Padronização ou Normalização

In [1]:
```
import pandas as pd
from matplotlib import pyplot as plt
import seaborn as sns

# Biblioteca para fazer a PADRONIZAÇÃO
from sklearn.preprocessing import StandardScaler

# Biblioteca para fazer a NORMALIZAÇÃO
from sklearn.preprocessing import MinMaxScaler

# Bilioteca para separação de dados em treino e teste
from sklearn.model_selection import train_test_split

# Biblioteca para calcular a acuracia do modelo
from sklearn.metrics import accuracy_score

# Algoritmo KNN
from sklearn.neighbors import KNeighborsClassifier

# Algoritmo SVM
from sklearn import svm
```

In [2]:
```
# Carregando o arquivo
df_original = pd.read_csv("https://archive.ics.uci.edu/ml/machine-learning-database
```

In [3]:
```
# Nomeando as colunas
df_original.columns = ['CLASS','ALCOHOL','MALICACID','ASH','ASHALCALINITY','MAGNESI
                       'NONFLAVONOIDSPHENOLS','PRONTHOCYANINS','COLORINTENSITY','HUE

# Visualizando as primeiras linhas do DataFrame
df_original.head()
```

Out[3]:

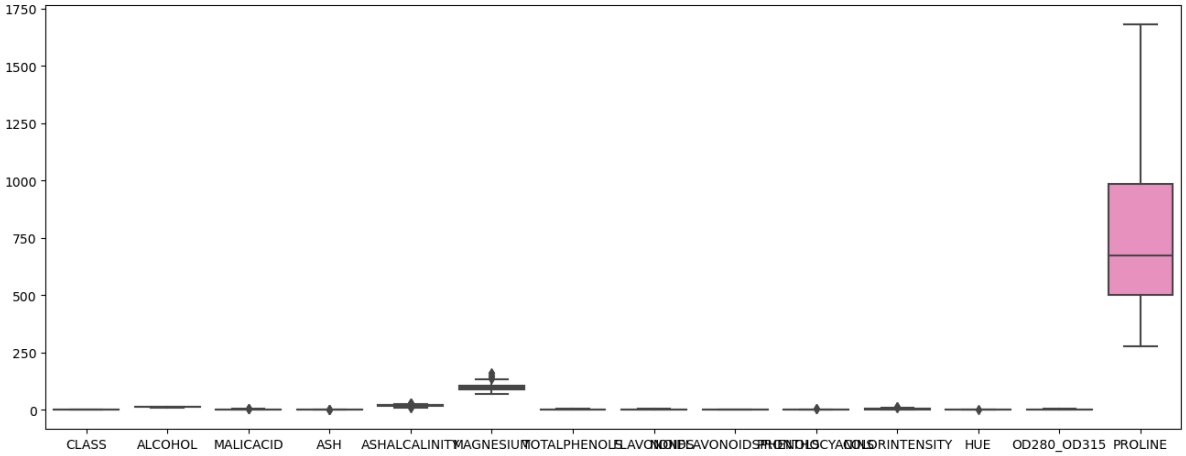
	CLASS	ALCOHOL	MALICACID	ASH	ASHALCALINITY	MAGNESIUM	TOTALPHENOLS	FLAVON(
0	1	13.20	1.78	2.14	11.2	100	2.65	
1	1	13.16	2.36	2.67	18.6	101	2.80	
2	1	14.37	1.95	2.50	16.8	113	3.85	
3	1	13.24	2.59	2.87	21.0	118	2.80	
4	1	14.20	1.76	2.45	15.2	112	3.27	

In [4]:
```
# Analisando um resumo das medidas
df_original.describe()
```

Out[4]:

	CLASS	ALCOHOL	MALICACID	ASH	ASHALCALINITY	MAGNESIUM	TOTALPHE
count	177.000000	177.000000	177.000000	177.000000	177.000000	177.000000	177.0(
mean	1.943503	12.993672	2.339887	2.366158	19.516949	99.587571	2.2(
std	0.773991	0.808808	1.119314	0.275080	3.336071	14.174018	0.6(
min	1.000000	11.030000	0.740000	1.360000	10.600000	70.000000	0.9(
25%	1.000000	12.360000	1.600000	2.210000	17.200000	88.000000	1.7(
50%	2.000000	13.050000	1.870000	2.360000	19.500000	98.000000	2.3(
75%	3.000000	13.670000	3.100000	2.560000	21.500000	107.000000	2.8(
max	3.000000	14.830000	5.800000	3.230000	30.000000	162.000000	3.8(

In [5]:
```
# Gerando um BoxPlot de todas variaveis
plt.figure(figsize=(16,6))
ax = sns.boxplot(data = df_original)
```



In [6]:
```
# Criando um objeto para PADRONIZAÇÃO dos dados
obj_padronizacao = StandardScaler().fit(df_original)
```

In [7]:
```
# Aplicando a PADRONIZAÇÃO
df_padronizado = obj_padronizacao.transform(df_original)
```

In [8]:
```
# Observe que é criado um ARRAY dos dados
df_padronizado
```

Out[8]:
```
array([[-1.22246766,  0.2558245 , -0.50162433, ...,  0.40709978,
         1.13169801,  0.97105248],
       [-1.22246766,  0.20622873,  0.01802001, ...,  0.3195674 ,
         0.80457911,  1.40099798],
       [-1.22246766,  1.70650069, -0.34931478, ..., -0.4244579 ,
         1.20281081,  2.34050852],
       ...,
       [ 1.36887097,  0.34261709,  1.73822194, ..., -1.60614514,
        -1.48525319,  0.28632445],
       [ 1.36887097,  0.21862767,  0.22408586, ..., -1.56237895,
        -1.39991783,  0.30224836],
       [ 1.36887097,  1.40892609,  1.57695301, ..., -1.51861275,
        -1.42836295, -0.58949046]])
```

In [9]:
```
# Transformando para DataFrame e nomeando as colunas
df_padronizado = pd.DataFrame(df_padronizado)
df_padronizado.columns = ['CLASS','ALCOHOL','MALICACID','ASH','ASHALCALINITY','MAGN
```

```
                                'NONFLAVONOIDSPHENOLS','PRONTHOCYANINS','COLORINTENSITY','HUE

# Visualizando os dados padronizados
df_padronizado.head()
```

Out[9]:

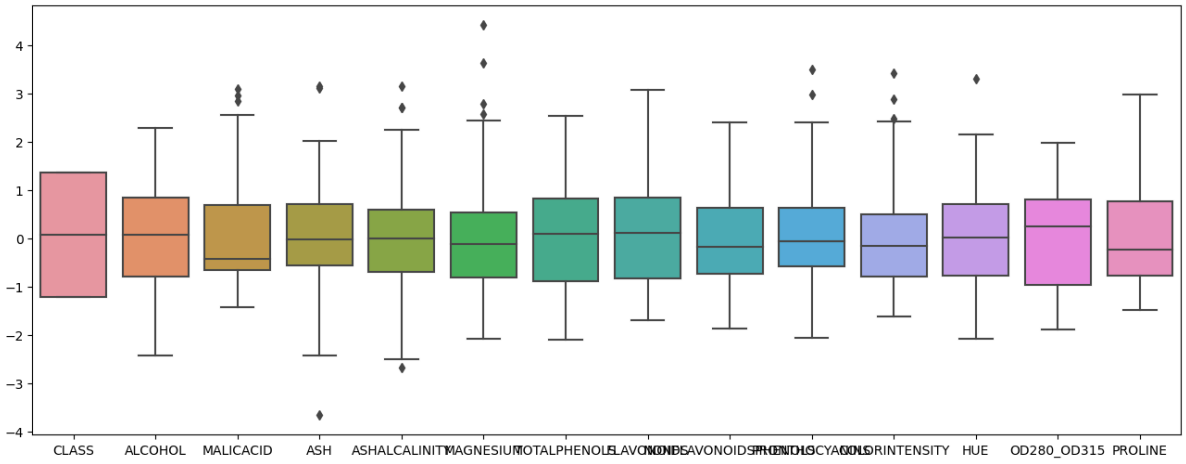
	CLASS	ALCOHOL	MALICACID	ASH	ASHALCALINITY	MAGNESIUM	TOTALPHENOLS	F
0	-1.222468	0.255824	-0.501624	-0.824485	-2.500110	0.029180	0.572666	
1	-1.222468	0.206229	0.018020	1.107690	-0.275639	0.099932	0.812784	
2	-1.222468	1.706501	-0.349315	0.487935	-0.816726	0.948953	2.493609	
3	-1.222468	0.305420	0.224086	1.836812	0.445811	1.302712	0.812784	
4	-1.222468	1.495719	-0.519543	0.305655	-1.297693	0.878201	1.565153	

In [10]:
```
# Visualizando as medidas dos dados PADRONIZADOS
df_padronizado.describe()
```

Out[10]:

	CLASS	ALCOHOL	MALICACID	ASH	ASHALCALINITY	MAGNESIUM
count	1.770000e+02	1.770000e+02	1.770000e+02	1.770000e+02	1.770000e+02	1.770000e+0
mean	1.327250e-15	-2.609338e-16	4.252719e-16	-4.378168e-16	-6.410440e-16	-1.028681e-1
std	1.002837e+00	1.002837e+00	1.002837e+00	1.002837e+00	1.002837e+00	1.002837e+0
min	-1.222468e+00	-2.434746e+00	-1.433400e+00	-3.668064e+00	-2.680472e+00	-2.093373e+0
25%	-1.222468e+00	-7.856866e-01	-6.628933e-01	-5.692924e-01	-6.964846e-01	-8.198411e-0
50%	7.320166e-02	6.984037e-02	-4.209899e-01	-2.245039e-02	-5.094986e-03	-1.123234e-0
75%	1.368871e+00	8.385748e-01	6.810145e-01	7.066723e-01	5.961134e-01	5.244425e-0
max	1.368871e+00	2.276852e+00	3.100048e+00	3.149233e+00	3.151249e+00	4.415790e+0

In [11]:
```
# Gerando os BoxPlot dos dados Padronizados
plt.figure(figsize=(16,6))
ax = sns.boxplot(data = df_padronizado)
```



In [ ]:
```
```

In [12]:
```
# Visualizando novamente o DataFrame Original
df_original.head()
```

Out[12]:

	CLASS	ALCOHOL	MALICACID	ASH	ASHALCALINITY	MAGNESIUM	TOTALPHENOLS	FLAVON(
0	1	13.20	1.78	2.14	11.2	100	2.65	
1	1	13.16	2.36	2.67	18.6	101	2.80	
2	1	14.37	1.95	2.50	16.8	113	3.85	
3	1	13.24	2.59	2.87	21.0	118	2.80	
4	1	14.20	1.76	2.45	15.2	112	3.27	

In [13]:
```
# Criando um objeto para NORMALIZAR os dados
obj_normalizacao = MinMaxScaler().fit(df_original)
```

In [14]:
```
# Aplicando a NORMALIZAÇÃO
df_normalizado = obj_normalizacao.transform(df_original)
```

In [15]:
```
# Visualizando os dados normalizados
df_normalizado
```

Out[15]:
```
array([[0.        , 0.57105263, 0.2055336 , ..., 0.46341463, 0.78021978,
        0.55064194],
       [0.        , 0.56052632, 0.3201581 , ..., 0.44715447, 0.6959707 ,
        0.64693295],
       [0.        , 0.87894737, 0.23913043, ..., 0.30894309, 0.7985348 ,
        0.85734665],
       ...,
       [1.        , 0.58947368, 0.69960474, ..., 0.08943089, 0.10622711,
        0.39728959],
       [1.        , 0.56315789, 0.36561265, ..., 0.09756098, 0.12820513,
        0.40085592],
       [1.        , 0.81578947, 0.66403162, ..., 0.10569106, 0.12087912,
        0.20114123]])
```

In [16]:
```
# Transformando para DataFrame e nomeando as colunas
df_normalizado = pd.DataFrame(df_normalizado)
df_normalizado.columns = ['CLASS','ALCOHOL','MALICACID','ASH','ASHALCALINITY','MAGN
                          'NONFLAVONOIDSPHENOLS','PRONTHOCYANINS','COLORINTENSITY','HUE


# Visualizando os dados NORMALIZADOS
df_normalizado.head()
```

Out[16]:

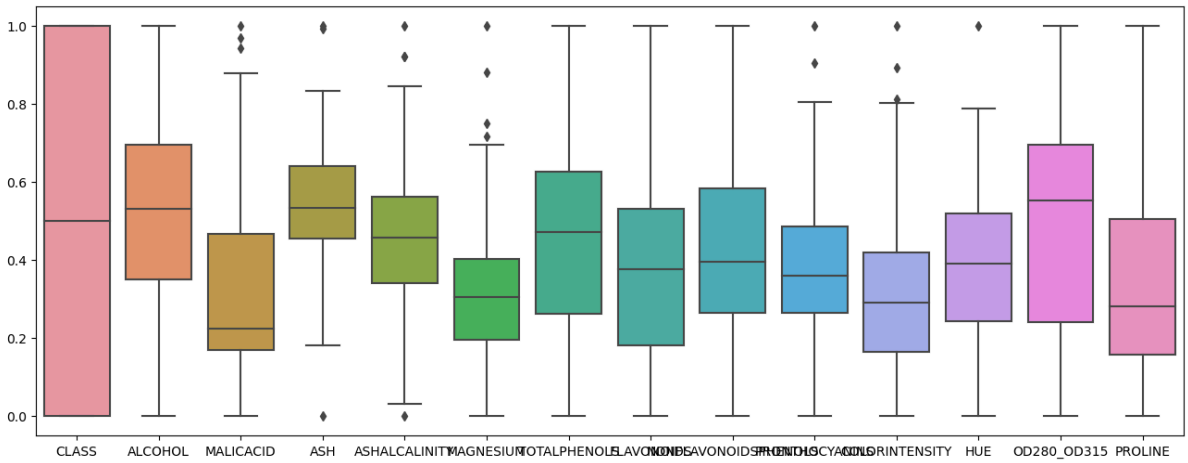
	CLASS	ALCOHOL	MALICACID	ASH	ASHALCALINITY	MAGNESIUM	TOTALPHENOLS	FLAV
0	0.0	0.571053	0.205534	0.417112	0.030928	0.326087	0.575862	
1	0.0	0.560526	0.320158	0.700535	0.412371	0.336957	0.627586	
2	0.0	0.878947	0.239130	0.609626	0.319588	0.467391	0.989655	
3	0.0	0.581579	0.365613	0.807487	0.536082	0.521739	0.627586	
4	0.0	0.834211	0.201581	0.582888	0.237113	0.456522	0.789655	

In [67]:
```
# Visualizando as medidas dos dados normalizados
df_normalizado.describe()
```

Out[67]:

	CLASS	ALCOHOL	MALICACID	ASH	ASHALCALINITY	MAGNESIUM	TOTALPHE
count	177.000000	177.000000	177.000000	177.000000	177.000000	177.000000	177.0
mean	0.471751	0.516756	0.316183	0.538053	0.459637	0.321604	0.4
std	0.386996	0.212844	0.221208	0.147102	0.171962	0.154065	0.2
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0
25%	0.000000	0.350000	0.169960	0.454545	0.340206	0.195652	0.2
50%	0.500000	0.531579	0.223320	0.534759	0.458763	0.304348	0.4
75%	1.000000	0.694737	0.466403	0.641711	0.561856	0.402174	0.6
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.0

In [68]:
```
#from matplotlib import pyplot as plt
plt.figure(figsize=(16,6))
ax = sns.boxplot(data = df_normalizado)
```



In [ ]:

In [ ]:

Algoritmo KNN

In [69]:
```
# Gerando um DataFrame das variáveis preditoras originais SEM a variável TARGET
VAR_PREDITORAS_ORIG = df_original.drop('CLASS', axis = 1)
```

In [70]:
```
# Gerando um DataFrame somente da variavel TARGET
VAR_TARGET = df_original['CLASS']
```

In [71]:
```
# Separando os dados em TREINO e TESTE (VARIAVEIS ORIGINAIS)
# 70% PARA TREINO E 30% PARA TESTE
X_train, X_test, Y_train, Y_test = train_test_split(VAR_PREDITORAS_ORIG, VAR_TARGET
```

In [72]:
```
knn = KNeighborsClassifier()
```

In [73]:
```
knn.fit(X_train, Y_train)
```

Out[73]:
```
KNeighborsClassifier()
```

In [78]:
```
import warnings
warnings.filterwarnings("ignore", category=FutureWarning, message=".*keepdims.*")
resultados = knn.predict(X_test)
```

In [79]:
```
score = accuracy_score(Y_test, resultados)
```

In [80]:
```
score
```

Out[80]:
```
0.6666666666666666
```

In [ ]:
```

```

In [82]:
```
# Gerando um DataFrame das variáveis preditoras NORMALIZADAS SEM a variável TARGET
VAR_PREDITORAS_NORM = df_normalizado.drop('CLASS', axis = 1)
```

In [83]:
```
# Gerando um DataFrame somente da variavel TARGET
# Obs: A variável TARGET NÃO É NORMALIZADA E NEM PADRONIZADA
VAR_TARGET = df_original['CLASS']
```

In [84]:
```
# Separando os dados em TREINO e TESTE (VARIAVEIS NORMALIZADAS)
# 70% PARA TREINO E 30% PARA TESTE
X_train, X_test, Y_train, Y_test = train_test_split(VAR_PREDITORAS_NORM, VAR_TARGET
```

In [85]:
```
knn = KNeighborsClassifier()
knn.fit(X_train, Y_train)
```

Out[85]:
```
KNeighborsClassifier()
```

In [86]:
```
resultados = knn.predict(X_test)
```

In [87]:
```
score = accuracy_score(Y_test, resultados)
```

In [88]:
```
score
```

Out[88]:
```
0.9074074074074074
```

In [ ]:
```

```

In [ ]:
```

```

In [89]:
```
# Gerando um DataFrame das variáveis preditoras PADRONIZADAS SEM a variável TARGET
VAR_PREDITORAS_PADRON = df_padronizado.drop('CLASS', axis = 1)
```

In [90]:
```
# Gerando um DataFrame somente da variavel TARGET
# Obs: A variável TARGET NÃO É NORMALIZADA E NEM PADRONIZADA
VAR_TARGET = df_original['CLASS']
```

In [91]:
```
# Separando os dados em TREINO e TESTE (VARIAVEIS PADRONIZADAS)
# 70% PARA TREINO E 30% PARA TESTE
X_train, X_test, Y_train, Y_test = train_test_split(VAR_PREDITORAS_PADRON, VAR_TARG
```

In [92]:
```
knn = KNeighborsClassifier()
knn.fit(X_train, Y_train)
```

Out[92]:
```
KNeighborsClassifier()
```

In [93]:
```
resultados = knn.predict(X_test)
```

In [94]:
```
score = accuracy_score(Y_test, resultados)
```

In [95]:
```
score
```

Out[95]:
```
0.9074074074074074
```

In [ ]:

In [ ]:

Algoritmo SVM

In [96]:
```
VAR_PREDITORAS_ORIG = df_original.drop('CLASS', axis = 1)
```

In [97]:
```
VAR_TARGET = df_original['CLASS']
```

In [98]:
```
X_train, X_test, Y_train, Y_test = train_test_split(VAR_PREDITORAS_ORIG, VAR_TARGET
```

In [99]:
```
svm = svm.SVC(kernel = 'linear')
svm.fit(X_train, Y_train)
```

Out[99]:
```
SVC(kernel='linear')
```

In [100…
```
resultados = svm.predict(X_test)
```

In [101…
```
score = accuracy_score(Y_test, resultados)
```

In [102…
```
score
```

Out[102]:
```
0.9074074074074074
```

In [ ]:

In [ ]:

In [103…
```
VAR_PREDITORAS_NORM = df_normalizado.drop('CLASS', axis = 1)
```

In [104…
```
VAR_TARGET = df_original['CLASS']
```

In [105…
```
X_train, X_test, Y_train, Y_test = train_test_split(VAR_PREDITORAS_NORM, VAR_TARGET
```

In [106…
```
svm = svm
svm.fit(X_train, Y_train)
```

Out[106]:
```
SVC(kernel='linear')
```

In [107…
```
resultados = svm.predict(X_test)
```

In [108…
```
score = accuracy_score(Y_test, resultados)
```

In [109…
```
score
```

Out[109]:
```
0.9259259259259259
```

In [ ]:
```
```

In [ ]:
```
```

In [110…
```
VAR_PREDITORAS_PADRON = df_padronizado.drop('CLASS', axis = 1)
```

In [111…
```
VAR_TARGET = df_original['CLASS']
```

In [112…
```
X_train, X_test, Y_train, Y_test = train_test_split(VAR_PREDITORAS_PADRON, VAR_TARG
```

In [113…
```
svm = svm
svm.fit(X_train, Y_train)
```

Out[113]:
```
SVC(kernel='linear')
```

In [114…
```
score = accuracy_score(Y_test, resultados)
```

In [115…
```
score = accuracy_score(Y_test, resultados)
```

In [116…
```
score
```

Out[116]:
```
0.9259259259259259
```

In [ ]:
```
```