PD7

June 15, 2021

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

[2]:
```
test = pd.read_csv('test.csv')
train = pd.read_csv('train.csv')
val = pd.read_csv('val.csv')
```

[3]:
```
print(train.shape)
train.head()
```

```
(95, 13)
```

[3]:

	Alcohol	Malic acid	Ash	Alcalinity of ash	Magnesium	Total phenols	\
0	12.72	1.75	2.28	22.5	84	1.38	
1	13.23	3.30	2.28	18.5	98	1.80	
2	12.58	1.29	2.10	20.0	103	1.48	
3	12.37	1.17	1.92	19.6	78	2.11	
4	13.84	4.12	2.38	19.5	89	1.80	

	Flavanoids	Nonflavanoid phenols	Proanthocyanins	Color intensity	Hue	\
0	1.76	0.48	1.63	3.30	0.88	
1	0.83	0.61	1.87	10.52	0.56	
2	0.58	0.53	1.40	7.60	0.58	
3	2.00	0.27	1.04	4.68	1.12	
4	0.83	0.48	1.56	9.01	0.57	

	OD280/OD315 of diluted wines	Proline
0	2.42	488
1	1.51	675
2	1.55	640
3	3.48	510
4	1.64	480

[4]:
```
print(test.shape)
test.head()
```

```
(17, 14)
```

[4]:
```
   class  Alcohol  Malic acid   Ash  Alcalinity of ash  Magnesium  \
0      0    13.34        0.94  2.36               17.0        110
1      0    12.00        0.92  2.00               19.0         86
2      0    11.84        0.89  2.58               18.0         94
3      0    12.47        1.52  2.20               19.0        162
4      0    11.81        2.12  2.74               21.5        134

   Total phenols  Flavanoids  Nonflavanoid phenols  Proanthocyanins  \
0           2.53        1.30                  0.55             0.42
1           2.42        2.26                  0.30             1.43
2           2.20        2.21                  0.22             2.35
3           2.50        2.27                  0.32             3.28
4           1.60        0.99                  0.14             1.56

   Color intensity   Hue  OD280/OD315 of diluted wines  Proline
0             3.17  1.02                          1.93      750
1             2.50  1.38                          3.12      278
2             3.05  0.79                          3.08      520
3             2.60  1.16                          2.63      937
4             2.50  0.95                          2.26      625
```

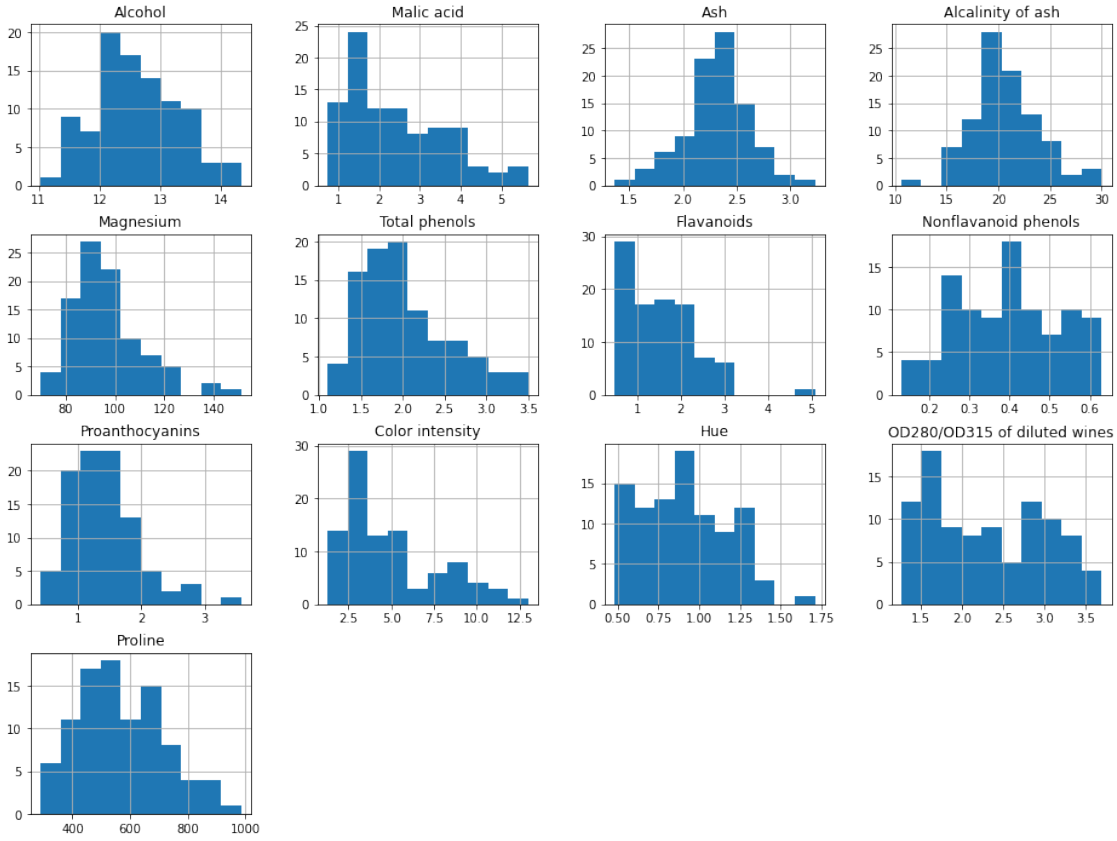
[5]:
```
print(val.shape)
val.head()
```

```
(17, 13)
```

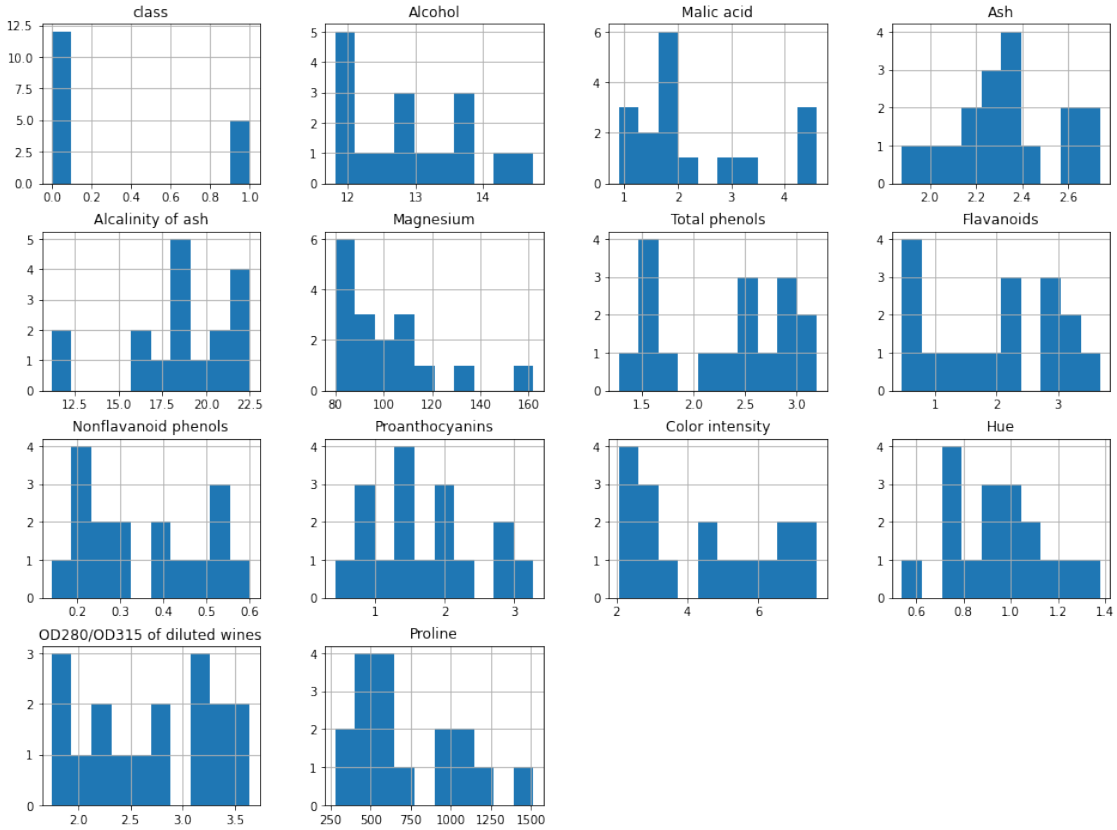
[5]:
```
   Alcohol  Malic acid   Ash  Alcalinity of ash  Magnesium  Total phenols  \
0    13.86        1.51  2.67               25.0         86           2.95
1    13.40        3.91  2.48               23.0        102           1.80
2    12.82        3.37  2.30               19.5         88           1.48
3    12.37        1.07  2.10               18.5         88           3.52
4    13.50        1.81  2.61               20.0         96           2.53

   Flavanoids  Nonflavanoid phenols  Proanthocyanins  Color intensity   Hue  \
0        2.86                  0.21             1.87             3.38  1.36
1        0.75                  0.43             1.41             7.30  0.70
2        0.66                  0.40             0.97            10.26  0.72
3        3.75                  0.24             1.95             4.50  1.04
4        2.61                  0.28             1.66             3.52  1.12

   OD280/OD315 of diluted wines  Proline
0                          3.16      410
1                          1.56      750
2                          1.75      685
3                          2.77      660
4                          3.82      845
```

[6]:
```
train.hist(figsize=(16,12))
plt.show()
```

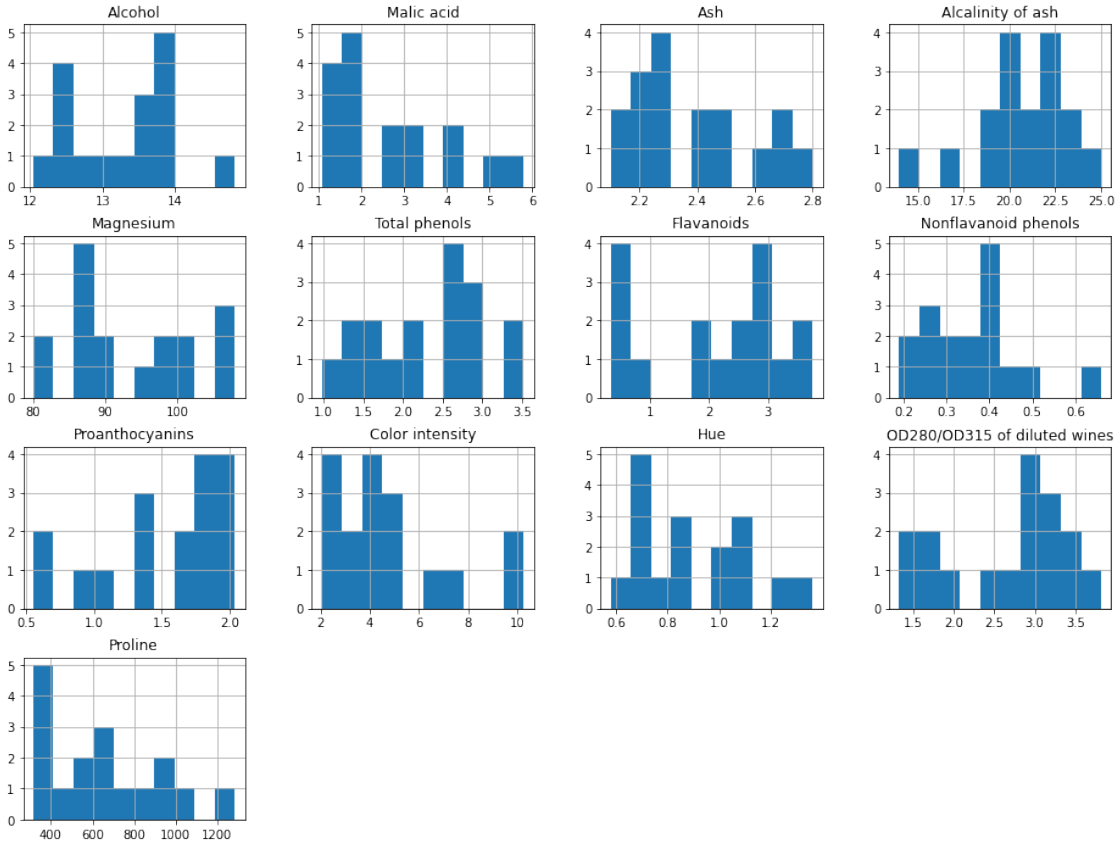
```
c:\users\user\appdata\local\programs\python\python38-32\lib\site-
packages\pandas\plotting\_matplotlib\tools.py:400: MatplotlibDeprecationWarning:
The is_first_col function was deprecated in Matplotlib 3.4 and will be removed
two minor releases later. Use ax.get_subplotspec().is_first_col() instead.
  if ax.is_first_col():
```



[7]:
```
test.hist(figsize=(16,12))
plt.show()
```



[8]:
```
val.hist(figsize=(16,12))
plt.show()
```



[9]:
```
print(train.info())
print(test.info())
print(val.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 95 entries, 0 to 94
Data columns (total 13 columns):
 #   Column                        Non-Null Count  Dtype
---  ------                        --------------  -----
 0   Alcohol                       95 non-null     float64
 1   Malic acid                    95 non-null     float64
 2   Ash                           95 non-null     float64
 3   Alcalinity of ash             95 non-null     float64
 4   Magnesium                     95 non-null     int64
 5   Total phenols                 95 non-null     float64
 6   Flavanoids                    95 non-null     float64
 7   Nonflavanoid phenols          95 non-null     float64
 8   Proanthocyanins               95 non-null     float64
 9   Color intensity               95 non-null     float64
 10  Hue                           95 non-null     float64
 11  OD280/OD315 of diluted wines  95 non-null     float64
```

```
 12  Proline                        95 non-null     int64
dtypes: float64(11), int64(2)
memory usage: 9.7 KB
None
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 17 entries, 0 to 16
Data columns (total 14 columns):
 #   Column                         Non-Null Count  Dtype
---  ------                         --------------  -----
 0   class                          17 non-null     int64
 1   Alcohol                        17 non-null     float64
 2   Malic acid                     17 non-null     float64
 3   Ash                            17 non-null     float64
 4   Alcalinity of ash              17 non-null     float64
 5   Magnesium                      17 non-null     int64
 6   Total phenols                  17 non-null     float64
 7   Flavanoids                     17 non-null     float64
 8   Nonflavanoid phenols           17 non-null     float64
 9   Proanthocyanins                17 non-null     float64
 10  Color intensity                17 non-null     float64
 11  Hue                            17 non-null     float64
 12  OD280/OD315 of diluted wines   17 non-null     float64
 13  Proline                        17 non-null     int64
dtypes: float64(11), int64(3)
memory usage: 1.9 KB
None
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 17 entries, 0 to 16
Data columns (total 13 columns):
 #   Column                         Non-Null Count  Dtype
---  ------                         --------------  -----
 0   Alcohol                        17 non-null     float64
 1   Malic acid                     17 non-null     float64
 2   Ash                            17 non-null     float64
 3   Alcalinity of ash              17 non-null     float64
 4   Magnesium                      17 non-null     int64
 5   Total phenols                  17 non-null     float64
 6   Flavanoids                     17 non-null     float64
 7   Nonflavanoid phenols           17 non-null     float64
 8   Proanthocyanins                17 non-null     float64
 9   Color intensity                17 non-null     float64
 10  Hue                            17 non-null     float64
 11  OD280/OD315 of diluted wines   17 non-null     float64
 12  Proline                        17 non-null     int64
dtypes: float64(11), int64(2)
memory usage: 1.8 KB
None
```

W danych nie ma braków, zbiór test ma dodatkową kolumnę class informującą czy wiersz jest obserwacją odstającą, po rozkładach widać prawdopodobne miejsca występowania outllierów w zbiorach val i test oraz ich brak w train.

[10]:
```
from sklearn.mixture import GaussianMixture
from sklearn.metrics import f1_score, precision_score, recall_score
```

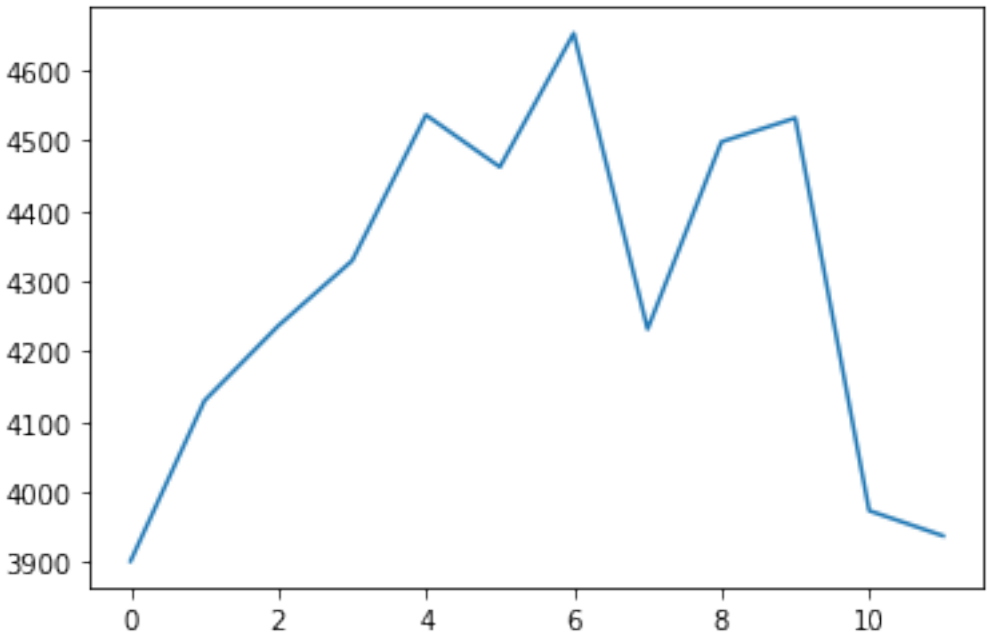
GMM zwraca prawdopodobieństwo przyporządkowania obserwacji do klastra, więc obserwacje o niskim i odstającym od reszty prawdopodobieństwie mogą być traktowane jako outliery.

Za pomocą funcji bic wyznaczmy odpowiednią liczbę klastrów - im mniejsza warość funkcji tym lepiej.

[11]:
```
bic = []
for i in range(1, 13):
    gmm = GaussianMixture(n_components = i, random_state = 29, covariance_type␣
 ↪= "full")
    gmm.fit(train)
    bic.append(gmm.bic(train))
```

[12]:
```
sns.lineplot(data = bic)
```

[12]:
```
<AxesSubplot:>
```



Najmniejszą wartość funkcja bic osiąga dla 1 klastra, stwórzmy więc właściwy model:

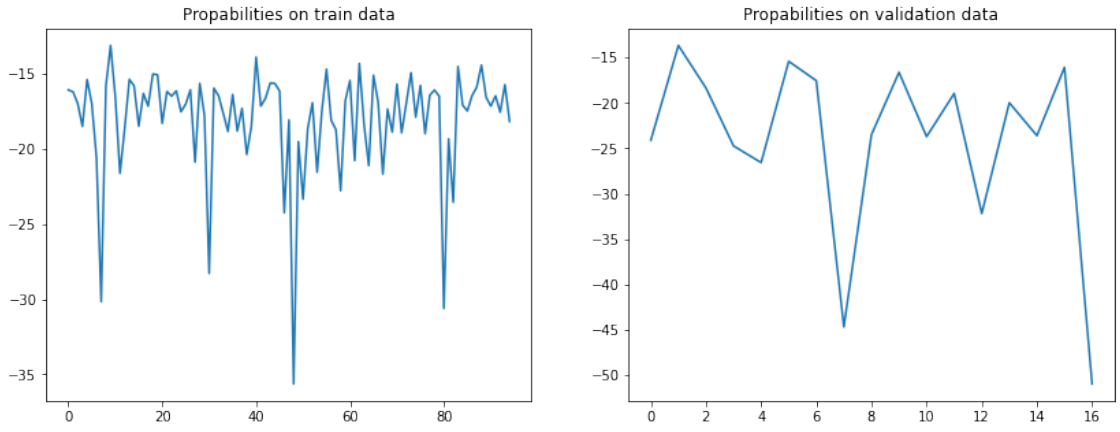
[13]:
```
gmm = GaussianMixture(covariance_type='full', n_components=1, random_state=29)
gmm.fit(train)
```

[13]:
```
GaussianMixture(random_state=29)
```

W zbiorze train nie ma outlierów a w val są, porównajmy więc prawdopodbieństwa i wyznaczmy na ich podstawie treshold.

[14]:
```
train_prop = gmm.score_samples(train)
val_prop = gmm.score_samples(val)
```

[15]:
```
fig, (ax1, ax2) = plt.subplots(1, 2, figsize = (14, 5))

sns.lineplot(data = train_prop, ax = ax1)
ax1.set_title('Propabilities on train data')

sns.lineplot(data = val_prop, ax = ax2)
ax2.set_title('Propabilities on validation data')

plt.show()
```



[16]:
```
#bierzemy treshold -30
t = -32
```

[17]:
```
test_prop = gmm.score_samples(test.drop('class', axis = 1))
```

[18]:
```
test_pred = np.where(test_prop < t, 1, 0)
```

[19]:
```
print(f'F1: {f1_score(test["class"], test_pred)}\n' +
      f'Precision: {precision_score(test["class"], test_pred)}\n' +
      f'Recall: {recall_score(test["class"], test_pred)}')
```

```
F1: 0.9090909090909091
Precision: 0.8333333333333334
Recall: 1.0
```

Metryki osiągają wysokie wyniki, zatem algorytm GMM nadaje się do wyszukiwania outlierów.