Seeds dataframe for unsupervised learning

May 27, 2023

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

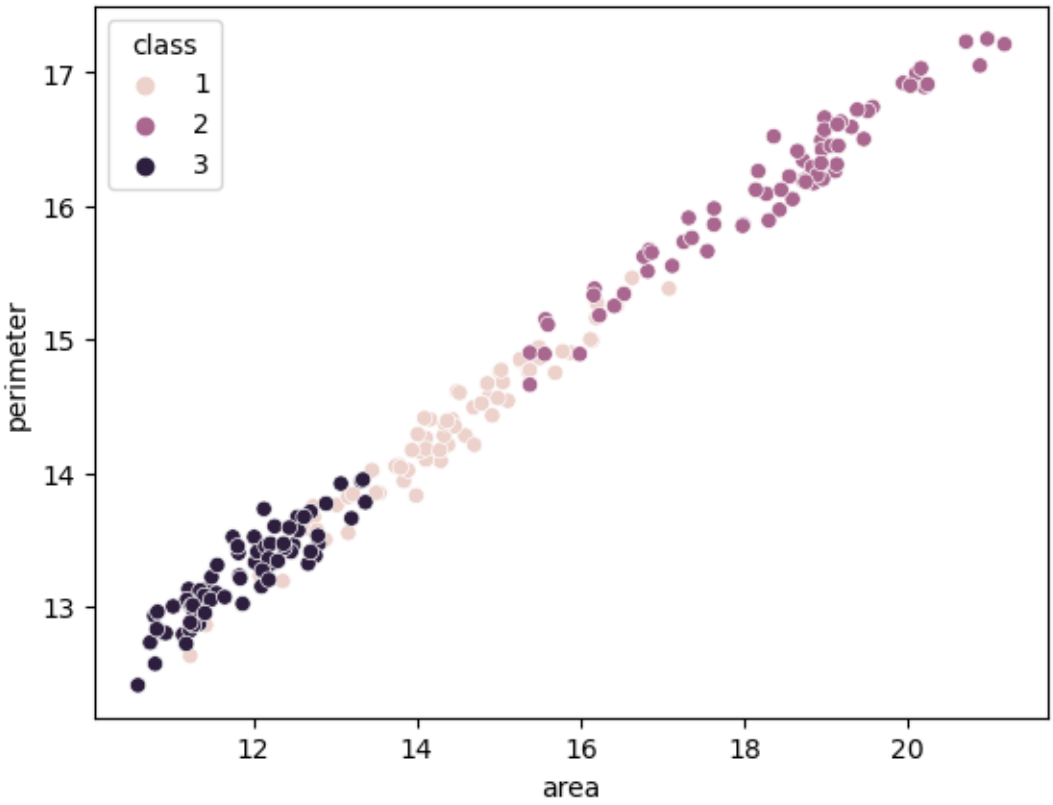
[2]:
```
cols = ["area", "perimeter", "compactness", "length", "width", "asymmetry",␣
 ↪"groove", "class"]

# sep="\s+" is necessary for using spaces as the separator
df = pd.read_csv("seeds_dataset.txt", names=cols, sep="\s+")
```

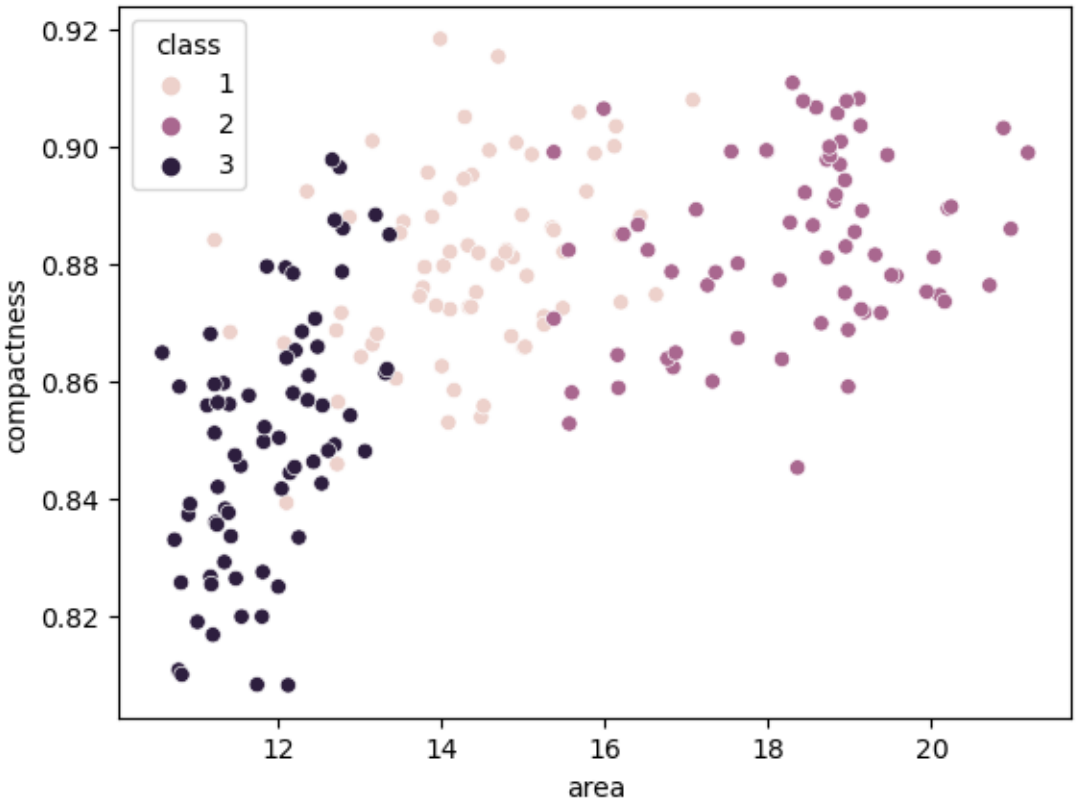
[3]:
```
df.head()
```

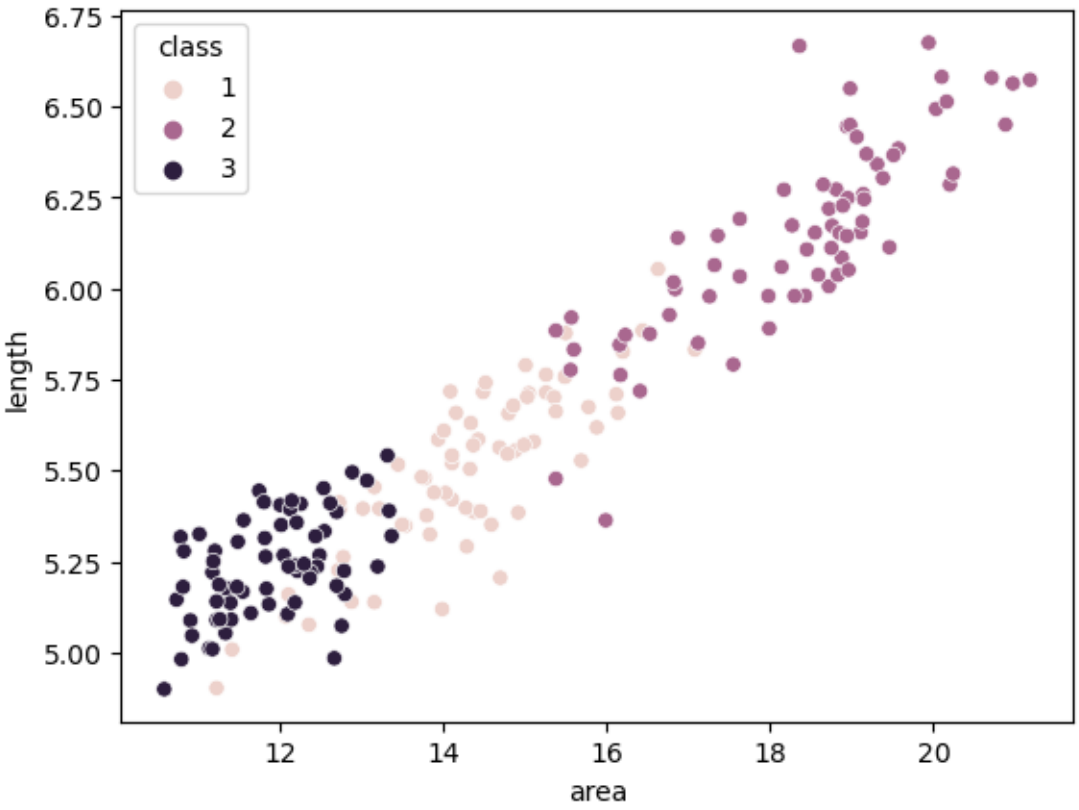
[3]:

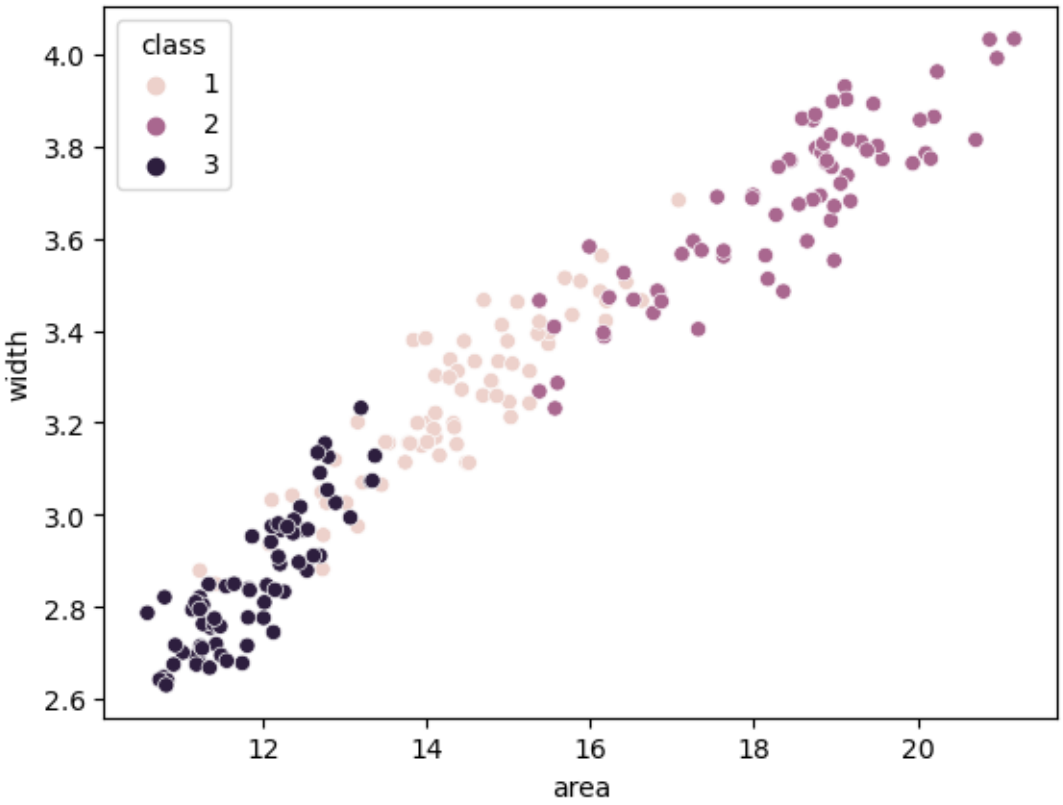
	area	perimeter	compactness	length	width	asymmetry	groove	class
0	15.26	14.84	0.8710	5.763	3.312	2.221	5.220	1
1	14.88	14.57	0.8811	5.554	3.333	1.018	4.956	1
2	14.29	14.09	0.9050	5.291	3.337	2.699	4.825	1
3	13.84	13.94	0.8955	5.324	3.379	2.259	4.805	1
4	16.14	14.99	0.9034	5.658	3.562	1.355	5.175	1

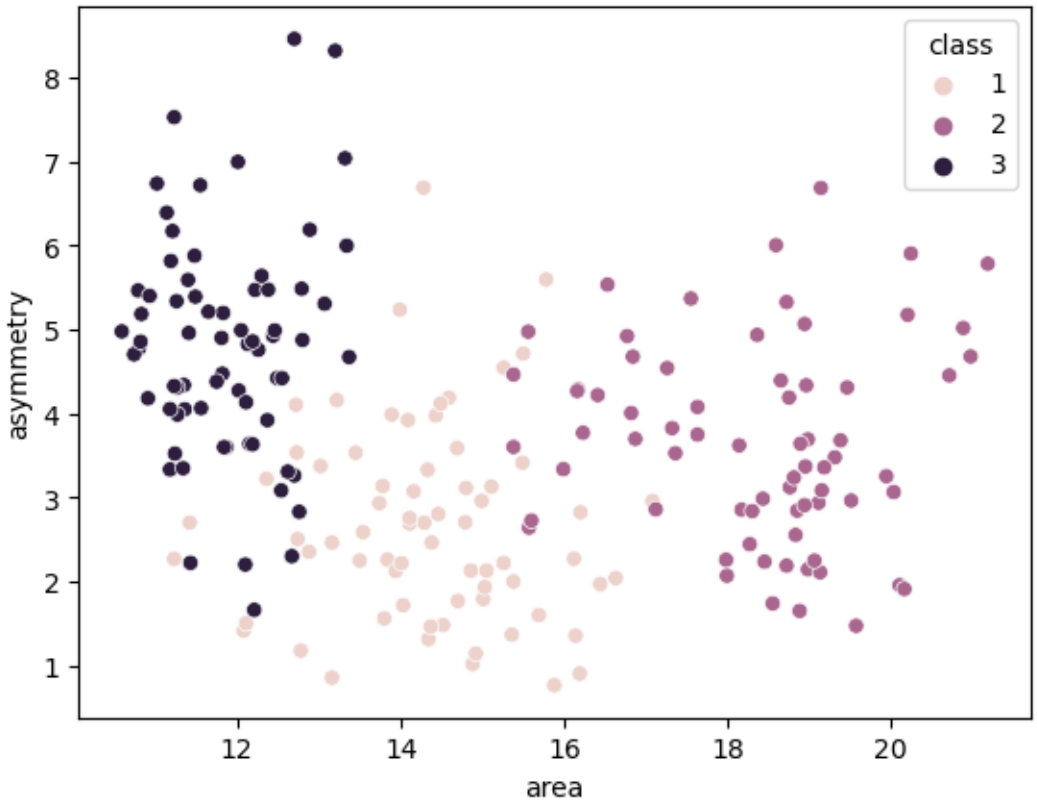
[6]:
```
for i in range(len(cols)-1):
    for j in range(i+1, len(cols)-1):
        x_label = cols[i]
        y_label = cols[j]
        sns.scatterplot(x=x_label, y=y_label, data=df, hue="class")
        plt.show()
```

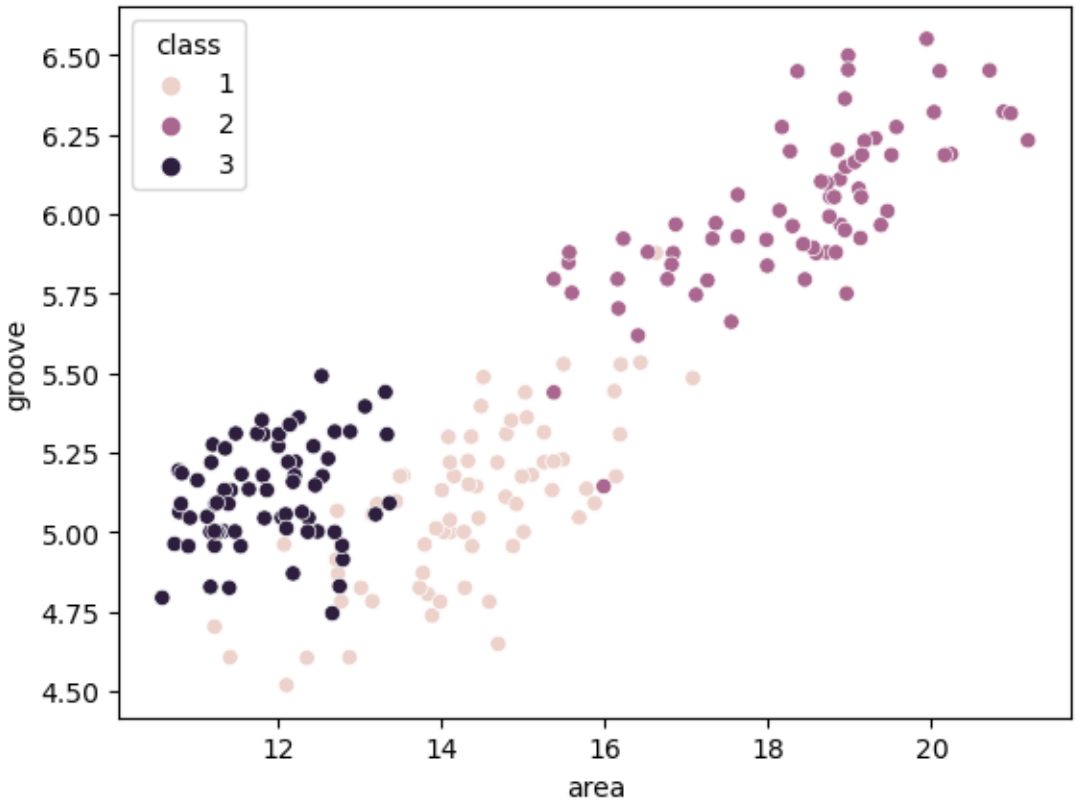

class
1
2
3
perimeter
17
16
15
14
13
12
14
16
18
20
area

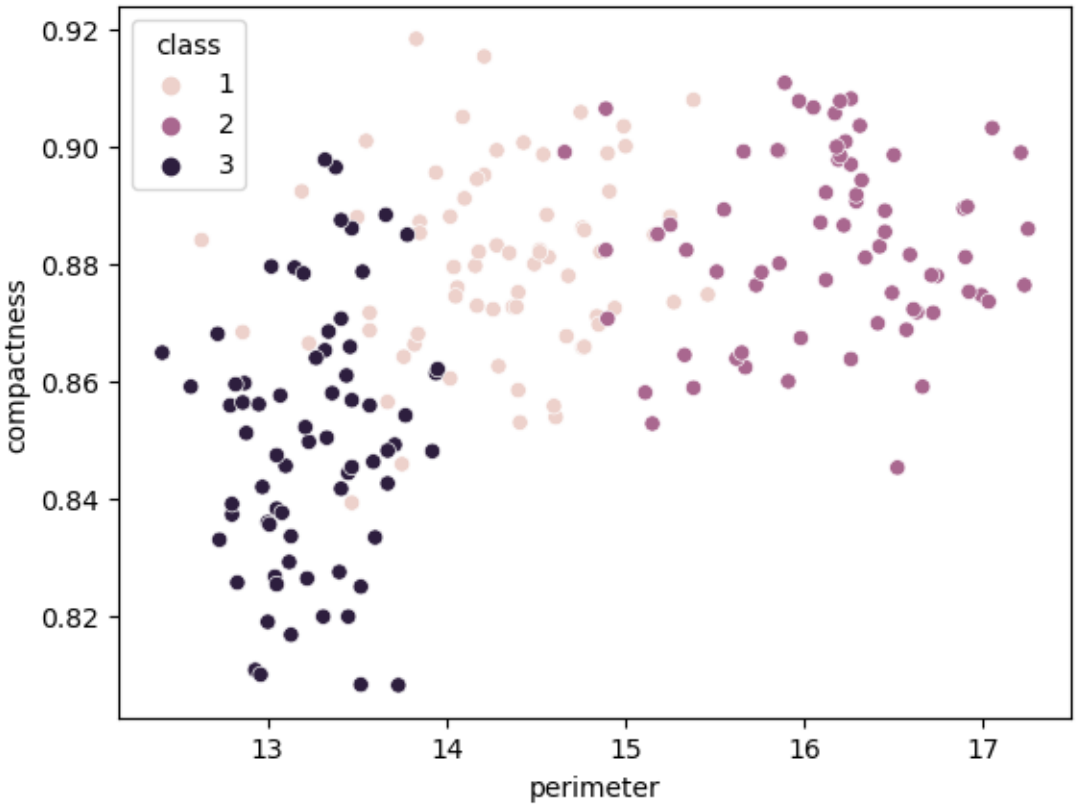

class
1
2
3
compactness
area
0.92
0.90
0.88
0.86
0.84
0.82
12
14
16
18
20

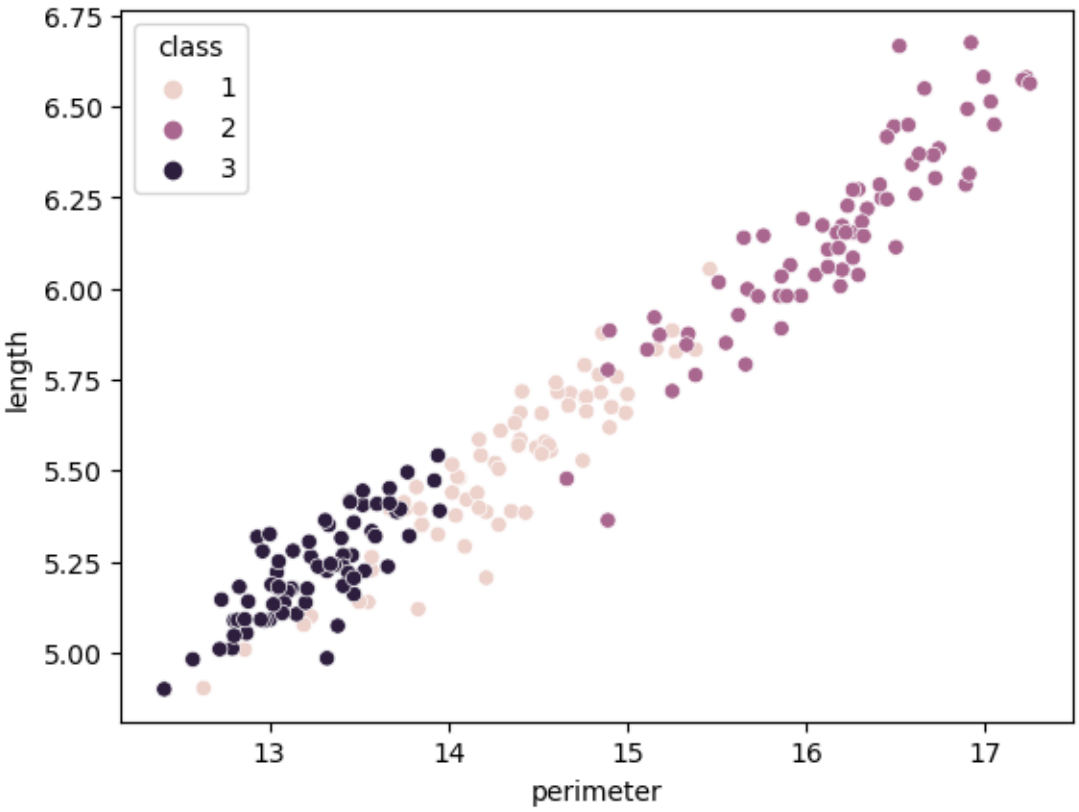

class
1
2
3
6.75
6.50
6.25
6.00
5.75
5.50
5.25
5.00
length
12
14
16
18
20
area

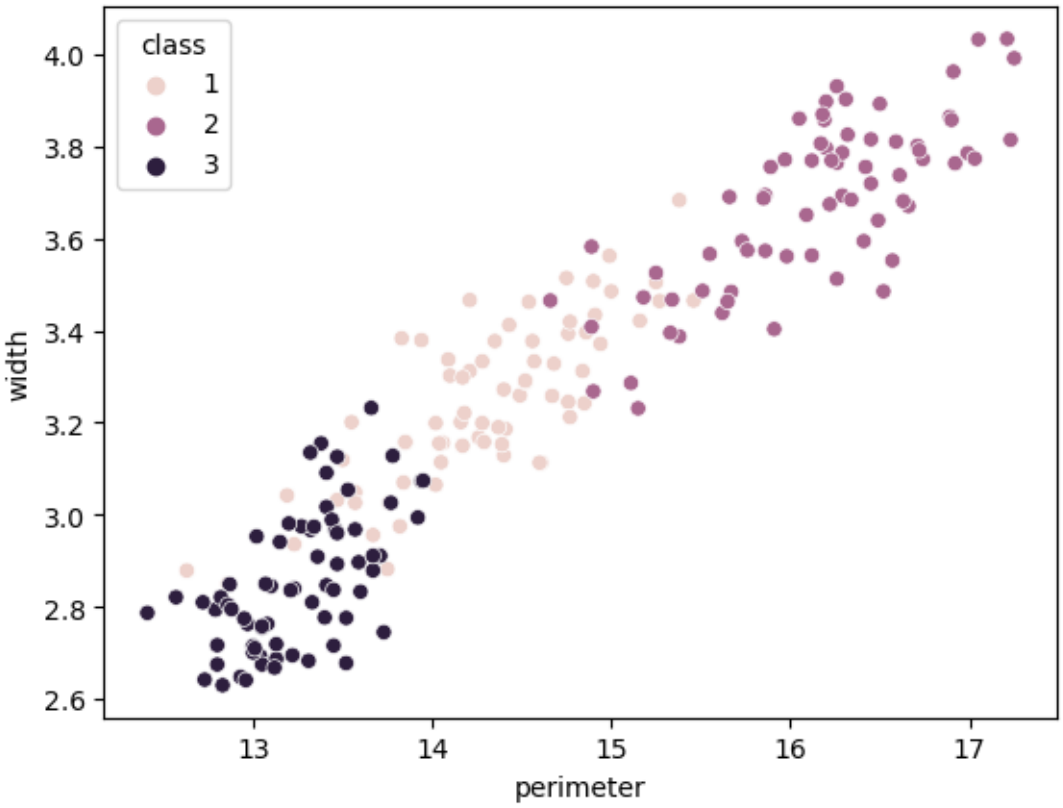

class
1
2
3
4.0
3.8
3.6
3.4
3.2
3.0
2.8
2.6
width
12
14
16
18
20
area

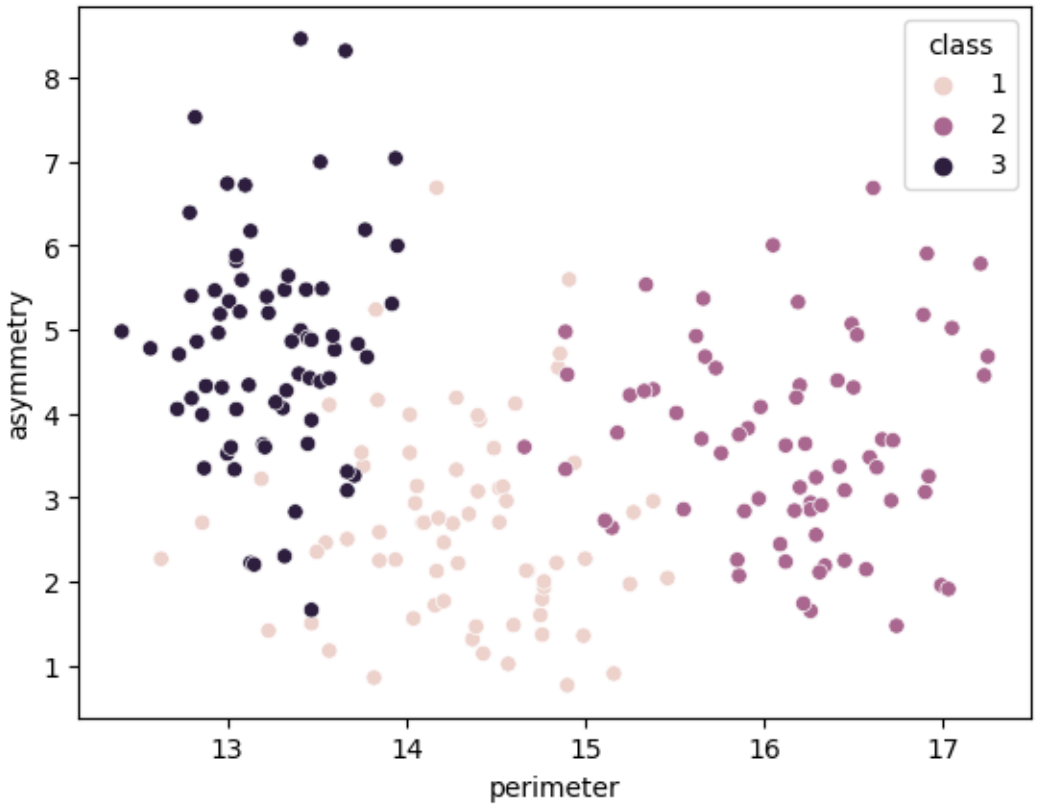

class
1
2
3
asymmetry
8
7
6
5
4
3
2
1
12
14
16
18
20
area

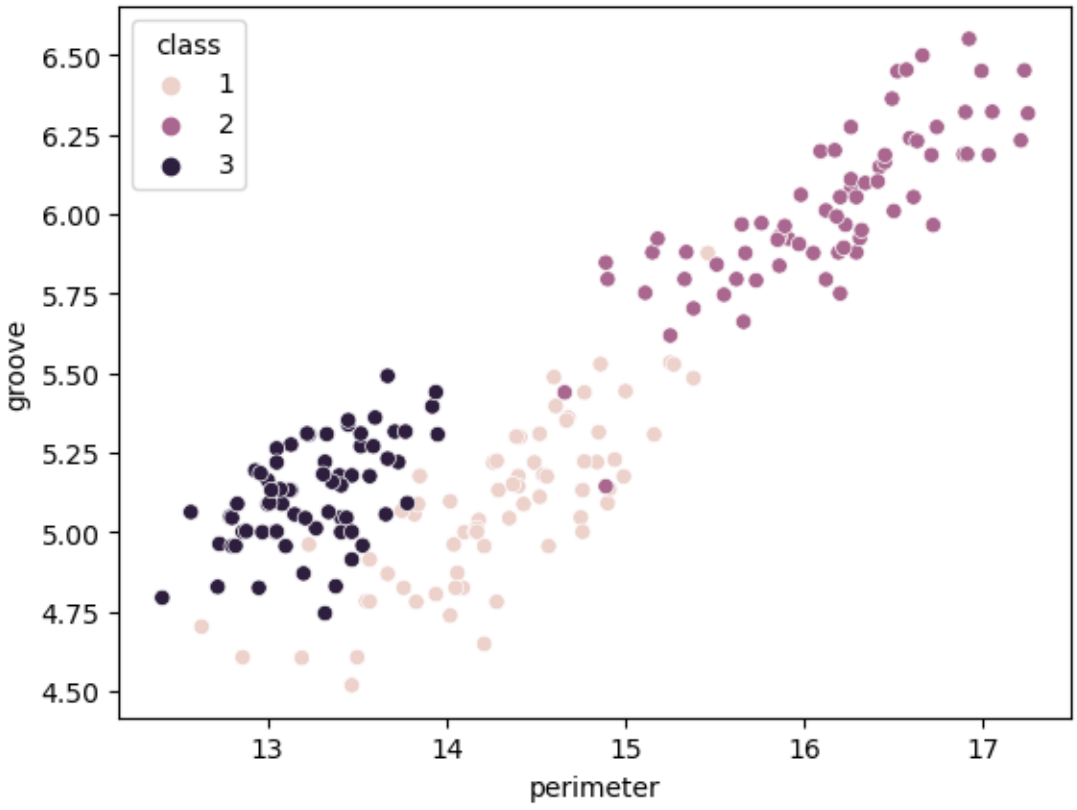

class
1
2
3
groove
area
6.50
6.25
6.00
5.75
5.50
5.25
5.00
4.75
4.50
12
14
16
18
20

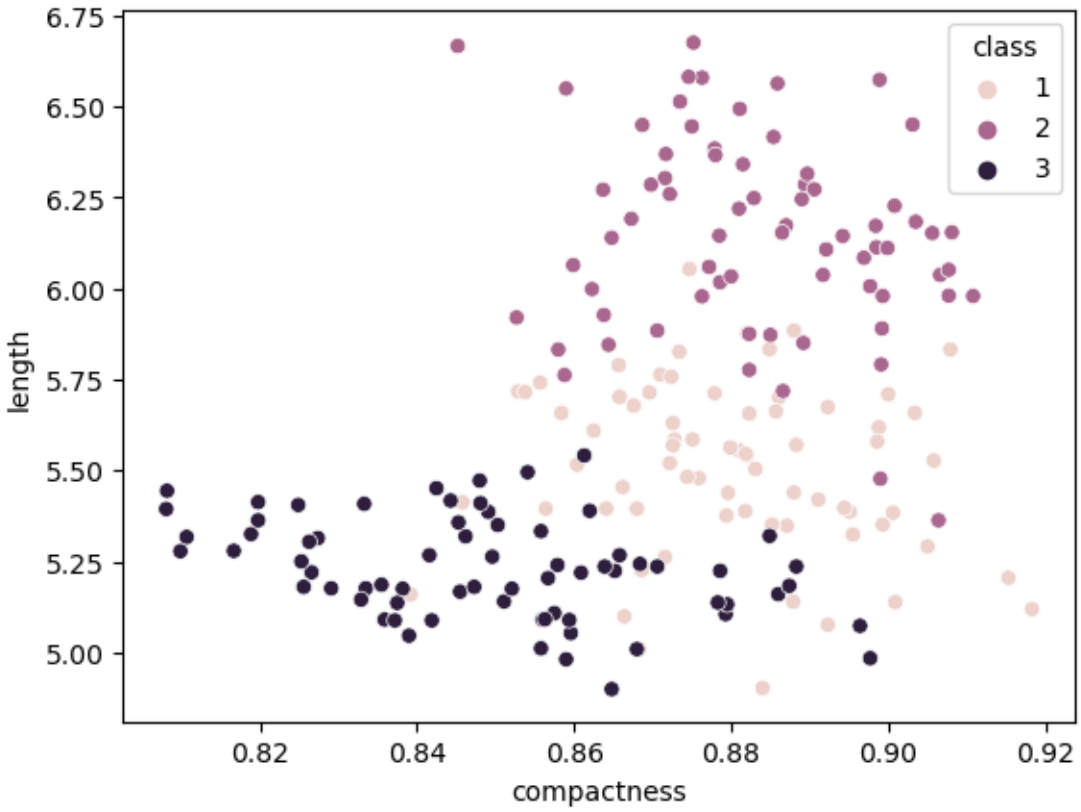

class
1
2
3
0.92
0.90
0.88
0.86
0.84
0.82
compactness
13
14
15
16
17
perimeter

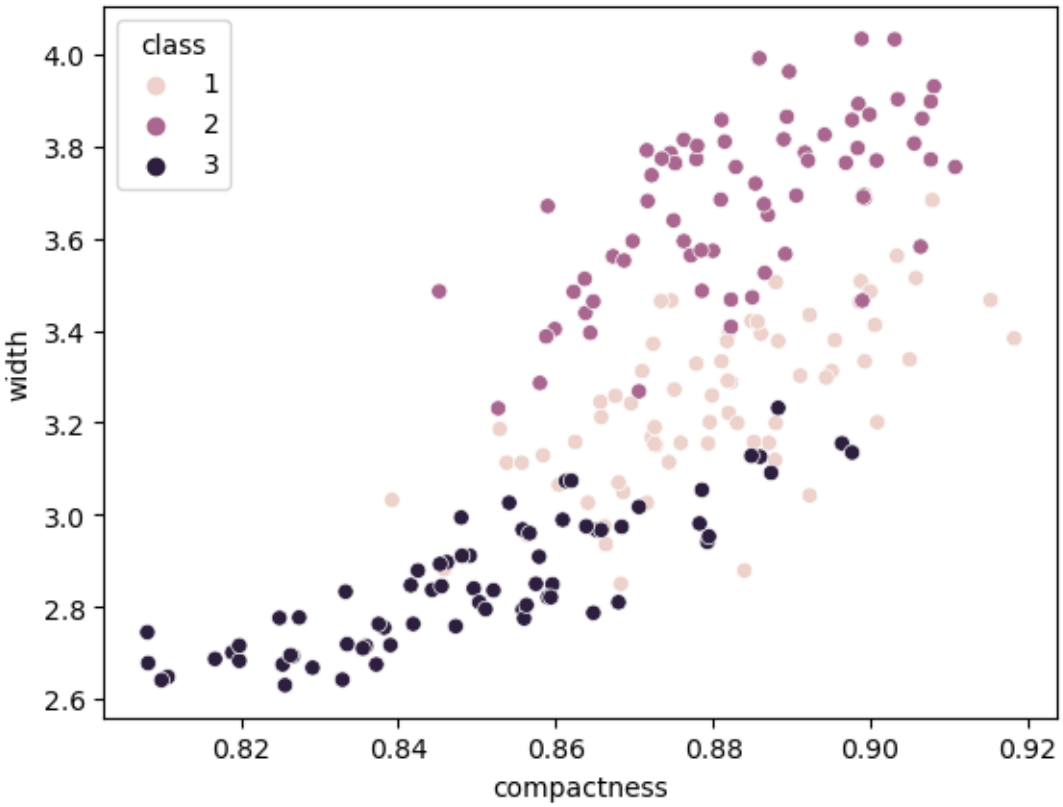

class
1
2
3
6.75
6.50
6.25
6.00
5.75
5.50
5.25
5.00
length
13
14
15
16
17
perimeter

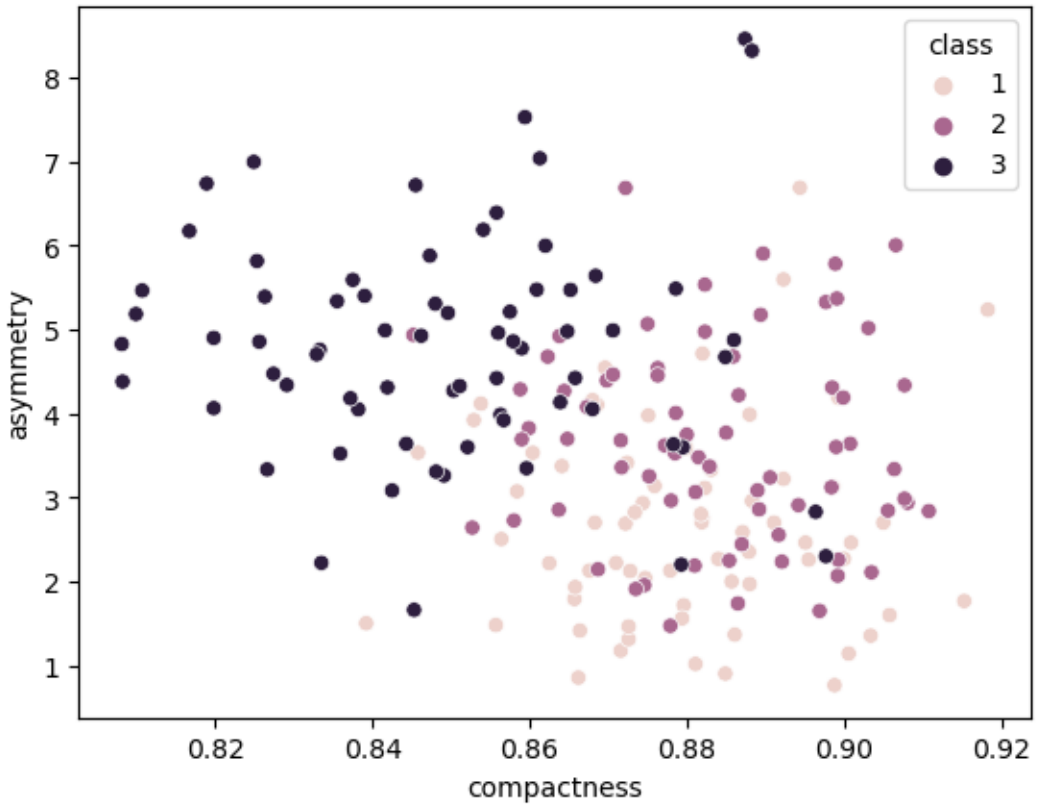

class
1
2
3
4.0
3.8
3.6
3.4
3.2
3.0
2.8
2.6
width
13
14
15
16
17
perimeter

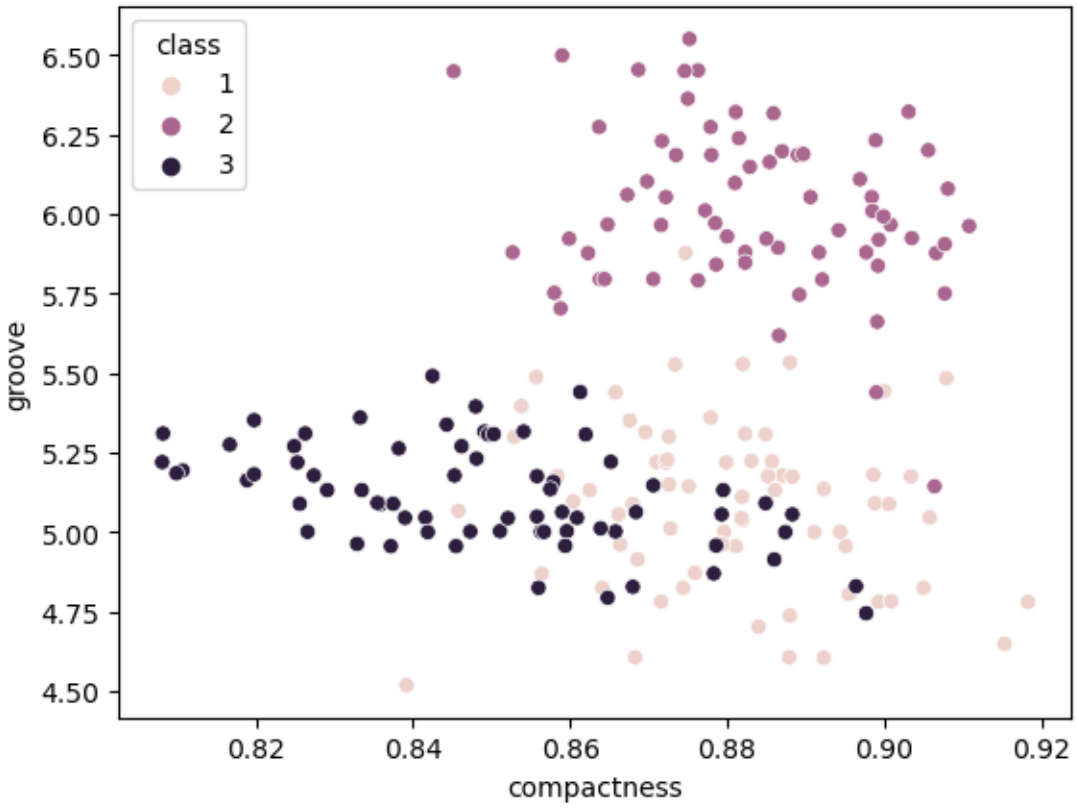

class
1
2
3
asymmetry
perimeter
8
7
6
5
4
3
2
1
13
14
15
16
17

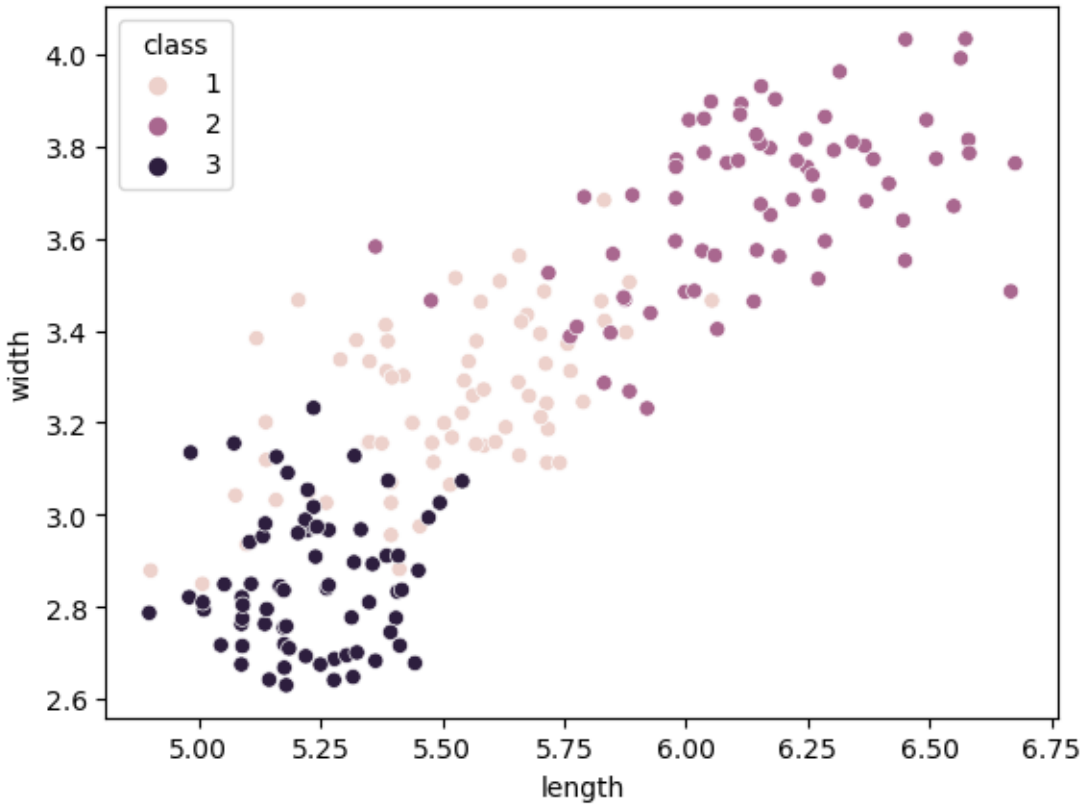

class
1
2
3
6.50
6.25
6.00
5.75
5.50
5.25
5.00
4.75
4.50
groove
13
14
15
16
17
perimeter

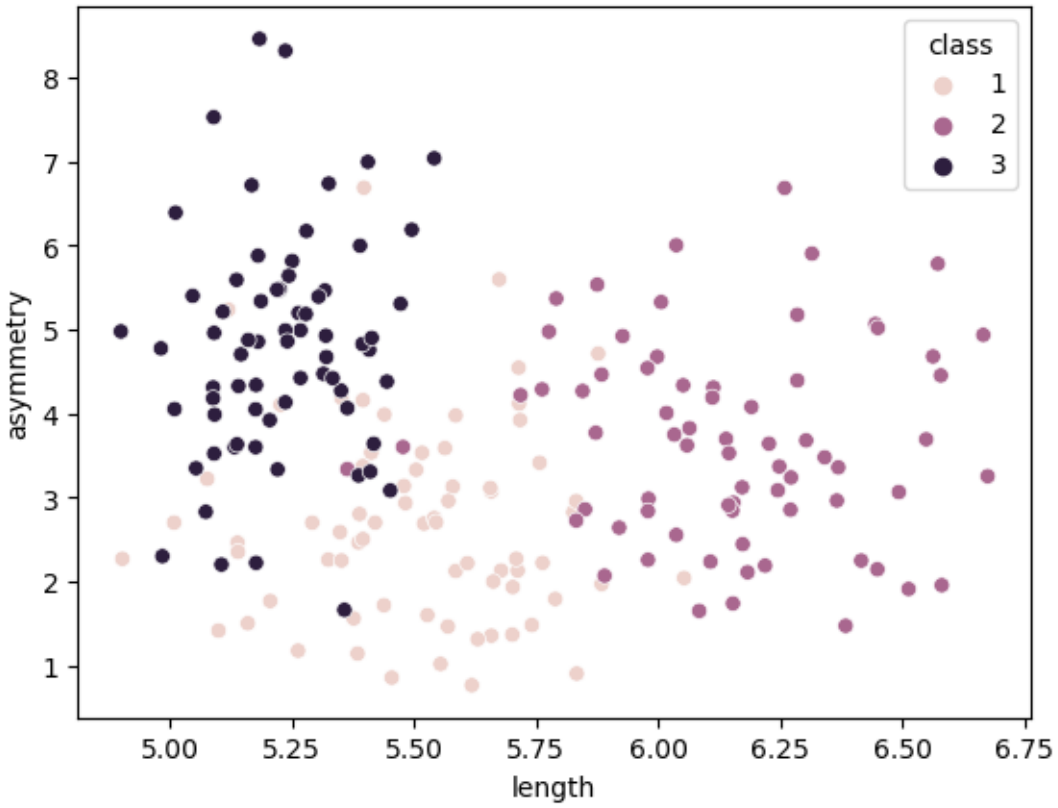

class
1
2
3
length
compactness
6.75
6.50
6.25
6.00
5.75
5.50
5.25
5.00
0.82
0.84
0.86
0.88
0.90
0.92

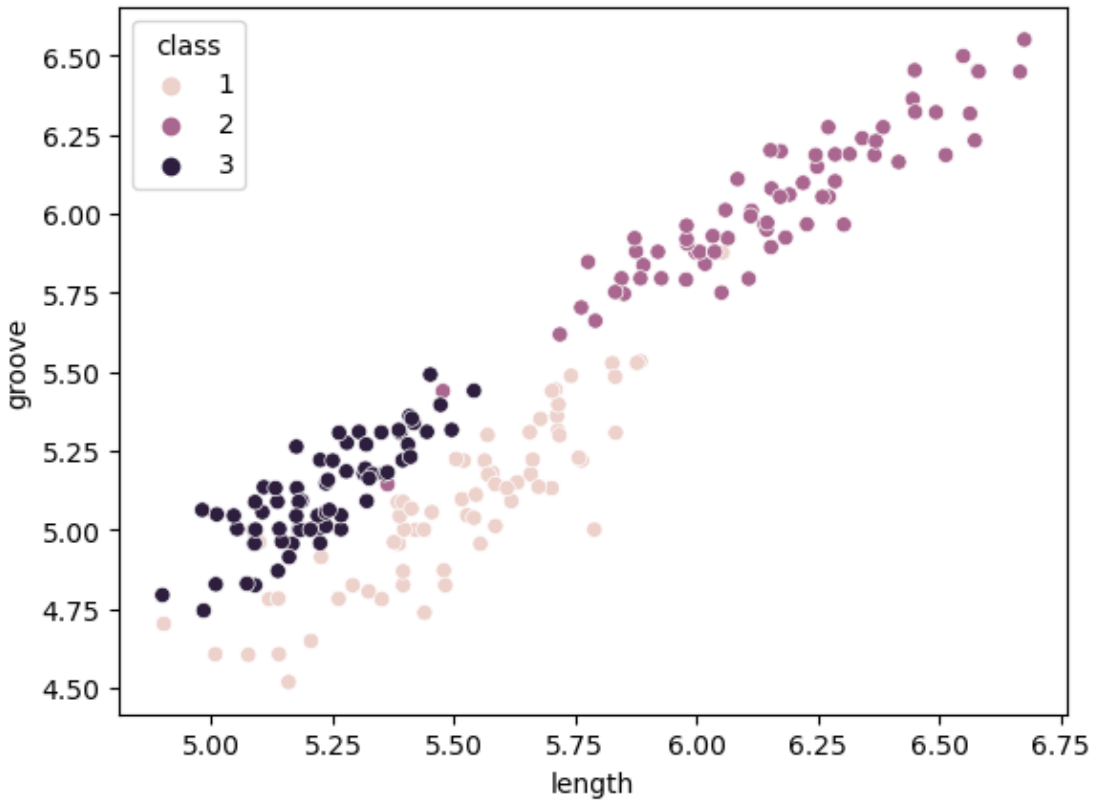

class
1
2
3
4.0
3.8
3.6
3.4
3.2
3.0
2.8
2.6
width
0.82
0.84
0.86
0.88
0.90
0.92
compactness

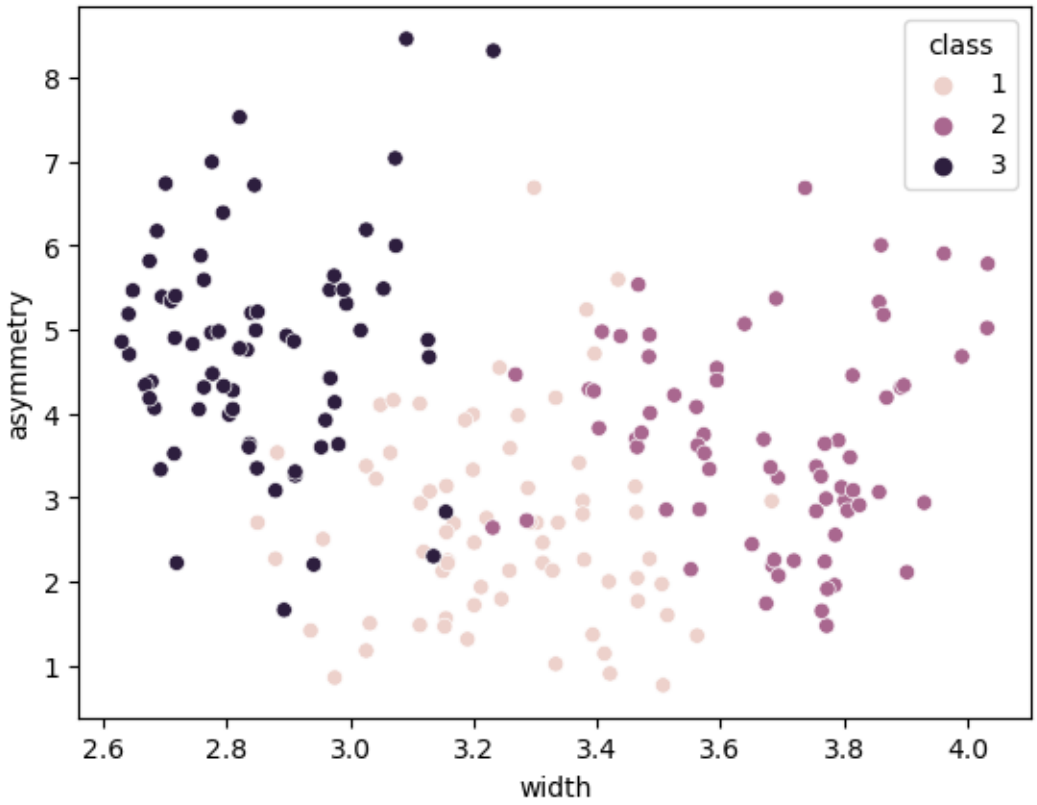

class
1
2
3
asymmetry
compactness
8
7
6
5
4
3
2
1
0.82
0.84
0.86
0.88
0.90
0.92

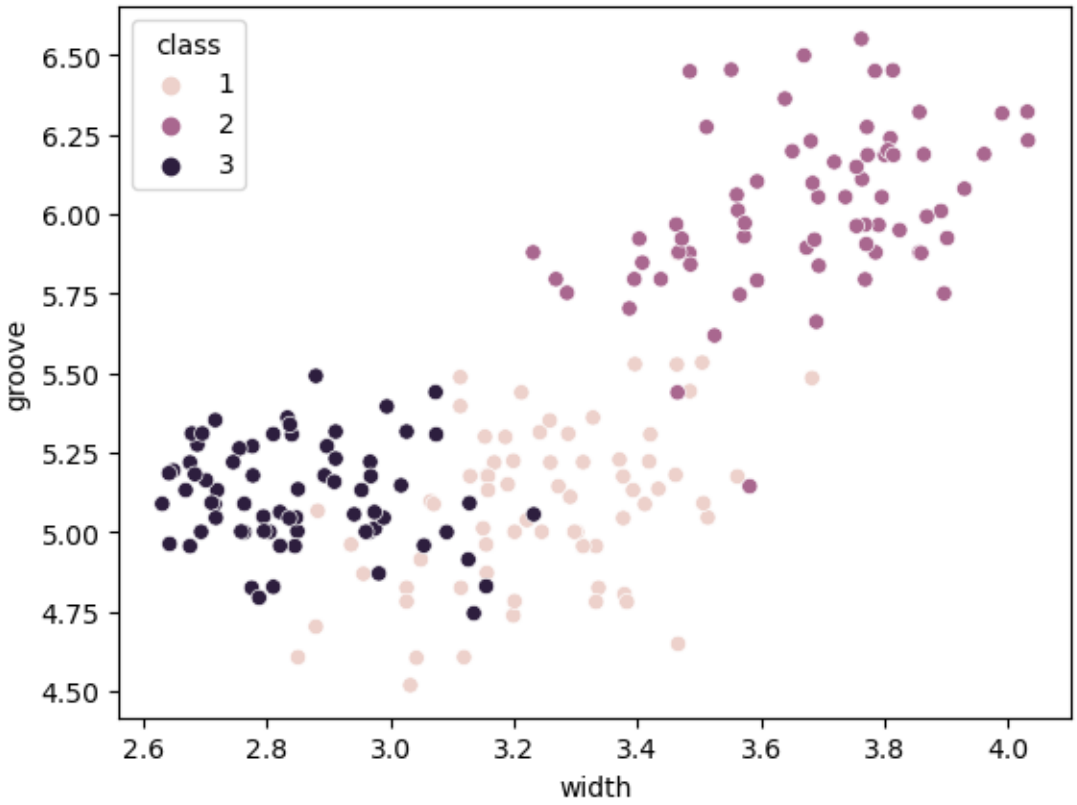

class
1
2
3
6.50
6.25
6.00
5.75
5.50
5.25
5.00
4.75
4.50
groove
0.82
0.84
0.86
0.88
0.90
0.92
compactness

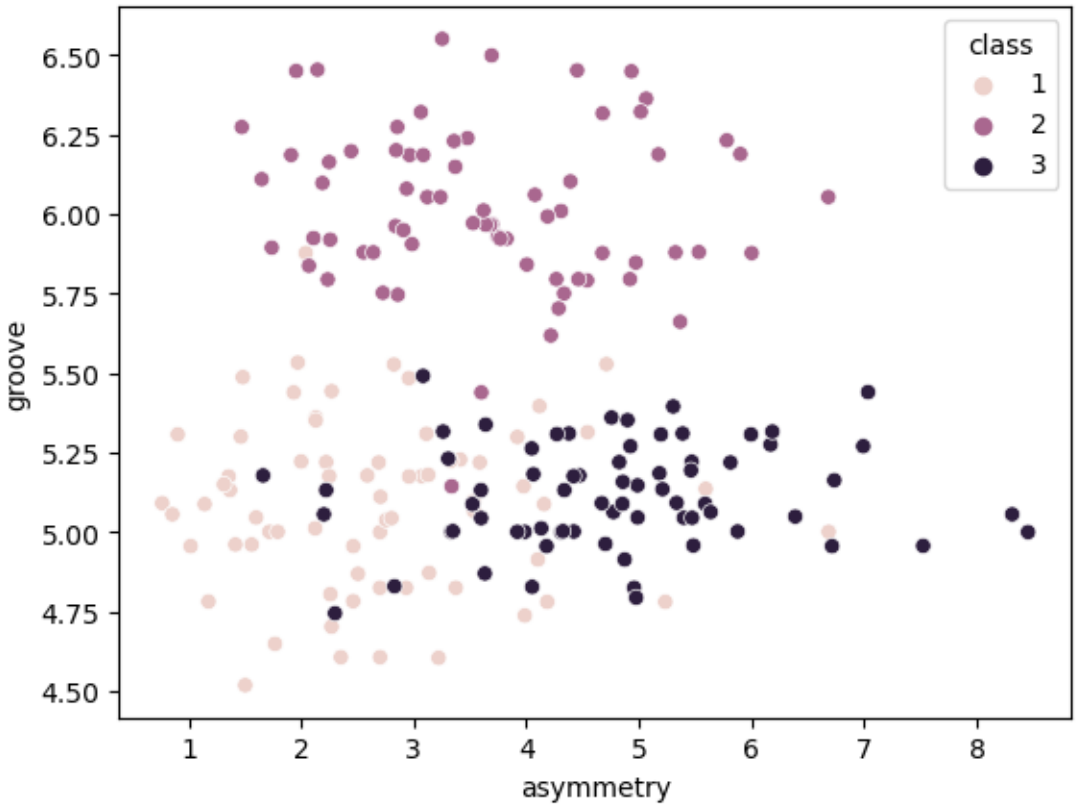

class
1
2
3
4.0
3.8
3.6
3.4
3.2
3.0
2.8
2.6
width
5.00
5.25
5.50
5.75
6.00
6.25
6.50
6.75
length


class
1
2
3
asymmetry
length
8
7
6
5
4
3
2
1
5.00
5.25
5.50
5.75
6.00
6.25
6.50
6.75


class
1
2
3
groove
length
6.50
6.25
6.00
5.75
5.50
5.25
5.00
4.75
4.50
5.00
5.25
5.50
5.75
6.00
6.25
6.50
6.75


class
1
2
3
asymmetry
width
8
7
6
5
4
3
2
1
2.6
2.8
3.0
3.2
3.4
3.6
3.8
4.0


class
1
2
3
groove
6.50
6.25
6.00
5.75
5.50
5.25
5.00
4.75
4.50
2.6
2.8
3.0
3.2
3.4
3.6
3.8
4.0
width



1 Clustering

We are gonna be using the KMeans algorithm for clustering. Really good for separating data into K groups. In our case, we have 3 different seeds we want to separate, so k will be 3 for the entire project.

[7]:
```
# The library that allows us to use KMeans
from sklearn.cluster import KMeans
```

[36]:
```
# Chossing our columns
x = "perimeter"
y = "asymmetry"
X = df[[x,y]].values
```

[63]:
```
# creating our KMeans model
kmeans = KMeans(n_clusters = 3).fit(X)
```

```
C:\Users\ezane\anaconda3\envs\tf\lib\site-
packages\sklearn\cluster\_kmeans.py:870: FutureWarning: The default value of
`n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init`
explicitly to suppress the warning
```

```
  warnings.warn(
C:\Users\ezane\anaconda3\envs\tf\lib\site-
packages\sklearn\cluster\_kmeans.py:1382: UserWarning: KMeans is known to have a
memory leak on Windows with MKL, when there are less chunks than available
threads. You can avoid it by setting the environment variable OMP_NUM_THREADS=1.
  warnings.warn(
```

[64]:
```
# Using the kmeans model
clusters = kmeans.labels_
```

[65]:
```
# Ajusting the predictions for the classes to match
# This cell is optional, but I think it helps with visualization
cluster_map = {0:1, 1:2, 2:3}
clusters = np.array([cluster_map[cluster] for cluster in clusters])
```

[51]:
```
# The predictions
clusters
```

[51]:
```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 3, 1, 3, 1, 1, 3, 3, 1, 1, 3, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 3, 1, 1, 1, 3,
       1, 1, 1, 1, 1, 1, 1, 3, 3, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 3, 1, 1,
       1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 3, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 3, 1, 2, 1, 1, 2, 3, 3, 3, 3, 3, 3, 1, 3, 1, 3, 3, 3, 3, 1,
       3, 3, 1, 3, 3, 3, 1, 3, 3, 3, 3, 1, 3, 1, 3, 1, 3, 3, 3, 3, 3, 3,
       3, 3, 3, 1, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 1, 3, 3, 3, 3, 3,
       1, 1, 3, 1, 3, 3, 3, 1, 3, 3, 1, 3])
```

[40]:
```
# The actual values
df["class"].values
```

[40]:
```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
       3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
       3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
       3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3], dtype=int64)
```

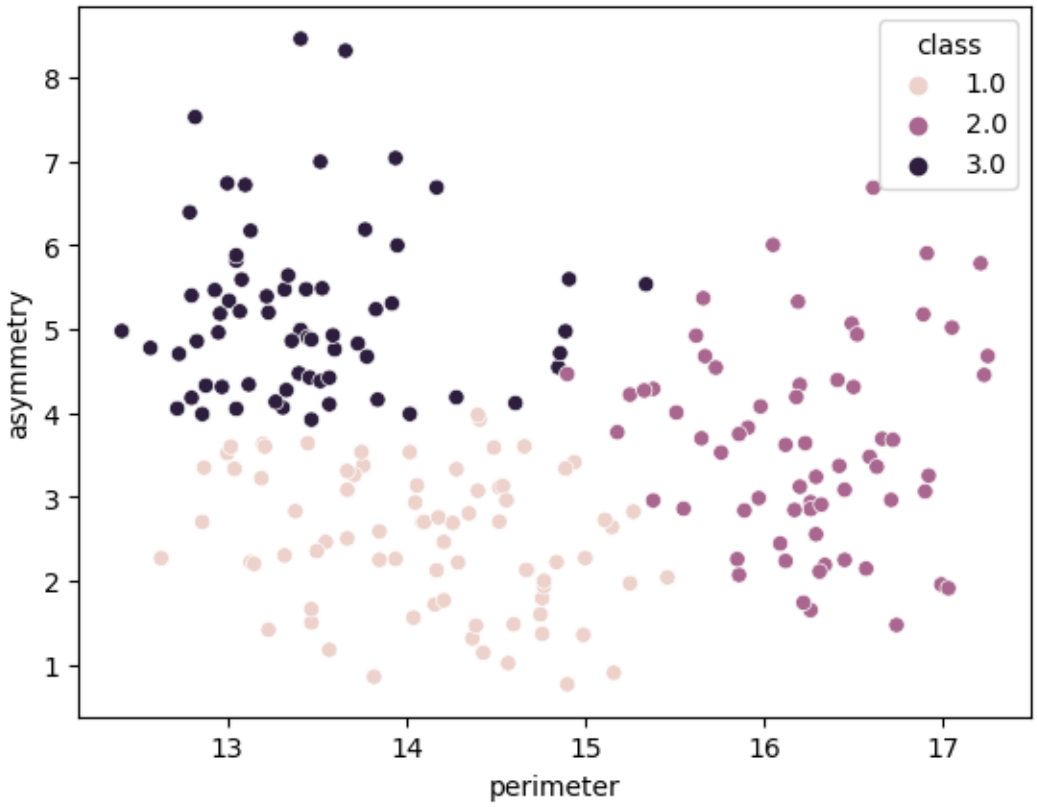
[60]:
```
# Calculating the percentage of classes our model got right
((df["class"].values==clusters).astype(int).sum())/len(clusters)
```

[60]:
```
0.8476190476190476
```

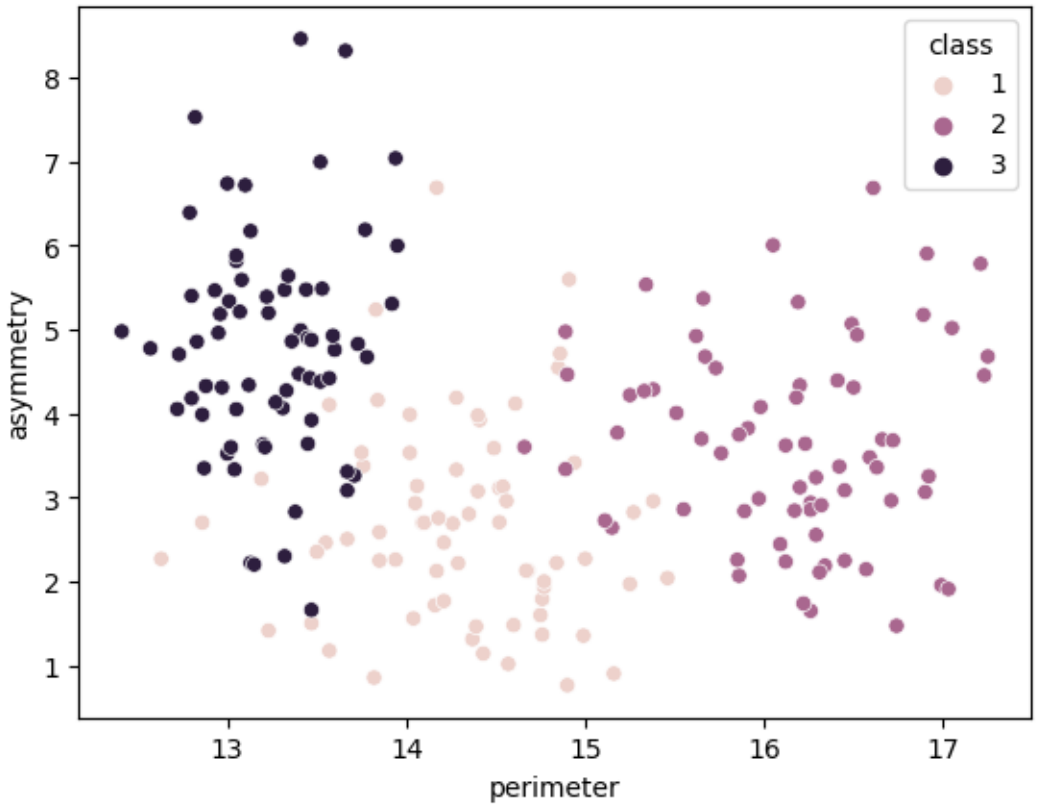
KMeans starts with random points, so it can lead to different outcomes depending on luck. Regardless, the model got 84% of cases right, pretty good.

[52]:
```
# Creating cluster_df with our predictions
cluster_df = pd.DataFrame(np.hstack((X,clusters.reshape(-1,1))),␣
 ↪columns=[x,y,"class"])
```

[53]:
```
# graph for our predictions
sns.scatterplot(x=x, y=y, hue="class", data=cluster_df)
plt.plot();
```



[54]:
```
# actual classes
sns.scatterplot (x=x, y=y, hue="class", data=df)
plt.show()
```



2 Higher dimensions

Now instead of using KMeans in 2d, lets use it for all the dimensions our data allows to see if our predictions are better.

[66]:
```
x = "compactness"
y = "asymmetry"
X = df[[x,y]].values
```

[67]:
```
# removing the classes column
X = df[cols[:-1]].values
```

[68]:
```
# creating and using our KMeans model in our data
kmeans = KMeans(n_clusters = 3).fit(X)

# getting the dataframe together with the predictions
cluster_df = pd.DataFrame(np.hstack((X, kmeans.labels_.reshape(-1,1))),␣
 ↪columns=df.columns)
```

```
C:\Users\ezane\anaconda3\envs\tf\lib\site-
packages\sklearn\cluster\_kmeans.py:870: FutureWarning: The default value of
```

```
`n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init`
explicitly to suppress the warning
  warnings.warn(
C:\Users\ezane\anaconda3\envs\tf\lib\site-
packages\sklearn\cluster\_kmeans.py:1382: UserWarning: KMeans is known to have a
memory leak on Windows with MKL, when there are less chunks than available
threads. You can avoid it by setting the environment variable OMP_NUM_THREADS=1.
  warnings.warn(
```

[75]:
```
cluster_map = {2.:1, 1.:2, 0.:3}
cluster_df["class"] = np.array([cluster_map[cluster] for cluster in␣
 ↪cluster_df["class"]])
```

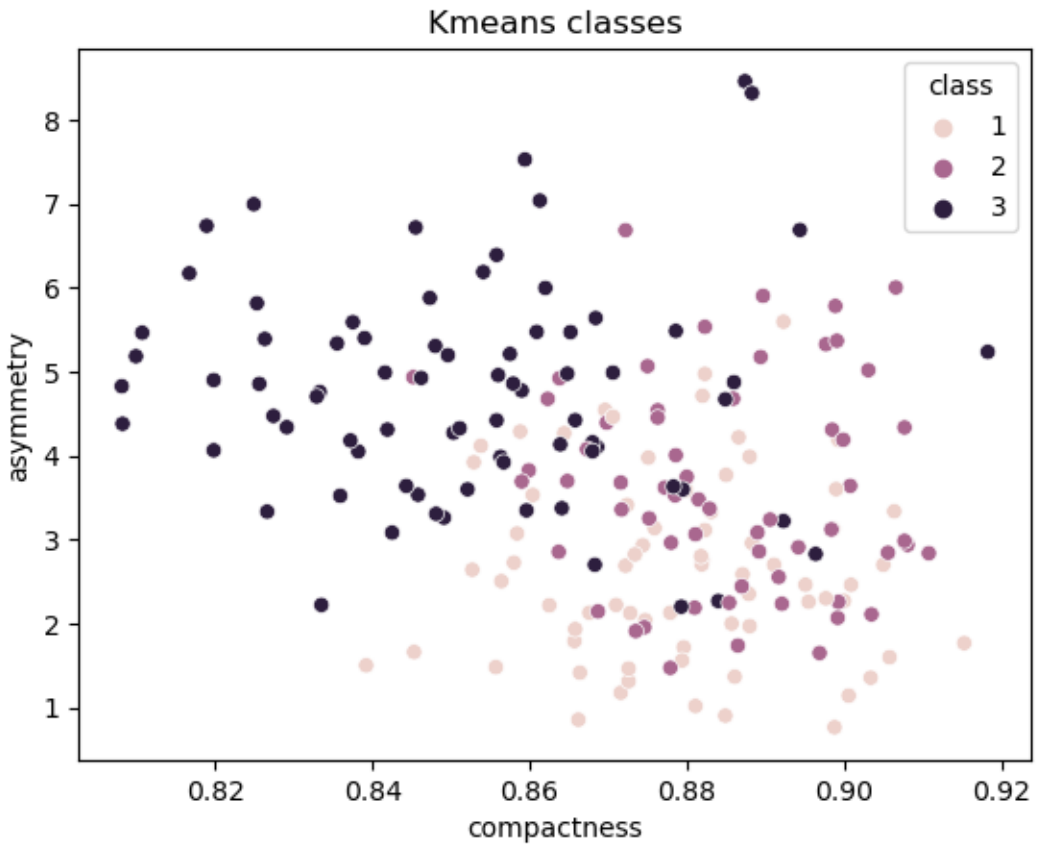
[76]:
```
cluster_df["class"].values
```

[76]:
```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 3, 1, 1, 3, 1, 1,
       1, 1, 1, 1, 3, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 3, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 3, 3, 3, 3, 1, 1,
       1, 1, 1, 3, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 1, 2, 2, 2, 2, 2, 2, 2,
       1, 1, 1, 1, 2, 1, 1, 1, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
       3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
       3, 3, 3, 1, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
       3, 3, 3, 1, 3, 3, 3, 3, 3, 3, 3, 3])
```

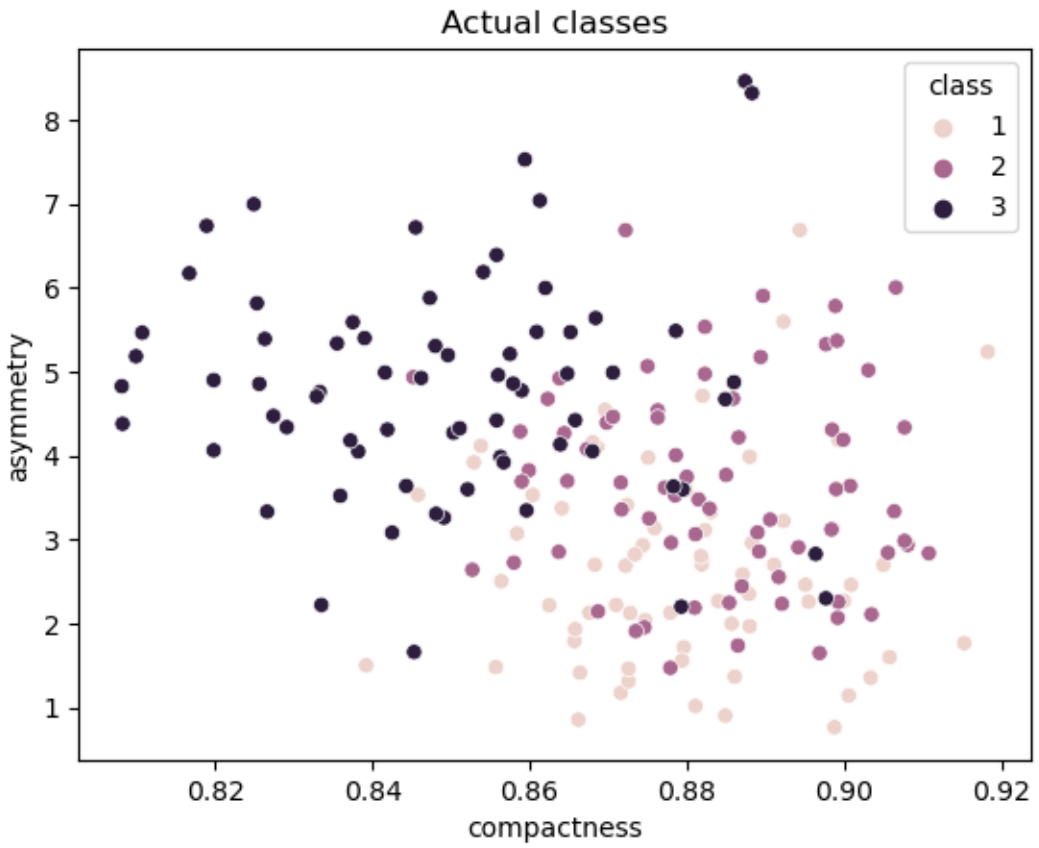
[77]:
```
df["class"].values
```

[77]:
```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
       3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
       3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
       3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3], dtype=int64)
```

[84]:
```
# K Means classes
sns.scatterplot(x=x, y=y, hue="class", data=cluster_df)
plt.title("Kmeans classes")
plt.show()
```



[87]:
```
# Actual classes
sns.scatterplot(x=x, y=y, hue="class", data=df)
plt.title("Actual classes")
plt.show()
```



[82]:
```
# Analising the accuracy of our kmeans prediction
((cluster_df["class"] == df["class"]).astype(int).sum())/len(df["class"])
```

[82]: 0.8952380952380953

Incredible accuracy considering how messy the graph is in 2 dimensions. This is the power of KMeans in higher dimensions, it makes use of all the relations between variables to make a prediction, not just the two we can see in the plot.

Note that the colors might not match when you run the code, but the classes will. This is because of the randomness in KMeans. Changing cluster_map will resolve the issue.

3 PCA

PCA stands for Principal Component Analysis, which is a tecnique to reduce the dimensions of a graph while retaining the most amount of information possible. For this project here we are gonna reduce the 7d graph with all the information about the seeds to 2d graph and then use KMeans. It will be interesting to compare using PCA and then KMeans agains just the KMeans algorithm with 7 dimension.

[89]:
```
# importing the library
from sklearn.decomposition import PCA
```

[90]:
```
# Creating and using our model
pca = PCA(n_components=2)
transformed_x = pca.fit_transform(X)
```

[91]:
```
X.shape
```

[91]:
```
(210, 7)
```

[92]:
```
transformed_x.shape
```

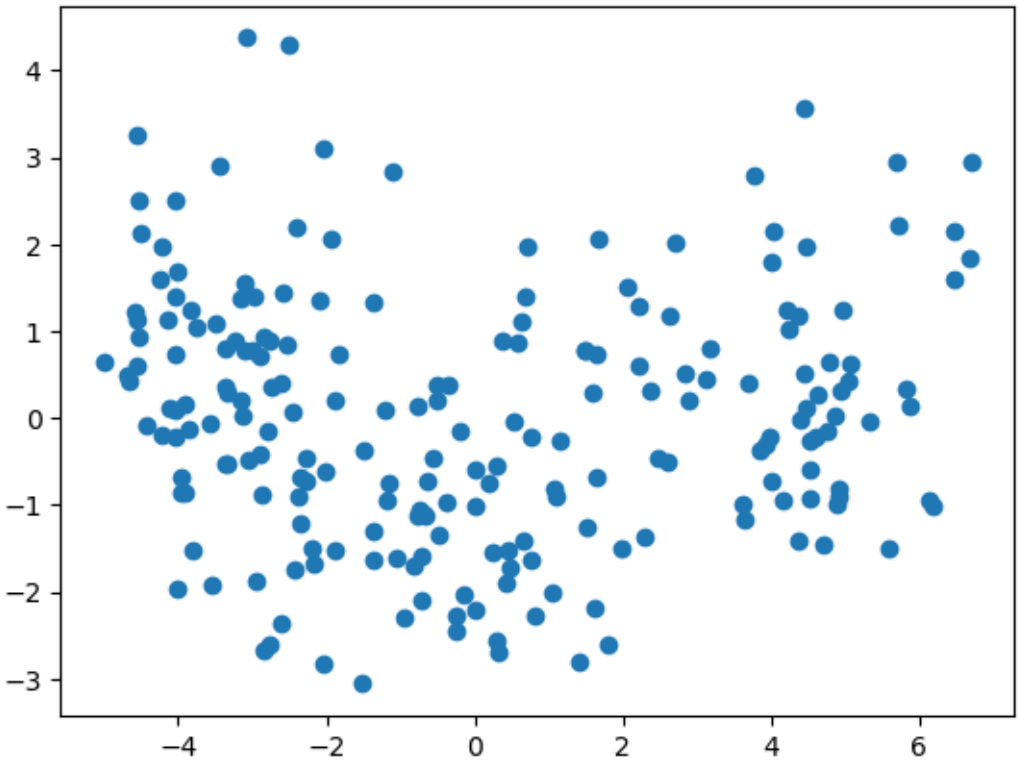
[92]:
```
(210, 2)
```

Look at how we reduced the shape from (210,7) to (210,2)

[93]:
```
transformed_x[:5]
```

[93]:
```
array([[ 0.66344838, -1.41732098],
       [ 0.31566651, -2.68922915],
       [-0.6604993 , -1.13150635],
       [-1.0552759 , -1.62119002],
       [ 1.61999921, -2.18338442]])
```

[94]:
```
# the 2d graph of 7 dimension...
plt.scatter(transformed_x[:,0], transformed_x[:,1])
plt.show()
```

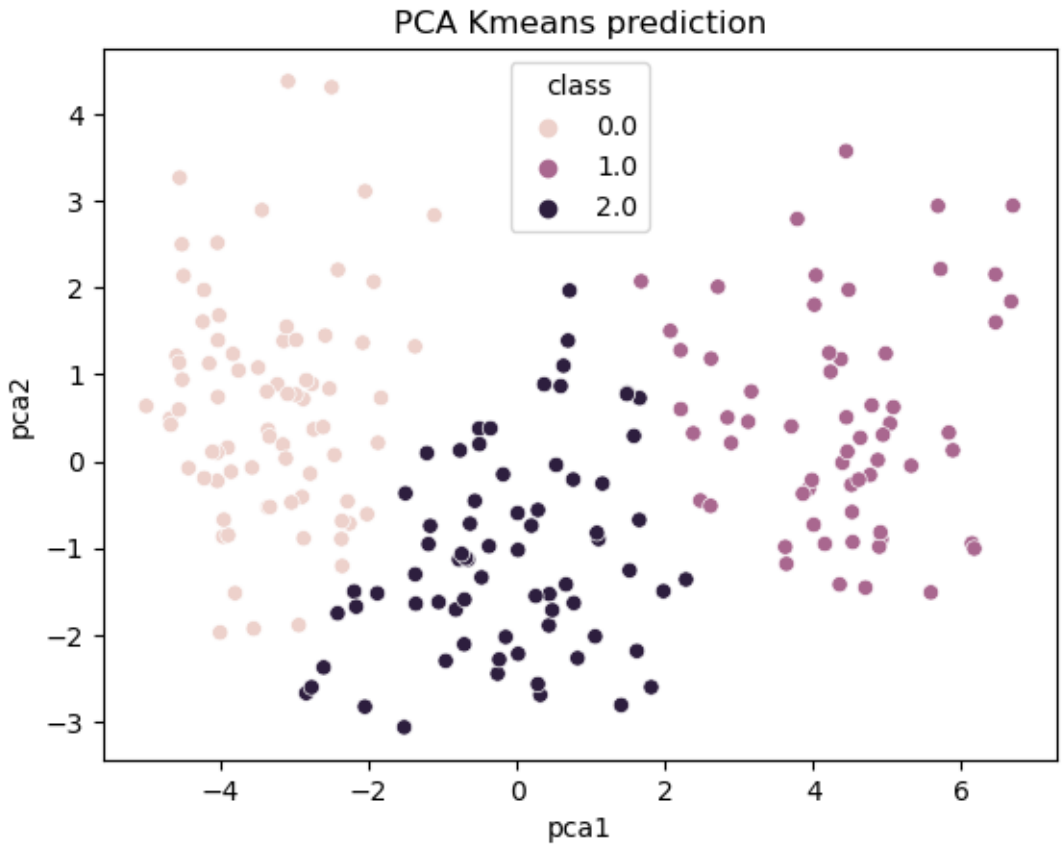


Well, what are the axis of this plot? No one knows, they are the decomposition of all the 7 metrics, so the axis have no meaning. Anyway, now we can use KMeans on this plot and see how it performs.

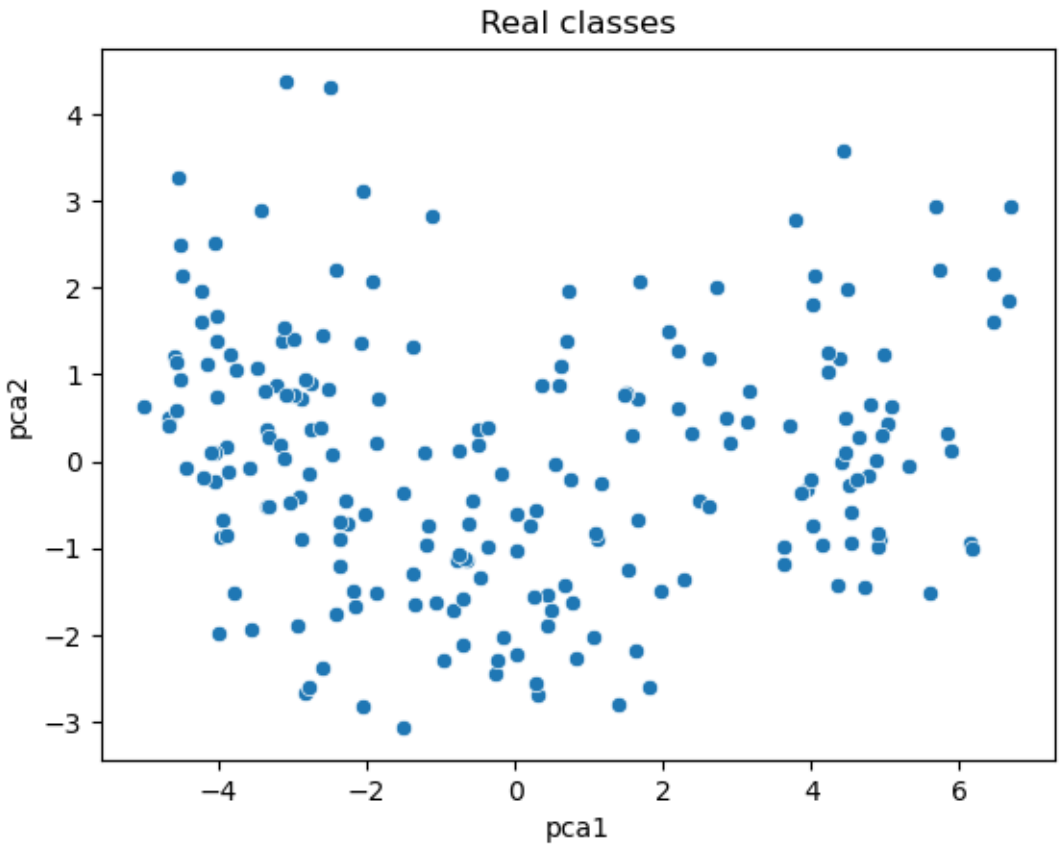
[98]:
```
# Getting the KMeans predictions for the PCA graph
kmeans_pca_df = pd.DataFrame(np.hstack((transformed_x, kmeans.labels_.
 ↪reshape(-1,1))), columns=["pca1", "pca2", "class"])
```

[97]:
```
# Getting the actual classes for the PCA graph
truth_pca_df = pd.DataFrame(np.hstack((transformed_x, df["class"].values.
 ↪reshape(-1, 1))), columns=["pca1", "pca2", "class"])
```

[100]:
```
sns.scatterplot(x="pca1",y="pca2", hue="class", data=kmeans_pca_df)
plt.title("PCA Kmeans prediction");
```



[102]:
```
sns.scatterplot(x="pca1", y="pca2", data=truth_pca_df)
plt.title("Real classes");
```


Real classes
pca2
pca1
4
3
2
1
0
−1
−2
−3
−4
−2
0
2
4
6