Unsupervised Machine Learning

Summary

The dataset used was taken from the UCI site and suitable for an Unsupervised Learning analisys of wine types.
These data are the results of a chemical analysis of wines grown in the same region in Italy but derived from three different cultivars. The analysis determined the quantities of 13 constituents found in each of the three types of wines.

The dataset has 178 instances and 13 attributes.

The attributes are:
1) Alcohol
2) Malic acid
3) Ash
4) Alcalinity of ash
5) Magnesium
6) Total phenols
7) Flavanoids
8) Nonflavanoid phenols
9) Proanthocyanins
10) Color intensity
11) Hue
12) OD280/OD315 of diluted wines
13) Proline

The goal of this project is find a possible segmentation into the dataset.

In [ ]:
```
def warn(*args, **kwargs):
    pass
import warnings
warnings.warn = warn
```

In [ ]:
```
# Import libraries
import seaborn as sns
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans, AgglomerativeClustering, DBSCAN
from scipy.cluster import hierarchy
from sklearn.metrics import silhouette_samples, silhouette_score
```

Data Exploration

In [ ]:
```
# Import the data
data = pd.read_csv("wine-clustering.csv")

data.head()
```

Out[ ]:

	Alcohol	Malic_Acid	Ash	Ash_Alcanity	Magnesium	Total_Phenols	Flavanoids	Nonflavanoid_Ph
0	14.23	1.71	2.43	15.6	127	2.80	3.06	
1	13.20	1.78	2.14	11.2	100	2.65	2.76	
2	13.16	2.36	2.67	18.6	101	2.80	3.24	
3	14.37	1.95	2.50	16.8	113	3.85	3.49	
4	13.24	2.59	2.87	21.0	118	2.80	2.69	

In [ ]:
```
data.shape
```

Out[ ]:
```
(178, 13)
```

In [ ]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 178 entries, 0 to 177
Data columns (total 13 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   Alcohol               178 non-null    float64
 1   Malic_Acid            178 non-null    float64
 2   Ash                   178 non-null    float64
 3   Ash_Alcanity          178 non-null    float64
 4   Magnesium             178 non-null    int64  
 5   Total_Phenols         178 non-null    float64
 6   Flavanoids            178 non-null    float64
 7   Nonflavanoid_Phenols  178 non-null    float64
 8   Proanthocyanins       178 non-null    float64
 9   Color_Intensity       178 non-null    float64
 10  Hue                   178 non-null    float64
 11  OD280                 178 non-null    float64
 12  Proline               178 non-null    int64  
dtypes: float64(11), int64(2)
memory usage: 18.2 KB
```

In [ ]:
```
data.describe
```

Out[ ]:
```
<bound method NDFrame.describe of      Alcohol  Malic_Acid   Ash  Ash_Alcanity  Magn
esium  Total_Phenols  \
0      14.23        1.71  2.43          15.6        127           2.80
1      13.20        1.78  2.14          11.2        100           2.65
2      13.16        2.36  2.67          18.6        101           2.80
3      14.37        1.95  2.50          16.8        113           3.85
4      13.24        2.59  2.87          21.0        118           2.80
..       ...         ...   ...           ...        ...            ...
173    13.71        5.65  2.45          20.5         95           1.68
174    13.40        3.91  2.48          23.0        102           1.80
175    13.27        4.28  2.26          20.0        120           1.59
176    13.17        2.59  2.37          20.0        120           1.65
177    14.13        4.10  2.74          24.5         96           2.05

     Flavanoids  Nonflavanoid_Phenols  Proanthocyanins  Color_Intensity   Hue  \
0          3.06                  0.28             2.29             5.64  1.04
```

```
1          2.76                0.26            1.28             4.38  1.05
2          3.24                0.30            2.81             5.68  1.03
3          3.49                0.24            2.18             7.80  0.86
4          2.69                0.39            1.82             4.32  1.04
..          ...                 ...             ...              ...   ...
173        0.61                0.52            1.06             7.70  0.64
174        0.75                0.43            1.41             7.30  0.70
175        0.69                0.43            1.35            10.20  0.59
176        0.68                0.53            1.46             9.30  0.60
177        0.76                0.56            1.35             9.20  0.61

     OD280  Proline
0     3.92     1065
1     3.40     1050
2     3.17     1185
3     3.45     1480
4     2.93      735
..     ...      ...
173   1.74      740
174   1.56      750
175   1.56      835
176   1.62      840
177   1.60      560

[178 rows x 13 columns]>
```

There are no null values

In [ ]:
```
data.isnull().sum()
```

Out[ ]:
```
Alcohol                 0
Malic_Acid              0
Ash                     0
Ash_Alcanity            0
Magnesium               0
Total_Phenols           0
Flavanoids              0
Nonflavanoid_Phenols    0
Proanthocyanins         0
Color_Intensity         0
Hue                     0
OD280                   0
Proline                 0
dtype: int64
```

Now we are looking fo possible duplicates

In [ ]:
```
print('Duplicates: ' + str(data.duplicated().sum()))
```

```
Duplicates: 0
```

Skewness

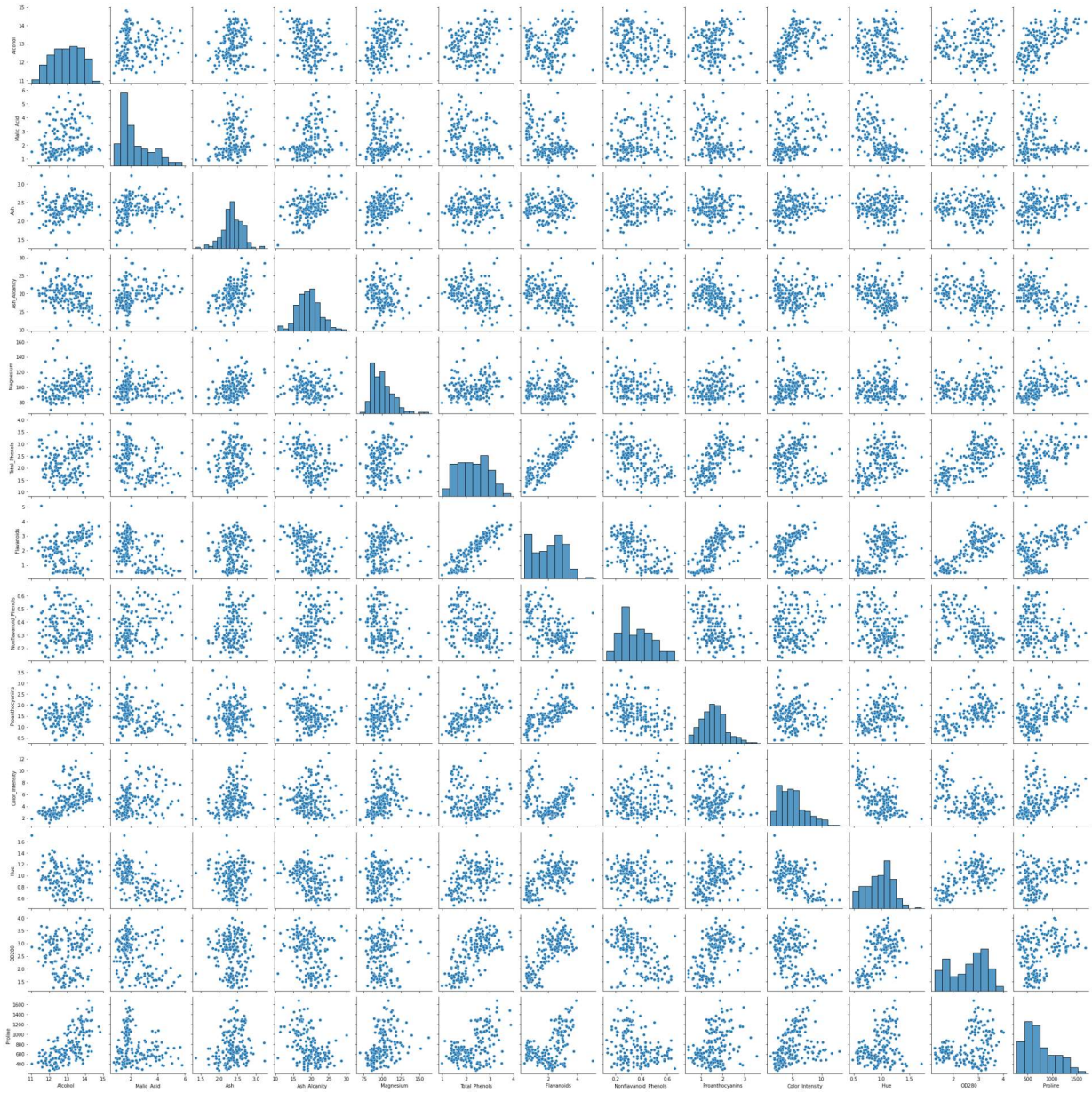
In [ ]:
```
log_columns = data.skew().sort_values(ascending=False)
log_columns = log_columns.loc[log_columns > 0.75]

log_columns
```

Out[ ]:
```
Magnesium          1.098191
Malic_Acid         1.039651
Color_Intensity    0.868585
Proline            0.767822
dtype: float64
```

Correlation

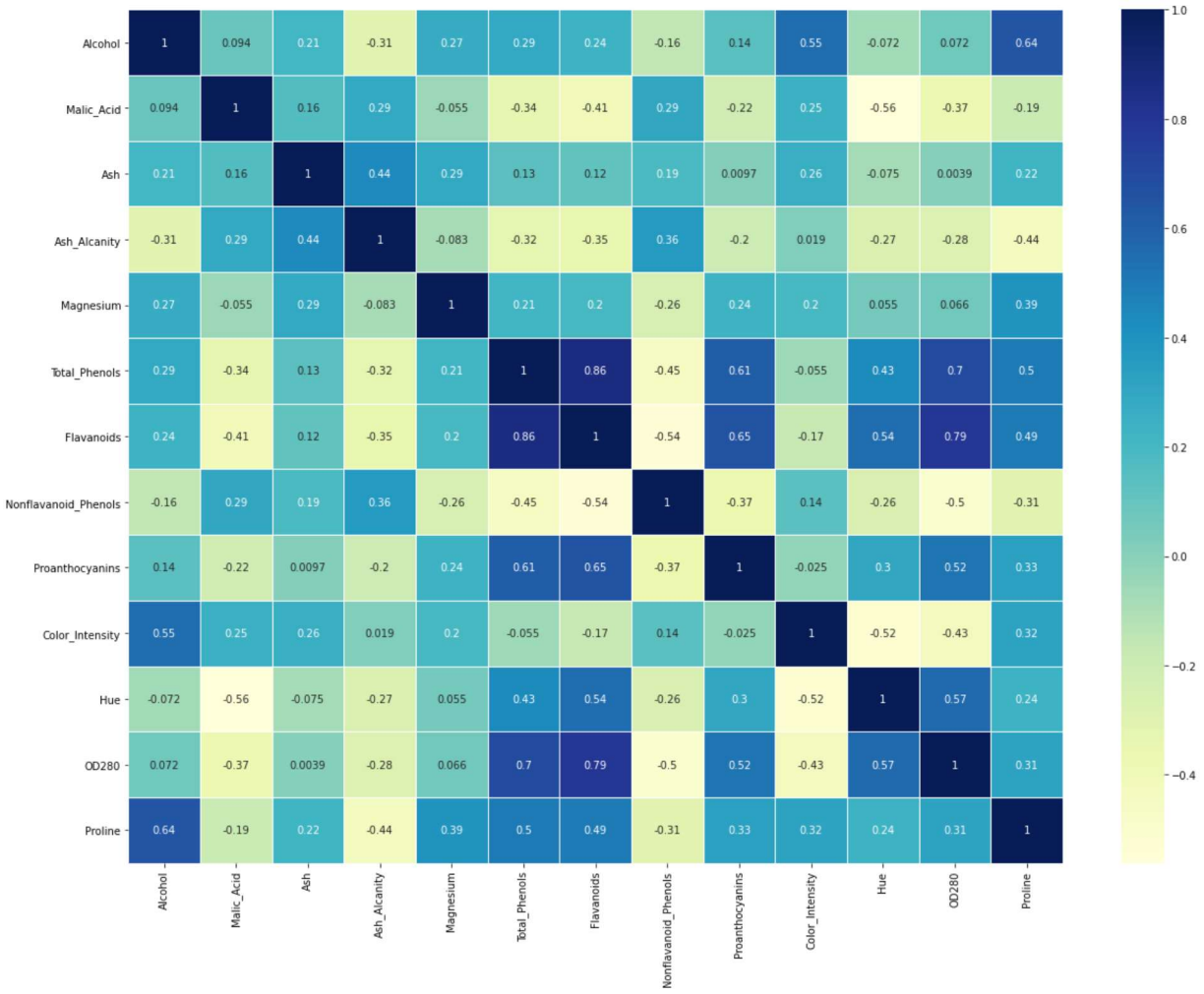
In [ ]:
```
sns.pairplot(data)
```

Out[ ]:
```
<seaborn.axisgrid.PairGrid at 0x128b2850190>
```



In [ ]:
```
corr = data.corr(method='pearson')
fig = plt.subplots(figsize = (20, 15))
sns.heatmap(corr,
            xticklabels=corr.columns,
            yticklabels=corr.columns,
            cmap='YlGnBu',
            annot=True,
            linewidth=0.5)
```

Out[ ]:
```
<AxesSubplot:>
```



As we can see, there are some notable correlations, but none of them is very high (|x|>0.9)

In [ ]:
```
# The log transformations
for col in log_columns.index:
    data[col] = np.log1p(data[col])
```

Scaling

In [ ]:
```
sc = StandardScaler()
feature_columns = [x for x in data.columns if x not in 'Net Income']
for col in feature_columns:
    data[col] = sc.fit_transform(data[[col]])

data.head(4)
```

Out[ ]:

	Alcohol	Malic_Acid	Ash	Ash_Alcanity	Magnesium	Total_Phenols	Flavanoids	Nonflavanc
0	1.518613	-0.511810	0.232053	-1.169593	1.844269	0.808997	1.034819	
1	0.246290	-0.429201	-0.827996	-2.490847	0.088329	0.568648	0.733629	
2	0.196879	0.184609	1.109334	-0.268738	0.161353	0.808997	1.215533	
3	1.691550	-0.236937	0.487926	-0.809251	0.985741	2.491446	1.466525	

Clustering

In [ ]:
```
```

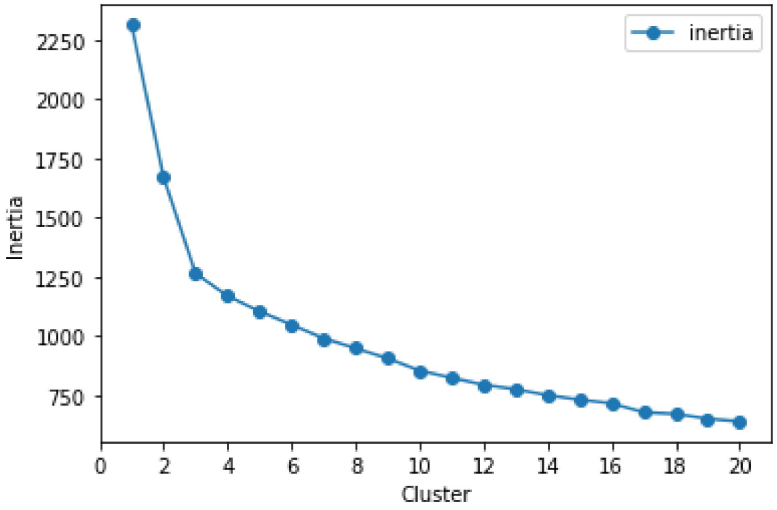
```
list_silhouette = []
```

K-MEANS

In [ ]:
```
km_list = list()

for clust in range(1,21):
    km = KMeans(n_clusters = clust, random_state = 42)
    km = km.fit(data)

    km_list.append(pd.Series({'clusters': clust,
                              'inertia': km.inertia_,
                              'model': km}))
```

In [ ]:
```
plot_data = (pd.concat(km_list, axis=1)
             .T
             [['clusters','inertia']]
             .set_index('clusters'))

ax = plot_data.plot(marker='o',ls='-')
ax.set_xticks(range(0,21,2))
ax.set_xlim(0,21)
ax.set(xlabel='Cluster', ylabel='Inertia')
```

Out[ ]:
```
[Text(0.5, 0, 'Cluster'), Text(0, 0.5, 'Inertia')]
```



In [ ]:
```
data_kmeans = data.copy()
data_kmeans.head()
```

Out[ ]:

	Alcohol	Malic_Acid	Ash	Ash_Alcanity	Magnesium	Total_Phenols	Flavanoids	Nonflavanc
0	1.518613	-0.511810	0.232053	-1.169593	1.844269	0.808997	1.034819	
1	0.246290	-0.429201	-0.827996	-2.490847	0.088329	0.568648	0.733629	
2	0.196879	0.184609	1.109334	-0.268738	0.161353	0.808997	1.215533	
3	1.691550	-0.236937	0.487926	-0.809251	0.985741	2.491446	1.466525	
4	0.295700	0.399085	1.840403	0.451946	1.303895	0.808997	0.663351	

In [ ]:

```
km = KMeans(n_clusters = 3, random_state=42)
km = km.fit(data)

data_kmeans['kmeans'] = km.predict(data)

list_silhouette.append(["K-MEANS", silhouette_score(data, km.labels_, metric = 'eucl
```

In [ ]:
```
(data_kmeans[['kmeans']]
 .groupby(['kmeans'])
 .size()
 .to_frame()
 .rename(columns={0:'number'}))
```

Out[ ]:

kmeans	number
0	65
1	62
2	51

Agglomerative

In [ ]:
```
data_agglom = data.copy()
data_agglom.head()
```

Out[ ]:

	Alcohol	Malic_Acid	Ash	Ash_Alcanity	Magnesium	Total_Phenols	Flavanoids	Nonflavanc
0	1.518613	-0.511810	0.232053	-1.169593	1.844269	0.808997	1.034819	
1	0.246290	-0.429201	-0.827996	-2.490847	0.088329	0.568648	0.733629	
2	0.196879	0.184609	1.109334	-0.268738	0.161353	0.808997	1.215533	
3	1.691550	-0.236937	0.487926	-0.809251	0.985741	2.491446	1.466525	
4	0.295700	0.399085	1.840403	0.451946	1.303895	0.808997	0.663351	

In [ ]:
```
ag = AgglomerativeClustering(n_clusters = 3, linkage='ward', compute_full_tree=True)
ag = ag.fit(data)
data_agglom['agglom'] = ag.fit_predict(data)

list_silhouette.append(["Agglomerative", silhouette_score(data, ag.labels_, metric =
```

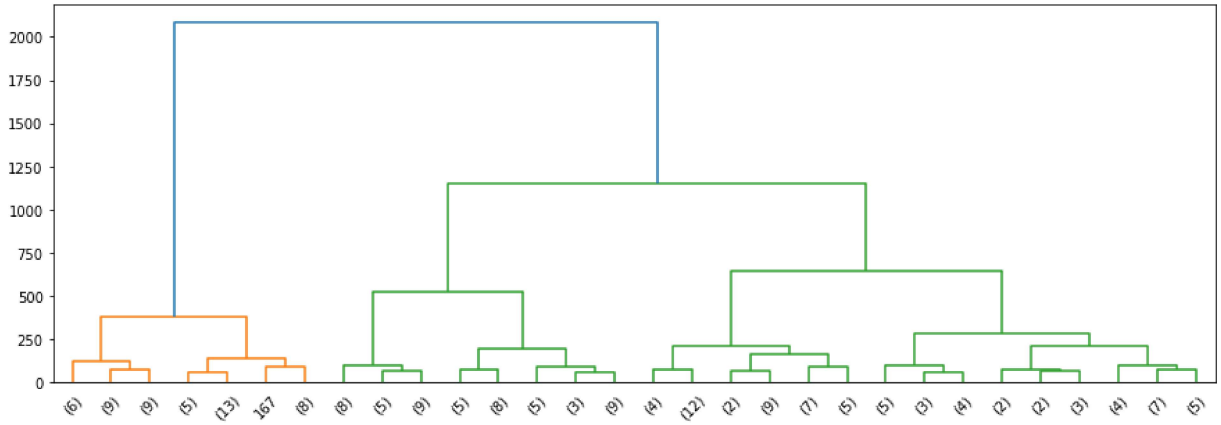
In [ ]:
```
(data_agglom[['agglom']]
 .groupby(['agglom'])
 .size()
 .to_frame()
 .rename(columns={0:'number'}))
```

Out[ ]:

agglom	number
0	64

	number
agglom	
1	52
2	62

In [ ]:
```
Z = hierarchy.linkage(ag.children_, method='ward')

fig, ax = plt.subplots(figsize=(15,5))

den = hierarchy.dendrogram(Z, orientation = 'top',
                           p = 30, truncate_mode = 'lastp',
                           show_leaf_counts = True, ax = ax)
```



DBSCAN

In [ ]:
```
from operator import itemgetter


eps = range(1,5)
min_samples = range(2,50,2)

best_pam = []

for e in eps:
    for min in min_samples:
        db = DBSCAN(eps = e, min_samples = min)

        db = db.fit(data)

        if any(db.labels_ == 1):
            best_pam.append([e,min, silhouette_score(data, db.labels_, metric = 'euc
        else:
            best_pam.append([e,min, 0])



sorted(best_pam, key = itemgetter(2), reverse = True)
```

Out[ ]:
```
[[3, 38, 0.27565094010739766],
 [3, 40, 0.26175215725505535],
 [3, 32, 0.24326425861844828],
 [3, 34, 0.24326425861844828],
 [3, 36, 0.23847005857888867],
 [3, 24, 0.22941164227792582],
```

```
[3, 22, 0.228990516479084],
[3, 26, 0.22094201626929436],
[3, 28, 0.21801385724934627],
[3, 30, 0.21801385724934627],
[3, 2, 0.17398258554621554],
[2, 10, 0.054775666513855556],
[2, 8, 0.05400368487128416],
[2, 6, 0.014819730593513405],
[2, 12, 0.014148026836539797],
[2, 4, 0.012085771610113129],
[1, 2, 0],
[1, 4, 0],
[1, 6, 0],
[1, 8, 0],
[1, 10, 0],
[1, 12, 0],
[1, 14, 0],
[1, 16, 0],
[1, 18, 0],
[1, 20, 0],
[1, 22, 0],
[1, 24, 0],
[1, 26, 0],
[1, 28, 0],
[1, 30, 0],
[1, 32, 0],
[1, 34, 0],
[1, 36, 0],
[1, 38, 0],
[1, 40, 0],
[1, 42, 0],
[1, 44, 0],
[1, 46, 0],
[1, 48, 0],
[2, 14, 0],
[2, 16, 0],
[2, 18, 0],
[2, 20, 0],
[2, 22, 0],
[2, 24, 0],
[2, 26, 0],
[2, 28, 0],
[2, 30, 0],
[2, 32, 0],
[2, 34, 0],
[2, 36, 0],
[2, 38, 0],
[2, 40, 0],
[2, 42, 0],
[2, 44, 0],
[2, 46, 0],
[2, 48, 0],
[3, 4, 0],
[3, 6, 0],
[3, 8, 0],
[3, 10, 0],
[3, 12, 0],
[3, 14, 0],
[3, 16, 0],
[3, 18, 0],
[3, 20, 0],
[3, 42, 0],
[3, 44, 0],
[3, 46, 0],
[3, 48, 0],
[4, 2, 0],
[4, 4, 0],
[4, 6, 0],
[4, 8, 0],
```

```
[4, 10, 0],
[4, 12, 0],
[4, 14, 0],
[4, 16, 0],
[4, 18, 0],
[4, 20, 0],
[4, 22, 0],
[4, 24, 0],
[4, 26, 0],
[4, 28, 0],
[4, 30, 0],
[4, 32, 0],
[4, 34, 0],
[4, 36, 0],
[4, 38, 0],
[4, 40, 0],
[4, 42, 0],
[4, 44, 0],
[4, 46, 0],
[4, 48, 0],
[2, 2, -0.041999159739381446]]
```

The best parameters are esp = 3 e min_samples = 38

In [ ]:
```
db = DBSCAN(eps = 3, min_samples = 38)

db = db.fit(data)

list_silhouette.append(["DBSCAN", silhouette_score(data, db.labels_, metric = 'eucli
```

Conclusions

In [ ]:
```
df2 = pd.DataFrame(list_silhouette ,columns = ['Algorithm','Silhouette'])
df2
```

Out[ ]:

	Algorithm	Silhouette
0	K-MEANS	0.298215
1	Agglomerative	0.294721
2	DBSCAN	0.275651

As can be seen, we have very close results for each of the algorithms tested.
The one that performs best, however, is the K-Means, with good Silhouette values.

Looking further, surely it would be interesting to conduct further tests with the techniques for handling dimensionality reduction, like PCA.

Raffaele Pane