## Import all the libraries etc. you need

In [1]:

```
import numpy as np
import pandas as pd
import seaborn as sb
import matplotlib.pyplot as plt
import math as math
import sklearn as sklearn
import random as random
import sklearn.metrics
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn import metrics
from sklearn.model_selection import LeaveOneOut
from sklearn.cluster import AgglomerativeClustering
from sklearn.metrics import silhouette_score
from sklearn.metrics import adjusted_rand_score
from scipy.cluster.hierarchy import dendrogram, linkage
```

## Read the data

- Read the ship data from the csv file into a Pandas dataframe.
- This file is already cleaned of outliers and missing values etc. Normally data cleaning is an important part of unsupervised learning, but since it has already been done in the previous exercises, we can start this one with already clean data.

In [2]:

```
# code here..

data = pd.read_csv('shipdata2021_ex4.txt')
print(data)

data['Ship_type'].unique()
```

```
          MMSI    Speed      COG Destination Ship_type  Gross_tonnage  Length  \
0    212209000  10.1377  64.3074      Hamina     Cargo           3416   94.91   
1    212436000  13.5256  77.0755      Hamina    Tanker           6280  116.90   
2    219082000   9.9416  74.6762      Hamina    Tanker           9980  141.20   
3    219083000  11.6038  74.7529      Hamina    Tanker           9980  141.20   
4    219426000  11.9203  56.3253      Hamina    Tanker           3219   99.90   
..         ...      ...      ...         ...       ...            ...     ...   
129  273374820  10.0396  74.6253     Vysotsk    Tanker           4979  139.90   
130  273385070   9.3507  74.5454     Vysotsk    Tanker           4979  139.90   
131  273388150   9.7668  68.7159     Vysotsk    Tanker           5075  140.85   
132  636092755  11.1554  73.7013     Vysotsk    Tanker          23240  183.00   
133  357100000  11.2703  59.3888     Vysotsk     Cargo          43717  229.04   

     Breadth  
0      15.34  
1      18.00  
2      21.90  
3      21.60  
4      15.00  
..       ...  
129    16.70  
130    16.94  
131    16.86  
132    27.37  
133    32.31  
```

```
[134 rows x 8 columns]
```

Out[2]:
```
array(['Cargo', 'Tanker', 'Tug'], dtype=object)
```

# Part 1: Preprocess and visualize the data

- Use "Speed", "COG", "Length", and "Gross_tonnage" as features for this exercise. You will also need the 'Ship_type' -column later to be used as labels for evaluating the performance of the clustering algorithm.
- Perform z-score standardization on the features to ensure that all features have the same scale.
- Project the data to two dimensions by using principal component analysis (PCA).

In [3]:
```
# code here...

zdata = data.copy()

zdata['Speed'] = (zdata['Speed'] - zdata['Speed'].mean()) / zdata['Speed'].std()
zdata['Length'] = (zdata['Length'] - zdata['Length'].mean()) / zdata['Length'].std()
zdata['COG'] = (zdata['COG'] - zdata['COG'].mean()) / zdata['COG'].std()
zdata['Gross_tonnage'] = (zdata['Gross_tonnage'] - zdata['Gross_tonnage'].mean()) /


zdata = zdata.drop(['Breadth'], axis=1)
zdata = zdata.drop(['MMSI'], axis=1)
zdata = zdata.drop(['Destination'], axis=1)

numericaldata = zdata.copy()
numericaldata = numericaldata.drop(['Ship_type'], axis=1)

pcaz = PCA(n_components=2)
pcazcomps = pcaz.fit_transform(numericaldata)
```

# Part 2: Perform clustering on the data and evaluate the results using silhouette score

- What is the significance of the linkage criterion in a hierarchical clustering algorithm?
- Perform agglomerative hierarchical clustering on the data, trying different values for the "linkage" parameter. Use the actual number of different ship types for the number of clusters to find and default values for other parameters.
- Use the z-score standardized 4-dimensional data for the clustering - not the PCA-transformed data!
- Evaluate the clustering performance for each linkage criterion using a metric called "silhouette score". What does silhouette score quantify and how is it computed?

In [4]:
```
# code here...

aggcluster = AgglomerativeClustering(n_clusters=3,linkage="ward")
aggcluster.fit_predict(numericaldata)
```

```
aggcluster1 = AgglomerativeClustering(n_clusters=3,linkage="complete")
aggcluster1.fit_predict(numericaldata)

aggcluster2 = AgglomerativeClustering(n_clusters=3,linkage="average")
aggcluster2.fit_predict(numericaldata)

aggcluster3 = AgglomerativeClustering(n_clusters=3,linkage="single")
aggcluster3.fit_predict(numericaldata)

score = silhouette_score(numericaldata, aggcluster.labels_)
score1 = silhouette_score(numericaldata, aggcluster1.labels_)
score2 = silhouette_score(numericaldata, aggcluster2.labels_)
score3 = silhouette_score(numericaldata, aggcluster3.labels_)



print('Silhouette score (linkage ward): %.3f' % score)
print('Silhouette score (linkage complete): %.3f' % score1)
print('Silhouette score (linkage average): %.3f' % score2)
print('Silhouette score (linkage single): %.3f' % score3)
```

```
Silhouette score (linkage ward): 0.440
Silhouette score (linkage complete): 0.264
Silhouette score (linkage average): 0.471
Silhouette score (linkage single): 0.292
```

### Part 2 : Answers here:

Linkage criteria defines the distance used between sets as function. Silhouette score tells how good the clustering technique is. It measures objects similarness to it's own cluster compared to other clusters. Silhouette score is calculated using the distance metric used in clustering.

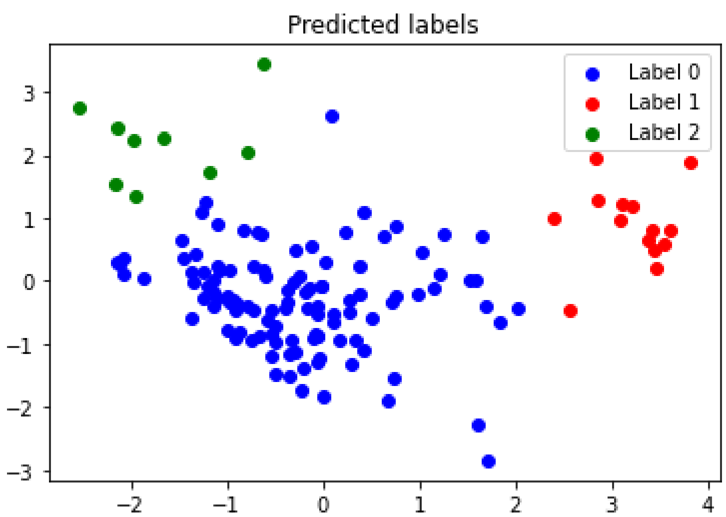
## Part 3a: Compare the clusters with the true labels

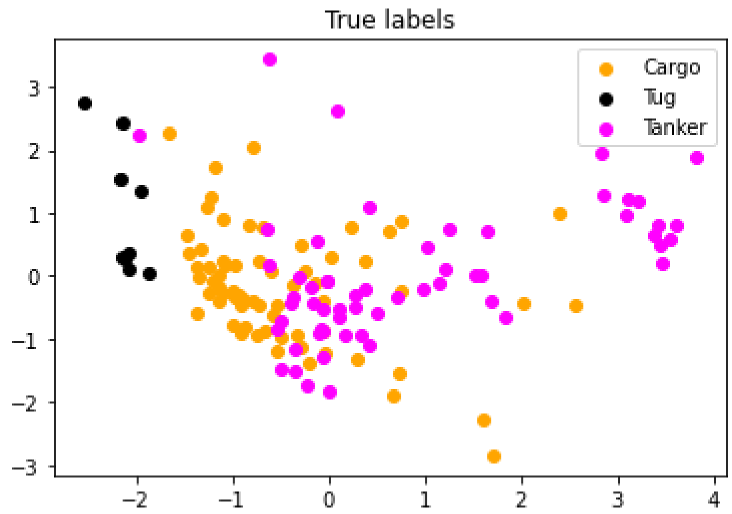
- If you performed the previous steps as instructed, the "average" linkage criterion should be the best performing linkage criterion (that is, with respect to the silhouette score).
- Perform agglomerative hierarchical clustering on the (z-score standardized, not pca-transformed) data using the "average" linkage criterion and the number of different ship types for the number of clusters to find. Again, use default values for other parameters. Visualize the clusters with a scatterplot by performing PCA transformation to two dimensions and color the scatterplot based on the predictions produced by the clustering algorithm.
- Visualize the data again using PCA, this time coloring the scatter plot based on the true class labels. Compare the two scatter plots: how well do the clusters found by the clustering algorithm match the true classes? Place the two scatter plots so that they can easily be compared (e.g. in subplots next to each other in the same figure).
- Based on the visual comparison between the clusters and true classes, would you say that the clustering was successful?
- For an objective evaluation of the clustering, compute the adjusted rand score (use the scikit-learn implementation) using the true labels and the labels predicted by clustering algorithm. How do you interpret the result?

- If the results seem unimpressive, don't get discouraged - clustering "real life" data sets is a difficult task, and a low rand score does not necessarily mean that you have made a mistake.

In [5]:

```
# code here...

labels = aggcluster2.fit_predict(numericaldata)

label0 = pcazcomps[labels == 0]
label1 = pcazcomps[labels == 1]
label2 = pcazcomps[labels == 2]

ndata = zdata.copy()

ndata = ndata['Ship_type']
ndata = ndata.to_numpy()


labels2 = ndata
labelCargo = pcazcomps[labels2 == 'Cargo']
labelTug = pcazcomps[labels2 == 'Tug']
labelTanker = pcazcomps[labels2 == 'Tanker']

plt.subplot(1,1,1)
plt.scatter(label0[:,0], label0[:,1], color='blue', label="Label 0")
plt.scatter(label1[:,0], label1[:,1], color='red', label="Label 1")
plt.scatter(label2[:,0], label2[:,1], color='green', label="Label 2")
plt.title("Predicted labels")
plt.legend()
plt.show()
plt.subplot(1,1,1)
plt.scatter(labelCargo[:,0], labelCargo[:,1], color='orange', label="Cargo")
plt.scatter(labelTug[:,0], labelTug[:,1], color='black', label="Tug")
plt.scatter(labelTanker[:,0], labelTanker[:,1], color='magenta', label="Tanker")
plt.title("True labels")
plt.legend()
plt.show()

randscore = adjusted_rand_score(labels2, labels)
print('rand score: %.3f' % randscore)
```






```
rand score: 0.084
```

### Part 3a : Answers here:

It seems like real data is not as clearly clustered, so the clusterer did not really do a good job when comparing with true values.

Rand score was also pretty low, around 8,4 %.

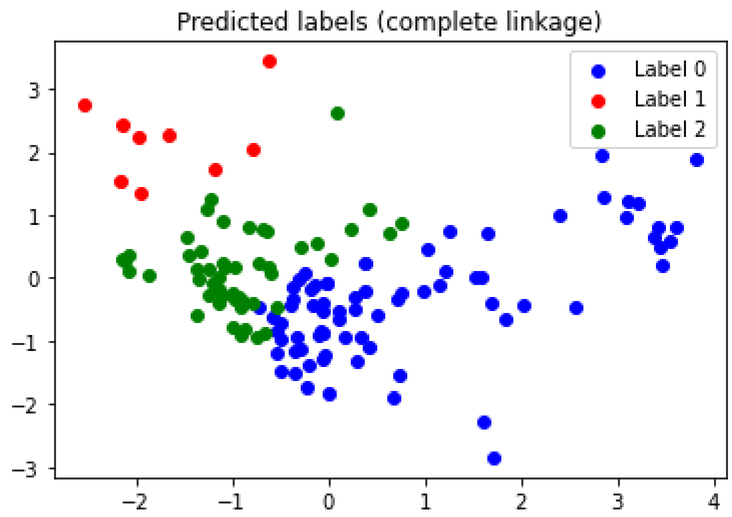
## Part 3b: Another linkage criterion

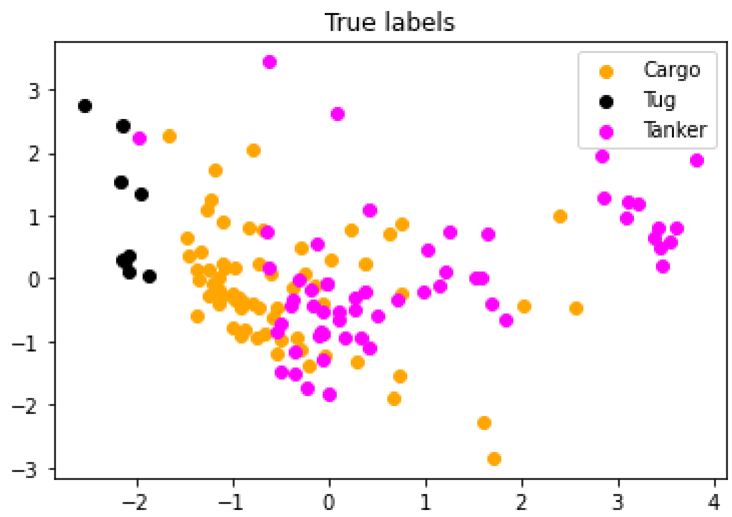
- Perform the same steps as in the previous task (3a), but this time using the "complete" linkage criterion. Visualize the clusters (predicted labels vs. the real labels) and compute the adjusted rand score for the predictions.
- Which linkage criterion performs better based on visual inspection and the adjusted rand score? How do the two criteria differ from each other?
- Compare the formulas for adjusted rand score and silhouette score. Can you explain (briefly) why a given linkage criterion can perform relatively well with respect to one metric and badly w.r.t. the other one?

In [6]:

```
# code here...

labelsc = aggcluster1.fit_predict(numericaldata)

labelc0 = pcazcomps[labelsc == 0]
labelc1 = pcazcomps[labelsc == 1]
labelc2 = pcazcomps[labelsc == 2]

plt.subplot(1,1,1)
plt.scatter(labelc0[:,0], labelc0[:,1], color='blue', label="Label 0")
plt.scatter(labelc1[:,0], labelc1[:,1], color='red', label="Label 1")
plt.scatter(labelc2[:,0], labelc2[:,1], color='green', label="Label 2")
plt.title("Predicted labels (complete linkage)")
plt.legend()
plt.show()
plt.subplot(1,1,1)
plt.scatter(labelCargo[:,0], labelCargo[:,1], color='orange', label="Cargo")
plt.scatter(labelTug[:,0], labelTug[:,1], color='black', label="Tug")
plt.scatter(labelTanker[:,0], labelTanker[:,1], color='magenta', label="Tanker")
plt.title("True labels")
plt.legend()
```

```
plt.show()

randscore = adjusted_rand_score(labels2, labelsc)
print('rand score: %.3f' % randscore)
```




```
rand score: 0.283
```

### Part 3a : Answers here:

This linkage criterion seems to work better based on visuals and rand score, which was around 28,3% this time.

With linkage criterion being complete, it allows elements in different clusters be closer to each other than using the average. Since in this data ships of different types can be really similar to each other, it works out better for this data. Silhouette score compares how similar a cluster is to other cluster. In average the clusters are more separate than in complete (and real data) so it gives better silhouette score where clusters can be really similar to each others.

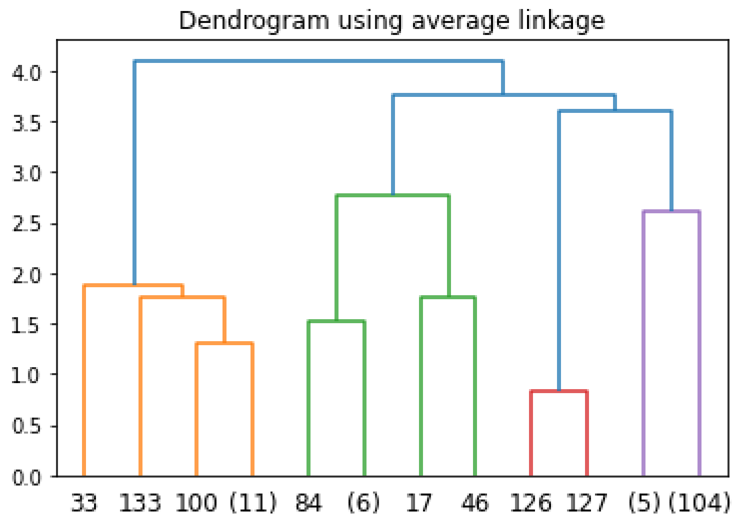
## Part 4: Plot the dendrogram

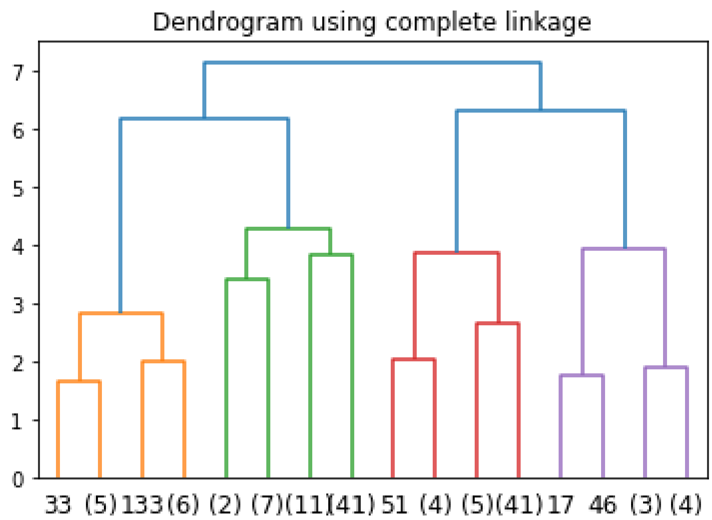
- As the last step, plot dendrograms to visualize the merging processes.
- For this you will need a linkage matrix - while you can extract one from a fitted AgglomerativeClustering object, it is much easier to use the scipy implementation (scipy.cluster.hierarchy.linkage).
- Compute the linkage matrix using both average and complete linkage, and plot the dendrograms using scipy.cluster.hierarchy.dendrogram). Truncate the dendrogram so that

three levels of the dendrogram tree are visible for better readability.

- How do you interpret the dendrograms? How do they differ?

In [7]:
```
# code here...

average_m = linkage(numericaldata, 'average')

plt.figure()
dgram_average = dendrogram(average_m, truncate_mode='level', p=3)
plt.title("Dendrogram using average linkage")
plt.show()

complete_m = linkage(numericaldata, 'complete')
plt.figure()
dgram_complete = dendrogram(complete_m, truncate_mode='level', p=3)
plt.title("Dendrogram using complete linkage")
plt.show()
```




### Part 4 : Answers here:

With complete relation is kind split into two at the highest level that do split also to two and then to two again. With average the splits are more inequal. There are move leaves when using complete than average. In complete clusters are more similar to each other than with average.