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
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
```

Data import

In [2]:
```
data_path = "https://raw.githubusercontent.com/vajnie/DADK_2021/main/shipdata_2021.c
```

In [3]:
```
#Import the data here
data = pd.read_csv(data_path) #data is presumed to be in the same folder.
```

---

1) Data preprocessing

Below are all the questions related to this topic, but you should put them under their own respective titles that are found below.

- a) First, find out how many different destinations there are in the data. Do you need to make any preprocessing? **1p**
- b) Destination harbor is a categorical variable. It needs to be converted into a numerical representation. Explain, why do you need to make this step? You can use get_dummies from pandas to implement one-hot coding for categorical features **1p**
- c) Plot Gross tonnage versus the ship Length. Use different colors for different ship types. According to the plot, there is one **clear outlier.** Correct the outlying value by changing it to the value 326 and rerun the plot after you have made the correction. **1p**
- d) It is good to exploit domain knowledge and make some reasonable transformation to the feature values to improve the expected results and/or to avoid redundancy. Find out what gross tonnage means. Make some transformation to Length values to acquire a linear relationship between the transformed length and Gross tonnage values. You can find the needed information https://en.wikipedia.org/wiki/Gross_tonnage. Look at the formula and think how you can get the two variables similarly scaled. If you are having trouble, just experiment with different transformations before asking help. By plotting you can see what the relationship looks like after a transformation you have tried **1p**

- e) The numerical variables have quite different ranges. To ensure that all variables can have the same importance on the model, perform Z-score standardization. Perform it for speed, the ***transformed length*** variable, and breadth **1p**

---

a) Find out how many different destinations there are in the data. Are there any mistakes and do you need to do any preprocessing? Give your code and answer below **1p**

In [4]:
```
### Code for 1.a)
#number of different destinations
data.Destination.unique().size

#Sillamäe and Sillamae are most likely supposed to be the same

datac = data.replace("Sillamae","Sillamäe", regex = True)

datac.Destination.unique().size
```

Out[4]:
```
16
```

* **Answer here** *

```
Sillamäe and Sillamae are supposed to be same destination. I fixed instances
of Sillamae to Sillamäe. So there are 16 unique destinations
```

---

b) Destination harbor is a categorical variable. It needs to be somehow converted into a numerical expression. Explain, why do you need to make this step?

You can use get_dummies from pandas to implement onehot coding for categorical features **1p**

In [5]:
```
### Code for 1.b)

datacc = pd.get_dummies(datac['Destination'])

datacc = pd.concat([datac, datacc], axis=1)
datacc = datacc.drop(["Destination"], axis=1)
```

* **Answer here** *

```
Machine Learning methods require input and output data to be numerical.
```

---

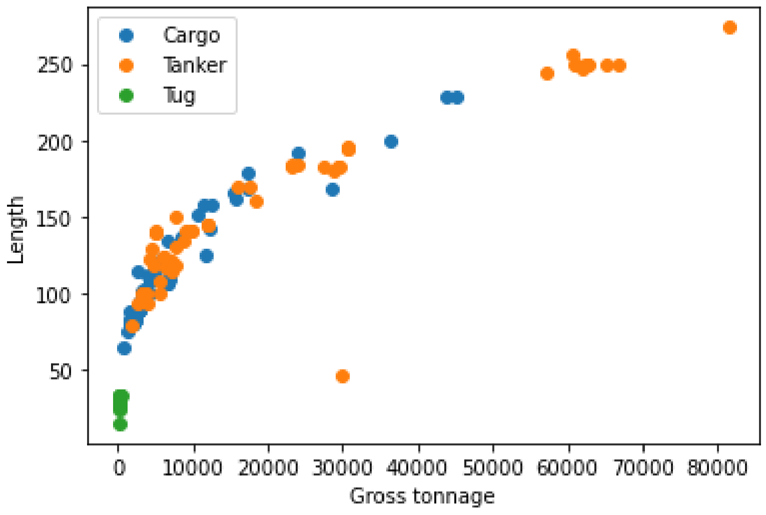
c) Plot Gross tonnage versus the ship Length.

- Use different colors for different ship types. According to the plot, there is one **clear** outlier. **Correct the outlying value by changing it to the value 326** and rerun the plot after you have made the correction. **1p**
- If you want to do things very nicely you could wrap the plotting in a function so you don't paste the same code twice, but this is not required.

In [6]:
```
### Code for 1 c) plot with the outlier

ship_types = datacc.groupby("Ship_type")
```

```
for name, group in ship_types:
    plt.plot(group["Gross_tonnage"], group["Length"],marker="o", linestyle="", label
plt.xlabel("Gross tonnage")
plt.ylabel("Length")
plt.legend()
```

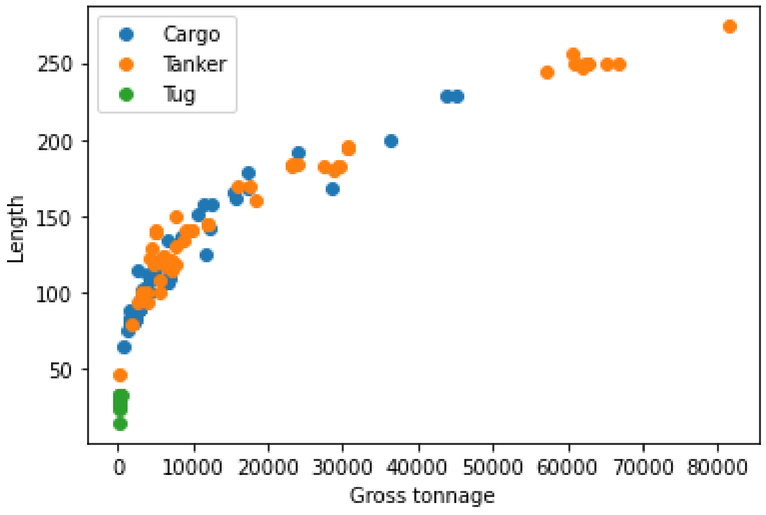
Out[6]: `<matplotlib.legend.Legend at 0x2200b517cd0>`



In [7]:

```
### Code for 1 c) find the outlier and replot

new_data = datacc.copy()
outlier = new_data.index[(new_data['Length'] < 55) & (new_data['Ship_type'] == "Tank

new_data.loc[outlier,"Gross_tonnage"]=326

ship_types1 = new_data.groupby("Ship_type")

for name, group in ship_types1:
    plt.plot(group["Gross_tonnage"], group["Length"],marker="o", linestyle="", label
plt.xlabel("Gross tonnage")
plt.ylabel("Length")
plt.legend()
```

Out[7]: `<matplotlib.legend.Legend at 0x2200b517b80>`

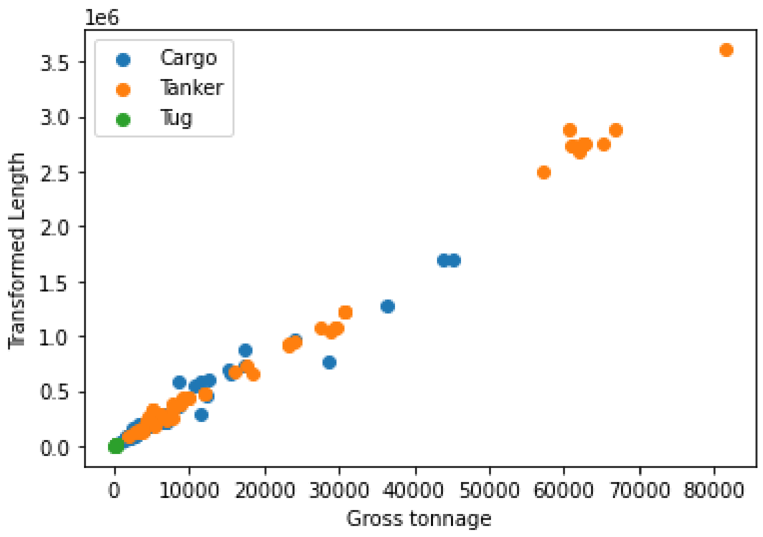


d) Exploit your domain knowledge and transform to improve the expected results and/or to avoid redundancy.

- Find out what gross tonnage means. Make some transformation to Length values to acquire a linear relationship between the transformed length and Gross tonnage values. Plot the situation after the transformation **1p**
- You can find the needed information in for example https://en.wikipedia.org/wiki/Gross_tonnage. Look at the formula and think how you can get the two variables similarly scaled. If you are having trouble, just experiment with different transformations before asking help. By plotting you can see what the relationship looks like after a transformation you have tried

In [9]:
```
### Code for 1d

transformd = new_data.copy()

transformd['LengthT'] = transformd['Length']*transformd['Length']*transformd['Breadt

ship_types2 = transformd.groupby("Ship_type")

for name, group in ship_types2:
    plt.scatter(group["Gross_tonnage"], group["LengthT"], label=name)
plt.xlabel("Gross tonnage")
plt.ylabel("Transformed Length")
plt.legend()
```

Out[9]:
```
<matplotlib.legend.Legend at 0x2200b79cfa0>
```



* **Answer here** * what does gross tonnage mean? What transformation did you do? Why is it useful?

```
I did the transformation of Length^2*Breadth. In the wikipedia article Gross
Tonnage is calculated with the help of ships volume that is in cubic meters.
I think the basis of that would be to multiple ships length, breadth and
height together. Height is not given but I found I got close solution to
linear when using value of length twice. Using length three times also seemed
to work.

Gross Tonnage is a measurement of ships volume. It is basically one way to
determine ships size for ports etc.
```

e) The numerical variables have quite different ranges. To ensure that all variables can have the same importance on the model, perform Z-score standardization. Perform it for speed, transformed length, and breadth **1p**

In [10]:
```
### Code for 1e

zdata = transformd.copy()

zdata['LengthT'] = (zdata['LengthT'] - zdata['LengthT'].mean()) / zdata['LengthT'].s

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

zdata['Breadth'] = (zdata['Breadth'] - zdata['Breadth'].mean()) / zdata['Breadth'].s
```

---

2. Classification

Predict the **ship type** using **speed, destination, transformed length, and breadth** as features. Find an estimation for the classification accuracy (number of correctly classified ships to the total number of ships) using **random training and test sets**.

Below is a summary of exercises for part 2

- a) Produce training and test data **1p**
  - Gather the normalized features and one-hot-coded destination columns as array **X** (input variables), and the ship type as array **y** (output variable)
  - Divide the data randomly into training (80%) and test (20%) sets
  - Do you need to use stratification? **Explain your decision**
- b) Train the model and test its performance **1p**
  - Use kNN classifier with k=3
  - Print out the confusion matrix. How does the model perform with different ship types?
  - What is the (total) classification accuracy?
    - Repeat the calculation 1000 times with different split of training/test data, and make a histogram of the results for classification accuracy **1p**
    - Discuss your results **1p**

---

a) Produce training and test data 1p

- Gather the normalized features and one-hot-coded destination columns as array X (input variables), and the ship type as array y (output variable)
- Divide the data randomly into training (80%) and test (20%) sets
- Do you need to use stratification? **Explain your decision**

In [11]:
```
### Code for 2a

datamodel = zdata.copy()

datamodel.drop(['MMSI', 'COG', 'Length'], axis=1)

datamodelCargo = datamodel.copy()

datamodelCargo = datamodelCargo.loc[datamodelCargo['Ship_type'] == "Cargo"]

datamodelTanker = datamodel.copy()
datamodelTanker = datamodelTanker.loc[datamodelTanker['Ship_type'] == "Tanker"]

datamodelTug = datamodel.copy()

datamodelTug = datamodelTug.loc[datamodelTug['Ship_type'] == "Tug"]

cargo80 = datamodelCargo.sample(frac=0.8, random_state=100)

cargo20 = datamodelCargo.drop(cargo80.index)

tanker80 = datamodelTanker.sample(frac=0.8, random_state=100)

tanker20 = datamodelTanker.drop(tanker80.index)

tug80 = datamodelTug.sample(frac=0.8, random_state=100)

tug20 = datamodelTug.drop(tug80.index)

all80 = pd.concat([cargo80, tanker80, tug80])
all20 = pd.concat([tanker20, cargo20, tug20])
```

*** Answer here (do you need to use strafication? Explain your decision) ***

I used stratification because there is way less samples of Tugs compared to other types of ships. There is otherwise too great of a risk for them not the accounted when splitting data.

---

b) Train the model and test its performance 1p

- Use a kNN classifier with k=3
- Print out the confusion matrix.
- How does the model perform with different ship types? Where do you think the differences come from?
- What is the (total) classification accuracy?

In [12]:

```
### Code for 2b

Xtrain = all80.drop(['Ship_type', 'MMSI', 'COG', 'Length'], axis=1)

Ytrain = all80['Ship_type']

Xtest = all20.drop(['Ship_type', 'MMSI', 'COG', 'Length'], axis=1)

Ytest= all20['Ship_type']

classifier = KNeighborsClassifier(n_neighbors=3)

classifier.fit(Xtrain, Ytrain)

ypred = classifier.predict(Xtest)

matrix = metrics.confusion_matrix(Ytest, ypred)
print(matrix)

classification = metrics.classification_report(Ytest, ypred)
print(classification)
```

```
[[10  3  0]
```

```
 [ 6  6  0]
 [ 0  0  2]]
              precision    recall  f1-score   support

       Cargo       0.62      0.77      0.69        13
      Tanker       0.67      0.50      0.57        12
         Tug       1.00      1.00      1.00         2

    accuracy                           0.67        27
   macro avg       0.76      0.76      0.75        27
weighted avg       0.67      0.67      0.66        27
```

* **Answer here** * - Discuss your results. What can you see? What do you think is relevant?

Accuracy is 67% overall. The best result model had with predicting Tugs. It might be because there way less tugs than other ship types, so there was less chance to go wrong. Cargos and Tankers had quite similar percentages of accuracy and there were almost equal amount of cargos and tankers being 13 and 12 while there was only 2 tugs.

---

3. Classification accuracy using leave-one-out cross validation

Again, predict the **ship type** using **speed, destination, transformed length, and breadth** of the ship as features. Find an estimation for the classification accuracy using *leave-one-out cross validation (LOO CV)*.

- a) Use leave-one-out cross validation to estimate the model performance **1p**
  - Use kNN classifier with k=3
  - What is the classification accuracy? Compare the result with the one you got in the previous task
- b) Which method gives better evaluation of the performance of the classifier with this data set? Explain your choice **1p**

In [13]:
```python
### Code for 3

from sklearn.model_selection import cross_val_score

cv = LeaveOneOut()

Xdata = datamodel.copy()
Xdata = Xdata.drop(['MMSI', 'COG','Ship_type','Length'], axis=1)

ydata = datamodel.copy()
ydata = ydata['Ship_type']

model = KNeighborsClassifier(n_neighbors=3)

scores = cross_val_score(model, Xdata, ydata, scoring="accuracy", cv=cv)

print('Accuracy: %.3f (%.3f)' % (np.mean(scores), np.std(scores)))
```

```
Accuracy: 0.724 (0.447)
```

a) What is the classification accuracy? Compare the result with the one you got in the previous task 1p

*** Answer here ***

```
Accuracy is 72.4%
```

b) Which method gives better evaluation of the performance of the classifier with this data set? Why? 1p

* **Answer here ***

2nd method. Although in my testing the accuracy on 1st method depended on the random seed (It was still ranging around 70%). I think 2nd method gives a better average since this dataset is not that large.