(adjusted_rand_score(iris_species, db_labels)*100),"%")
```

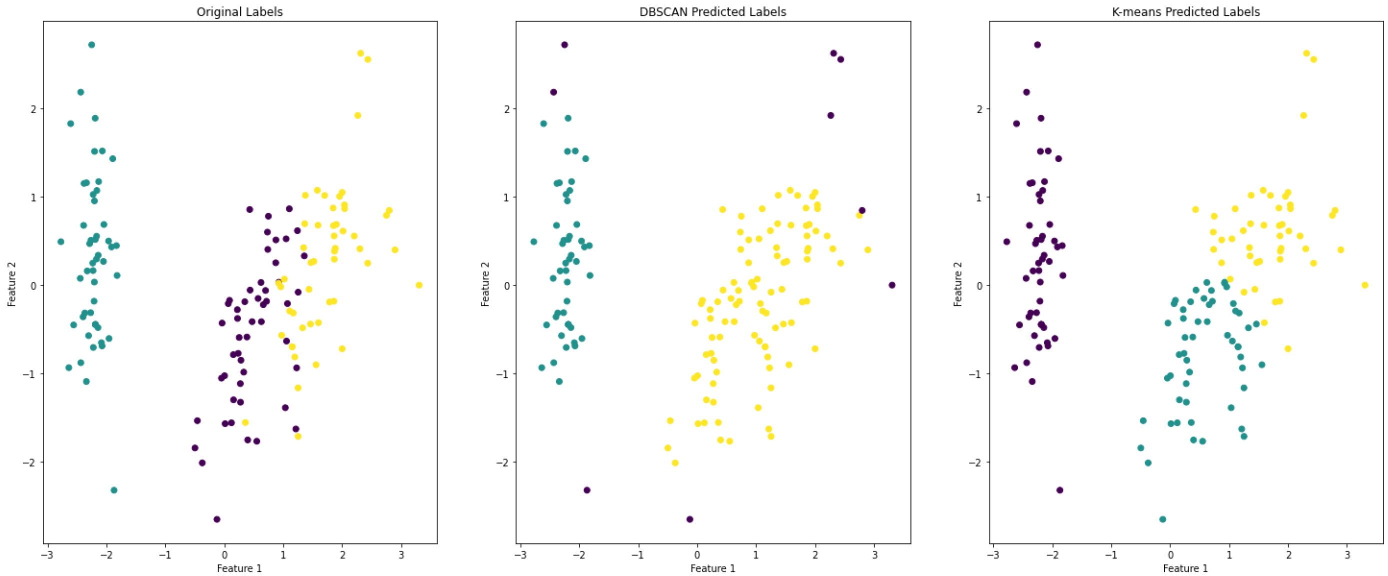
```
Measuring performance of the 2 methods :

K-Means = 62 %
DBSCAN = 52 %
```

In [54]:
```
# Calculate the silhouette score
db_score = metrics.silhouette_score(pca_features, db_labels)
km_score = metrics.silhouette_score(pca_features, km_labels)

# Print the silhouette scores
print("Silhouette score for DBSCAN:", db_score)
print("Silhouette score for K-Means:", km_score)
```

```
Silhouette score for DBSCAN: 0.5470524869561282
Silhouette score for K-Means: 0.5081546339516392
```

In [55]:
```
species = {'Iris-setosa': 1,'Iris-versicolor': 0, 'Iris-virginica' : 2}

species_map = [species[item] for item in iris_species]
```

In [56]:
```
fig, ax = plt.subplots(1,3,figsize=(25, 10))

ax[0].scatter(pca_features[:, 0], pca_features[:, 1], c=species_map)
ax[1].scatter(pca_features[:, 0], pca_features[:, 1], c=db_labels)
ax[2].scatter(pca_features[:, 0], pca_features[:, 1], c=km_labels)

ax[0].set_xlabel('Feature 1')
ax[0].set_ylabel('Feature 2')
ax[0].set_title('Original Labels')

ax[1].set_xlabel('Feature 1')
ax[1].set_ylabel('Feature 2')
ax[1].set_title('DBSCAN Predicted Labels')

ax[2].set_xlabel('Feature 1')
ax[2].set_ylabel('Feature 2')
ax[2].set_title('K-means Predicted Labels')

plt.show()
```



On applying PCA to reduce the number of features and implementing DBSCAN and K-Means on the IRIS data, we can see the Predicted labels vs the original labels in the charts above.

Visually, it is very evident that K-Means was able to identify the 3 classes more accurately then DBSCAN. Furthermore, on computing the Adjusted Rand Score we can see that K-Means has a better performance (62%) when compared to the DBSCAN output (52%) for this scenario.

We can observe that the silhouette scores for DBSCAN are better than that of K-Means mainly because in DBSCAN we have more distinguishable clusters whereas in K-Means the distance between 2 of the 3 clusters is not very significant.

For IRIS dataset, K-Means performs better than DBSCAN for clustering our data as the clusters are spherical in shape.

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.

In [57]:
```
## Reload the the first 10 components of delta dataset
reduced = np.load('delta_reduced.npy')
```

Clusters generated using the K-means Clustering method applied in Problem 2

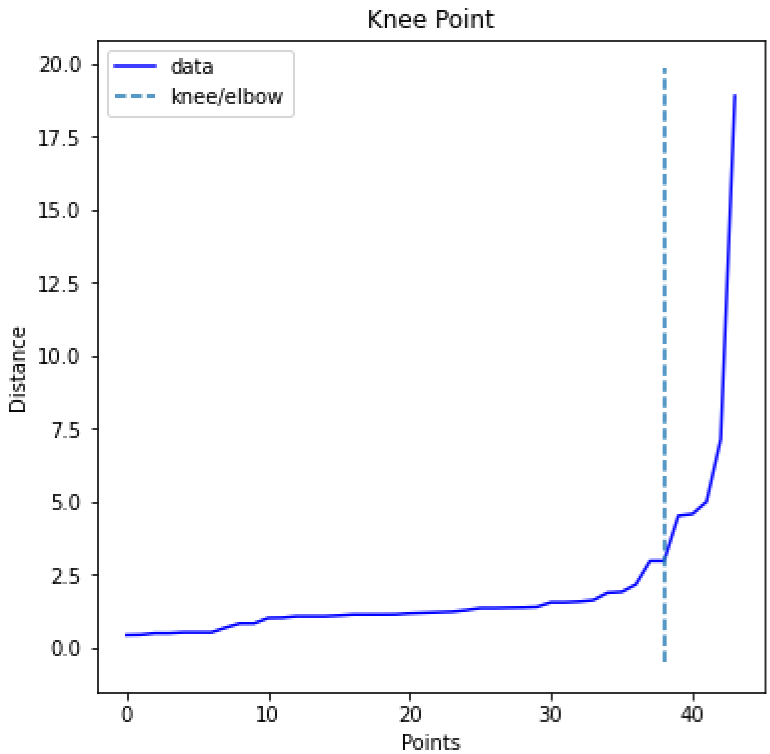
In [58]:
```
km_delta = df.iloc[:,-2:]
km_delta_count = km_delta.groupby("Clusters")['Aircraft'].count()
km_delta_count
```

Out[58]:
```
Clusters
0    24
1    15
2     4
3     1
Name: Aircraft, dtype: int64
```

In [59]:
```
#Applying DBSCAN

from sklearn.neighbors import NearestNeighbors
nearest_neighbors = NearestNeighbors(n_neighbors=3)
neighbors = nearest_neighbors.fit(reduced)
distances, indices = neighbors.kneighbors(reduced)
distances = np.sort(distances[:,2], axis=0)
```

```
from kneed import KneeLocator
i = np.arange(len(distances))
knee = KneeLocator(i, distances, S=1, curve='convex', direction='increasing', interp_met
fig = plt.figure(figsize=(10, 10))
knee.plot_knee()
plt.xlabel("Points")
plt.ylabel("Distance")

print(distances[knee.knee])
```

```
2.9657186547402756
<Figure size 720x720 with 0 Axes>
```



We apply DBSCAN with min_samples = 3 and eps = 3 as this gives us 4 clusters with minimal noise

In [60]:
```
db_delta_model = DBSCAN(eps=3, min_samples=3)
db_delta_labels = db_delta_model.fit_predict(reduced)

db_delta = pd.DataFrame({'DB_clusters': db_delta_labels, 'Aircraft': km_delta['Aircraft'

db_delta.groupby("DB_clusters")['Aircraft'].count()
```

Out[60]:
```
DB_clusters
-1      5
 0     23
 1      8
 2      5
 3      3
Name: Aircraft, dtype: int64
```

K-Means clusters generated above from Problem 2 are :

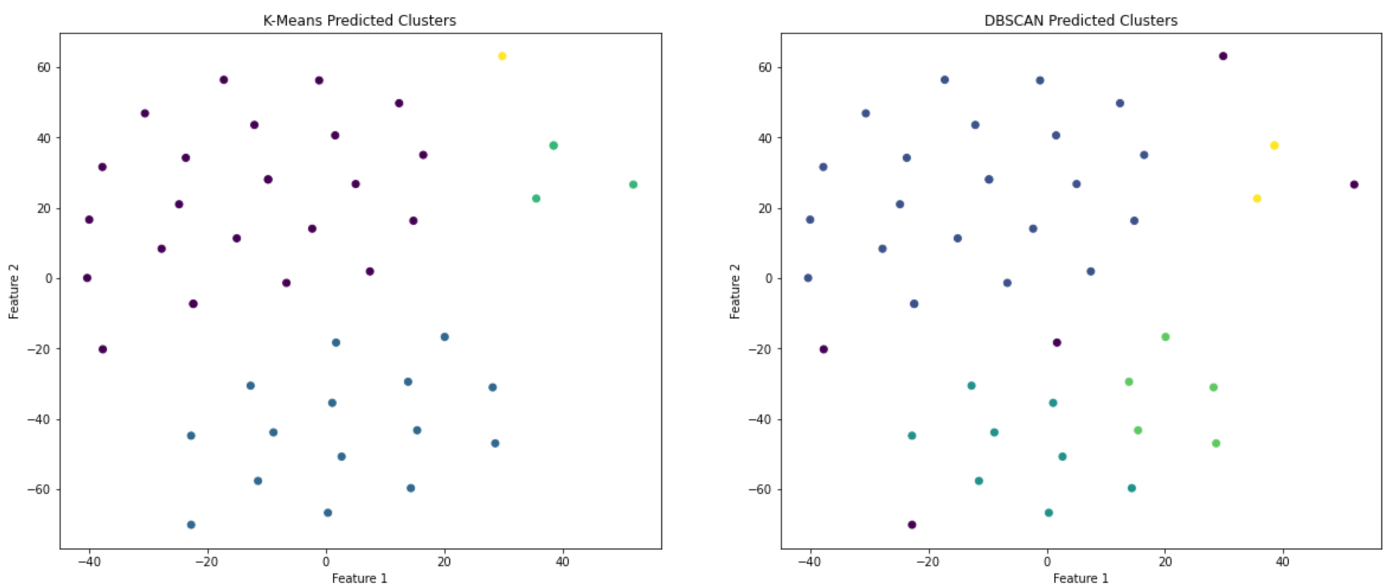
In [61]:
```
km_delta.groupby("Clusters")['Aircraft'].count()
```

Out[61]:
```
Clusters
0    24
1    15
2     4
3     1
Name: Aircraft, dtype: int64
```

Comparing the 2 outputs above we can see that DBSCAN outputs are very similar to our K-Means clusters.

Using TSNE we can visualise the results in a 2-D visual in order to interpret the performance of DBSCAN versus K-Means.

In [62]:
```
from sklearn.manifold import TSNE

tsne = TSNE(n_components=2, learning_rate=75)
X_tsne = tsne.fit_transform(reduced)

xs, ys  = X_tsne[:,0], X_tsne[:,1]


fig, ax = plt.subplots(1,2,figsize=(20, 8))

ax[0].scatter(xs, ys, c=km_delta['Clusters'])
ax[1].scatter(xs, ys, c=db_delta['DB_clusters'])

ax[0].set_xlabel('Feature 1')
ax[0].set_ylabel('Feature 2')
ax[0].set_title('K-Means Predicted Clusters')

ax[1].set_xlabel('Feature 1')
ax[1].set_ylabel('Feature 2')
ax[1].set_title('DBSCAN Predicted Clusters')


plt.show()
```



We know from output of Problem 2 that the K-Means model clusters our data points into 4 logical groups based on the Aircraft specifications.

On employing DBSCAN to our data and generating 4 clusters and visualising the outputs using TSNE we can observe that DBSCAN does cluster our data into 4 groups with very minimal noise (just 5 data points) but is not as accurate as our K-means clusters.

For our reduced delta aircraft dataset with 10 features, K-Means clusters are more significant when compared to the clusters generated by DBSCAN.

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 [63]:
```
# Load movements dataset
movements = pd.read_csv("movements.csv")
```

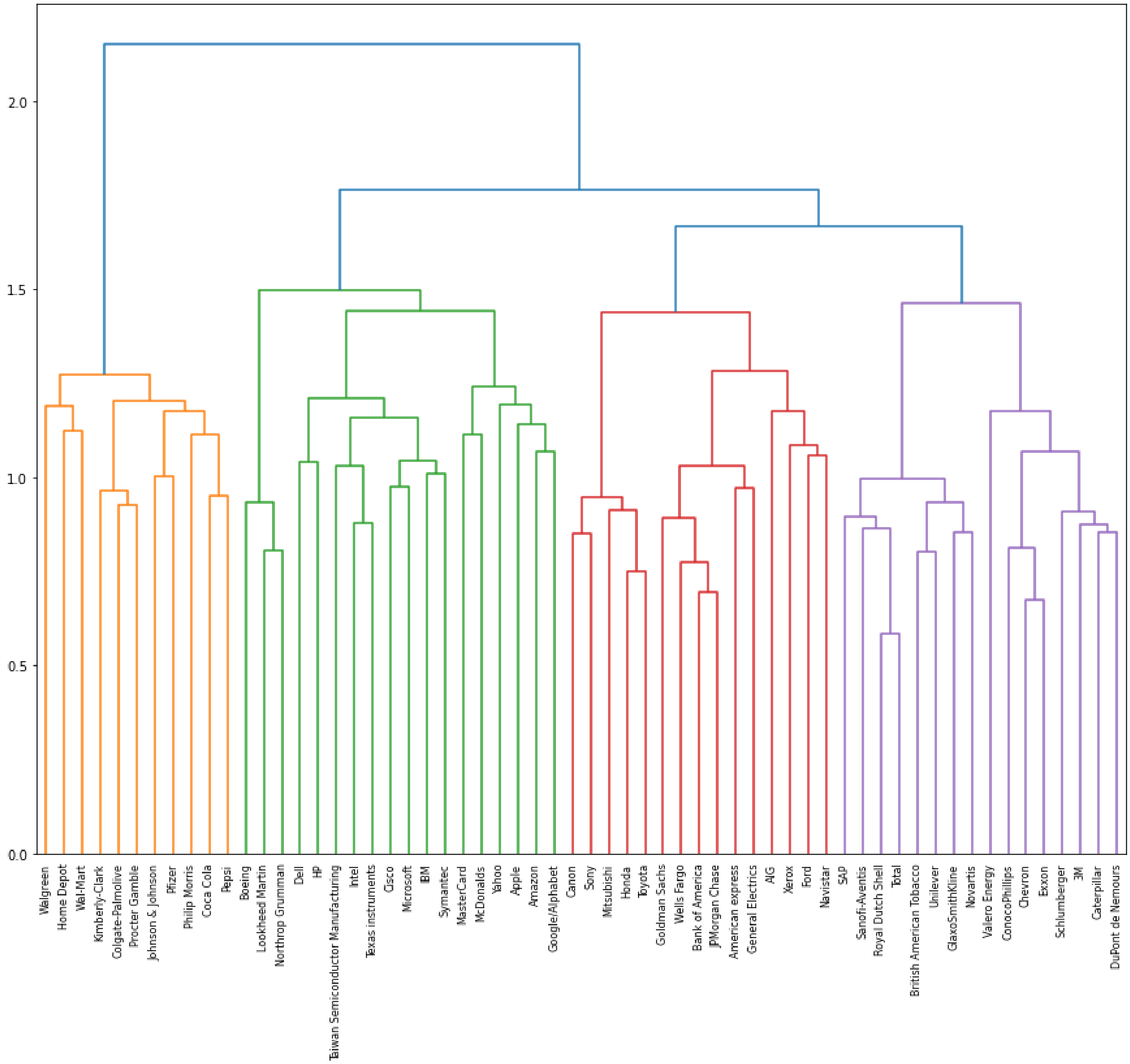
In [64]:
```
movements.head()
```

Out[64]:

	Unnamed: 0	2010-01-04	2010-01-05	2010-01-06	2010-01-07	2010-01-08	2010-01-11	2010-01-12	2010-01-13	2
0	Apple	0.580000	-0.220005	-3.409998	-1.170000	1.680011	-2.689994	-1.469994	2.779997	-0
1	AIG	-0.640002	-0.650000	-0.210001	-0.420000	0.710001	-0.200001	-1.130001	0.069999	-0
2	Amazon	-2.350006	1.260009	-2.350006	-2.009995	2.960006	-2.309997	-1.640007	1.209999	-
3	American express	0.109997	0.000000	0.260002	0.720002	0.190003	-0.270001	0.750000	0.300004	0
4	Boeing	0.459999	1.770000	1.549999	2.690003	0.059997	-1.080002	0.360000	0.549999	0

5 rows × 964 columns

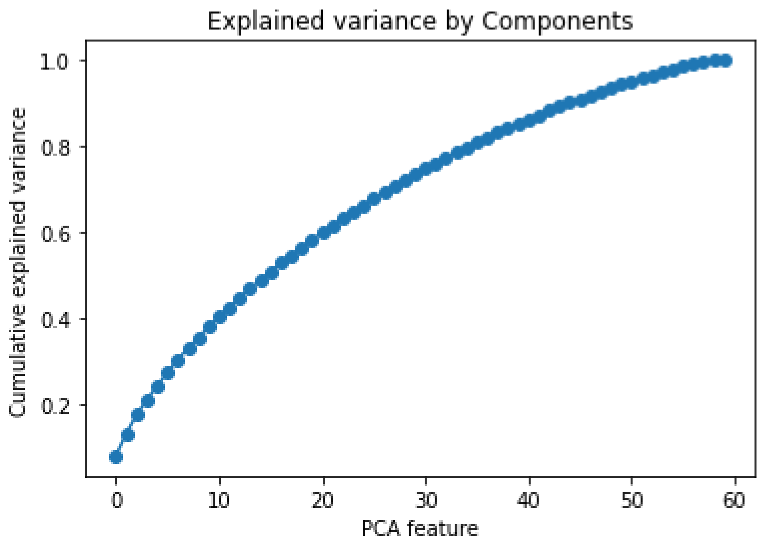
In [65]:
```
# Apply Hierarchical Clustering

# Import required packages for pre and post processing
from sklearn.preprocessing import normalize
from scipy.cluster.hierarchy import linkage, dendrogram, fcluster


normalized_movements = normalize(movements.iloc[:,1:].values)
plt.figure(figsize=(15,12))

# Calculate the linkage: mergings
mergings = linkage(normalized_movements, method='ward')
hc_labels = fcluster(mergings, 1.5, criterion='distance')
companies = movements.iloc[:,0].values
# Plot the dendrogram
dendrogram(
    mergings,
    labels=companies,
    leaf_rotation=90.,
    leaf_font_size=8
)


plt.show()
```



From the output of hierarchical clustering we can see that our data is grouped into 4 logical clusters based on daily price movements. Now we can try generating these 4 clusters using K-Means and DBSCAN

In [66]:
```
X,Y = movements.iloc[:, 1:].values, movements.iloc[:, 0].values

# Normalise the data
X_norm = normalize(X)
```

In [67]:
```
# Create a PCA instance: pca
pca = PCA()

# Fit pca to 'X'
pca.fit(X_norm)

# Plot the explained variances
plt.plot(range(0,pca.n_components_), pca.explained_variance_ratio_.cumsum(), marker='o')
plt.title('Explained variance by Components')
plt.xlabel('PCA feature')
plt.ylabel('Cumulative explained variance')
plt.show()
```



In [68]:

```
# Create a PCA model with 40 components: pca
pca = PCA(n_components=40)

# Fit the PCA instance to the scaled samples
pca.fit(X_norm)

# Transform the scaled samples: pca_features
pca_features = pca.transform(X_norm)

# Print the shape of pca_features
print(pca_features.shape)
print("Variance explained by each of the n_components: ",pca.explained_variance_ratio_)
print("Total variance explained by the n_components: ",sum(pca.explained_variance_ratio_
```

```
(60, 40)
Variance explained by each of the n_components:  [0.08044047 0.05190949 0.04361654 0.034
5087  0.03274123 0.03145806
 0.0303444  0.02587708 0.02502402 0.02442041 0.02384288 0.02215196
 0.02184713 0.02123589 0.02016123 0.01997043 0.01945132 0.01846491
 0.01809117 0.01756118 0.01692913 0.01625154 0.01610724 0.01571442
 0.01530809 0.01512413 0.01497176 0.01416402 0.01373971 0.01343992
 0.01307107 0.01292622 0.01215918 0.01201814 0.01185984 0.01160154
 0.01123418 0.01113252 0.01055232 0.01037657]
Total variance explained by the n_components:  0.8518000352707084
```

We can go ahead with 40 components as it explains about 85% of the total variance.

For K-Means we can try generating 4 clusters as repersented in the Hierarchical clustering.

In [69]:

```
# Apply KMeans

# Create a KMeans model with 4 clusters: model
km_model = KMeans(n_clusters=4, random_state=check_random_state(0), n_init=10)

# Use fit_predict to fit model and obtain cluster labels: labels
km_labels = km_model.fit_predict(pca_features)

km_df = pd.DataFrame({'labels': km_labels, 'companies': Y})

print('KMeans Clusters')
km_df.groupby('labels').count()
```

```
KMeans Clusters
```

Out[69]:

	companies
labels	

0	9
1	15
2	15
3	21

Since the early leaves at the botton of the dendogram groups the companies in pairs, we can take min_samples as 2 and try generating clusters with minimal noise using DBSCAN.

In [70]:
```
# Apply DBSCAN

db_model = DBSCAN(eps=0.8, min_samples=2)
db_labels = db_model.fit_predict(pca_features)

db_df = pd.DataFrame({'labels': db_labels, 'companies': Y})

print('DBSCAN Clusters')
db_df.groupby('labels').count()
```

```
DBSCAN Clusters
```

Out[70]:

	companies
labels	
-1	26
0	29
1	3
2	2

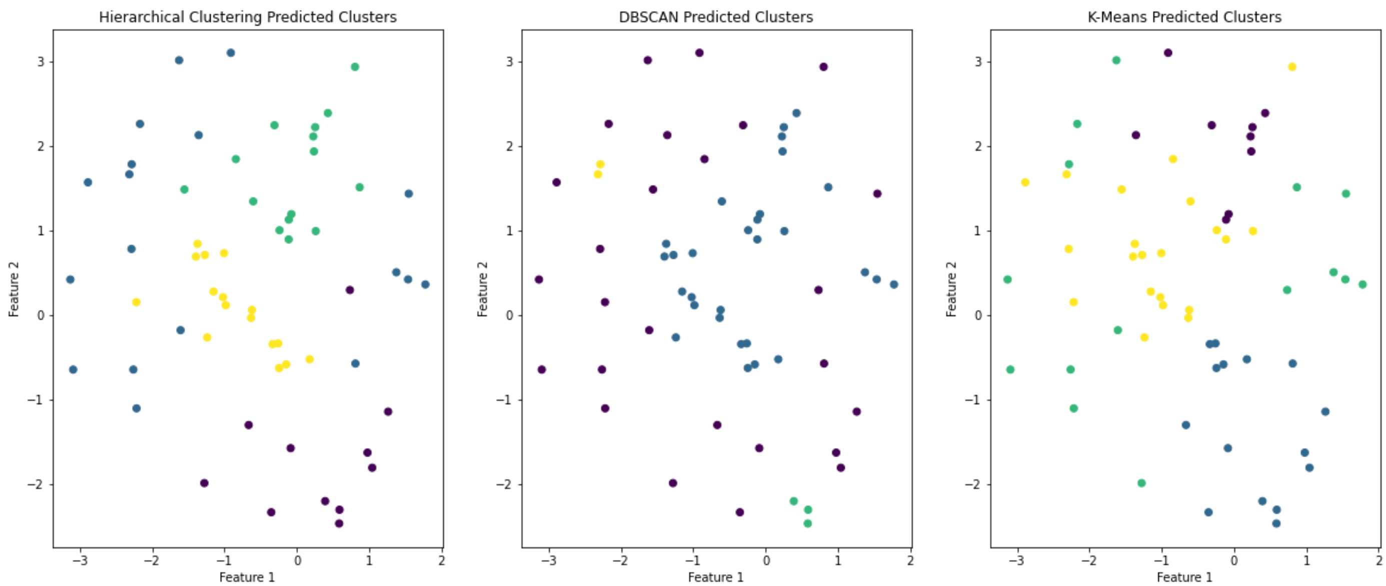
In [71]:
```
print ( "Measuring performance of the 2 methods against Hierarchical clusters: \n")
#k-means performance:
print("K-Means =", int(adjusted_rand_score(hc_labels, km_labels)*100),"%")

#DBSCAN performance:
print("DBSCAN =", int(adjusted_rand_score(hc_labels, db_labels)*100),"%")
```

```
Measuring performance of the 2 methods against Hierarchical clusters:

K-Means = 30 %
DBSCAN = 24 %
```

We see from the computation above that DBSCAN has a better clustering performance (30%) when compared to K-means (24%). We can now visually compare the performance of DBSCAN, K-Means and the Hierarchical clustering using TSNE.

In [72]:
```
from sklearn.manifold import TSNE

tsne = TSNE(n_components=2, learning_rate=75)
X_tsne = tsne.fit_transform(pca_features)

xs, ys  = X_tsne[:,0], X_tsne[:,1]

fig, ax = plt.subplots(1,3,figsize=(20, 8))

ax[0].scatter(xs, ys, c=hc_labels)
ax[1].scatter(xs, ys, c=db_df['labels'])
ax[2].scatter(xs, ys, c=km_df['labels'])
```

```
ax[0].set_xlabel('Feature 1')
ax[0].set_ylabel('Feature 2')
ax[0].set_title('Hierarchical Clustering Predicted Clusters')

ax[1].set_xlabel('Feature 1')
ax[1].set_ylabel('Feature 2')
ax[1].set_title('DBSCAN Predicted Clusters')

ax[2].set_xlabel('Feature 1')
ax[2].set_ylabel('Feature 2')
ax[2].set_title('K-Means Predicted Clusters')


plt.show()
```



We can see that DBSCAN performs better than K-Means on noisy data. However, we can understand from the above implementations that Hierarchical clustering has the best performance for data with higher number of dimensions.

Hierarchical clustering is better than K-means and DBSCAN here because :

Data is more complex and has a hierarchical structure, and the number of clusters is not known beforehand. Also, hierarchical clustering can identify nested clusters, which may not be captured well by K-means or DBSCAN.

Problem 4

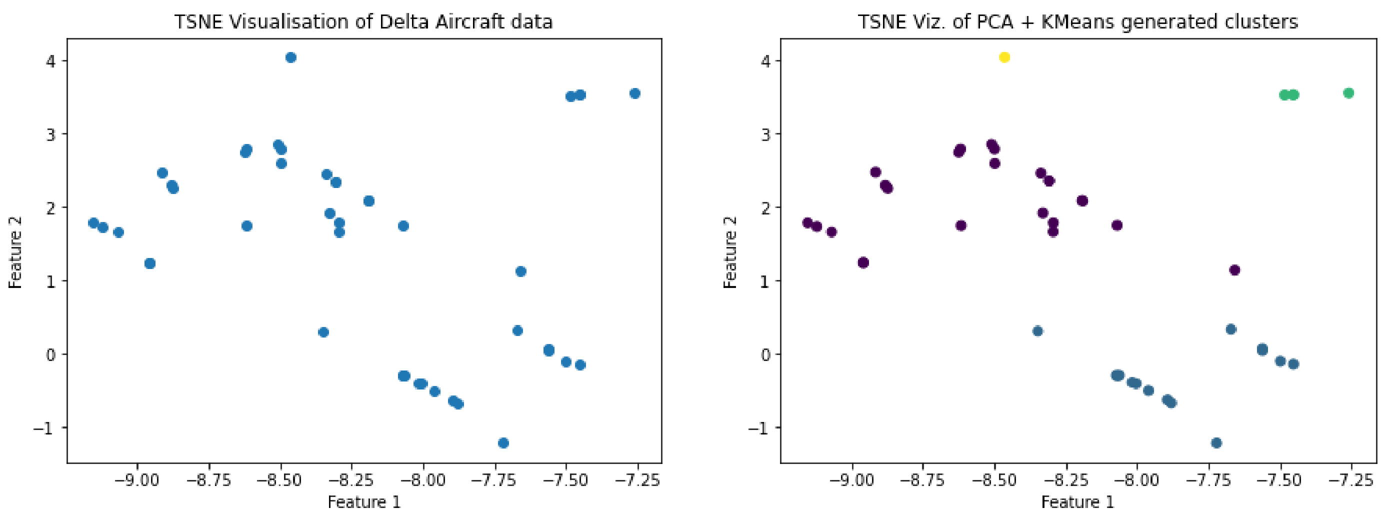
Apply t-SNE reduction to delta.csv file and compare/discuss the results with PCA. Please submit your code and output, and write down 3-4 sentences that you observed from the results.

In [73]:
```
#Reading original delta.csv data
delta_df = pd.read_csv('delta.csv', index_col='Aircraft')

#Standardizing the data
scaled = standardize(delta_df)
```

In [74]:
```
#Apply TSNE to scaled original delta.csv data

from sklearn.manifold import TSNE
```

```
tsne = TSNE(n_components=2, learning_rate=50)

# Apply fit_transform to scaled delta data
X_tsne = tsne.fit_transform(scaled)

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

# Select the 1st feature: ys
ys = X_tsne[:,1]
```

In [75]:
```
fig, ax = plt.subplots(1,2,figsize=(15, 5))

ax[0].scatter(xs, ys)
ax[1].scatter(xs, ys, c=km_delta['Clusters'])

ax[0].set_xlabel('Feature 1')
ax[0].set_ylabel('Feature 2')
ax[0].set_title('TSNE Visualisation of Delta Aircraft data')

ax[1].set_xlabel('Feature 1')
ax[1].set_ylabel('Feature 2')
ax[1].set_title('TSNE Viz. of PCA + KMeans generated clusters')

plt.show()
```



PCA is a fast and efficient technique for reducing the dimensionality of the data in a linear manner, while t-SNE is a powerful technique for exploring the relationships between data points in a non-linear manner.

t-SNE is useful when you want to explore the structure of the data and identify patterns and relationships between data points. t-SNE is especially useful in this case when working with high-dimensional data, as it can preserve local structures and help with visualization.

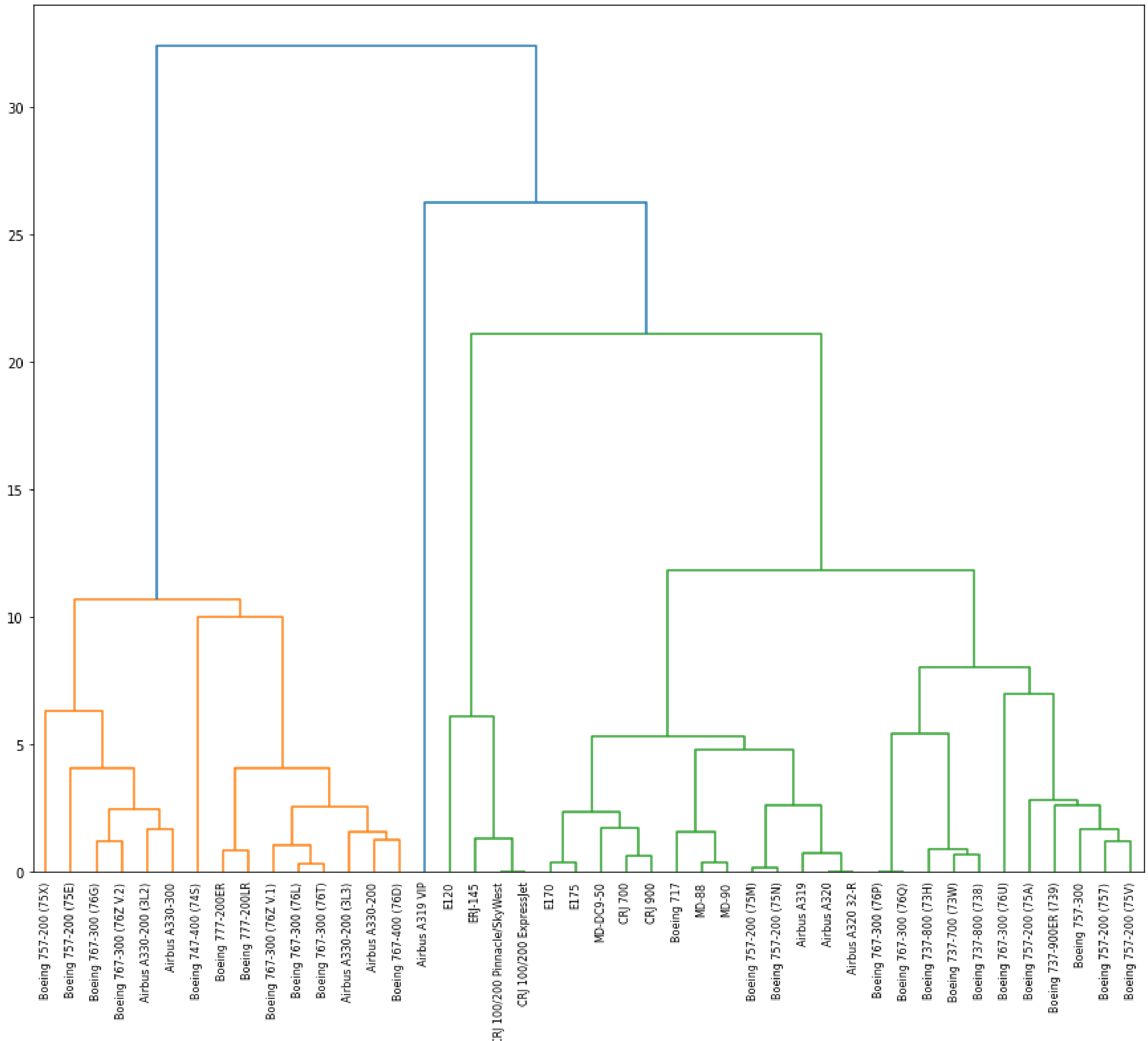
From the above visualisations, we can see that TSNE does help us discern clusters almost as well as the clusters generated by K-Means with PCA on the Delta data. TSNE does help us identufy the number of clusters in high dimensions.

Problem 5 (Bonus)

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

In [76]:

```
delta_df = pd.read_csv('delta.csv', index_col='Aircraft')
standardized_delta = StandardScaler().fit_transform(delta_df.iloc[:,:].values)

plt.figure(figsize=(15,12))

# Calculate the linkage: mergings
mergings = linkage(standardized_delta, method='ward')
aircraft = delta_df.index.values
hc_delta_labels = fcluster(mergings, 20, criterion='distance')
# Plot the dendrogram
dendrogram(
    mergings,
    labels=aircraft,
    leaf_rotation=90.,
    leaf_font_size=8
)

plt.show()
```



In [77]:

```
km_delta['Hierarchical_Clusters'] = hc_delta_labels
km_delta
```

Out[77]:

	Clusters	Aircraft	Hierarchical_Clusters
Aircraft			

Airbus A319	0	Airbus A319	3
Airbus A319 VIP	3	Airbus A319 VIP	4
Airbus A320	0	Airbus A320	3
Airbus A320 32-R	0	Airbus A320 32-R	3
Airbus A330-200	1	Airbus A330-200	1
Airbus A330-200 (3L2)	1	Airbus A330-200 (3L2)	1
Airbus A330-200 (3L3)	1	Airbus A330-200 (3L3)	1
Airbus A330-300	1	Airbus A330-300	1
Boeing 717	0	Boeing 717	3
Boeing 737-700 (73W)	0	Boeing 737-700 (73W)	3
Boeing 737-800 (738)	0	Boeing 737-800 (738)	3
Boeing 737-800 (73H)	0	Boeing 737-800 (73H)	3
Boeing 737-900ER (739)	0	Boeing 737-900ER (739)	3
Boeing 747-400 (74S)	1	Boeing 747-400 (74S)	1
Boeing 757-200 (75A)	0	Boeing 757-200 (75A)	3
Boeing 757-200 (75E)	1	Boeing 757-200 (75E)	1
Boeing 757-200 (75M)	0	Boeing 757-200 (75M)	3
Boeing 757-200 (75N)	0	Boeing 757-200 (75N)	3
Boeing 757-200 (757)	0	Boeing 757-200 (757)	3
Boeing 757-200 (75V)	0	Boeing 757-200 (75V)	3
Boeing 757-200 (75X)	1	Boeing 757-200 (75X)	1
Boeing 757-300	0	Boeing 757-300	3
Boeing 767-300 (76G)	1	Boeing 767-300 (76G)	1
Boeing 767-300 (76L)	1	Boeing 767-300 (76L)	1
Boeing 767-300 (76P)	0	Boeing 767-300 (76P)	3
Boeing 767-300 (76Q)	0	Boeing 767-300 (76Q)	3
Boeing 767-300 (76T)	1	Boeing 767-300 (76T)	1
Boeing 767-300 (76U)	0	Boeing 767-300 (76U)	3
Boeing 767-300 (76Z V.1)	1	Boeing 767-300 (76Z V.1)	1
Boeing 767-300 (76Z V.2)	1	Boeing 767-300 (76Z V.2)	1
Boeing 767-400 (76D)	1	Boeing 767-400 (76D)	1
Boeing 777-200ER	1	Boeing 777-200ER	1
Boeing 777-200LR	1	Boeing 777-200LR	1
CRJ 100/200 Pinnacle/SkyWest	2	CRJ 100/200 Pinnacle/SkyWest	2
CRJ 100/200 ExpressJet	2	CRJ 100/200 ExpressJet	2
CRJ 700	0	CRJ 700	3
CRJ 900	0	CRJ 900	3
E120	2	E120	2
E170	0	E170	3

E175	0	E175	3
ERJ-145	2	ERJ-145	2
MD-88	0	MD-88	3
MD-90	0	MD-90	3
MD-DC9-50	0	MD-DC9-50	3

The Hierarchical clustering generates the same clusters as the K-means clustering method carried out in the initial part of this document in Problem 2.

In [78]:
```
print ("Comparing the clusters generated by Hierarchical Clustering vs Kmeans: \n")

print("Accuracy =", int(adjusted_rand_score(hc_delta_labels, km_delta['Clusters'].values
```

```
Comparing the clusters generated by Hierarchical Clustering vs Kmeans:

Accuracy = 100 %
```

In the agglomerative hierarchical clustering, the Aircrafts are initially treated as individual clusters and are successively merged into larger clusters as the algorithm progresses. The dendrogram represents the relationships between the clusters based on their similarity. The bottom of the dendrogram represents the individual objects, while the top represents the merged clusters.

The algorithm computes the similarity (or distance) between each pair of clusters, merges the two closest clusters into a single cluster and repeat this until all the data points are in a single cluster.

Comparing the physical features of Airbus A330-200 (3L2) and Airbus A330-300 to visualize similarity as they are clustered together in the first step of the Hierarchical Clustering process :

In [79]:
```
df.query('Aircraft in ("Airbus A330-200 (3L2)", "Airbus A330-300")').T
```

Out[79]:

Aircraft	Airbus A330-200 (3L2)	Airbus A330-300
Seat Width (Club)	0.0	0.0
Seat Pitch (Club)	0	0
Seat (Club)	0	0
Seat Width (First Class)	0.0	0.0
Seat Pitch (First Class)	0.0	0.0
Seats (First Class)	0	0
Seat Width (Business)	21.0	20.0
Seat Pitch (Business)	80.0	60.0
Seats (Business)	34	34
Seat Width (Eco Comfort)	18.0	18.0
Seat Pitch (Eco Comfort)	35.0	35.0
Seats (Eco Comfort)	32	32
Seat Width (Economy)	18.0	18.0
Seat Pitch (Economy)	30.5	30.5
Seats (Economy)	168	232
Accommodation	243	298

Cruising Speed (mph)	531	531
Range (miles)	6536	5343
Engines	2	2
Wingspan (ft)	197.83	197.83
Tail Height (ft)	59.83	56.33
Length (ft)	188.67	208.83
Wifi	0	0
Video	1	1
Power	1	1
Satellite	0	0
Flat-bed	0	0
Sleeper	1	1
Club	0	0
First Class	0	0
Business	1	1
Eco Comfort	1	1
Economy	1	1
Clusters	1	1
Aircraft	Airbus A330-200 (3L2)	Airbus A330-300

We can clearly observe that data points that are being clustered in the early leaves of the dendogram have similar physical features.