# pr_dom5

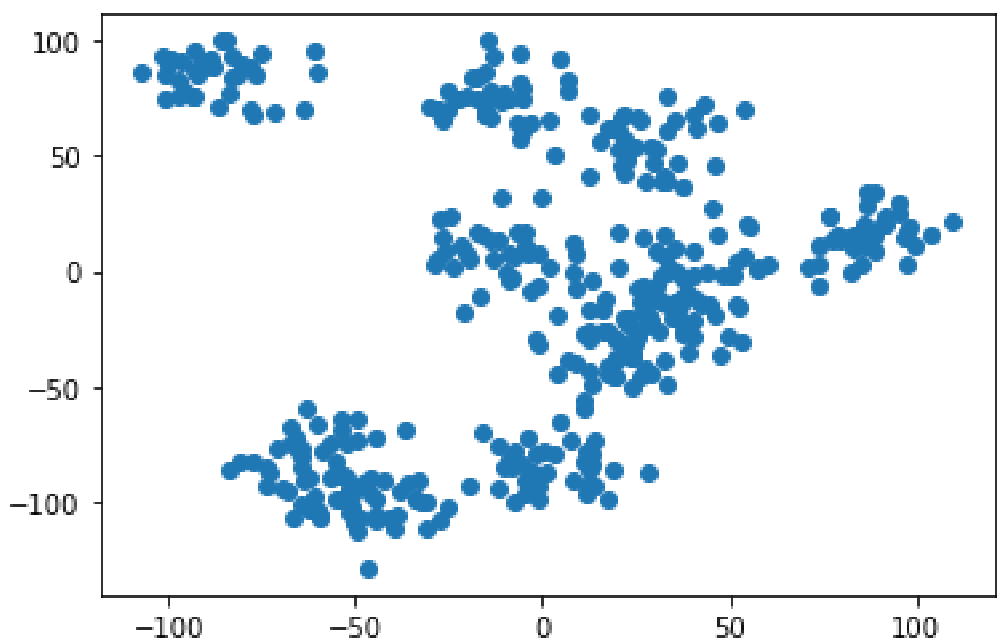
May 18, 2021

[1]:
```
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.cluster import KMeans, AgglomerativeClustering
from scipy.cluster import hierarchy
from sklearn.preprocessing import normalize
import numpy as np
```

[2]:
```
df=pd.read_csv('clustering.csv')
```

[3]:
```
df.columns=["x","y"]
df
plt.scatter(df['x'],df['y'])
```

[3]:
```
<matplotlib.collections.PathCollection at 0x29b7450aaf0>
```



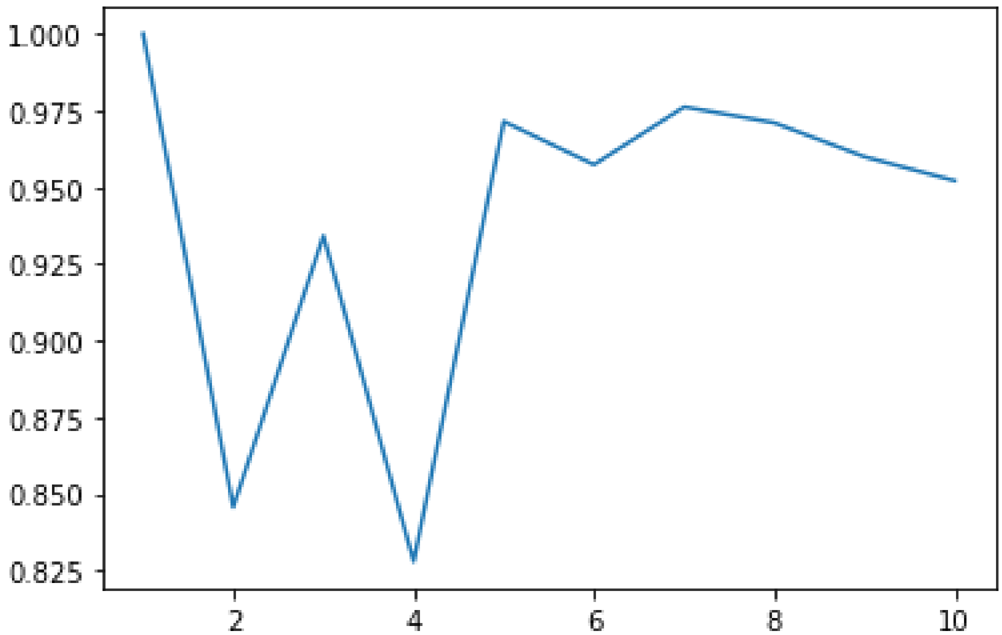
# 1 KMeans

## 1.1 Metrics

### 1.1.1 Stability

[4]:
```
from sklearn.utils import resample
import math
from sklearn import metrics
def kmeans_stability_mean(df,cluster_num):
    n=100
    ans=[]
    metr=[]
    labels_true=KMeans(n_clusters=cluster_num).fit_predict(df)
    mean=0
    for i in range(n):
        bootstrap=resample(df, replace=True, n_samples=math.floor(df.shape[0]*0.
 ↪8))
        model=KMeans(n_clusters=cluster_num)
        model.fit(bootstrap)
        ans.append( model.predict(df))
        metr.append(metrics.adjusted_mutual_info_score(labels_true,ans[i]))
        mean+=metr[i]
    mean=mean/n
    return mean
```

[10]:
```
vec=[]
vec1=[]
for i in range(10):
    n_clusters=i+1
    vec.append(kmeans_stability_mean(df,n_clusters))
    vec1.append(n_clusters)
plt.plot(vec1,vec)
plt.show()
```




Za pomocą bootstrapa określiliśmy, że najstabilniejszy jest podział na 5 lub 7 klastrów dla KMeans, dlatego postaramy się wybierać ilość klastrów ze zbioru {5,7,8,9}.

#### 1.1.2 Metoda łokcia

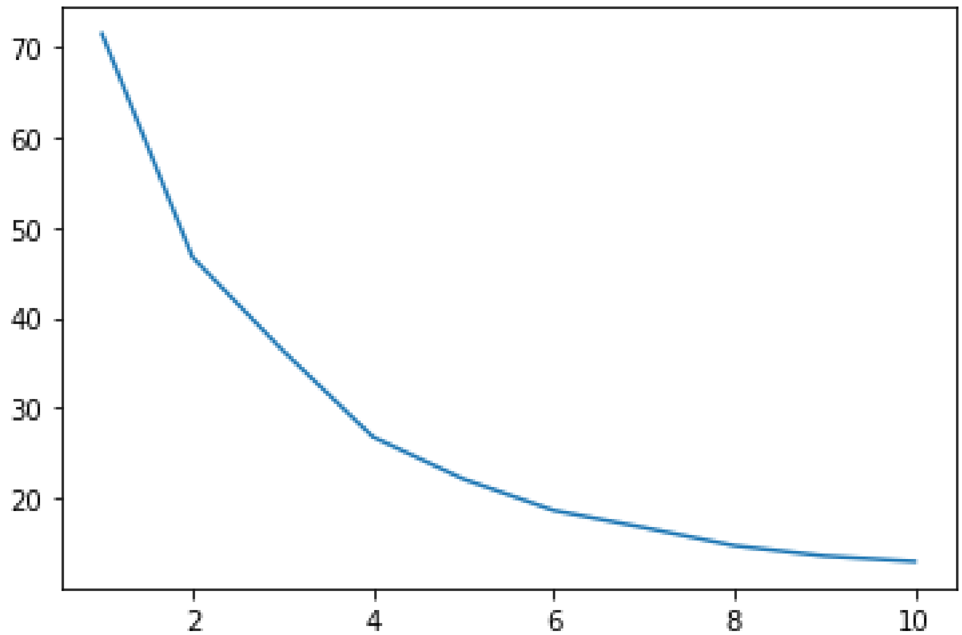
[40]:
```
# Code copied from laboratory file
from scipy.spatial import distance
def count_clustering_scores(X, cluster_num, model_class, score_fun):
    if isinstance(cluster_num, int):
        cluster_num_iter = [cluster_num]
    else:
        cluster_num_iter = cluster_num
    scores = []
    for k in cluster_num_iter:
        model_instance = model_class(n_clusters=k)
        labels = model_instance.fit_predict(X)
        wcss = score_fun(X, labels)
        scores.append(wcss)

    if isinstance(cluster_num, int):
        return scores[0]
    else:
        return scores
def count_clustering_scores_agl(X, cluster_num, model_class, score_fun):
    if isinstance(cluster_num, int):
        cluster_num_iter = [cluster_num]
```

```
    else:
        cluster_num_iter = cluster_num
    scores = []
    for k in cluster_num_iter:
        model_instance = model_class(n_clusters=k,linkage="ward")
        labels = model_instance.fit_predict(X)
        wcss = score_fun(X, labels)
        scores.append(wcss)

    if isinstance(cluster_num, int):
        return scores[0]
    else:
        return scores
def mean_dist_to_center(X, label):
    clusters = set(label)
    inclust_dist_list = []
    for cluster_i in clusters:
        cluster_i_idx = np.where(label == cluster_i)
        cluster_i_mean = np.mean(X[cluster_i_idx], axis=0, keepdims=True)
        inclust_dist = np.mean(distance.cdist(X[cluster_i_idx], cluster_i_mean))
        inclust_dist_list.append(inclust_dist)
    return np.mean(inclust_dist_list)
```

[27]:
```
vec2=[]
vec3=[]
for i in range(10):
    n_clusters=i+1
    vec2.append(count_clustering_scores(np.array(df), n_clusters, KMeans,␣
 ↪mean_dist_to_center))
    vec3.append(n_clusters)
plt.plot(vec3,vec2)
plt.show()
```

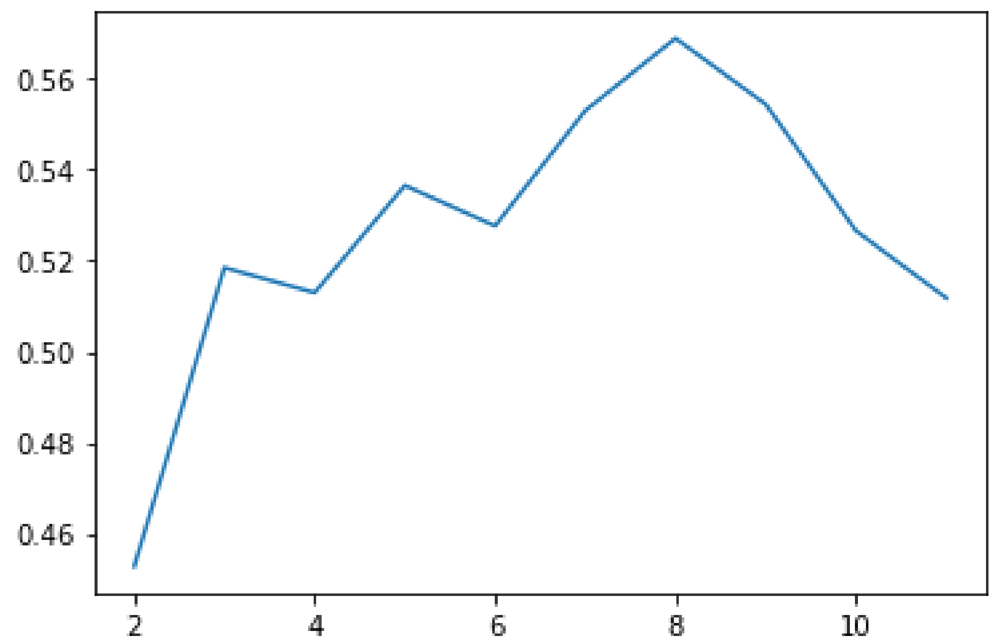



Za pomocą metody łokcia ciężko jest określić optymalną ilość klastrów może to być 2, 4, 6 lub 8. Dlatego spróbujemy metoda silhouette.

#### 1.1.3 Metoda silhouette

[28]:
```
from sklearn.metrics import silhouette_score
```

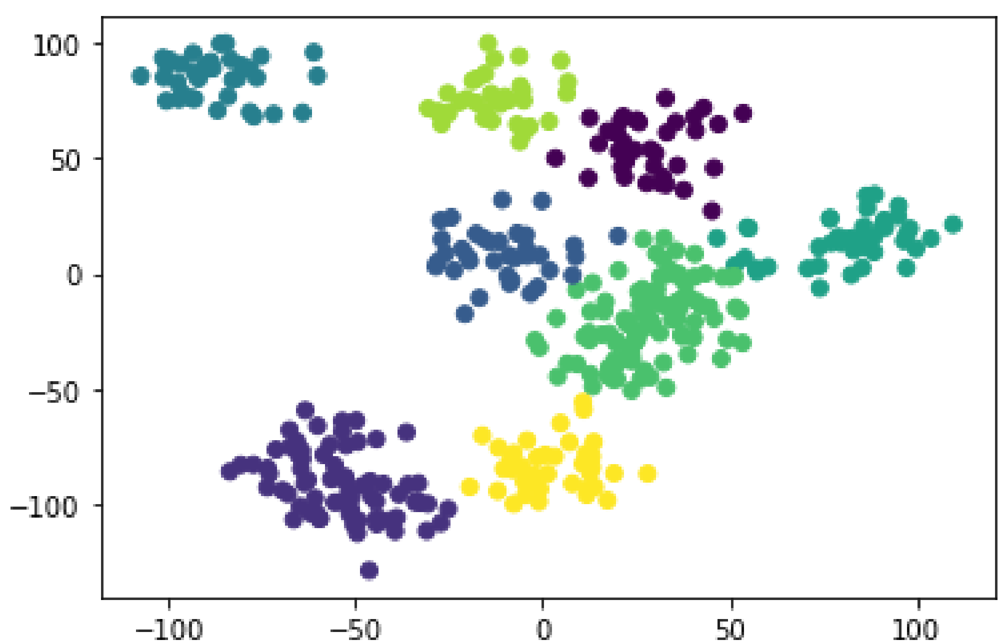
[32]:
```
vec=[]
vec1=[]
for i in range(10):
    n_clusters=i+2
    vec.append(count_clustering_scores(df, n_clusters, KMeans, silhouette_score))
    vec1.append(n_clusters)
plt.plot(vec1,vec)
plt.show()
```




Tu już łatwo zauważyć, że powinniśmy wybrać 8 klastrów.

[35]:
```
km=KMeans(n_clusters=8)
plt.scatter(df['x'],df['y'],c=km.fit_predict(df))
```

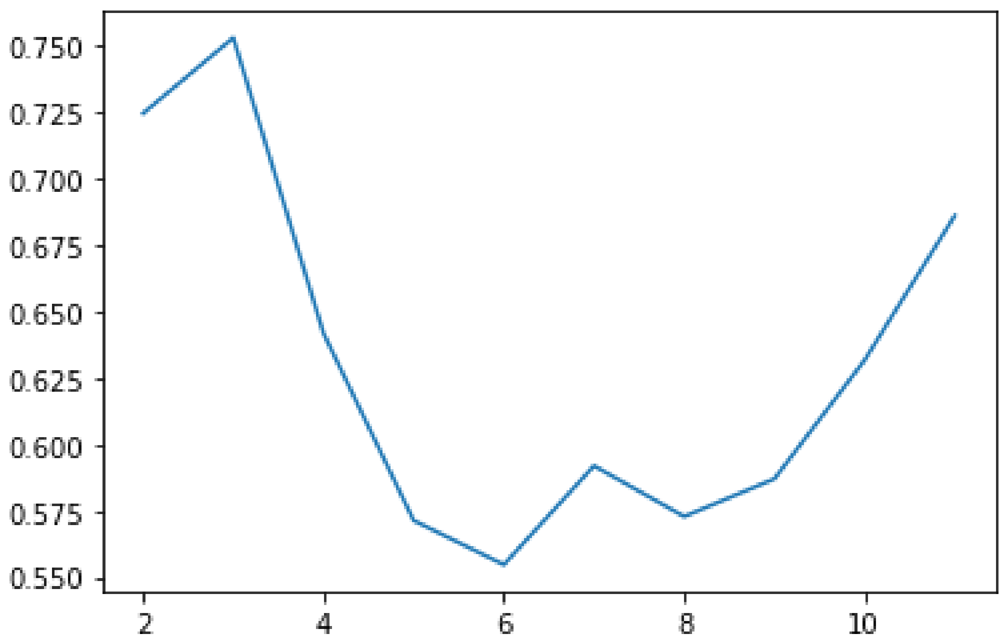
[35]:
```
<matplotlib.collections.PathCollection at 0x237ed7221f0>
```



## 2 Agglomerative Clustering

Zacznijmy od obliczenia liczby klastrów za pomocą Daviesa- Bouldina

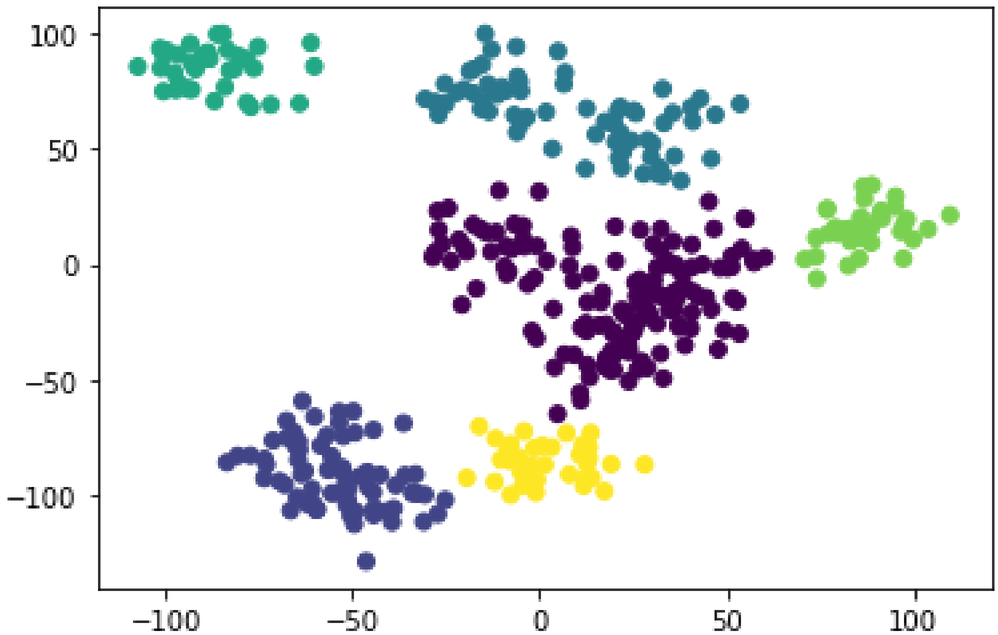
[41]:
```
vec=[]
vec1=[]
for i in range(10):
    n_clusters=i+2
    vec.append(count_clustering_scores_agl(df, n_clusters,␣
 ↪AgglomerativeClustering, metrics.davies_bouldin_score))
    vec1.append(n_clusters)
plt.plot(vec1,vec)
plt.show()
```




Tu już łątwo określić optymalną liczbę klastrów -6

[42]:
```
ag=AgglomerativeClustering(n_clusters=6, linkage='ward')
plt.scatter(df['x'],df['y'],c=ag.fit_predict(df))
```

[42]: <matplotlib.collections.PathCollection at 0x237ed820eb0>



100
50
0
−50
−100
−100
−50
0
50
100


```
[ ]:
```