

# Actividad 3.5 Clasificación de vinos

Enlace al repositorio de github: [Github](#)

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.

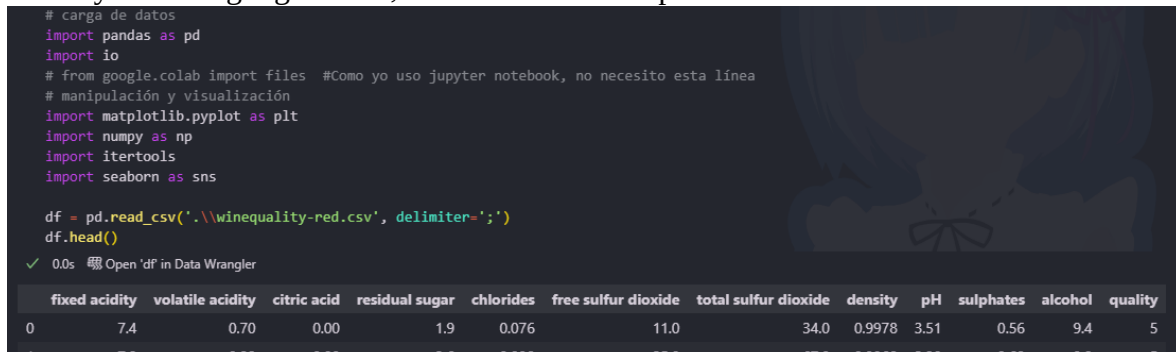
## - Importación del dataset

Importamos el dataset y las librerías

Como yo no uso google.colab, no me hace falta importarlo.

```
# carga de datos
import pandas as pd
import io
# from google.colab import files #Como yo uso jupyter notebook, no necesito esta línea
# manipulación y visualización
import matplotlib.pyplot as plt
import numpy as np
import itertools
import seaborn as sns

df = pd.read_csv('..\\winequality-red.csv', delimiter=';')
df.head()
```



|   | fixed acidity | volatile acidity | citric acid | residual sugar | chlorides | free sulfur dioxide | total sulfur dioxide | density | pH   | sulphates | alcohol | quality |
|---|---------------|------------------|-------------|----------------|-----------|---------------------|----------------------|---------|------|-----------|---------|---------|
| 0 | 7.4           | 0.70             | 0.00        | 1.9            | 0.076     | 11.0                | 34.0                 | 0.9978  | 3.51 | 0.56      | 9.4     | 5       |
| 1 | 7.8           | 0.88             | 0.00        | 2.6            | 0.098     | 25.0                | 67.0                 | 0.9968  | 3.20 | 0.68      | 9.8     | 5       |

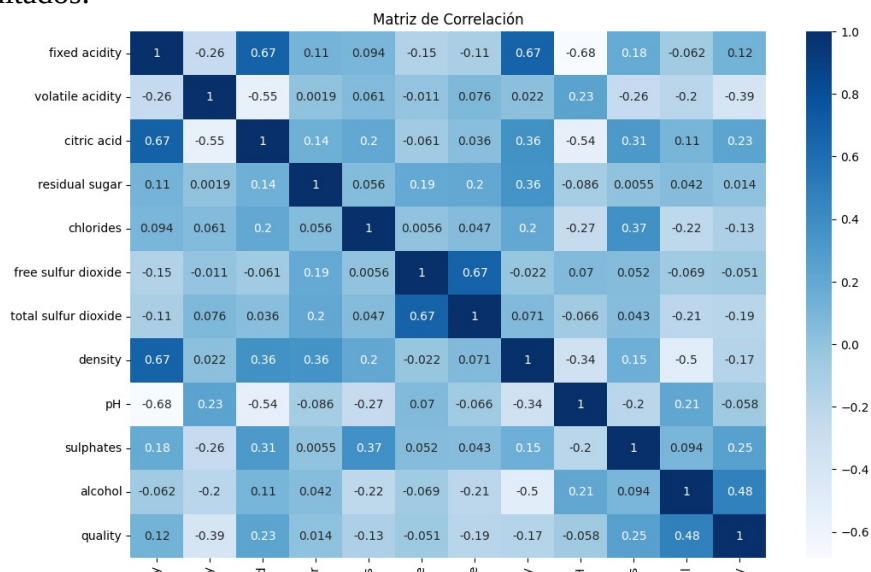
## Mostrar matriz de correlación

Para la matriz de correlación, he utilizado la siguiente función:

```
def plot_correlation_matrix(df):
    plt.figure(figsize=(12, 8))
    correlation_matrix = df.corr()
    sns.heatmap(correlation_matrix, annot=True, cmap=plt.cm.Blues)
    plt.title('Matriz de Correlación')
    plt.show()

plot_correlation_matrix(df)
```

Aquí los resultados:



**Aplicar cualquier otra técnica de selección de características que consideres adecuados y justificar tu propuesta.**

Con este código, podemos seleccionar las 5 mejores características, haciendo que el modelo pueda alcanzar resultados similares sin tener que realizar tantos cálculos:

```
from sklearn.feature_selection import SelectKBest, f_classif

# Seleccionamos las 5 características más importantes
selector = SelectKBest(score_func=f_classif, k=5)
X_new = selector.fit_transform(X, y)

# Mostramos las características seleccionadas
selected_features = X.columns[selector.get_support()]
```

**Realizar una comparativa de la precisión en el entrenamiento de los diferentes modelos de NaivaBayes y KNN. Aplicando Cross Validation.**

Aquí podemos ver una comparativa de los diferentes modelos y sus resultados:

```
# classifiers
names = ["GaussianNB", "MultiNomialNB", 'BernouilliNB', 'ComplementNB', 'KNeighborsClassifier']
classifiers = [GaussianNB(), MultinomialNB(), BernouilliNB(), ComplementNB(), knn(n_neighbors=5)]

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

    # FIT THE MODEL
    clf.fit(X_train, y_train)
    # PREDIT AND SCORE

    score = cross_val_score(clf, X_train, y_train, cv=5) #Returns the mean cross-validation accuracy
    print (f"Modelo: {name} = ", score.mean())
```

✓ 0.1s

Modelo: GaussianNB = 0.5397421524663677  
Modelo: MultiNomialNB = 0.44056694426649584  
Modelo: BernouilliNB = 0.4218089365791159  
Modelo: ComplementNB = 0.4977858744394618  
Modelo: KNeighborsClassifier = 0.5021941063420884

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

***Entrenarlo y obtener la matriz de confusión.***

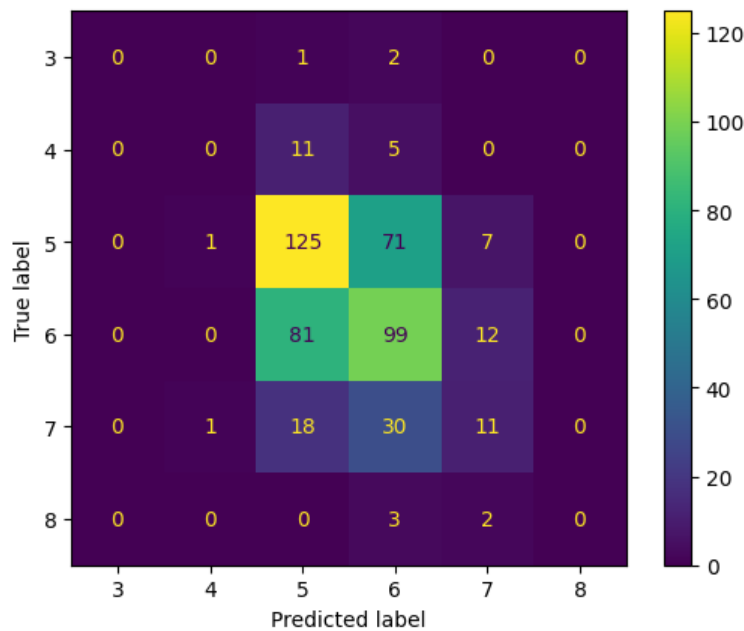
Con esto, podemos ver la matriz de confusión:

```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

clf = knn()
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)

labels = unique_labels(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred, labels=labels)
ConfusionMatrixDisplay(conf_matrix, display_labels=labels).plot()
```

Aquí los resultados:



### ***Exportar a un fichero los parámetros del modelo entrenado.***

Con este código, podemos exportar el modelo:

```
# Para exportar el modelo a un fichero
import sklearn.externals
import joblib
joblib.dump(clf, 'Ejemp_3_3_modelo_entrenado.pkl')
✓ 0.0s
['Ejemp_3_3_modelo_entrenado.pkl']
```

### ***Importar los parámetros del modelo.***

Con este código, podemos importar los parámetros del modelo ya entrenado:

```
# Para importar el modelo entrenado y ejecutar de nuevo test
clf_entrenado = joblib.load('Ejemp_3_3_modelo_entrenado.pkl')
clf_entrenado.score(X_test, y_test) # Obtenemos la precisión
#clf_entrenado.score(X_train, y_train) # Obtenemos la precisión
✓ 0.0s
0.6125
```

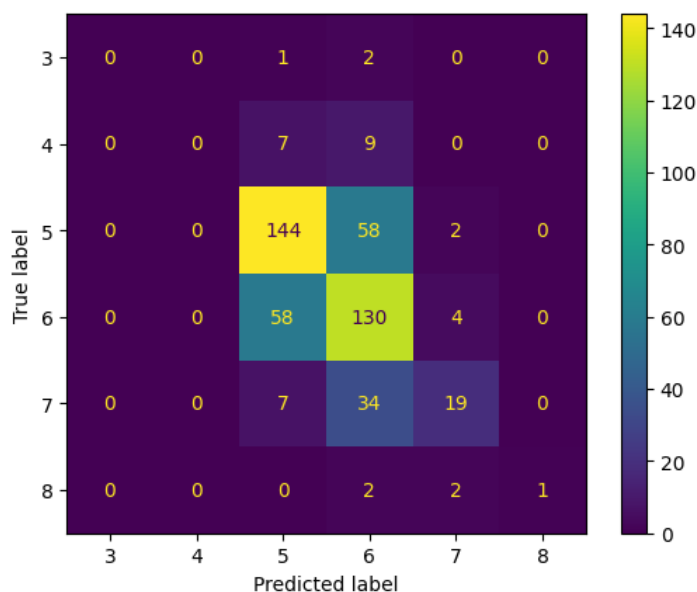
**Aplicar el modelo (predict) a todos los datos del dataset y obtener la matriz de confusión.**

Aplicamos el modelo usando el método predict:

```
#Volvemos a realizar la validación final y obtenemos la matriz de confusión
y_pred = clf_entrenado.predict(X_test)
conf_matrix = confusion_matrix(y_test, y_pred, labels=labels)
ConfusionMatrixDisplay(conf_matrix, display_labels=labels).plot()
```

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

Aquí podemos ver los resultados:



**Probar a utilizar el cuaderno con el dataset de los vinos blancos y realizar captura de los resultados obtenidos. (utilizar el dataset winequality-white.csv)**

Cambiamos el archivo:

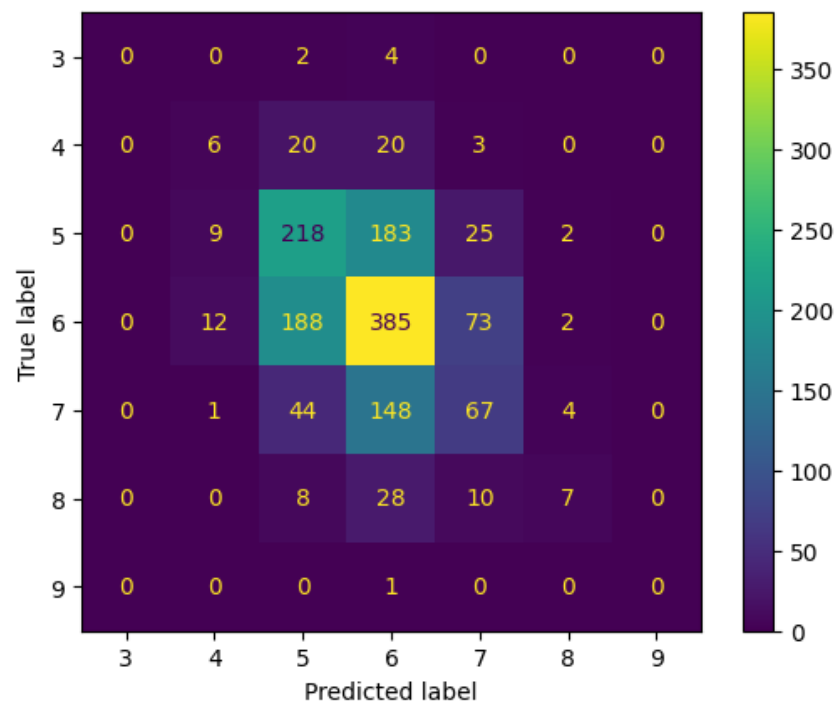
```
df = pd.read_csv('.\\winequality-white.csv', delimiter=';')
df.head()
```

Y obtenemos los resultados:

- Precisión de los modelos:

```
Modelo: GaussianNB = 0.4428311804388074
Modelo: MultiNomialNB = 0.3862318316273329
Modelo: BernouilliNB = 0.4480743121023175
Modelo: ComplementNB = 0.36142708178161775
```

Primera matriz de confusión:



Segunda matriz de confusión:

