In [1]:
```
# Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

# Show the plots inline in the notebook
%matplotlib inline
```

Know your data: an introduction to the data and domain knowledge

The data used in this exercise is a subset from the Marine Traffic portal. More information available for example here:

- https://www.marinetraffic.com/blog/information-transmitted-via-ais-signal/
- https://www.diva-portal.org/smash/get/diva2:833998/FULLTEXT01.pdf
- https://www.marinetraffic.com/en/data/

The exercise data has the following columns/attributes:

- **MMSI**
  - Maritime Mobile Service Identity. A radio-identification number that uniquely identifies a ship. The first three numbers tell the nationality of the ship - for example finnish ships would have the number 266 preceding them. The following six digits are the identifying part unique to each ship.
- **Speed**
  - The speed (in knots x10) that the subject vessel is reporting according to AIS transmissions
- **COG**
  - Course Over Ground
    COG=3600 means “not available”
- **Destination**
  - The Destination of the subject vessel according to the AIS transmissions
- **Ship_type**
  - The Shiptype of the subject vessel according to AIS transmissions -
- **Gross_tonnage**
  - unitless measure that calculates the moulded volume of all enclosed spaces of a ship
- **Length**
  - The overall Length (in metres) of the subject vessel
- **Breadth**
  - The Breadth (in metres) of the subject vessel

---

1. Data import

Datasets for this exercise are available via the following url-paths

- https://raw.githubusercontent.com/vajnie/DADK_2021/main/shipdata1_2021.csv
- https://raw.githubusercontent.com/vajnie/DADK_2021/main/shipdata2_2021.csv

---

a) First load data files shipdata1.csv and shipdata2.csv using pandas.

- *Note! Files were prepared by two different persons, so there are differences in the file formatting!*

In [2]:

```
# Here are the paths to the files
##################################
# Load shipdata1.csv
url1 = 'https://raw.githubusercontent.com/vajnie/DADK_2021/main/shipdata1_2021.csv'
# Load shipdata2.csv
url2 = 'https://raw.githubusercontent.com/vajnie/DADK_2021/main/shipdata2_2021.csv'
#Print/show in notebook first 5 rows of both dataframes.
```

In [3]:

```
# Your script for 1.a) here

d1 = pd.read_csv(url1)
d2 = pd.read_csv(url2)
```

b) Print/show in notebook first 5 rows of both dataframes.

In [4]:

```
# Script here
print(d1.head(5))
print(d2.head(5))
```

```
        MMSI    Speed      COG Destination Ship_type  Gross_tonnage  Length  \
0  212209000  10.1000  64.3000      Hamina     Cargo           3416    94.9
1  212436000  13.5256  77.0755      Hamina    Tanker           6280   116.9
2  219082000   9.9000  74.7000      Hamina    Tanker           9980   141.2
3  219083000  11.6038  74.8000      Hamina    Tanker           9980   141.2
4  219426000  11.9203  56.3253      Hamina    Tanker           3219    99.9

   Breadth
0     15.3
1     18.0
2     21.9
3     21.6
4     15.0
        MMSI    Speed      COG Destination Ship_type  Gross_tonnage  Length  \
0  538002778  11,3631  74,6552      Porvoo    Tanker          30641     195
1  636016752     11,7     74,6      Porvoo    Tanker           3853    92,9
2  244870429  11,7126  69,5662      Porvoo    Tanker           7251     115
3  305653000  10,8253  56,4266      Porvoo     Cargo           6668  107,03
4  235060255  11,7311     80,9    Primorsk    Tanker          23353   184,0

  Breadth
0   32,24
1    15,3
2    18,6
3   18,42
4    27,7
```

c) For the vessel with **MMSI 231844000**, search for gross tonnage, length and breadth from one of the datasets

In [5]:
```
# Script here
##########

vessel1 = d1.loc[d1['MMSI'] == 231844000]
print(vessel1[['Gross_tonnage' , 'Length', 'Breadth']])
```

```
    Gross_tonnage  Length  Breadth
49           2876    88.9     13.2
```

2. Fix numeric data

a) The dataframes have one systematic difference in numerical values. Look at the previous printouts: **What is the difference?**

* **One uses . to differiate decimals (the international way) while other uses , to differiate decimals (the Finnish way) ***

b) Fix this issue so that you correct shipdata2 dataframe to similar formatting as in shipdata1.

In [6]:
```
# Script here

d2f = d2.replace(',','.', regex = True)
```

c) Print first 5 rows of the now fixed shipdata2 dataframe.

In [7]:
```
# Script here

print(d2f.head(5))
```

```
        MMSI    Speed      COG Destination Ship_type  Gross_tonnage  Length  \
0  538002778  11.3631  74.6552      Porvoo    Tanker          30641     195   
1  636016752     11.7     74.6      Porvoo    Tanker           3853    92.9   
2  244870429  11.7126  69.5662      Porvoo    Tanker           7251     115   
3  305653000  10.8253  56.4266      Porvoo     Cargo           6668  107.03   
4  235060255  11.7311     80.9    Primorsk    Tanker          23353   184.0   

  Breadth  
0   32.24  
1    15.3  
2    18.6  
3   18.42  
4    27.7  
```

3. Combine dataframes together

Tip for this section: Each subtask can be easily done with one (or two) line(s) of code when using Pandas.

a) Add an additional column/attribute Origin which indicates the origin of the data (values 1, 2, according to shipdata name). This is often helpful for possible detective work, if there are any further direpancies in the data.

In [8]:
```
# Script here
d1['Origin']=1
d2f['Origin']=2
```

b) Combine the two separate dataframes as one new dataframe.

In [9]:
```
# Script here

data1 = d1
df = data1.append(d2f)
```

c) Check a sample of the new dataframe by taking a random sample of six rows and printing it.

In [10]:
```
# Script here
print(df.sample(n=6))
```

```
         MMSI    Speed      COG Destination Ship_type  Gross_tonnage  Length  \
23  245241000     11.2     74.5       Kotka     Cargo           2862    94.7
28  210974000     10.3     74.5  Kronshtadt     Cargo           2984    90.0
25  249856000  12.1048  69.8444    Ust-Luga    Tanker          61000   249.9
32  305813000  10.0533  84.8581    Ust-Luga     Cargo           2452   87.84
1   636016752     11.7     74.6      Porvoo    Tanker           3853    92.9
43  305293000   9.8454  77.8443       Muuga     Cargo          15633  161.09

   Breadth  Origin
23    13.4       1
28    15.4       1
25    43.8       2
32    12.9       2
1     15.3       2
43   25.43       1
```

d) Check the shape of the new dataframe, try using df.info(). What information can you find in the output?

In [11]:
```
# Script here
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 136 entries, 0 to 66
Data columns (total 9 columns):
 #   Column         Non-Null Count  Dtype 
---  ------         --------------  ----- 
 0   MMSI           136 non-null    int64 
 1   Speed          136 non-null    object
 2   COG            136 non-null    object
 3   Destination    136 non-null    object
 4   Ship_type      135 non-null    object
 5   Gross_tonnage  136 non-null    int64 
 6   Length         136 non-null    object
 7   Breadth        136 non-null    object
 8   Origin         136 non-null    int64 
dtypes: int64(3), object(6)
memory usage: 10.6+ KB
```

*** There are 136 entries, non of them are null. ***

---

4. Data cleaning

a) Check if there are any missing data.

In [12]:

```
# Script here

df1 = df.loc[df['Origin'] == 1]
df2 = df.loc[df['Origin'] == 2]
print(df1.equals(d1))
print(d2f.equals(df2))
```

```
False
True
```

b) Check if there any duplicate data; any vessel in the dataframe several times?

In [13]:

```
# Script here

print(pd.unique(df['MMSI']).size)
print(len(df.index))
```

```
134
136
```

c) Resolve missing data and remove duplicate data.

In [14]:

```
# Script here
#comparing the lengts and unique mmsi names in the arrays to figure out how many ite
datas = [d1, d2f, df]
datas1 = [d1, d2f]
newdata = pd.concat(datas, ignore_index = True)
newdata2 = pd.concat(datas1, ignore_index = True)

print(len(d1.index))
print(len(d2.index))
print(pd.unique(d1['MMSI']).size)
print(pd.unique(d2['MMSI']).size)
print(len(newdata.index))
print(len(newdata2.index))
print(pd.unique(newdata2['MMSI']).size)

newdata2 = newdata2.drop_duplicates(subset=['MMSI'], keep='first')
```

```
69
67
69
67
272
136
134
```

d) Print out proof that there are no more missing or duplicate data

In [15]:

```
# Script here

newdata2origin1 = newdata2.loc[newdata2['Origin'] == 1]
newdata2origin2 = newdata2.loc[newdata2['Origin'] == 2]
print(pd.unique(newdata2origin1['MMSI']).size)
print(pd.unique(newdata2origin2['MMSI']).size)

ogdatacommon = pd.merge(d1, d2f, on=['MMSI'], how='inner')

print(ogdatacommon)

print('136-134=2, what is the amount of ships that exist in both datasets given. The
```

```
69
65
        MMSI  Speed_x    COG_x Destination_x Ship_type_x Gross_tonnage_x  \
0  538002778  11.3631  74.6552        Porvoo      Tanker           30641
1  636016752  11.7000  74.6000        Porvoo      Tanker            3853

   Length_x  Breadth_x  Origin_x  Speed_y    COG_y Destination_y Ship_type_y  \
0     195.0      32.24         1  11.3631  74.6552        Porvoo      Tanker
1      92.9      15.30         1     11.7     74.6        Porvoo      Tanker

   Gross_tonnage_y Length_y Breadth_y  Origin_y
0            30641      195     32.24         2
1             3853     92.9      15.3         2
136-134=2, what is the amount of ships that exist in both datasets given. The combin
ed list should have 134 uniques ships
```

---

5. Descriptive statistics

a) Check data types and correct if needed

Because Python does not require separate variable declaration, it is always a good practice to check the data types. Check the data types for the attributes and

- convert MMSI to object or string if needed (e.g. to exclude from numeric comparison)
- convert object or string typed numeric attributes to float.

In [16]:

```
# Script here

newdata2['MMSI'] = newdata2['MMSI'].astype('str')

newdata2['Speed'] = newdata2['Speed'].astype('float64')
newdata2['Length'] = newdata2['Length'].astype('float64')
newdata2['Breadth'] = newdata2['Breadth'].astype('float64')
newdata2['COG'] = newdata2['COG'].astype('float64')


print(newdata2.dtypes)
```

```
MMSI              object
Speed            float64
COG              float64
Destination       object
Ship_type         object
Gross_tonnage      int64
Length           float64
Breadth          float64
Origin             int64
dtype: object
```

b) Print count, mean. Std, min, quartiles (25%, 50%, 75%) and max for all numeric attributes

- *This can be done with one line - if your answer gets long consider changing your approach.*

In [17]:

```
# Script here
newdata2.describe()
```

Out[17]:

	Speed	COG	Gross_tonnage	Length	Breadth	Origin
count	134.000000	134.000000	134.000000	134.000000	134.000000	134.000000

	Speed	COG	Gross_tonnage	Length	Breadth	Origin
mean	10.453009	78.271204	13535.291045	136.669776	20.186716	1.485075
std	1.955434	15.731984	18433.591631	124.040522	9.943960	0.501653
min	5.500000	53.326400	100.000000	15.000000	5.000000	1.000000
25%	8.961525	71.053100	2551.250000	88.925000	12.900000	1.000000
50%	10.300000	74.850000	5348.500000	115.000000	16.900000	1.000000
75%	11.709450	81.236900	15558.250000	160.810000	24.880000	2.000000
max	17.082500	157.267300	81502.000000	1399.000000	48.040000	2.000000

c) Choose Breadth and two (2) other numeric attributes that you would like to focus and know more.

In [18]:

```
# Script here

newdata2[['Breadth','Length','Speed']].describe()
```

Out[18]:

	Breadth	Length	Speed
count	134.000000	134.000000	134.000000
mean	20.186716	136.669776	10.453009
std	9.943960	124.040522	1.955434
min	5.000000	15.000000	5.500000
25%	12.900000	88.925000	8.961525
50%	16.900000	115.000000	10.300000
75%	24.880000	160.810000	11.709450
max	48.040000	1399.000000	17.082500

d) Descriptive statistics by Ship_type

- Print the descriptive statistics now by each ship type for those three attributes used in the previous task.
  "by" here means that you group by that variable.
- *Tip! A wide Pandas table can be easily rotated using transpose, for better readability in the jupyter notebook.*

In [19]:

```
# Script here
newdata2.groupby(['Ship_type'])[['Breadth','Length','Speed']].describe()
```

Out[19]:

	Breadth											
	count	mean	std	min	25%	50%	75%	max	count	mean	...	75⁹
Ship_type												
Cargo	67.0	16.841493	5.984697	10.5	12.5	14.4	18.75	32.31	67.0	111.993582	...	123.7
Tanker	57.0	26.023509	11.045589	8.1	16.9	22.2	32.20	48.04	57.0	160.599649	...	195.0
Tug	9.0	8.511111	1.749603	5.0	7.5	9.0	9.80	10.50	9.0	28.555556	...	32.9

3 rows × 24 columns

e) How many ship types there are? Which Ship type has the largest breadth?

*** 3 Ship types. Tanker has the largest breadth***

---

7. Visualizing attribute value distributions

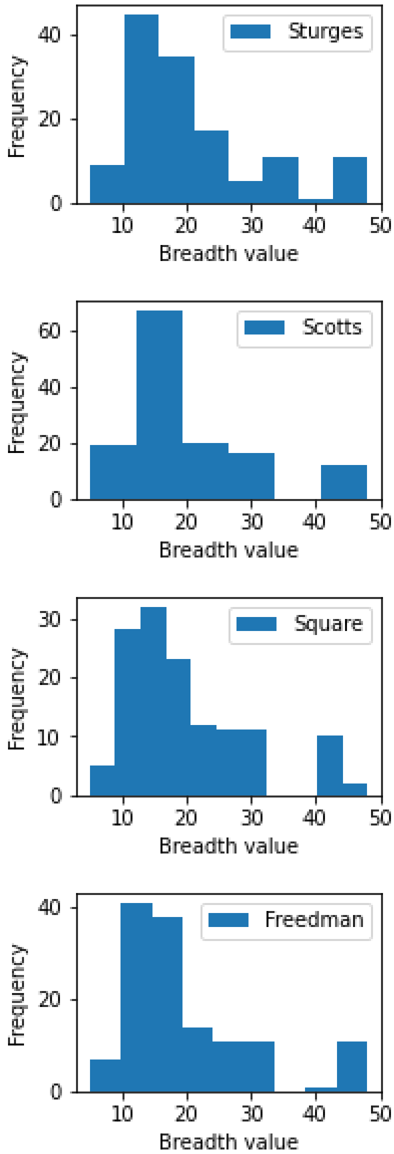
a) Plot four histrograms of the Breadth using the Sturges', Scott's, square root and Freedman-Diaconis' methods to determine the number of bins. How are the numbers of bins calculated? Compare the distributions of different ship types. Do you think this a feasible attribute for classification, why?

- Tip: it would be nice to use subplots when you have more than one plot.

In [20]:

```
### Script here
maxvalue = newdata2['Breadth'].max()
minvalue = newdata2['Breadth'].min()
breadthsd = newdata2['Breadth'].std()


scottsh = (3.49 * breadthsd) / 134**(1/3)
scotts = (maxvalue - minvalue) / scottsh

square = 134**(1/2)

q75, q25 = np.percentile(newdata2['Breadth'], [75,25])
irq = q75 - q25

freedh = 2 * ( irq / 134**(1/3))

freed = (maxvalue - minvalue) / freedh

sturges = 8
##log base 2 134 + 1 and rounded to nearest ceiling = 8

##manual rounding by me the human
scotts = 6

square = 11

freed = 9

plt.subplot(2,2,1)

plt.hist(newdata2['Breadth'], bins=sturges, label="Sturges")
plt.ylabel("Frequency")
plt.xlabel("Breadth value")
plt.legend()
plt.show()

plt.subplot(2,2,2)
plt.hist(newdata2['Breadth'], bins=scotts, label="Scotts")
plt.ylabel("Frequency")
plt.xlabel("Breadth value")
plt.legend()
```

```
plt.show()

plt.subplot(2,2,3)
plt.hist(newdata2['Breadth'], bins = square, label="Square")
plt.legend()
plt.ylabel("Frequency")
plt.xlabel("Breadth value")
plt.show()

plt.subplot(2,2,4)
plt.hist(newdata2['Breadth'], bins = freed, label="Freedman")
plt.ylabel("Frequency")
plt.xlabel("Breadth value")

plt.legend()
plt.show()
```



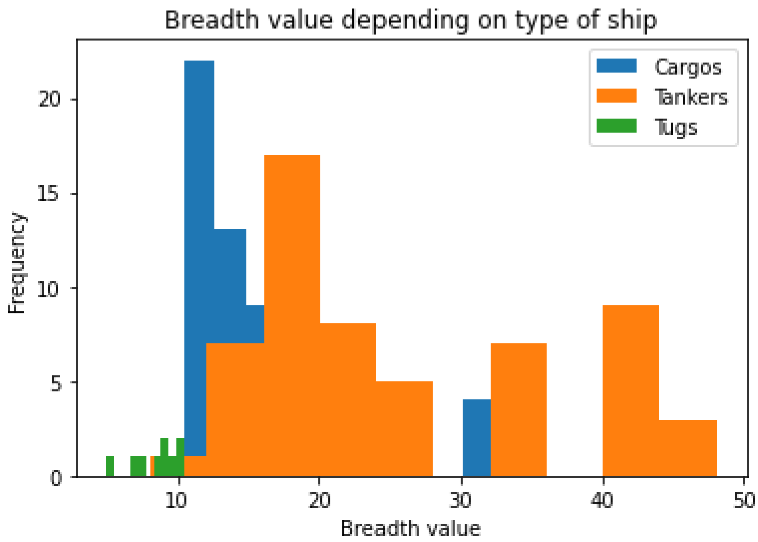
*** Bins are calculated using there https://en.wikipedia.org/wiki/Histogram#Number_of_bins_and_width. Ship type distribution is not possible to detailed with this type of graph, so it is not a good way. ***

b) Compare the distributions of the Breath variable between different ship types. Do you think this a feasible attribute for classification, why?

- *Tip What kind of plot can you do on a continuous variable? You only need to produce one plot, not multiple.*

In [21]:
```
# Script here
cargos = newdata2[newdata2['Ship_type'] == 'Cargo']
tankers = newdata2[newdata2['Ship_type'] == 'Tanker']
tugs = newdata2[newdata2['Ship_type'] == 'Tug']

plt.hist(cargos['Breadth'], label='Cargos')
plt.legend()
plt.hist(tankers['Breadth'], label="Tankers")
plt.legend()
plt.hist(tugs['Breadth'],label='Tugs')
plt.legend()
plt.title("Breadth value depending on type of ship")
plt.xlabel("Breadth value")
plt.ylabel("Frequency")
```

Out[21]:
```
Text(0, 0.5, 'Frequency')
```

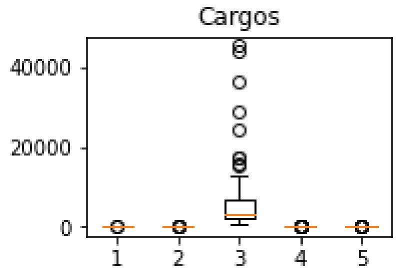


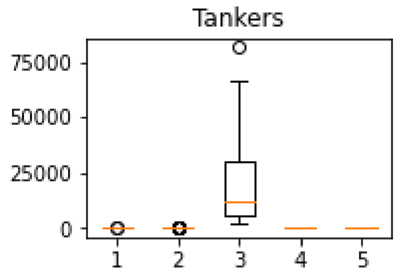
*** Yes, it does tell how breadth values differ between ship types.

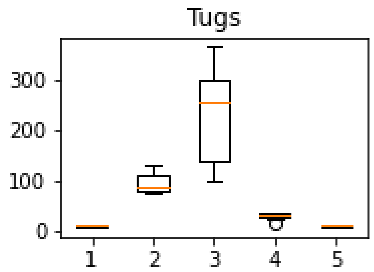
c) Explain what a boxplot is. Plot them for the numeric attributes (excluding 'Origin') grouped by the ship type. Do you see outliers that require some action?

In [22]:
```
# Script here

boxdata1 = cargos.drop(['Origin', 'Destination', 'Ship_type', 'MMSI'], axis=1)
boxdata2 = tankers.drop(['Origin', 'Destination', 'Ship_type', 'MMSI'], axis=1)
boxdata3 = tugs.drop(['Origin', 'Destination', 'Ship_type', 'MMSI'], axis=1)

plt.subplot(2,2,1)
plt.boxplot(boxdata1)
plt.title("Cargos")
plt.show()

plt.subplot(2,2,2)
plt.boxplot(boxdata2)
plt.title("Tankers")
plt.show()

plt.subplot(2,2,3)
plt.boxplot(boxdata3)
```

```
plt.title("Tugs")
plt.show()
```







*** Boxplot is a plot where it groups numerics so they can be compared. It easily displays outliers. There are outliers in every ship type, most in Cargo ships. ***

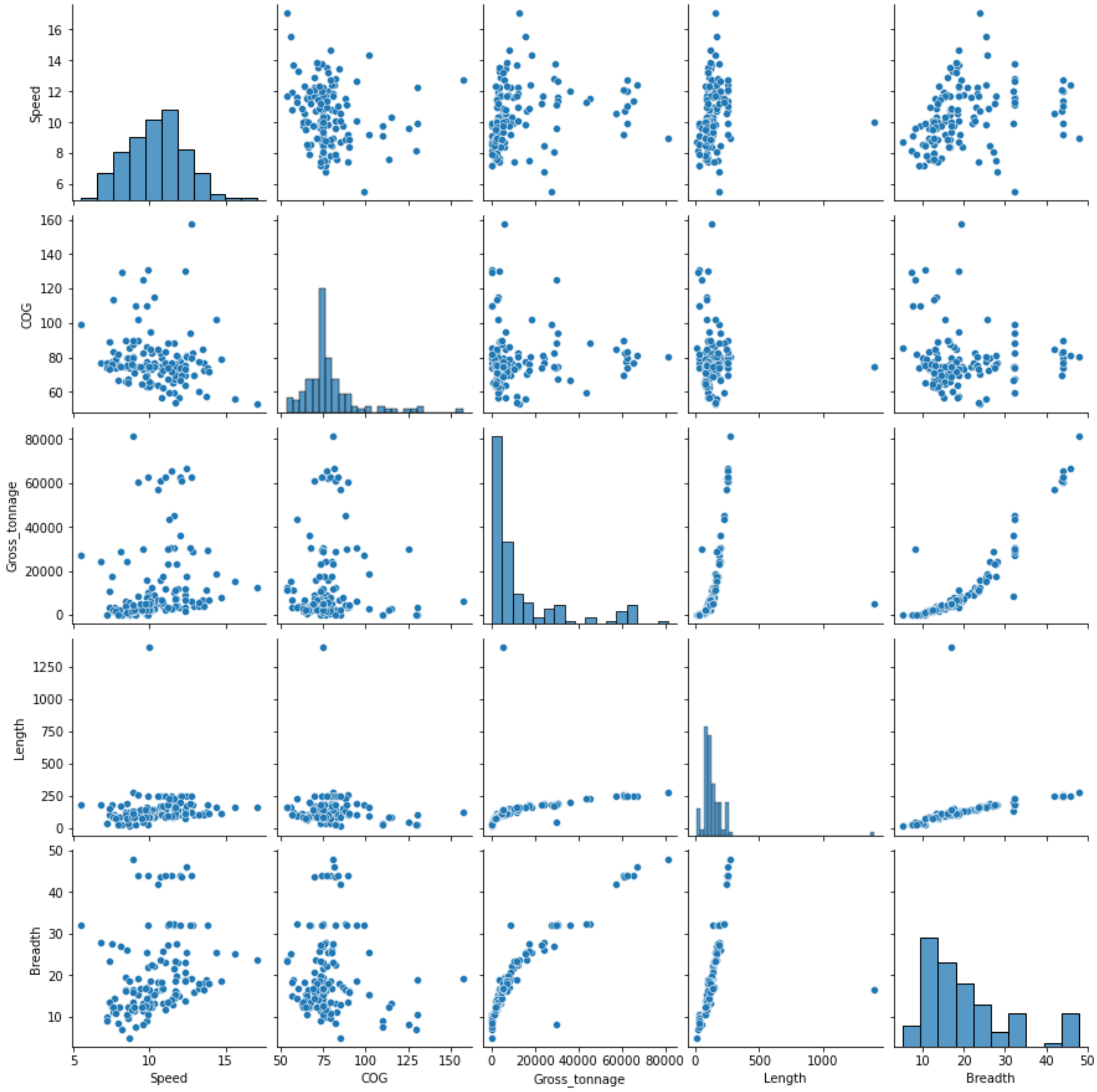
8. Relationships between attributes

a) Plot pairwise scatter plots of the numerical attributes. What kind of relationships can you see? Can you see any outliers?

- *this can be done in one line*

In [23]:
```
# Script here
pairdata = newdata2.drop(['Origin', 'Destination', 'Ship_type', 'MMSI'], axis=1)



sns.pairplot(pairdata)
```

Out[23]:
```
<seaborn.axisgrid.PairGrid at 0x1a53dff0100>
```



*** I can see outliers. Many of the data does grow little bit like exponential functions, while others grow more staticly (? basically only goes up or right). Some of them are really widely spread with hard to make a function to describe.***

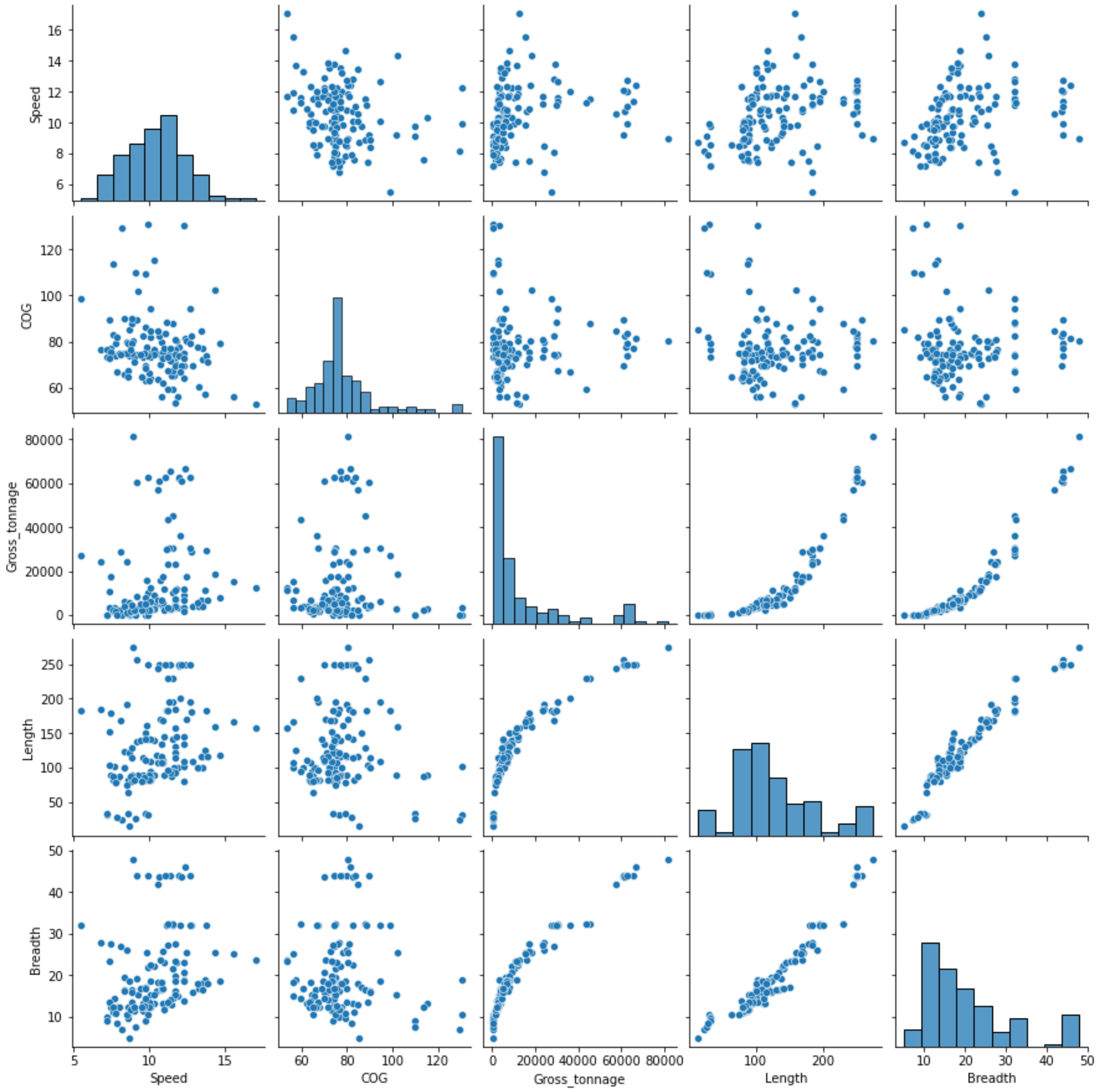
b) Make a new clean dataframe without outlier(s) and replot. What difference do you see?

- *include the most relevant attributes only, or limit to those needed in next task*

In [24]:
```
# Script here
olddata = pairdata.copy()
newdata3 = newdata2.drop(newdata2.index[newdata2['Length'] > 1000], inplace=True)

numericaldata = newdata2.drop(['Origin', 'Destination', 'Ship_type', 'MMSI'], axis=1

numericaldata.drop(numericaldata.index[numericaldata['COG'] > 140], inplace = True)

newdata4 = numericaldata.copy()

numericaldata.drop(numericaldata.index[(numericaldata['Gross_tonnage'] > 20000) & (n



newdata5 = numericaldata.copy()
```

```
numericaldata.drop(numericaldata.index[(numericaldata['Breadth'] > 30) & (numericald
sns.pairplot(numericaldata)
```

Out[24]: <seaborn.axisgrid.PairGrid at 0x1a53e39b130>



*** Now datapoints are more grouped together than before. There are still that seem like outliers in some of plots, but not in others. Like in Gross_tonnage x Speed and Gross tonnage x COG seems to be outliers, but it seems ok in 'reverse' plots***

---

9. Correlation and heatmap

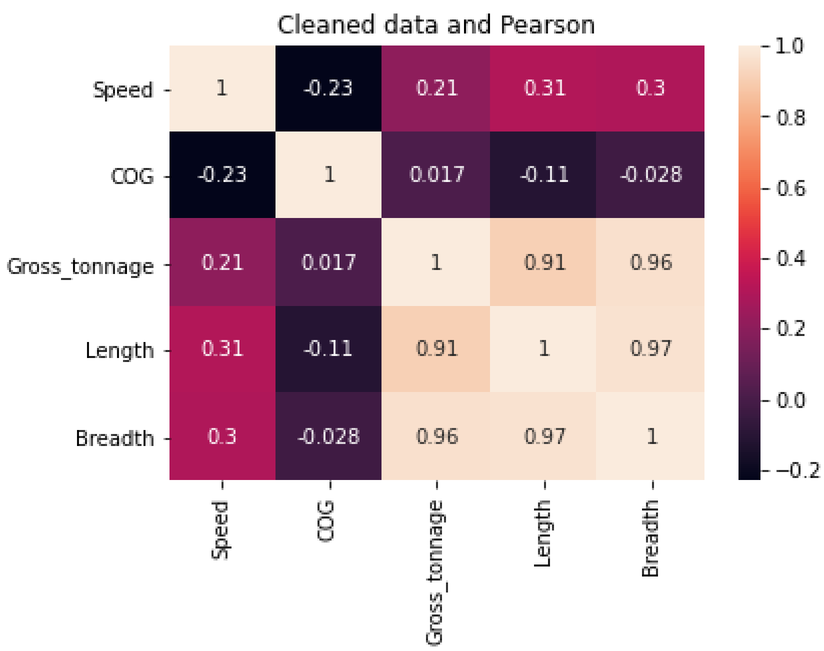
a) Explain what are

- Pearson's correlation
- Spearman's rho and
- Kendall's tau?

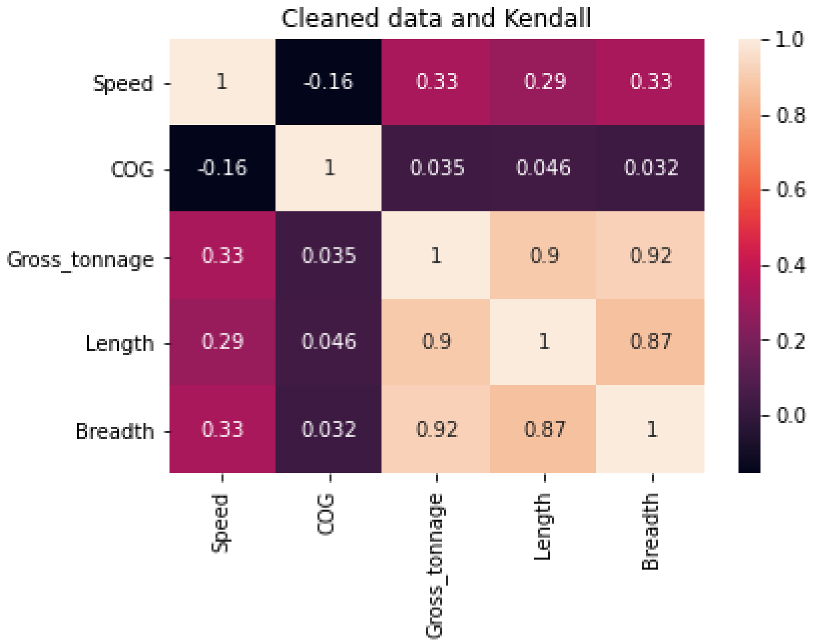
*** Pearson's correlation is the correlation between two linear datasets. Spearmans rho is the correlation between datasets that is not necessarily linear. Kendall's tau correlation will be high when data is similar and low when it is dissimilar.***

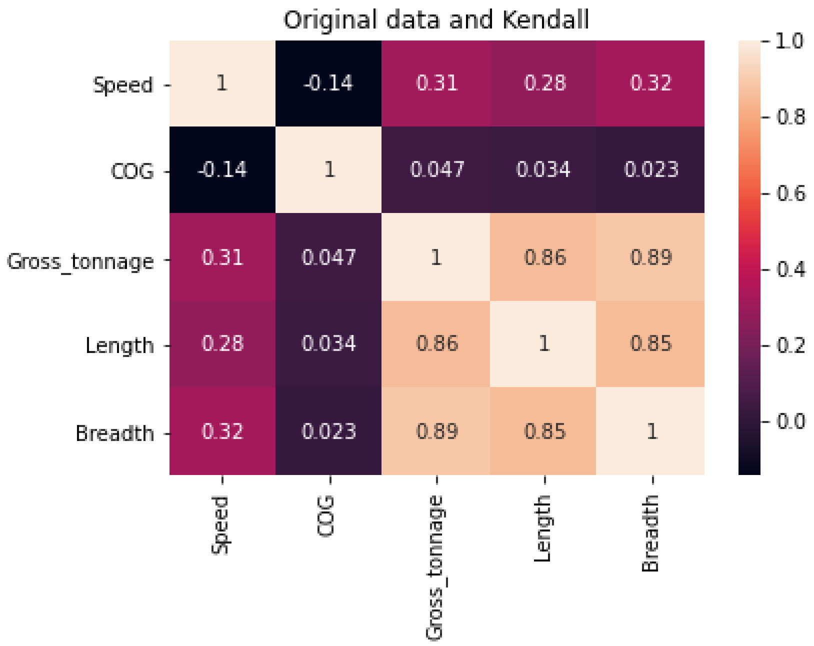
b) Calculate the correlation coefficient matrices. What kind of relationships there are between the attributes? You can use a heatmap to visualize the matrices and more easily see the strength of the relationship. **See what kind of a difference there is between the cleaned dataset and the non-cleaned dataset.**

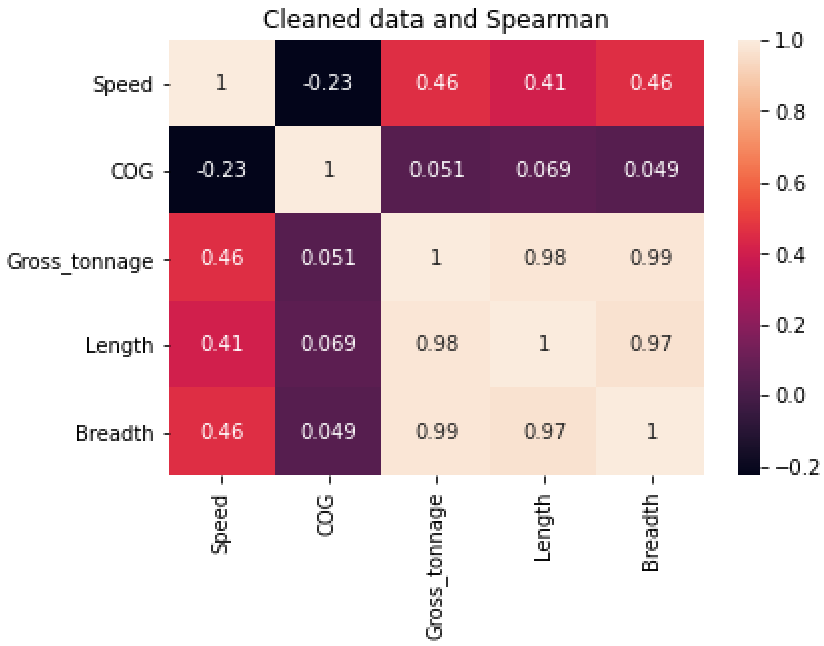
In [25]:
```
#Script here

matrix1 = numericaldata.corr(method='pearson')
matrix2 = numericaldata.corr(method='kendall')
matrix3 = numericaldata.corr(method="spearman")

matrix4 = olddata.corr(method="pearson")
matrix5 = olddata.corr(method="kendall")
matrix6 = olddata.corr(method="spearman")


sns.heatmap(matrix1, xticklabels=matrix1.columns, yticklabels=matrix1.columns, annot
plt.title("Cleaned data and Pearson")
plt.show()

sns.heatmap(matrix4, xticklabels=matrix4.columns, yticklabels=matrix4.columns, annot
plt.title("Original data and Pearson")
plt.show()

sns.heatmap(matrix2, xticklabels=matrix2.columns, yticklabels=matrix2.columns, annot
plt.title("Cleaned data and Kendall")
plt.show()

sns.heatmap(matrix5, xticklabels=matrix5.columns, yticklabels=matrix5.columns, annot
plt.title("Original data and Kendall")
plt.show()

sns.heatmap(matrix3, xticklabels=matrix3.columns, yticklabels=matrix3.columns, annot
plt.title("Cleaned data and Spearman")
plt.show()

sns.heatmap(matrix6, xticklabels=matrix6.columns, yticklabels=matrix6.columns, annot
plt.title("Original data and Spearman")
plt.show()
```

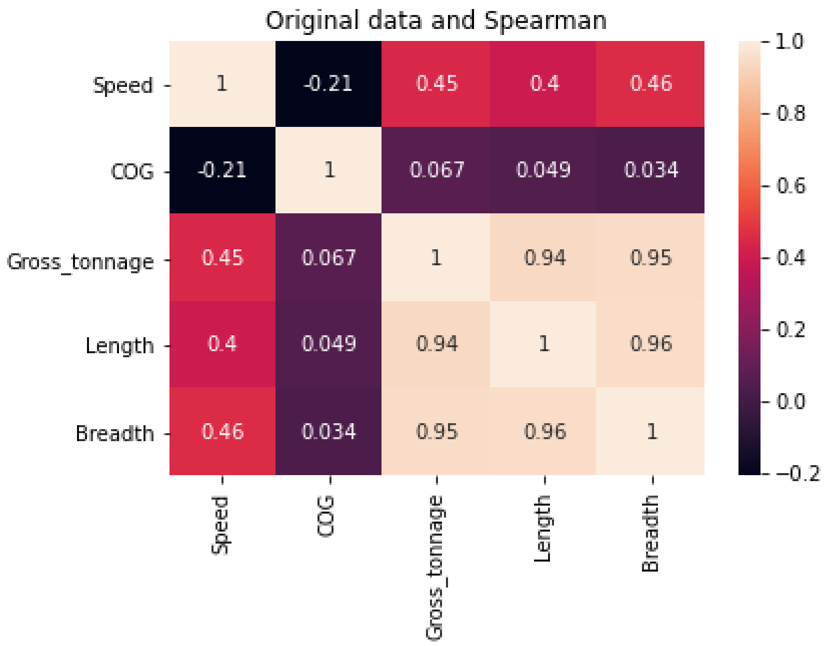



Original data and Pearson
Speed
COG
Gross_tonnage
Length
Breadth
1 -0.16 0.2 0.13 0.3
-0.16 1 0.021 -0.077 -0.057
0.2 0.021 1 0.38 0.94
0.13 -0.077 0.38 1 0.42
0.3 -0.057 0.94 0.42 1
1.0
0.8
0.6
0.4
0.2
0.0


Cleaned data and Kendall
Speed
COG
Gross_tonnage
Length
Breadth
1 -0.16 0.33 0.29 0.33
-0.16 1 0.035 0.046 0.032
0.33 0.035 1 0.9 0.92
0.29 0.046 0.9 1 0.87
0.33 0.032 0.92 0.87 1
1.0
0.8
0.6
0.4
0.2
0.0


Original data and Kendall
Speed
COG
Gross_tonnage
Length
Breadth
1 -0.14 0.31 0.28 0.32
-0.14 1 0.047 0.034 0.023
0.31 0.047 1 0.86 0.89
0.28 0.034 0.86 1 0.85
0.32 0.023 0.89 0.85 1
1.0
0.8
0.6
0.4
0.2
0.0





There is much more lower values in matrixes when using original data compared to cleaned up versions. Meaning there is much less correlation in the original dataset with the outliers intact. Most drastic difference is when using Pearson's correlation as the method

*** Principal component analysis is a way to find parts of the data that will give out the general idea of what the data tells but without losing significant information. It is downsizing to show the general idea and the challenge lays in finding out what data to leave out and what to keep in. ***

- **b)** Do it with and without z-score standardization.

In [26]:
```
# Script: PCA with z-score standardization

pdata = numericaldata.to_numpy()

pdata_z = (pdata - np.mean(pdata, axis= 0)) / np.std(pdata, axis=0)

pdatazf = pd.DataFrame(pdata_z, index=numericaldata.index, columns=numericaldata.col
```

```
pcaz = PCA(n_components=2)
pcacompz = pcaz.fit_transform(pdatazf)

pcaz.components_
```

Out[26]:
```
array([[ 0.2351347 , -0.06071611,  0.54970416,  0.56163108,  0.56870123],
       [ 0.58430233, -0.78175612, -0.1858042 , -0.03923437, -0.1067034 ]])
```

In [27]:
```
# Script: PCA without z-score standardization

pca = PCA(n_components=2)

pdataf = pd.DataFrame(pdata, index=numericaldata.index, columns=numericaldata.column
pcacomps = pca.fit(pdataf)

pca.components_
```

Out[27]:
```
array([[ 2.19171557e-05,  1.28324473e-05,  9.99995830e-01,
         2.84122693e-03,  5.15701939e-04],
       [-2.54554695e-02,  2.38999277e-01,  2.78953925e-03,
        -9.66655025e-01, -8.83269268e-02]])
```

In [28]:
```
# Script: Explore variation
print('Non z-score used:'+format(pca.explained_variance_ratio_))
print('Z-score used:'+format(pcaz.explained_variance_ratio_))
```

```
Non z-score used:[9.99997795e-01 1.71480662e-06]
Z-score used:[0.60286208 0.23353913]
```

*** When I did not calculate z-score, most of data was not lost. When z-score was used component 1 represents around 60 % of data and component 2 23% which means almost 20% of data was lost. When used z-score the data does not wary as much, since it's limits seem to be around -0.8 and 0.6 for both x and y. In the first values can be around -1 to 0.6 in y axis and 0.0 to 1 in x axis. When standardized some of the more domineering outliers are not taken into account, but since it can lose data along the way it might make it harder to get proper idea of the data. When not standardized almost no data was lost, but it was much more one sided with one component representing near 99% of the data. Because of that it is also bad for getting the general idea of the data**