Actividad 3.5 – 🍷 Clasificación de vinos:

El objeto de esta actividad es poner en práctica los conocimientos adquiridos hasta el momento para ellos vamos a utilizar el siguiente dataset que contiene una serie de características físico-químicas que determina la calidad del vino en una escala de valores del 1 al 10.

El enlace donde se encuentran los dataset es el siguiente: https://archive.ics.uci.edu/ml/datasets/Wine+Quality

Como proyecto de partida se puede utilizar el ejemplo:

Título: Ejemplo_3_3_Clasificación_con_Naive_Bayes_(Heart_Diseases)

Url: https://colab.research.google.com/drive/1J_QQh_tkRngskGWRubrmcHC2J5HGLvrH?usp=sharing

Importación de los datasets (**utilizar el dataset RedWine**)

```
# importación de librerías

import numpy as np

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap

import sklearn.externals
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix, ConfusionMatrixDisplay
from sklearn.model_selection import KFold
from sklearn.naive_bayes import GaussianNB, MultinomialNB, BernoulliNB, ComplementNB, Categor
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.utils.multiclass import unique_labels

import joblib


# importación de los datos

# df_wines = pd.read_csv('winequality-red.csv', sep=';')

# Vinos tintos
# df_wines = pd.read_csv('http://archive.ics.uci.edu/ml/machine-learning-databases/wine-quali
```

```
# Vinos blancos
df_wines = pd.read_csv('http://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality

df_wines
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	
0	7.0	0.27	0.36	20.7	0.045	45.0	170.0	1.00100	3
1	6.3	0.30	0.34	1.6	0.049	14.0	132.0	0.99400	3
2	8.1	0.28	0.40	6.9	0.050	30.0	97.0	0.99510	3
3	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.99560	3
4	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.99560	3
...	...	...	...	...	...	...	...	...	
4893	6.2	0.21	0.29	1.6	0.039	24.0	92.0	0.99114	3
4894	6.6	0.32	0.36	8.0	0.047	57.0	168.0	0.99490	3
4895	6.5	0.24	0.19	1.2	0.041	30.0	111.0	0.99254	2
4896	5.5	0.29	0.30	1.1	0.022	20.0	110.0	0.98869	3
4897	6.0	0.21	0.38	0.8	0.020	22.0	98.0	0.98941	3

4898 rows × 12 columns

```
# valores unicos de la columna 'quality'
# se usa en la matriz de confusión

qualities = df_wines.quality
uniques = sorted(pd.unique(qualities).tolist())
uniques
```

```
[3, 4, 5, 6, 7, 8, 9]
```

```
# estadísticas descriptivas

df_wines.describe()
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide
count	4898.000000	4898.000000	4898.000000	4898.000000	4898.000000	4898.000000
mean	6.854788	0.278241	0.334192	6.391415	0.045772	35.308085
std	0.843868	0.100795	0.121020	5.072058	0.021848	17.007137
min	3.800000	0.080000	0.000000	0.600000	0.009000	2.000000
25%	6.300000	0.210000	0.270000	1.700000	0.036000	23.000000
50%	6.800000	0.260000	0.320000	5.200000	0.043000	34.000000
75%	7.300000	0.320000	0.390000	9.900000	0.050000	46.000000
max	14.200000	1.100000	1.660000	65.800000	0.346000	289.000000

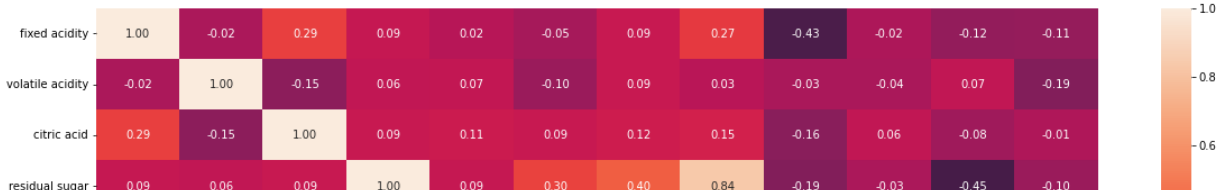
```
# separamos datos de entrada y salida

x_wines = df_wines.drop('quality', axis=1)
y_wines = df_wines['quality']

# separamos train y test

x_train, x_test, y_train, y_test = train_test_split(x_wines, y_wines, test_size=0.3, random_s
```

Mostrar la matriz de correlación de variables

```
# Mostramos la matriz de correlación

df_wines.corr()
```

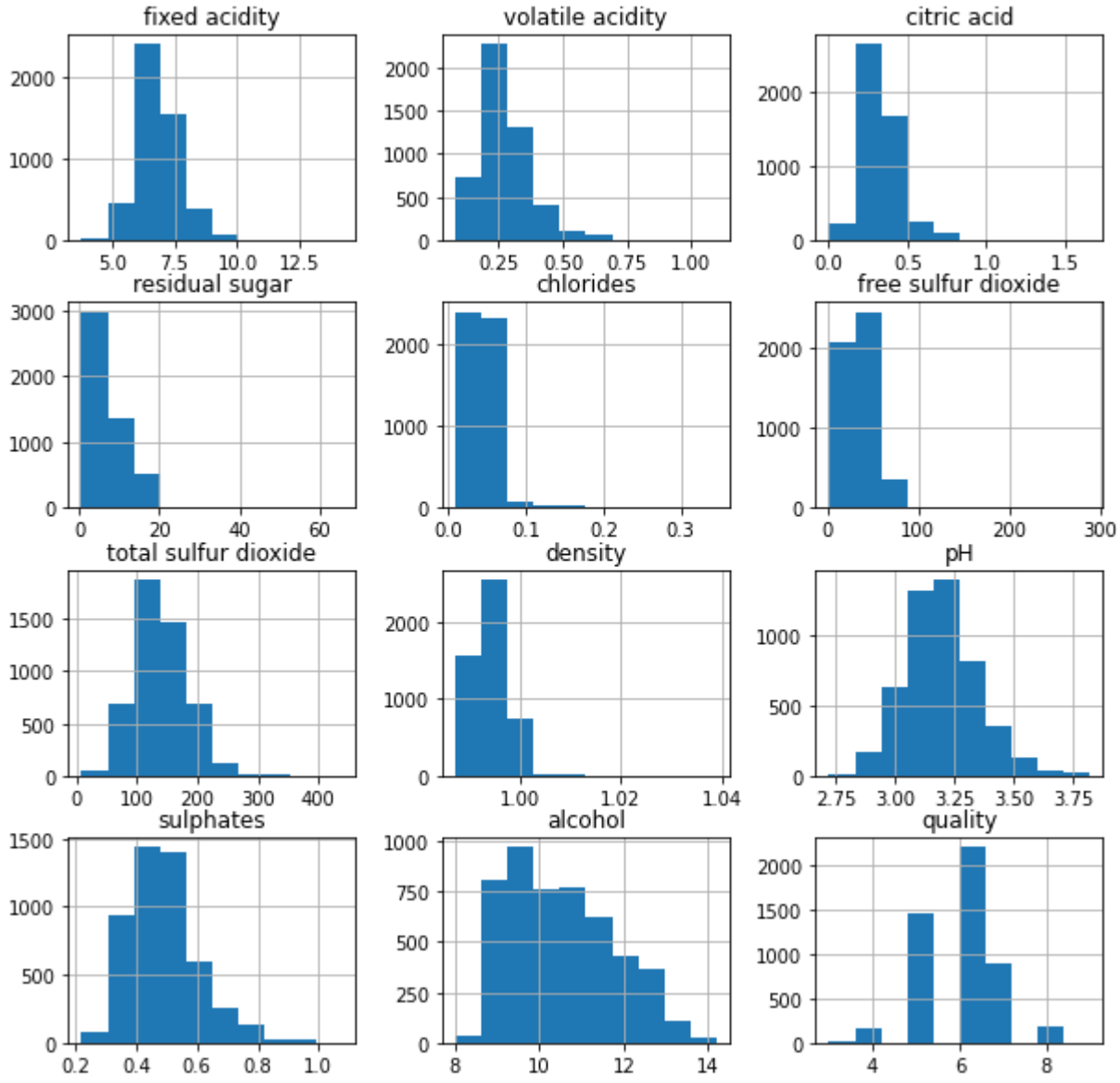
	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide
fixed acidity	1.000000	-0.022697	0.289181	0.089021	0.023086	-0.049396	0.091070
volatile acidity	-0.022697	1.000000	-0.149472	0.064286	0.070512	-0.097012	0.089261
citric acid	0.289181	-0.149472	1.000000	0.094212	0.114364	0.094077	0.121131
residual sugar	0.089021	0.064286	0.094212	1.000000	0.088685	0.299098	0.401439
chlorides	0.023086	0.070512	0.114364	0.088685	1.000000	0.101392	0.198910
free sulfur dioxide	-0.049396	-0.097012	0.094077	0.299098	0.101392	1.000000	0.615501

```
# Mostramos la matriz de correlación
# como un mapa de calor

fig, ax = plt.subplots(figsize=(20,10))
sns.heatmap(df_wines.corr(), annot=True, fmt=".2f")
ax.set_xticklabels(
    labels=df_wines.columns.values,
    rotation=45,
    horizontalalignment='center'
);
plt.show()
```



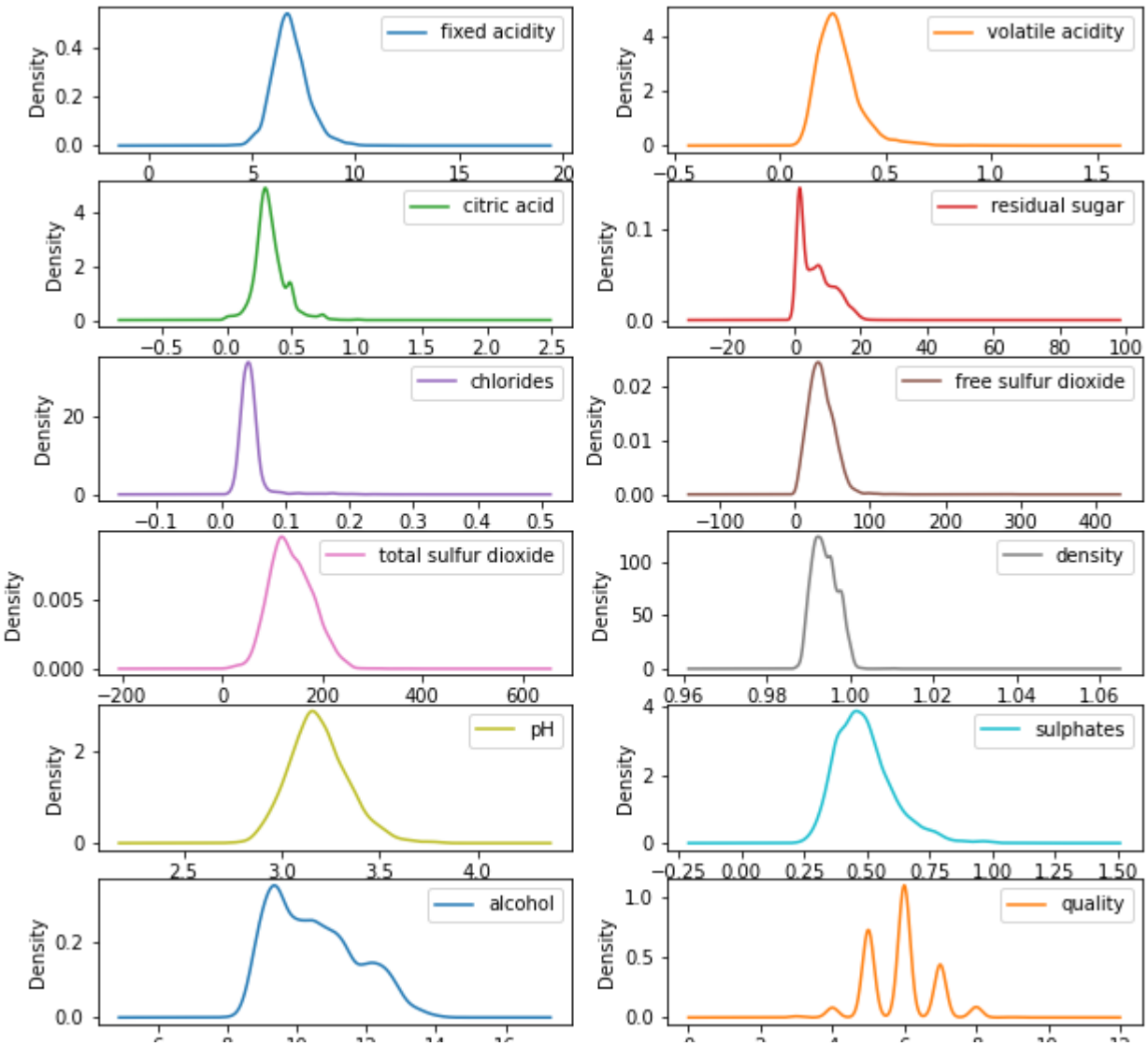
Mostrar comparativa por pares de variables (sns.pairplot)


```
# Mostramos los histogramas

df_wines.hist(figsize=(10,10))
plt.show()
```



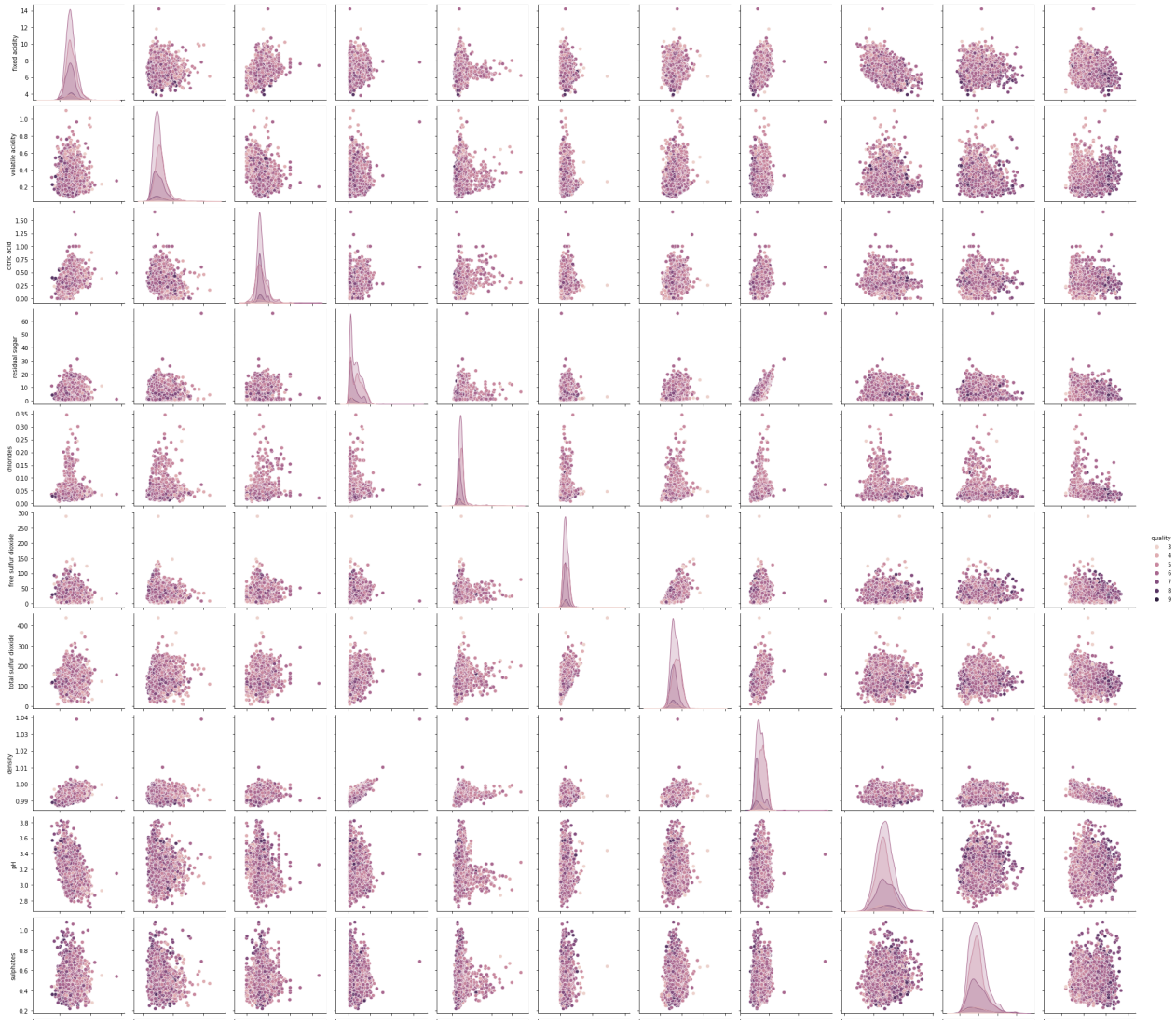

```
# Mostramos las gráficas de densidad
```

```
df_wines.plot(kind='density', subplots=True, layout=(6,2), figsize=(10,10), sharex=False)
plt.show()
```




```
# Mostramos el diagrama de pares (pairplot)

sns.pairplot(df_wines, hue='quality')
```

```
<seaborn.axisgrid.PairGrid at 0x7f1608eb1ac0>
```



Realizar una comparativa de la precisión en el entrenamiento de los diferentes modelos NaiveBayes

Sin CrossValidation

```
# DA FALLO SI AÑADO EL MODELO CategoricalNB()

# Modelos NaiveBayes
names = ["GaussianNB", "MultiNomialNB", 'BernouilliNB', 'ComplementNB']
classifiers = [GaussianNB(), MultinomialNB(), BernoulliNB(), ComplementNB()]

for name, clf in zip(names, classifiers):

  # Entrena el modelo
  clf.fit(x_train, y_train)
```

```
  # Predice y puntua
  # Devuelve la precisión media de las etiquetas y los datos de prueba proporcionados
  score = clf.score(x_test, y_test)
  print ("Modelo: %s = %6.2f" % (name, score))
```

```
Modelo: GaussianNB =   0.45
Modelo: MultiNomialNB =   0.40
Modelo: BernouilliNB =   0.45
Modelo: ComplementNB =   0.37
```

Con CrossValidation

```
cv = KFold(n_splits = 5, shuffle = True)
total_scores = []
for name, clf in zip(names, classifiers):
  fold_accuracy = []
  for train_fold, test_fold in cv.split(x_train):

    # División train test aleatoria
    # Extrae la información (iloc), atendiendo a los indices obtenidos por CrossValidation
    f_train_x = x_train.iloc[train_fold]
    f_train_y = y_train.iloc[train_fold]

    # Entrenamiento y ejeución del modelo
    clf.fit(f_train_x, f_train_y)

    # Realizamos la predicción (Final evaluation)
    # y guardamos la precisión para calcular la media posteriormente
    y_pred = clf.predict(x_train.iloc[test_fold])

    # Evaluación del modelo
    acc = accuracy_score(y_train.iloc[test_fold], y_pred)
    fold_accuracy.append(acc)

  total_scores.append(sum(fold_accuracy)/len(fold_accuracy))

for i in range(len(names)):
  print ("Modelo: %s = %6.2f" % (names[i], total_scores[i]))
```

```
Modelo: GaussianNB =   0.45
Modelo: MultiNomialNB =   0.40
Modelo: BernouilliNB =   0.45
Modelo: ComplementNB =   0.35
```

Una vez decides el modelo que consideras mejor, entonces realizar las siguientes tareas:

```
# Elegimos el modelo: GaussianNB() sin CrossValidation
# El que obtiene más puntuación
```

Entrenarlo y obtener la matriz de confusión

```
# Instanciamos el modelo

model = GaussianNB()

# Entrenamiento con los datos

model.fit(x_train, y_train)
```

```
     GaussianNB()
```

```
# Predicción con nuevos datos

y_model = model.predict(x_test)
y_model
```

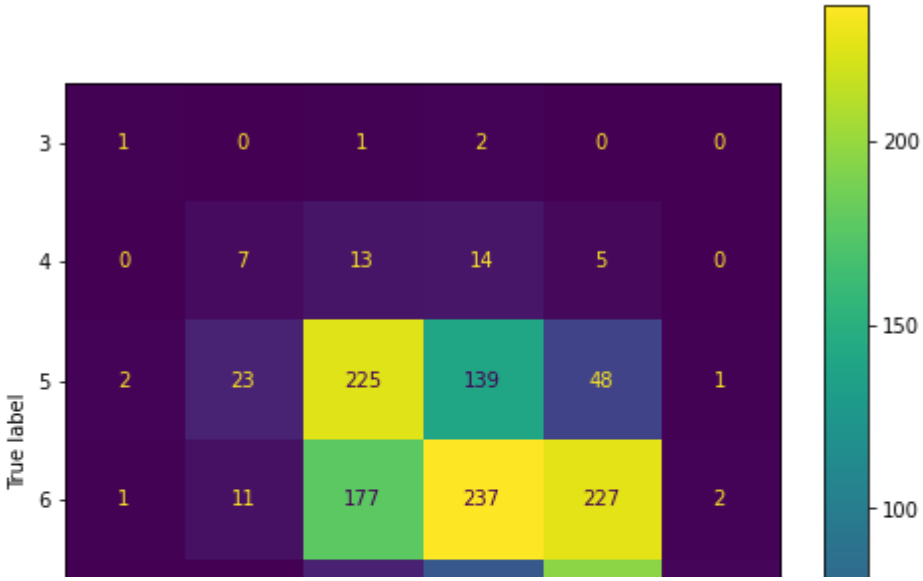
```
     array([5, 5, 7, ..., 6, 6, 7])
```

```
# Evaluación y precisión del modelo

accuracy_score(y_test, y_model)
```

```
     0.454421768707483
```

```
# Matriz de confusión

cm = confusion_matrix(y_test, y_model)
print(cm)
display_cm = ConfusionMatrixDisplay(cm, display_labels=uniques)
fig, ax = plt.subplots(figsize=(8,8))
display_cm.plot(ax=ax)
plt.show()
```

```
[[  1   0   1   2   0   0]
 [  0   7  13  14   5   0]
 [  2  23 225 139  48   1]
 [  1  11 177 237 227   2]
 [  0   0  22  65 196   3]
 [  0   1   2   8  35   2]]
```



Exportar a un fichero los parámetros del modelo entrenado

```
# Exportamos el modelo

joblib.dump(model,'vinos.pkl')
```

```
['vinos.pkl']
```

Importar los parámetros del modelo

```
# Importamos el modelo

import_model = GaussianNB()
import_model = joblib.load('vinos.pkl')
import_model.score(x_test, y_test)
```

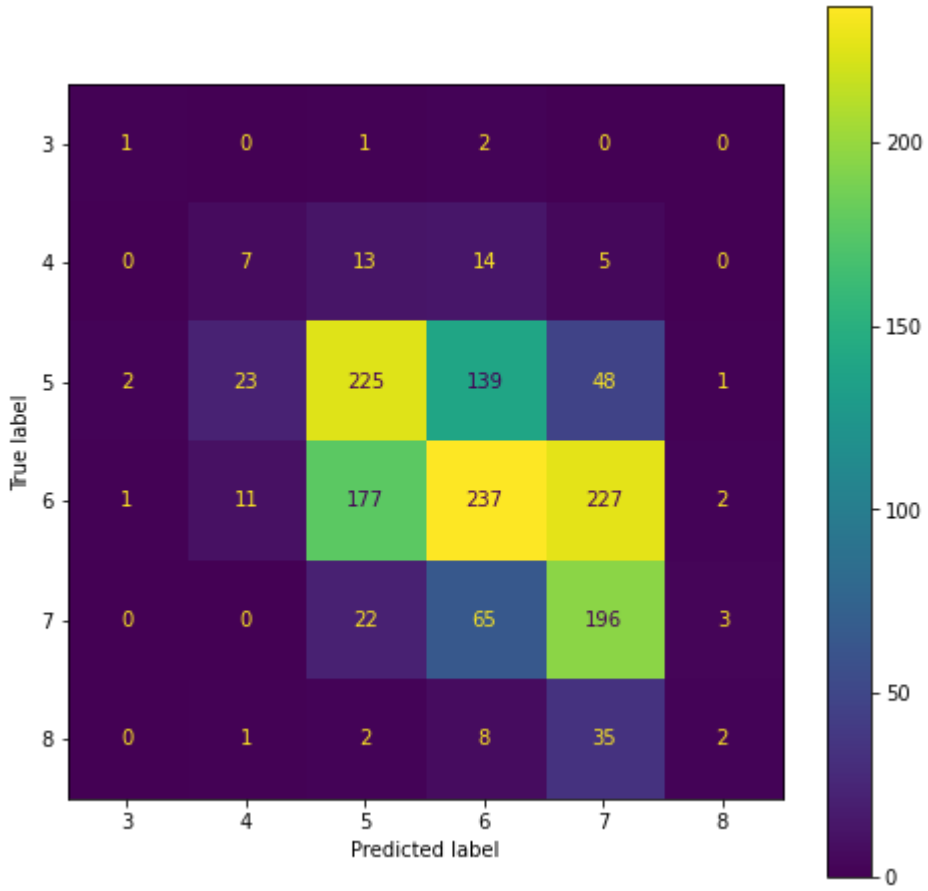
```
0.454421768707483
```

Aplicar el modelo (predict) a todos los datos del dataset y obtener la matriz de confusión

```
# Realizamos la predicción
```

```
y_pred_import = import_model.predict(x_test)

# Matriz de Confusión

cm_import = confusion_matrix(y_test, y_pred_import)
print(cm_import)
display_cm_import = ConfusionMatrixDisplay(cm_import, display_labels=uniques)
fig, ax = plt.subplots(figsize=(8,8))
display_cm_import.plot(ax=ax)
plt.show()
```

```
[[  1   0   1   2   0   0]
 [  0   7  13  14   5   0]
 [  2  23 225 139  48   1]
 [  1  11 177 237 227   2]
 [  0   0  22  65 196   3]
 [  0   1   2   8  35   2]]
```

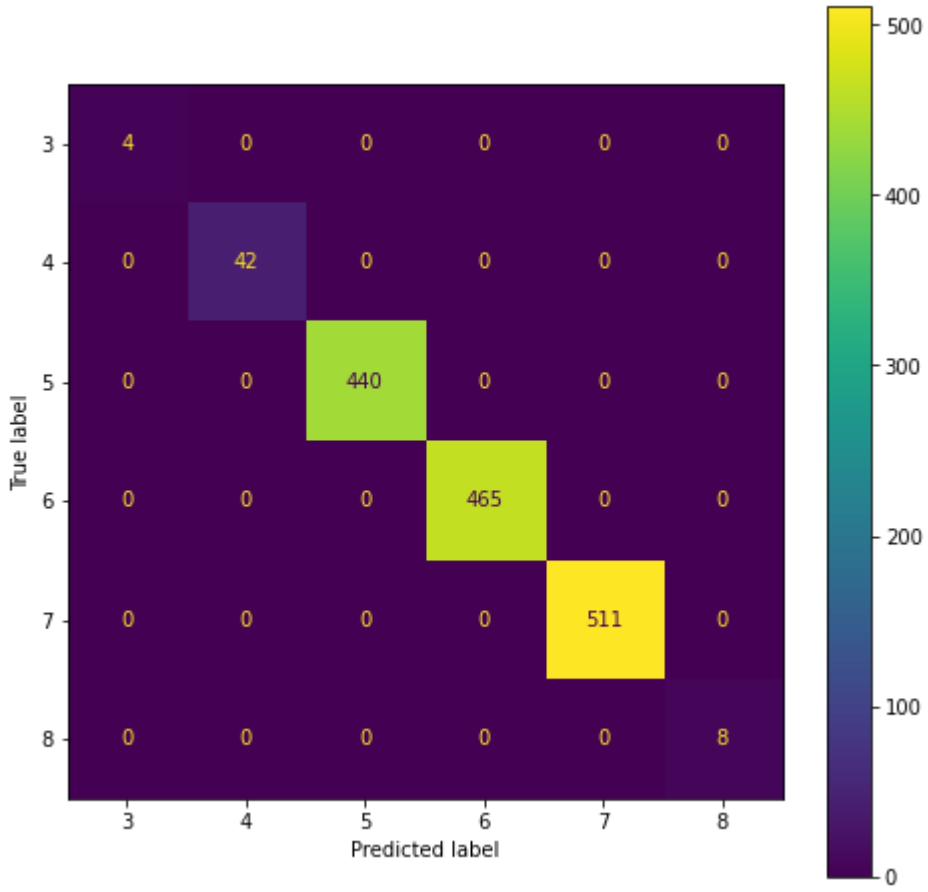


Comparar el resultado obtenido con el valor de calidad indicado en el dataset por medio de una matriz de confusión

```
# Matriz de Confusión

cm_import = confusion_matrix(y_model, y_pred_import)
```

```
print(cm_import)
display_cm_import = ConfusionMatrixDisplay(cm_import, display_labels=uniques)
fig, ax = plt.subplots(figsize=(8,8))
display_cm_import.plot(ax=ax)
plt.show()
```

```
[[  4   0   0   0   0   0]
 [  0  42   0   0   0   0]
 [  0   0 440   0   0   0]
 [  0   0   0 465   0   0]
 [  0   0   0   0 511   0]
 [  0   0   0   0   0   8]]
```



Obtener la precisión del resultado obtenido, para determinar si coincide con la precisión que se calculó durante el entrenamiento

```
import_model.score(x_test, y_pred_import)
```

```
1.0
```

Probar a utilizar el cuaderno con el dataset de los vinos blancos y concluir si hay variaciones en los métodos gausianos utilizados y en

los resultados finales obtenidos