

# Modelos para la Clasificación de Vinos según su Calidad

Pablo Álvaro, Guillermo Balen, Paula Gutiérrez

# **1) LIBRERIAS**



```
1 import warnings
2 import numpy as np # type: ignore
3 import pandas as pd # type: ignore
4 import matplotlib.pyplot as plt # type: ignore
5 import seaborn as sns # type: ignore
6 from IPython.display import Image, display # type: ignore
7 from scipy.stats import boxcox # type: ignore
8
9 from keras.layers import Dense, Dropout, Input # type: ignore
10
11 from sklearn.ensemble import RandomForestClassifier # type: ignore
12 from sklearn.inspection import permutation_importance # type: ignore
13 from sklearn.linear_model import LogisticRegression # type: ignore
14 from sklearn.metrics import accuracy_score, precision_score, recall_score, confusion_matrix, classification_report, f1_score # type: ignore
15 from sklearn.model_selection import train_test_split, cross_val_score, GridSearchCV # type: ignore
16 from sklearn.preprocessing import MinMaxScaler # type: ignore
17 from sklearn.svm import SVC # type: ignore
18 from sklearn.tree import plot_tree # type: ignore
19
20 import xgboost as xgb # type: ignore
21
22 import tensorflow as tf
23 from tensorflow.keras.models import Sequential # type: ignore
24 from tensorflow.keras.layers import Dense, Dropout # type: ignore
25 from tensorflow.keras.utils import plot_model, to_categorical # type: ignore
26
27 # Quitar future warnings
28 warnings.simplefilter(action='ignore', category=FutureWarning)
29 warnings.simplefilter(action='ignore', category=DeprecationWarning)
```

## **2) DATASET Y DEFINICIÓN DE VARIABLE OBJETIVO**



<https://archive.ics.uci.edu/dataset/186/wine+quality>



```
1 white = pd.read_csv("winequality-white.csv", sep=";")
2
3 red = pd.read_csv("winequality-red.csv", sep=";")
4 # Mostramos la informacion de white_wine
5 print(white.info())
6 white.head(5)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4898 entries, 0 to 4897
Data columns (total 12 columns):
 #   Column           Non-Null Count Dtype  
 --- 
 0   fixed acidity    4898 non-null  float64 
 1   volatile acidity 4898 non-null  float64 
 2   citric acid      4898 non-null  float64 
 3   residual sugar   4898 non-null  float64 
 4   chlorides        4898 non-null  float64 
 5   free sulfur dioxide 4898 non-null  float64 
 6   total sulfur dioxide 4898 non-null  float64 
 7   density          4898 non-null  float64 
 8   pH               4898 non-null  float64 
 9   sulphates        4898 non-null  float64 
 10  alcohol          4898 non-null  float64 
 11  quality          4898 non-null  int64  
dtypes: float64(11), int64(1)
memory usage: 459.3 KB
None
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.0	0.27	0.36	20.7	0.045	45.0	170.0	1.0010	3.00	0.45	8.8	6
1	6.3	0.30	0.34	1.6	0.049	14.0	132.0	0.9940	3.30	0.49	9.5	6
2	8.1	0.28	0.40	6.9	0.050	30.0	97.0	0.9951	3.26	0.44	10.1	6
3	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956	3.19	0.40	9.9	6
4	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956	3.19	0.40	9.9	6

```
1 print(red.info())
2 red.head(5)
✓ 0.0s
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1599 entries, 0 to 1598
Data columns (total 12 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   fixed acidity    1599 non-null   float64 
 1   volatile acidity 1599 non-null   float64 
 2   citric acid      1599 non-null   float64 
 3   residual sugar   1599 non-null   float64 
 4   chlorides        1599 non-null   float64 
 5   free sulfur dioxide 1599 non-null   float64 
 6   total sulfur dioxide 1599 non-null   float64 
 7   density          1599 non-null   float64 
 8   pH               1599 non-null   float64 
 9   sulphates        1599 non-null   float64 
 10  alcohol          1599 non-null   float64 
 11  quality          1599 non-null   int64  
dtypes: float64(11), int64(1)
memory usage: 150.0 KB
```

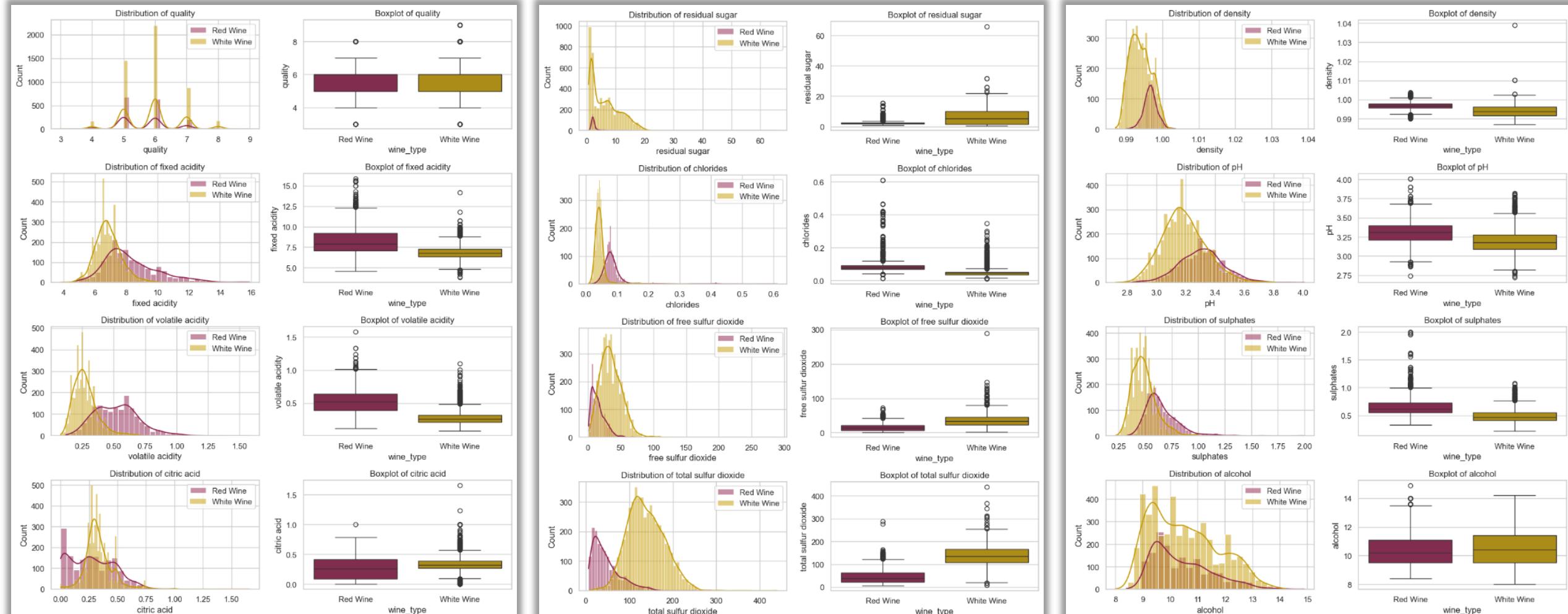
```
None
```

	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
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9.8	5
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9.8	6
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5

## 2.1) Comparación Red y White wine



```
1 def plot_histograms(df_1, df_2, df_1_name='Dataset 1', df_2_name='Dataset 2', x_axis_size=10, y_axis_size=3, color_1='red', color_2='blue'): ...
2 plot_histograms(red, white, df_1_name='Red Wine', df_2_name='White Wine', x_axis_size=10, y_axis_size=3, color_1='#8E1F4C', color_2='#C49A00')
```



Tienen ciertas diferencias por lo que hemos decidido tratarlos por separado.

## **2.2) Establecimiento de la Variable Objetivo**

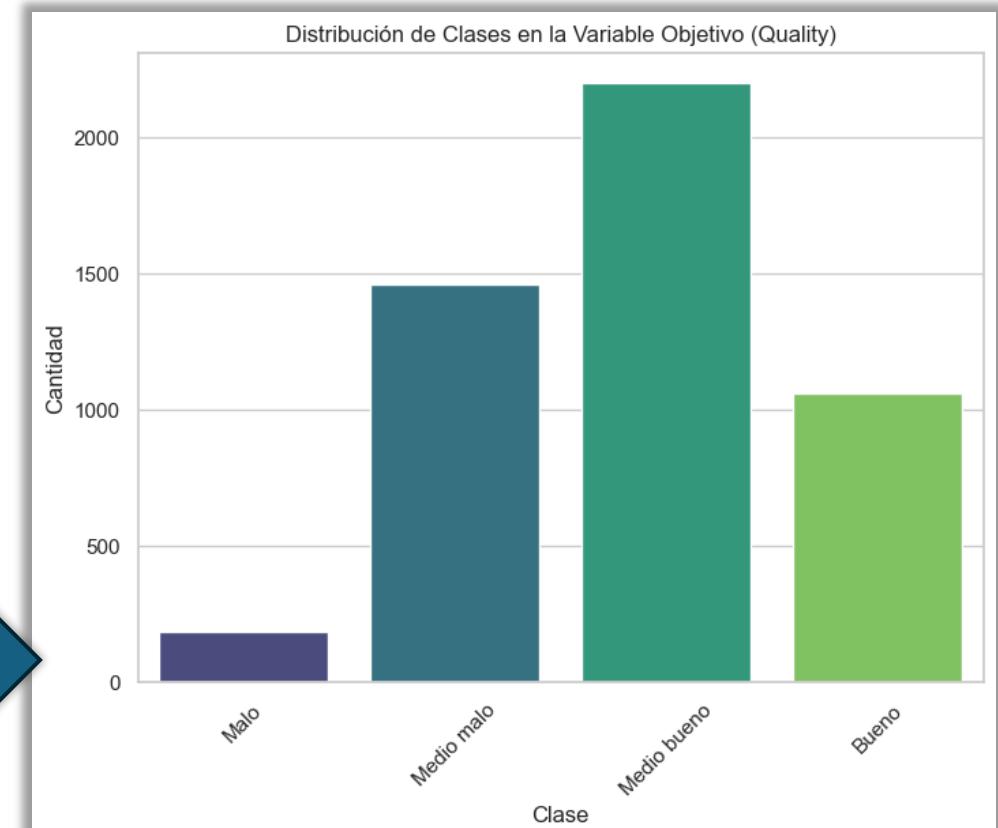
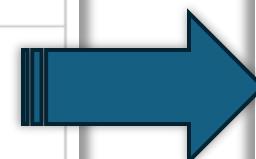
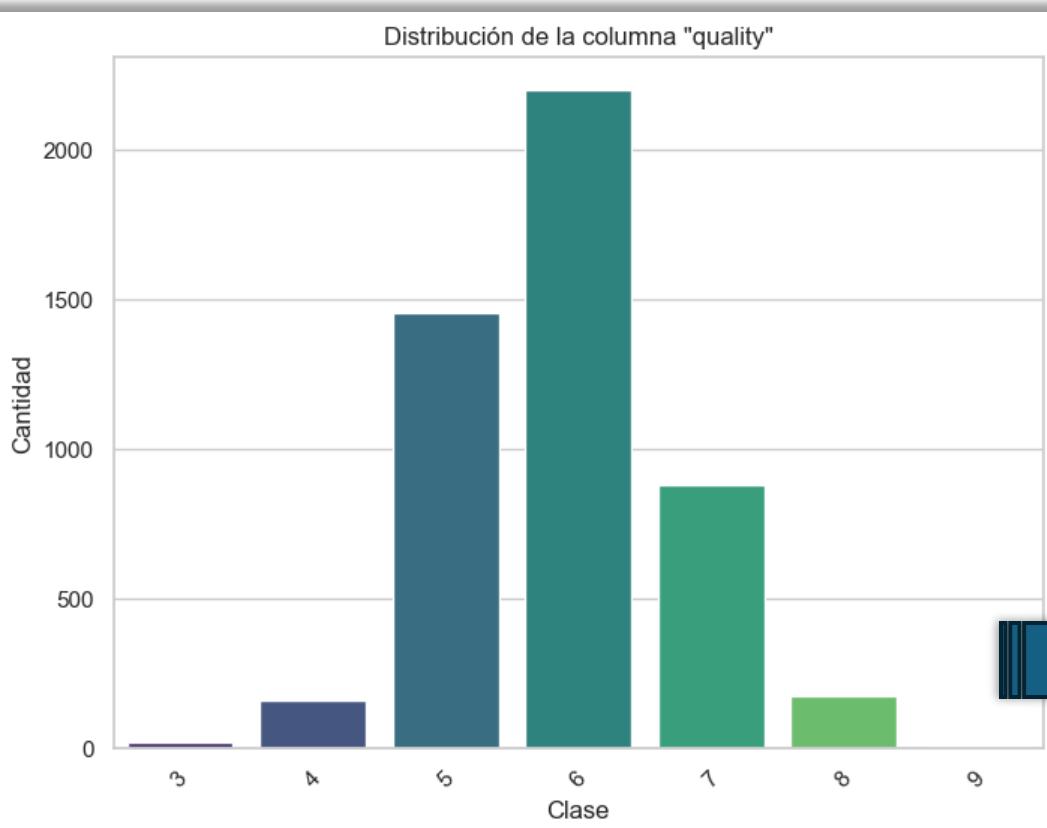
```
7 white['quality'] = pd.cut(white['quality'], bins = [0, 4, 5, 6, 10], labels = ['Malo', 'Medio malo', 'Medio bueno', 'Bueno'])  
8 white['quality'].value_counts()
```

✓ 0.0s

quality

Medio bueno	2198
Medio malo	1457
Bueno	1060
Malo	183

Name: count, dtype: int64

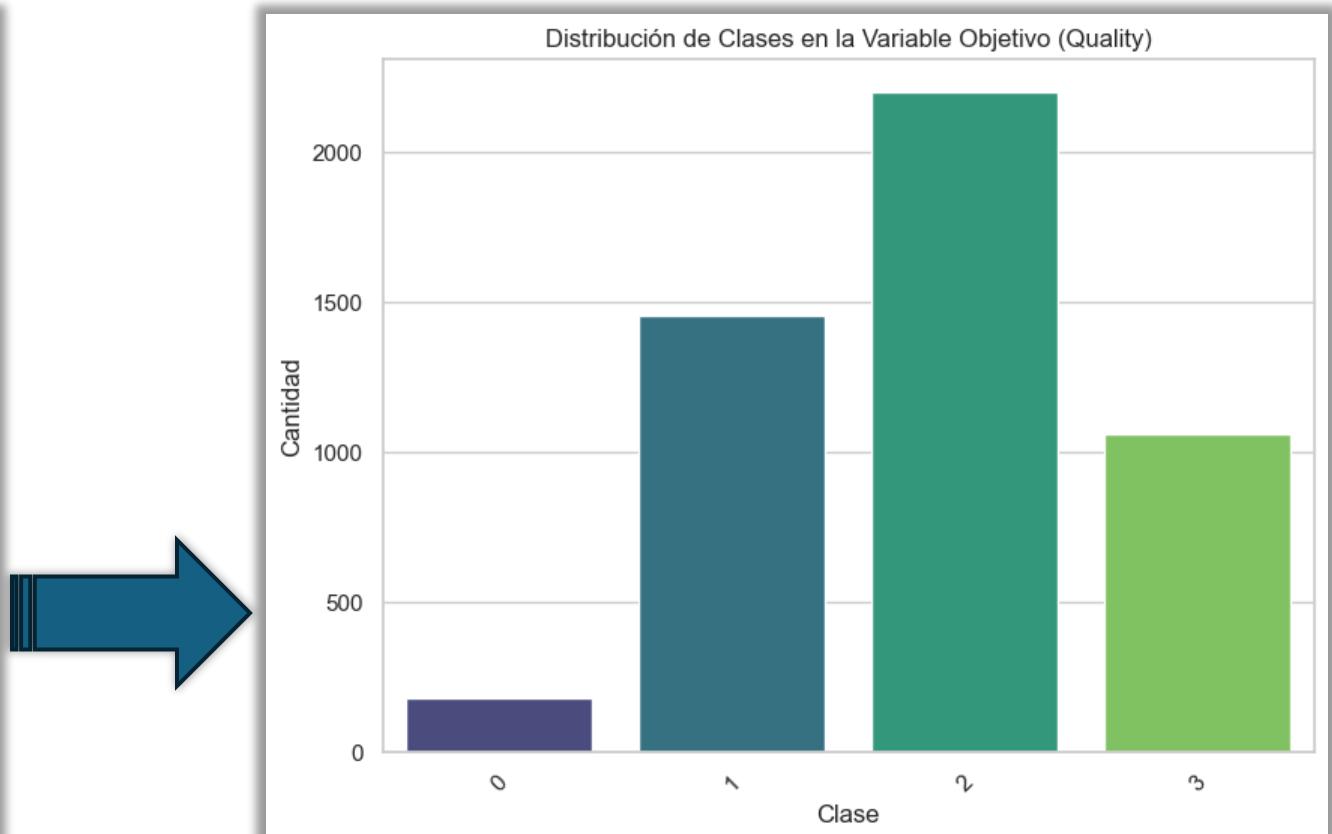
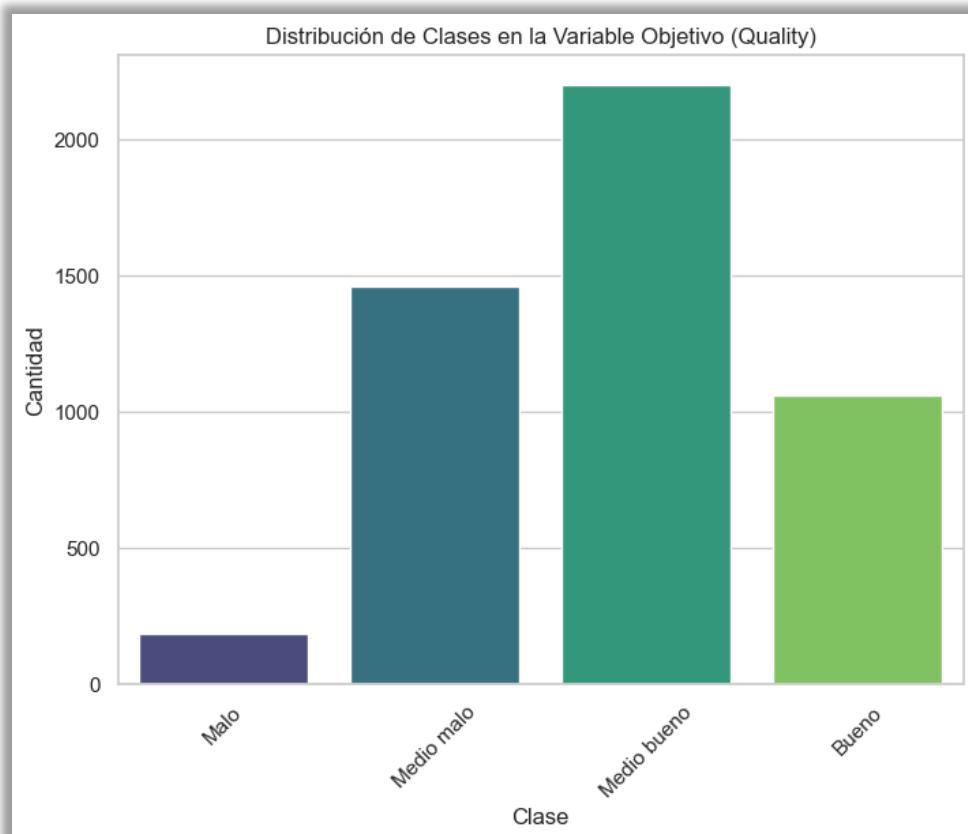


```

1 # Poner los malos como 0, los medios malos como 1, los medios buenos como 2 y los buenos como 3
2 white['quality'] = white['quality'].map({'Malo': 0, 'Medio malo': 1, 'Medio bueno': 2, 'Bueno': 3})
3 white['quality'].value_counts()
4
✓ 0.0s

quality
2 2198
1 1457
3 1060
0 183
# Importante guardar una lista llamada labels para el futuro graficas con las etiquetas
labels = ['Malo', 'Medio malo', 'Medio bueno', 'Bueno']

```



# **3) EDA (EXPLORATORY DATA ANALYSIS)**

```

1 print("----Nulos en white_wine----")
2 print(white.isnull().sum(), "\n")
3
4 white.describe()
✓ 0s

```

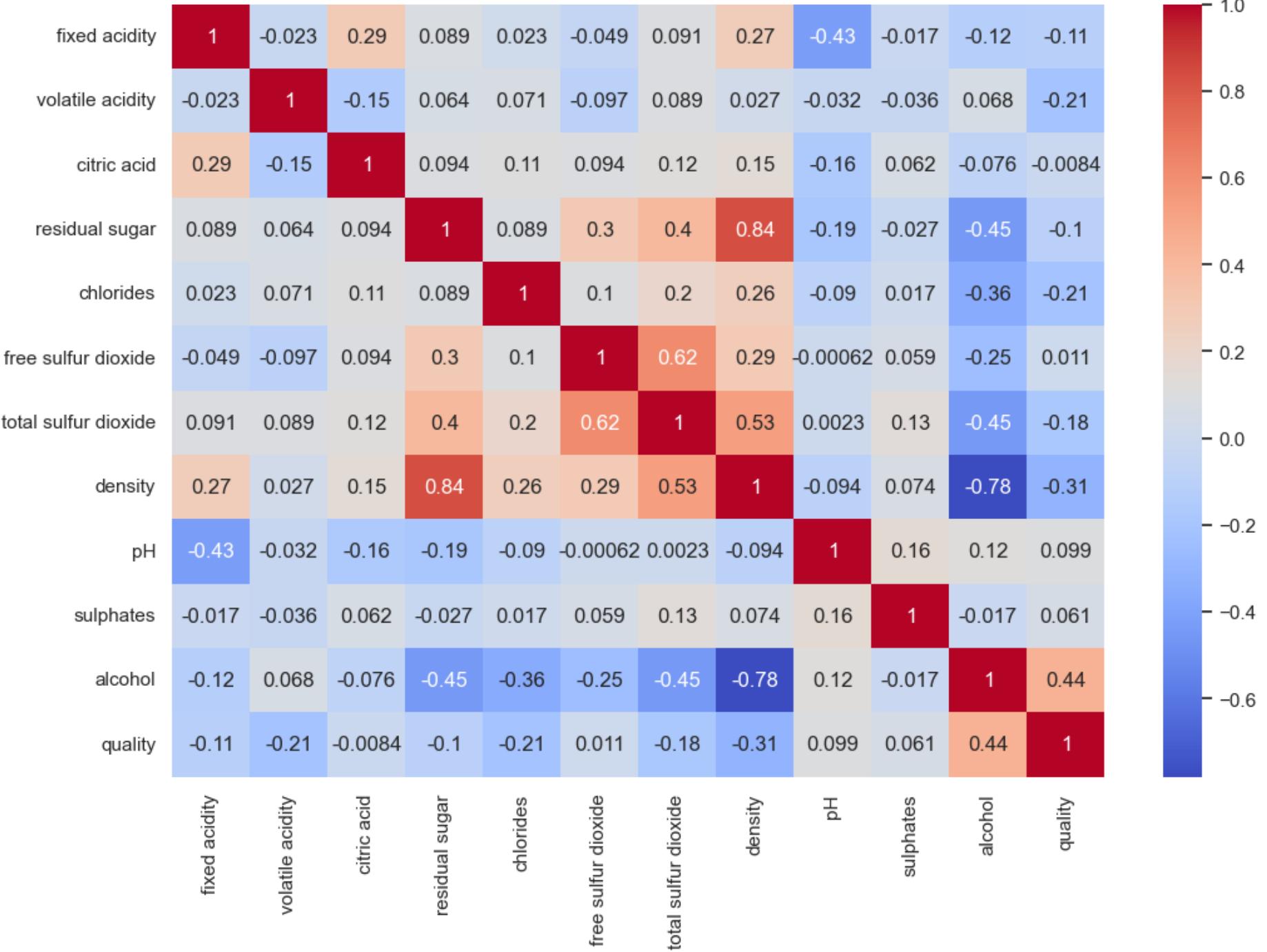
----Nulos en white\_wine----

fixed acidity	0
volatile acidity	0
citric acid	0
residual sugar	0
chlorides	0
free sulfur dioxide	0
total sulfur dioxide	0
density	0
pH	0
sulphates	0
alcohol	0
quality	0
dtype:	int64

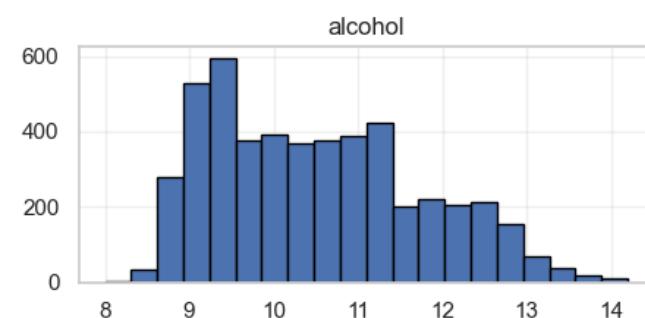
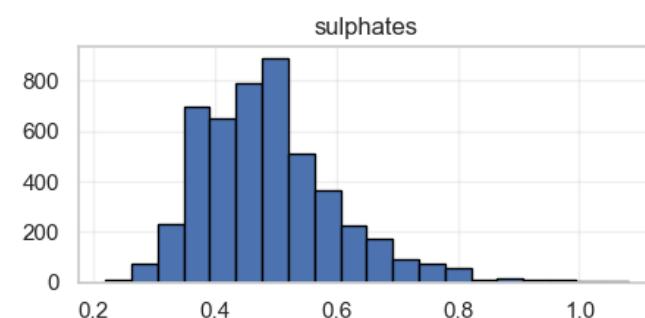
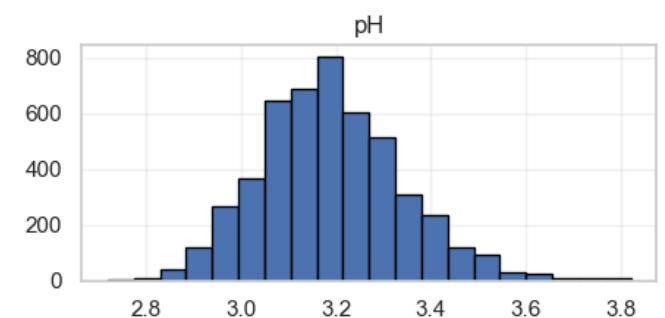
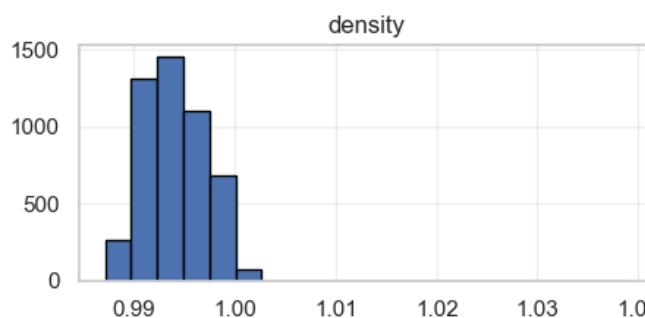
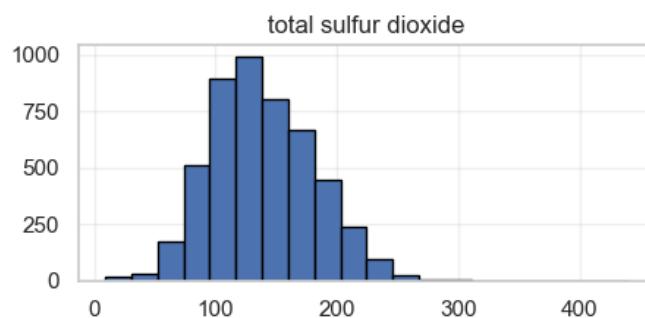
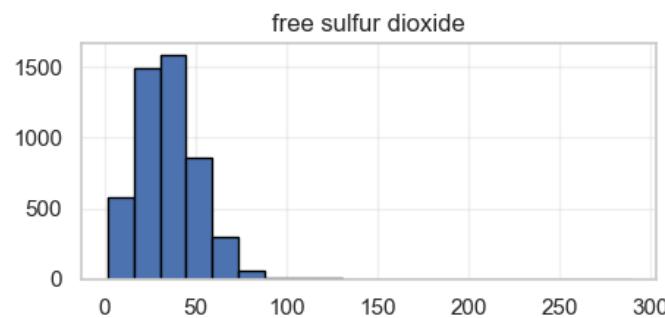
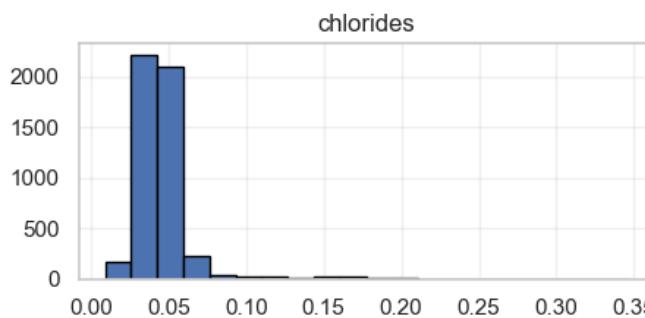
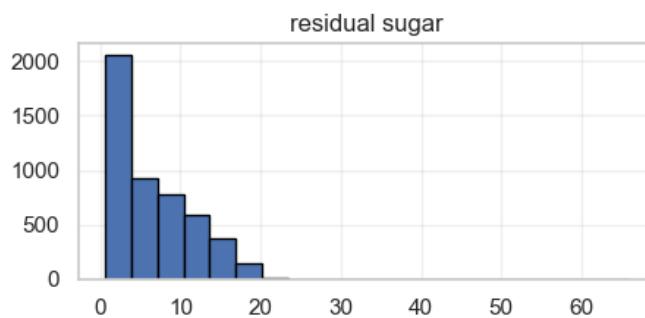
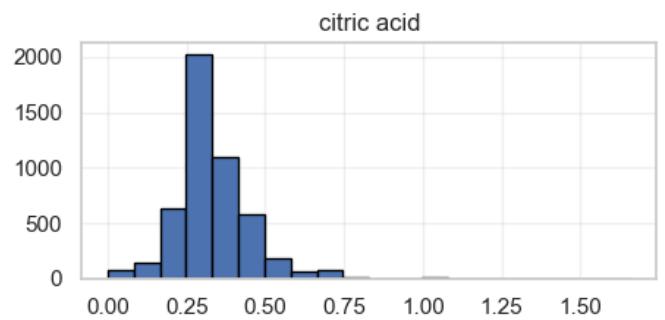
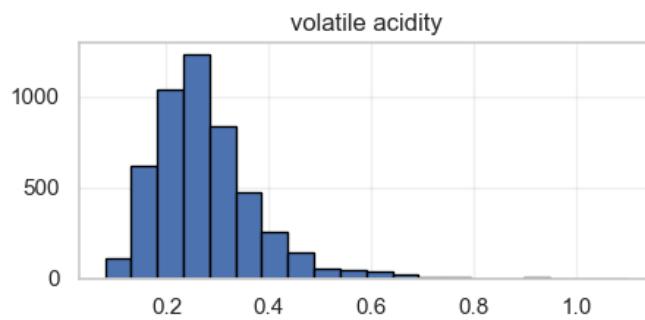
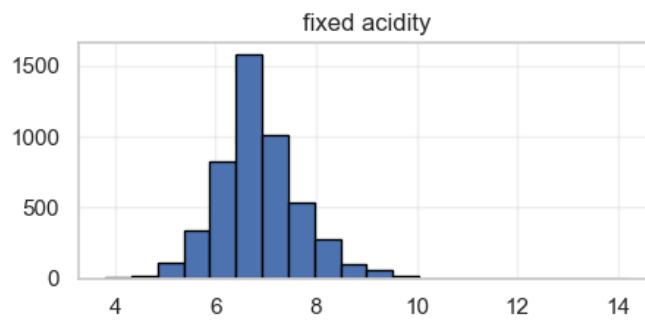
Vemos que no hay valores nulos, que las columnas son todas numéricas y que no hay columnas con valores negativos

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol
count	4898.000000	4898.000000	4898.000000	4898.000000	4898.000000	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	138.360657	0.994027	3.188267	0.489847	10.514267
std	0.843868	0.100795	0.121020	5.072058	0.021848	17.007137	42.498065	0.002991	0.151001	0.114126	1.230621
min	3.800000	0.080000	0.000000	0.600000	0.009000	2.000000	9.000000	0.987110	2.720000	0.220000	8.000000
25%	6.300000	0.210000	0.270000	1.700000	0.036000	23.000000	108.000000	0.991723	3.090000	0.410000	9.500000
50%	6.800000	0.260000	0.320000	5.200000	0.043000	34.000000	134.000000	0.993740	3.180000	0.470000	10.400000
75%	7.300000	0.320000	0.390000	9.900000	0.050000	46.000000	167.000000	0.996100	3.280000	0.550000	11.400000
max	14.200000	1.100000	1.660000	65.800000	0.346000	289.000000	440.000000	1.038980	3.820000	1.080000	14.200000

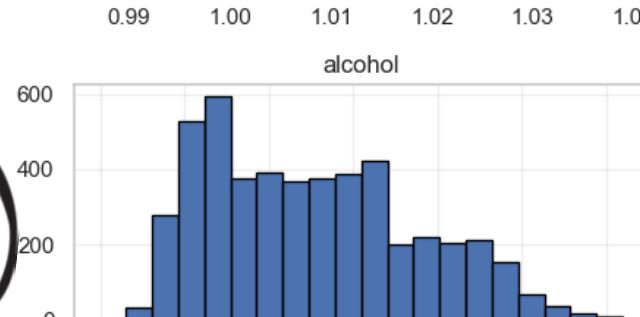
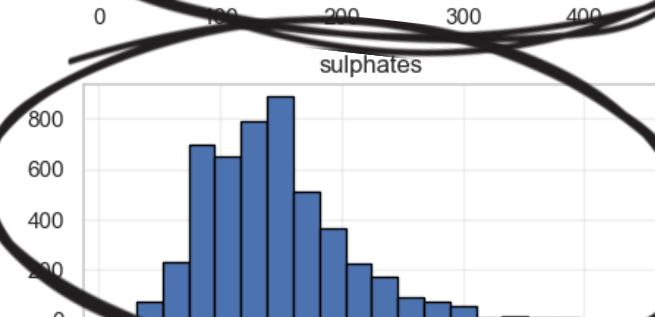
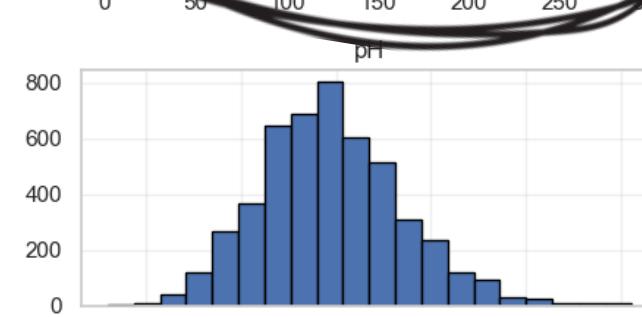
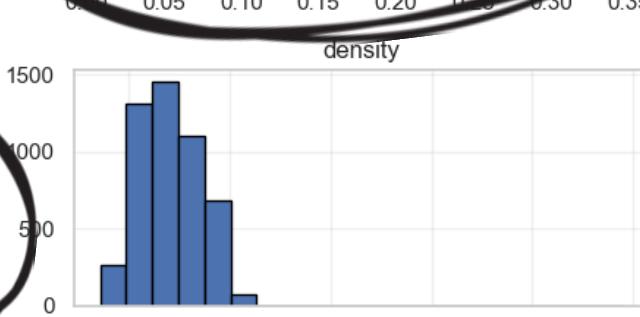
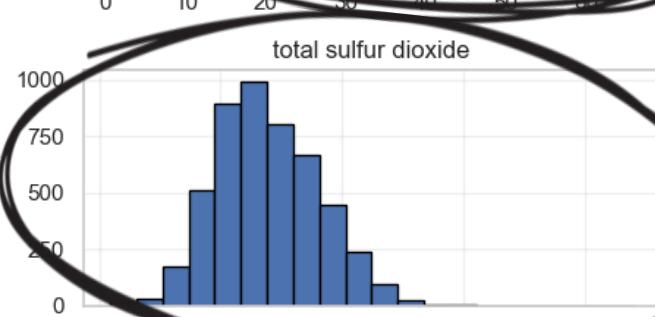
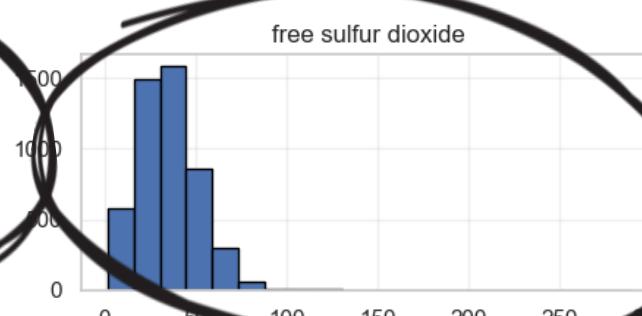
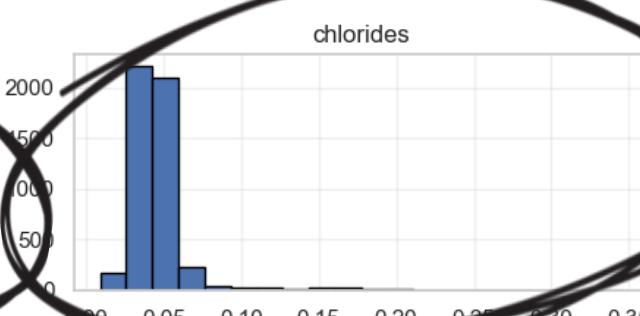
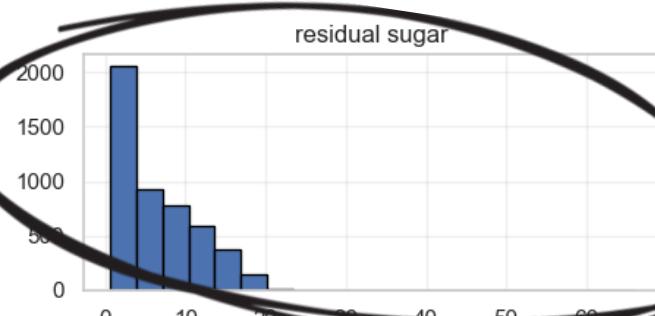
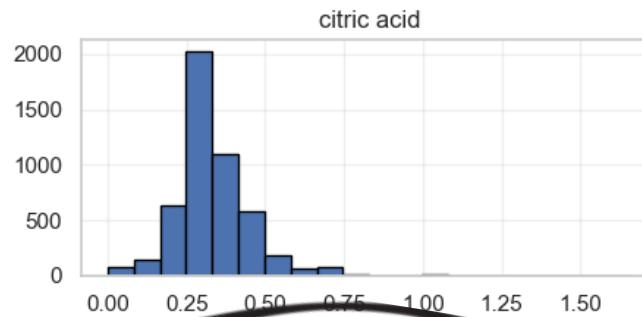
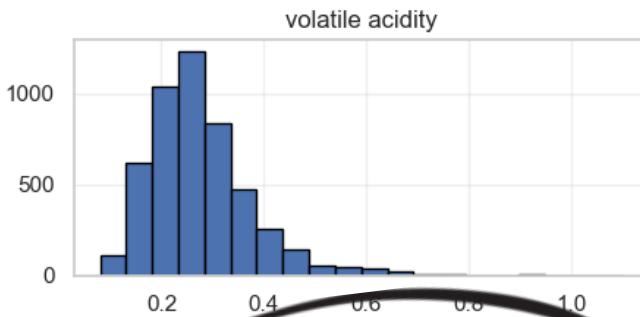
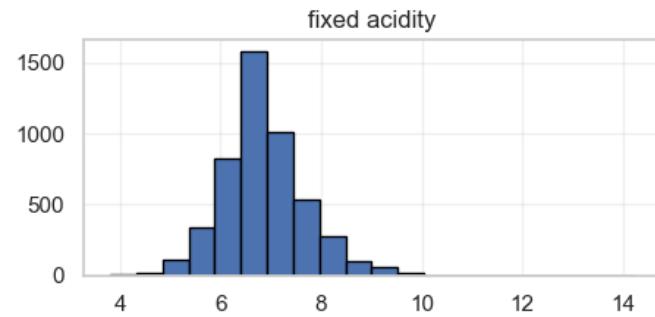
Matriz de correlación de white\_wine



## Distribución de las variables



## Distribución de las variables



## **4) SPLITTING DE DATOS**

# Splitting de datos

80% train - 20% test

```
1 # Establecemos una semilla para reproducibilidad
2 seed = np.random.seed(42)
3
4 # Dividimos los datos en conjuntos de entrenamiento y prueba
5 x_white = white.drop(columns=['quality'])
6 y_white = white['quality']
7
8 # Split 80/20
9 x_train_white, x_test_white, y_train_white, y_test_white = train_test_split(x_white, y_white, test_size=0.2, random_state=seed)
✓ 0.0s
```

```
1 # Contamos cuántas observaciones hay en cada conjunto
2 print(f'Conjunto de entrenamiento: {x_train_white.shape[0]} observaciones')
3 print(f'Conjunto de prueba: {x_test_white.shape[0]} observaciones')
✓ 0.0s
```

Conjunto de entrenamiento: 3918 observaciones  
Conjunto de prueba: 980 observaciones

- **x\_train\_white** y **y\_train\_white**: datos de entrenamiento.
- **x\_test\_white** y **y\_test\_white**: datos de prueba.

# **5) PREPROCESAMIENTO**

## **5.1) Transformación BOX-COX**

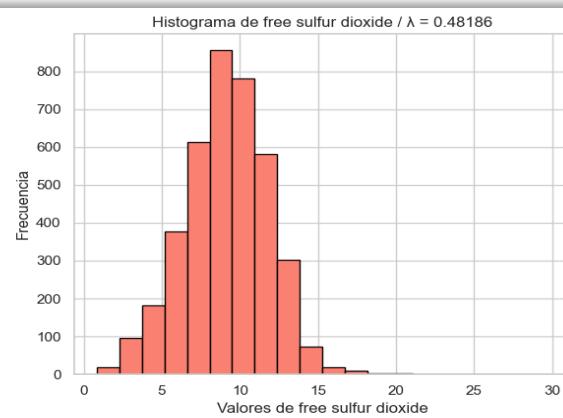
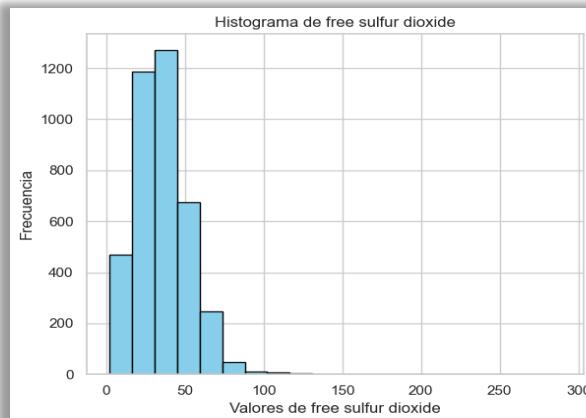
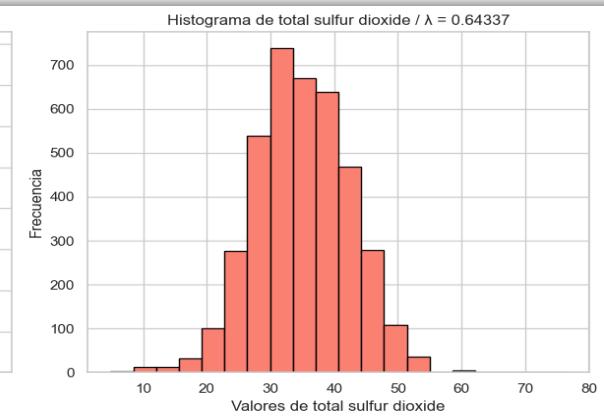
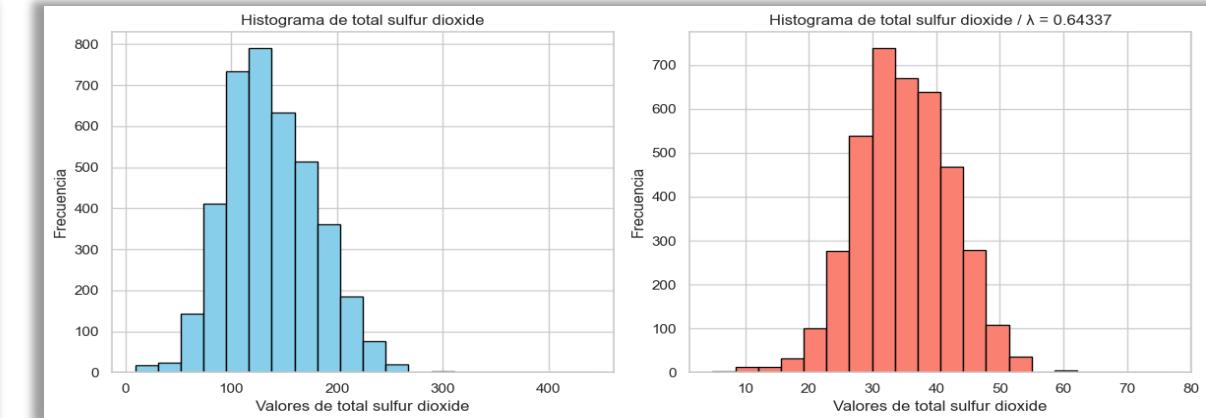
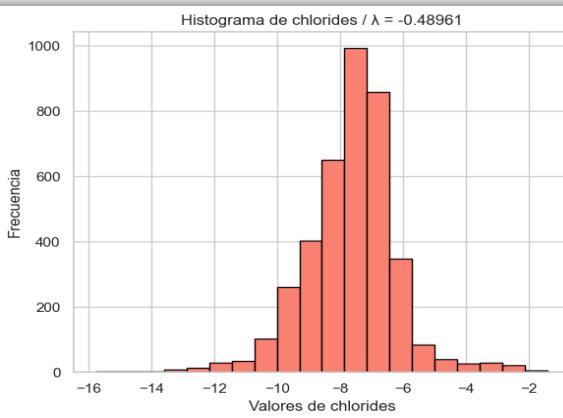
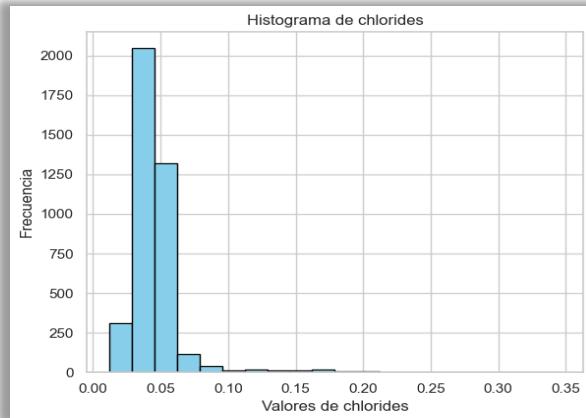
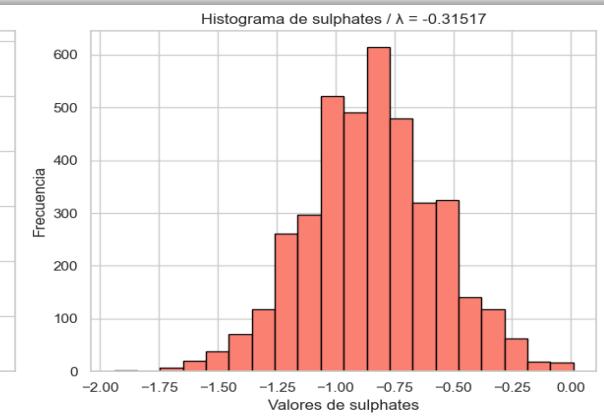
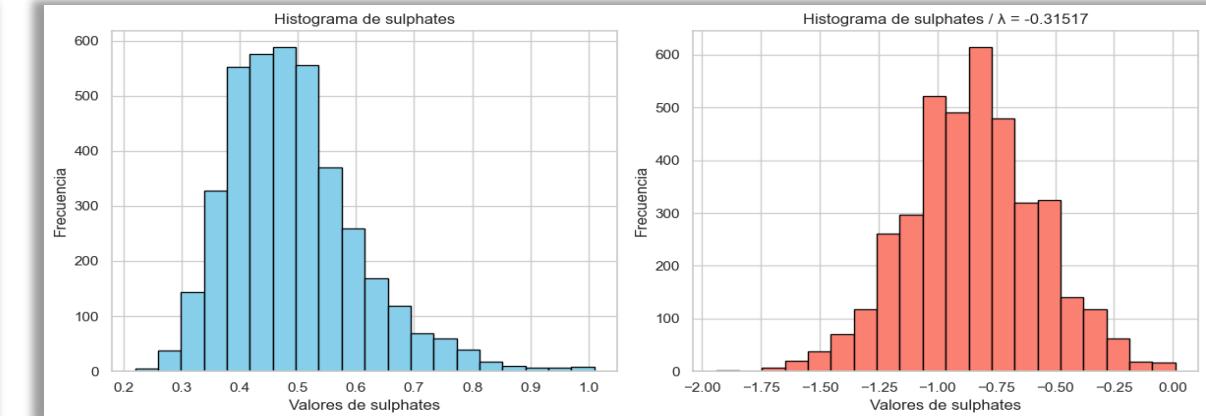
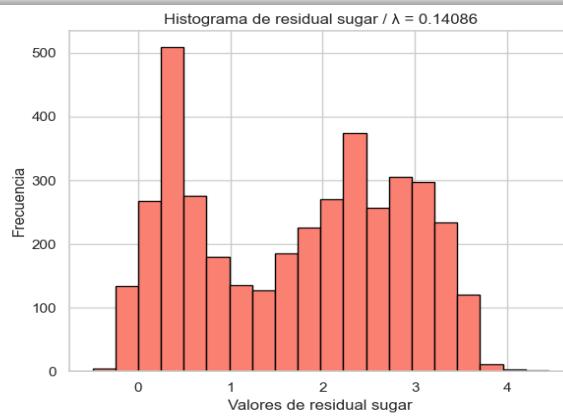
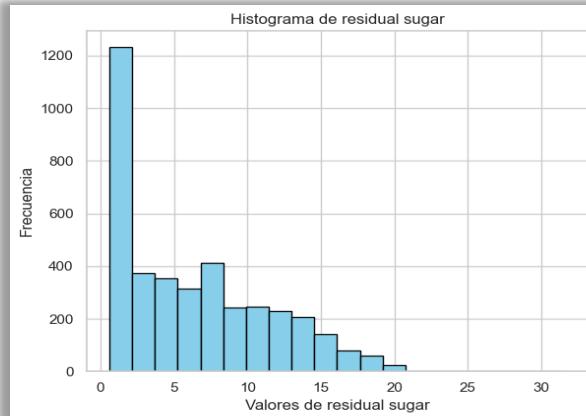
```
1 # Columnas a transformar: 'residual sugar', 'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'sulphates'
2
3 # Duplicamos el dataset para comparar los histogramas
4 white_wine_manual = X_train_white.copy()
5
6 # Elegimos los valores de lambda
7 lambda_residual_sugar = 0.20
8 lambda_chlorides = -0.5
9 lambda_free_sulfur_dioxide = 0.5
10 lambda_total_sulfur_dioxide = 0.6
11 lambda_sulphates = -0.3
12
13 # Transformación Box-Cox
14 white_wine_manual['residual sugar transformed'] = boxcox(white_wine_manual['residual sugar'], lmbda=lambda_residual_sugar)
15 white_wine_manual['chlorides transformed'] = boxcox(white_wine_manual['chlorides'], lmbda=lambda_chlorides)
16 white_wine_manual['free sulfur dioxide transformed'] = boxcox(white_wine_manual['free sulfur dioxide'], lmbda=lambda_free_sulfur_dioxide)
17 white_wine_manual['total sulfur dioxide transformed'] = boxcox(white_wine_manual['total sulfur dioxide'], lmbda=lambda_total_sulfur_dioxide)
18 white_wine_manual['sulphates transformed'] = boxcox(white_wine_manual['sulphates'], lmbda=lambda_sulphates)
19
20 # Comparamos los histogramas
21 comparar_histogramas(X_train_white, white_wine_manual, 'residual sugar', 'residual sugar transformed', lambda_residual_sugar)
22 comparar_histogramas(X_train_white, white_wine_manual, 'chlorides', 'chlorides transformed', lambda_chlorides)
23 comparar_histogramas(X_train_white, white_wine_manual, 'free sulfur dioxide', 'free sulfur dioxide transformed', lambda_free_sulfur_dioxide)
24 comparar_histogramas(X_train_white, white_wine_manual, 'total sulfur dioxide', 'total sulfur dioxide transformed', lambda_total_sulfur_dioxide)
25 comparar_histogramas(X_train_white, white_wine_manual, 'sulphates', 'sulphates transformed', lambda_sulphates)
```

# $\lambda$ AUTOMATICO

```
5 # Diccionario para almacenar los valores de lambda
6 lambdas = {}
7
8 # Columnas a transformar: 'residual sugar', 'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'sulphates'
9 columns_to_transform = ['residual sugar', 'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'sulphates']
10
11 # Aplicamos la transformación Box-Cox a X_train_white_wine
12 for column in columns_to_transform:
13     # Dado que Box-Cox solo acepta valores positivos, añadimos una pequeña constante si es necesario
14     X_train_white_transformed[column], lambda_value = boxcox(X_train_white_transformed[column] + 1e-6)
15
16     # Guardar el nombre de la columna y el valor de lambda en el diccionario
17     lambdas[f'{column} λ']= float(round(lambda_value, 5))
18
19 # Aplicamos la transformacion Box-Cox a X_test_white_wine con los valores de lambda obtenidos
20
21 for column in columns_to_transform:
22     # Dado que Box-Cox solo acepta valores positivos, añadimos una pequeña constante si es necesario
23     X_test_white_transformed[column] = boxcox(X_test_white_transformed[column] + 1e-6, lmbda=lambdas[f'{column} λ'])
24
✓ 0.1s
```

```
1 lambdas
✓ 0.0s 📈 Open 'lambdas' in Data Wrangler
{'residual sugar λ': 0.14086,
 'chlorides λ': -0.48961,
 'free sulfur dioxide λ': 0.48186,
 'total sulfur dioxide λ': 0.64337,
 'sulphates λ': -0.31517}
```

Para encontrar el valor óptimo de  $\lambda$ ,  
Box-Cox maximiza la función de verosimilitud  
logarítmica.



Residual Sugar  $\lambda = 0.14086$   
 Chlorides  $\lambda = -0.48961$   
 Free Sulfur Dioxide  $\lambda = 0.48186$   
 Total Sulfur Dioxide  $\lambda = 0.64337$   
 Sulphates  $\lambda = -0.31517$

## **5.2) Estandarización**



```
1 # Definimos el escalador
2 scaler = MinMaxScaler()
3
4 # Escalamos las variables independientesn para el conjunto de train y test
5 X_train_white_scaled = scaler.fit_transform(X_train_white_transformed)
6 X_test_white_scaled = scaler.transform(X_test_white_transformed)
7
8 # Convertimos los arrays resultantes a DataFrames
9 X_train_white_scaled = pd.DataFrame(X_train_white_scaled, columns=X_train_white_transformed.columns)
10 X_test_white_scaled = pd.DataFrame(X_test_white_scaled, columns=X_test_white_transformed.columns)
```

# **6) MODELADO**

# Principales Métricas de Evaluación para Clasificación Multinomial

- **Exactitud (Accuracy):**

- Proporción de predicciones correctas sobre el total de predicciones en todas las clases.

$$\text{Exactitud} = \frac{\text{Número de predicciones correctas}}{\text{Total de predicciones}}$$

- **Precisión (Precision) macro promedio:**

- Mide cuántas de las predicciones positivas de cada clase son realmente correctas, promediado sobre todas las clases.

$$\text{Precisión}_{macro} = \frac{1}{C} \sum_{i=1}^C \frac{TP_i}{TP_i + FP_i}$$

- **Sensibilidad (Recall) macro promedio:**

- Mide cuántos de los casos positivos reales de cada clase fueron correctamente identificados, promediado sobre todas las clases.

$$\text{Sensibilidad}_{macro} = \frac{1}{C} \sum_{i=1}^C \frac{TP_i}{TP_i + FN_i}$$

- **F1-score macro promedio:**

- Media armónica entre precisión y sensibilidad para cada clase, promediado sobre todas las clases.

$$\text{F1-score}_{macro} = \frac{1}{C} \sum_{i=1}^C 2 \times \frac{\text{Precisión}_i \times \text{Sensibilidad}_i}{\text{Precisión}_i + \text{Sensibilidad}_i}$$

# 6.1) Regresión Logistica

# Regresión Logística

```
1 # Definir el modelo de regresión logística
2 logreg = LogisticRegression(max_iter=1000, random_state=seed)
3
4 # Realizar validación cruzada en el conjunto de entrenamiento
5 scores = cross_val_score(logreg, X_train_white_scaled, y_train_white, cv=5)
6
7 # Mostrar los resultados de la validación cruzada
8 print(f"Puntajes de validación cruzada: {scores}")
9 print(f"Accuracy promedio: {scores.mean():.4f}")
✓ 0.2s
```

```
Puntajes de validación cruzada: [0.55357143 0.53826531 0.57780612 0.55555556 0.55810983]
Accuracy promedio: 0.5567
```

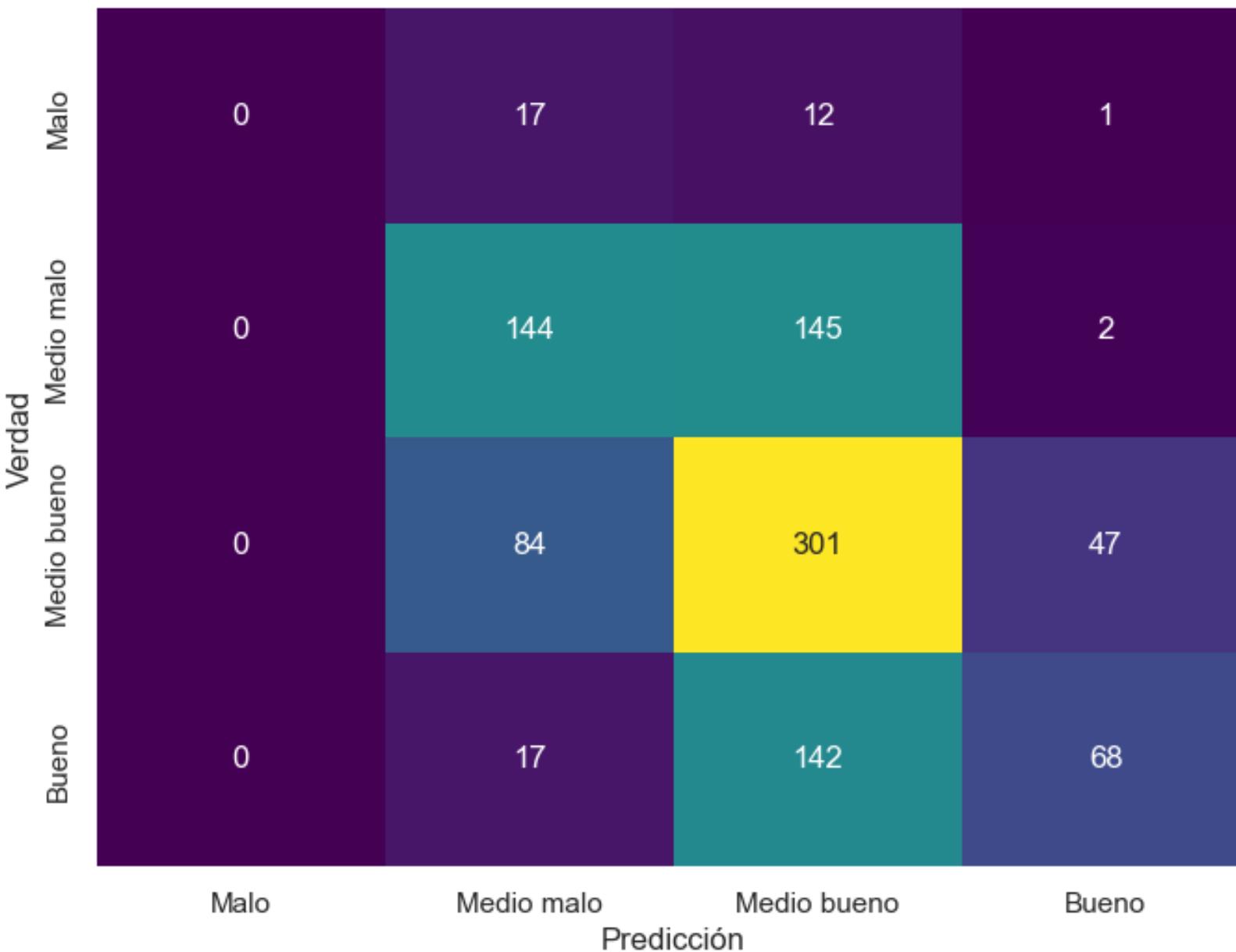
Informe de clasificación en el conjunto de prueba:				
	precision	recall	f1-score	support
0	0.00	0.00	0.00	30
1	0.55	0.49	0.52	291
2	0.50	0.70	0.58	432
3	0.58	0.30	0.39	227
accuracy			0.52	980
macro avg	0.41	0.37	0.37	980
weighted avg	0.52	0.52	0.50	980

Precisión (accuracy) en el conjunto de prueba: 0.5235  
Precisión (precision) en el conjunto de prueba: 0.5178  
Recall (sensitividad) en el conjunto de prueba: 0.5235  
F1-Score en el conjunto de prueba: 0.5031

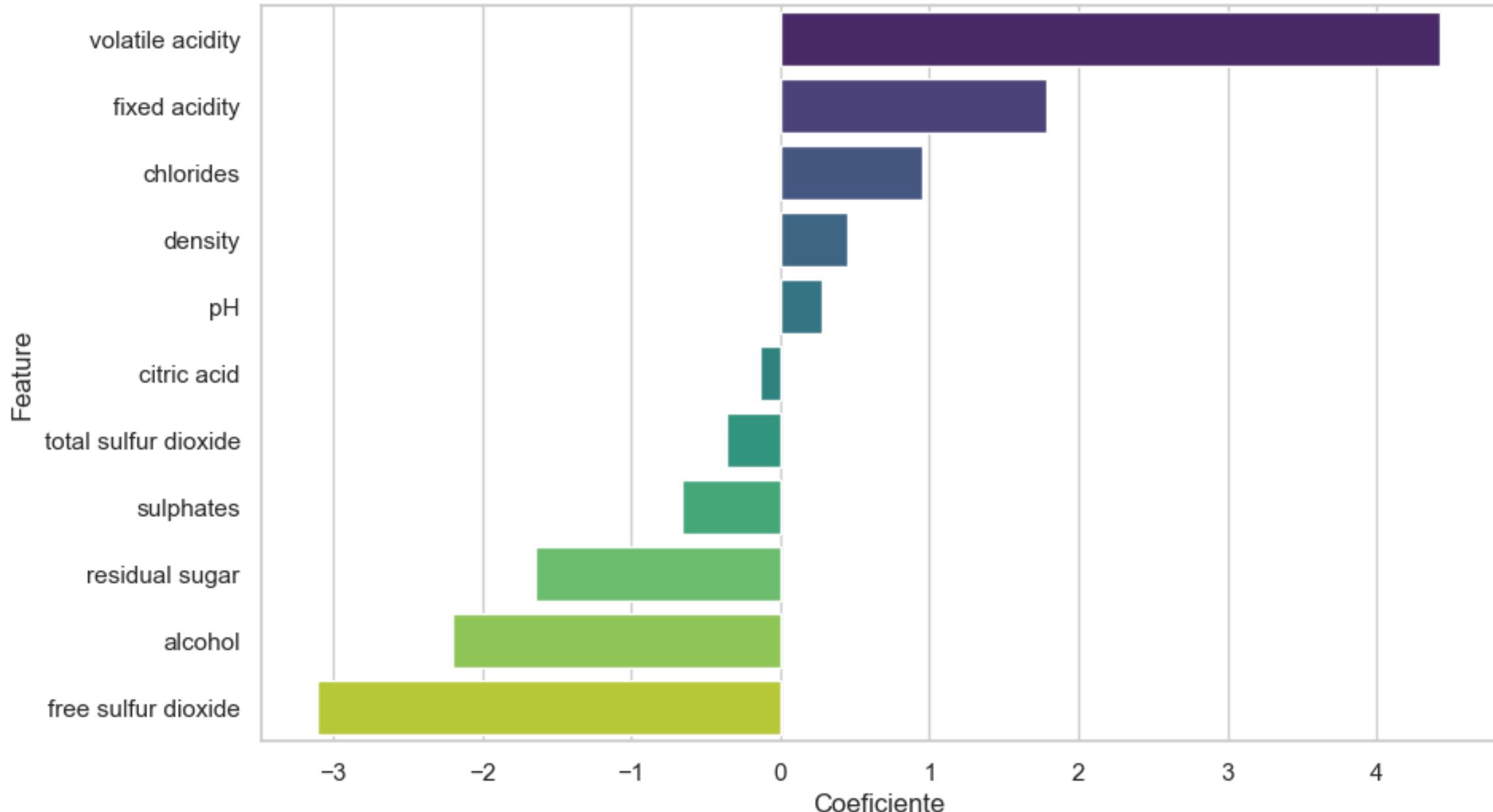


```
1 # Ajustar el modelo con los datos de entrenamiento
2 logreg.fit(X_train_white_scaled, y_train_white)
3
4 # Predecir sobre el conjunto de prueba
5 y_pred_test = logreg.predict(X_test_white_scaled)
6
7 # Informe de clasificación en el conjunto de prueba
8 print("Informe de clasificación en el conjunto de prueba:\n", classification_report(y_test_white, y_pred_test, zero_division=0))
```

Matriz de Confusión - Regresión Logística (Conjunto de Prueba)



### Importancia de los Features - Regresión Logística



## **6.2) ElasticNet**



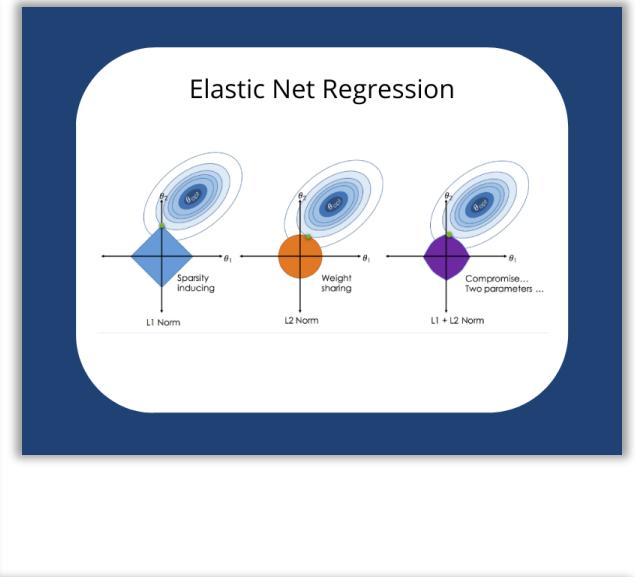
```

1 # Definir el modelo de regresión logística con penalización Elastic Net
2 elastic_net_logreg = LogisticRegression(penalty='elasticnet', solver='saga', max_iter=1000, random_state=seed)
3
4 # Definir el grid de hiperparámetros para GridSearchCV
5 param_grid = {
6     'C': [0.001, 0.1, 1, 10, 100],
7     'l1_ratio': [0, 0.25, 0.5, 0.75, 1], # ElasticNet mixing parameter
8 }
9
10 # Configurar GridSearchCV
11 grid_search = GridSearchCV(estimator=elastic_net_logreg, param_grid=param_grid, cv=5, n_jobs=-1, verbose=2)
12
13 # Ajustar el modelo con GridSearchCV
14 grid_search.fit(X_train_white_scaled, y_train_white)
15
16 # Obtener el mejor modelo
17 best_en_logreg = grid_search.best_estimator_
18
19 # Probar el modelo en el conjunto de test
20 y_pred_test = best_en_logreg.predict(X_test_white_scaled)

```

- $l1\_ratio = 1 \Rightarrow$  Regularización Lasso (solo penalización L1)
- $l1\_ratio = 0 \Rightarrow$  Regularización Ridge (solo penalización L2)
- $0 < l1\_ratio < 1 \Rightarrow$  Elastic Net (combinación de Lasso y Ridge)

- Mayor  $C \Rightarrow$  Menos regularización, mayor riesgo de sobreajuste.
- Menor  $C \Rightarrow$  Más regularización, menor riesgo de sobreajuste.

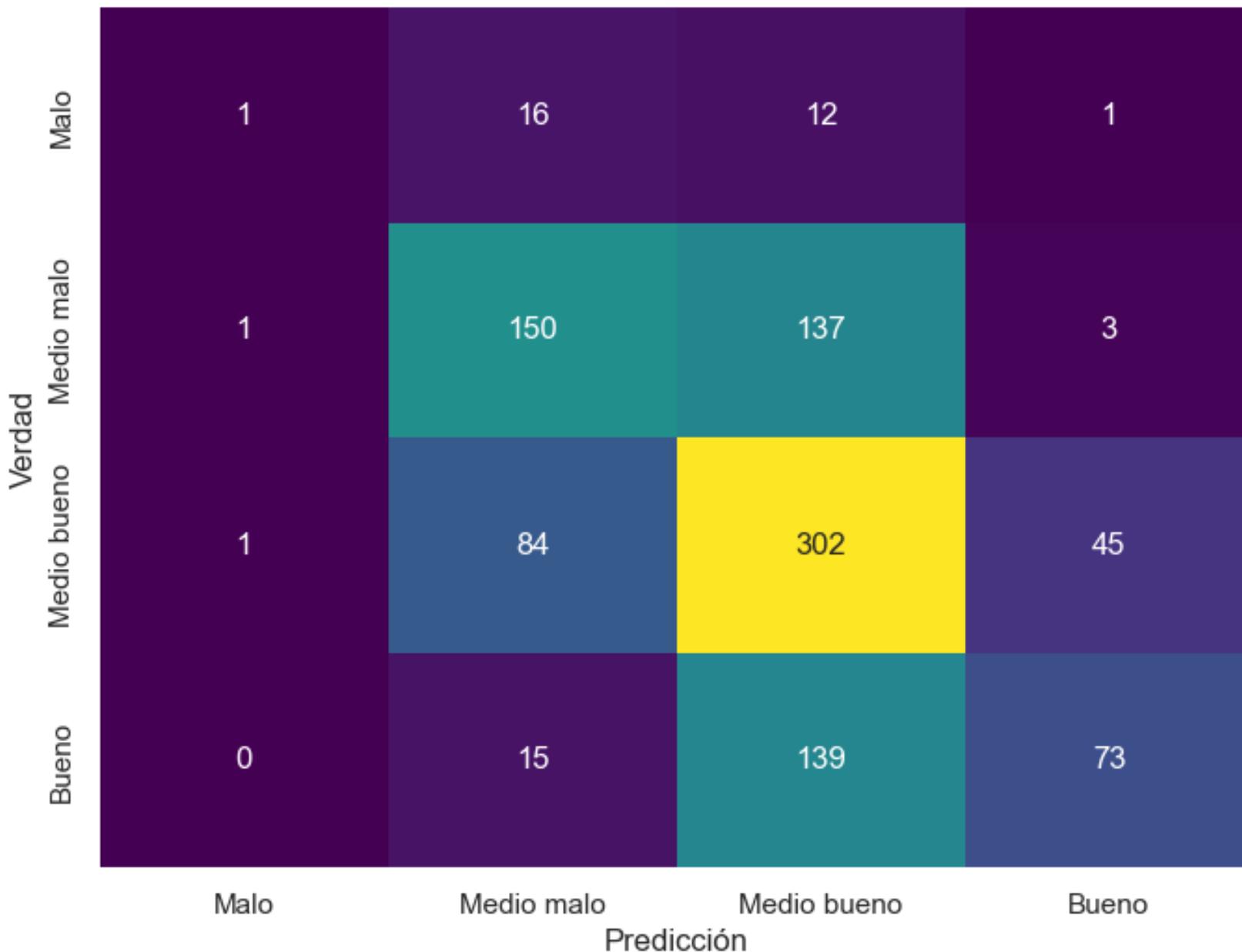


Fitting 5 folds for each of 25 candidates, totalling 125 fits  
Mejores hiperparámetros encontrados: {'C': 10, 'l1\_ratio': 0.5}  
Informe de clasificación en el conjunto de prueba (Elastic Net):

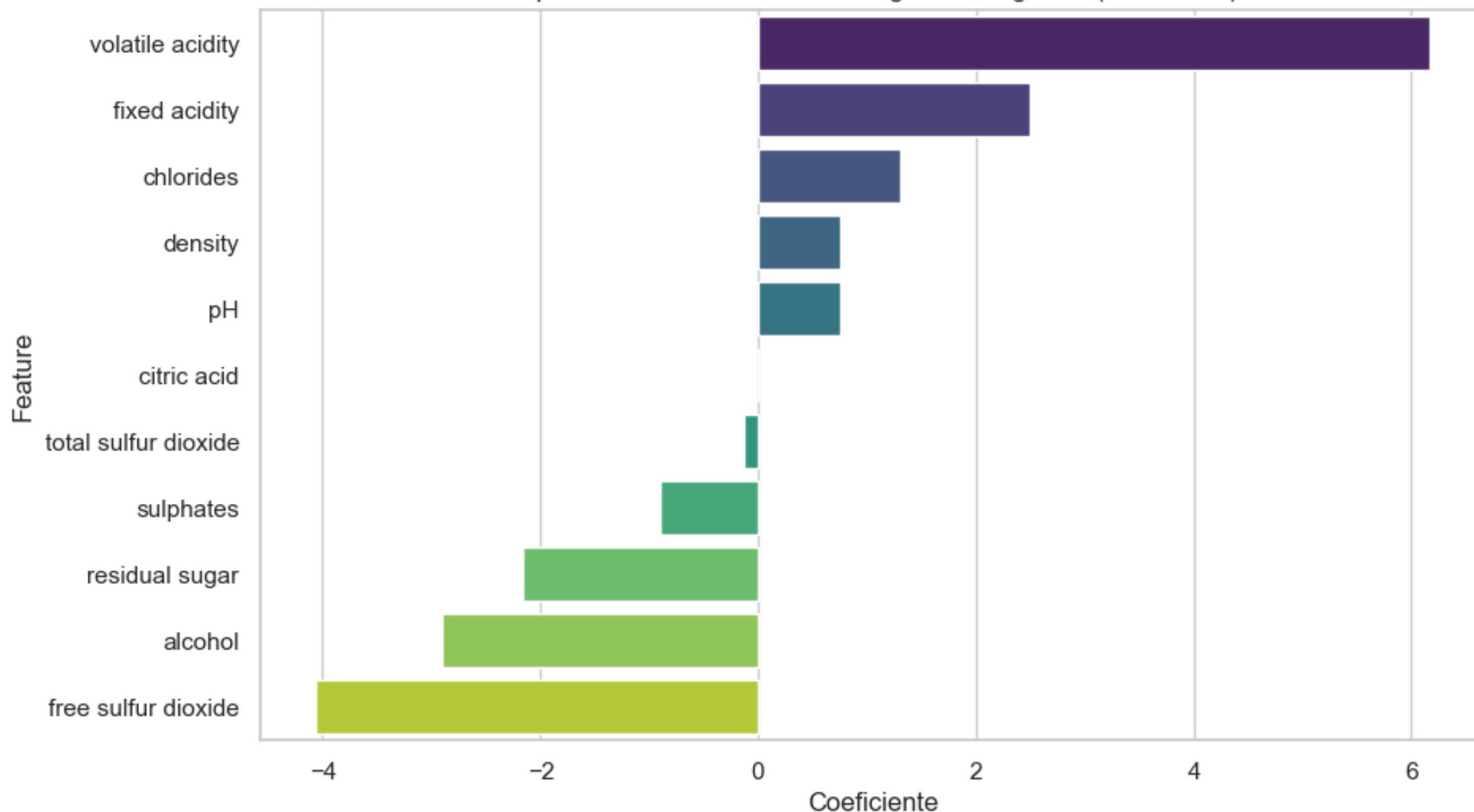
	precision	recall	f1-score	support
0	0.33	0.03	0.06	30
1	0.57	0.52	0.54	291
2	0.51	0.70	0.59	432
3	0.60	0.32	0.42	227
accuracy			0.54	980
macro avg	0.50	0.39	0.40	980
weighted avg	0.54	0.54	0.52	980

Precisión (accuracy) en el conjunto de prueba: 0.5367  
Precisión (precision) en el conjunto de prueba: 0.5425  
Recall (sensitividad) en el conjunto de prueba: 0.5367  
F1-Score en el conjunto de prueba: 0.5195

Matriz de Confusión - Regresión Logística (Elastic Net) en el Conjunto de Prueba



Importancia de los Features - Regresión Logística (Elastic Net)



## **6.3) Random Forest**

```

1 # Definir el modelo de Random Forest
2 random_forest = RandomForestClassifier(random_state=42)
3
4 # Definir el grid de hiperparámetros para GridSearchCV
5 param_grid = {
6     'n_estimators': [100, 200],           Hiperparámetros óptimos encontrados por GridSearchCV:
7     'max_depth': [10, 15],             {'max_depth': 15, 'max_features': 'log2', 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 200}
8     'min_samples_split': [2, 4],
9     'min_samples_leaf': [1, 2],
10    'max_features': ['sqrt', 'log2', None]
11 }
12
13 # Configurar GridSearchCV
14 grid_search = GridSearchCV(estimator=random_forest, param_grid=param_grid, cv=5, n_jobs=-1, verbose=2)
15
16 # Ajustar el modelo con GridSearchCV
17 grid_search.fit(X_train_white_scaled, y_train_white)
18
19 # Obtener el mejor modelo
20 best_rf = grid_search.best_estimator_
21
22 # Evaluar el modelo en el conjunto de prueba
23 y_pred_test = best_rf.predict(X_test_white_scaled)
24

```

Informe de clasificación en el conjunto de prueba (Random Forest):

	precision	recall	f1-score	support
0	0.78	0.23	0.36	30
1	0.71	0.68	0.70	291
2	0.67	0.78	0.72	432
3	0.83	0.68	0.75	227
accuracy			0.71	980
macro avg	0.75	0.59	0.63	980
weighted avg	0.72	0.71	0.71	980

Precisión en el conjunto de prueba (accuracy): 0.7122  
 Precisión en el conjunto de prueba (precision): 0.7220  
 Recall (sensitividad) en el conjunto de prueba: 0.7122  
 F1-Score en el conjunto de prueba: 0.7088

Hiperparámetros óptimos encontrados por GridSearchCV:

```
{'max_depth': 15, 'max_features': 'log2', 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 200}
```

Hiperparámetros de Random Forest

- **n\_estimators:**

- Número de árboles en el bosque.
- Un valor más alto generalmente mejora la precisión, pero también incrementa el costo computacional.
- Ejemplos: 100 y 200.

- **max\_depth:**

- Profundidad máxima de los árboles.
- Limitar la profundidad ayuda a prevenir el sobreajuste.
- Valores más bajos (ej. 10) limitan la complejidad del modelo, mientras que valores más altos (ej. 15) permiten que los árboles crezcan más y sean más complejos.

- **min\_samples\_split:**

- Número mínimo de muestras requeridas para dividir un nodo.
- Valores más bajos (ej. 2) permiten divisiones más profundas y árboles más complejos.
- Valores más altos (ej. 4) limitan la profundidad y ayudan a evitar el sobreajuste.

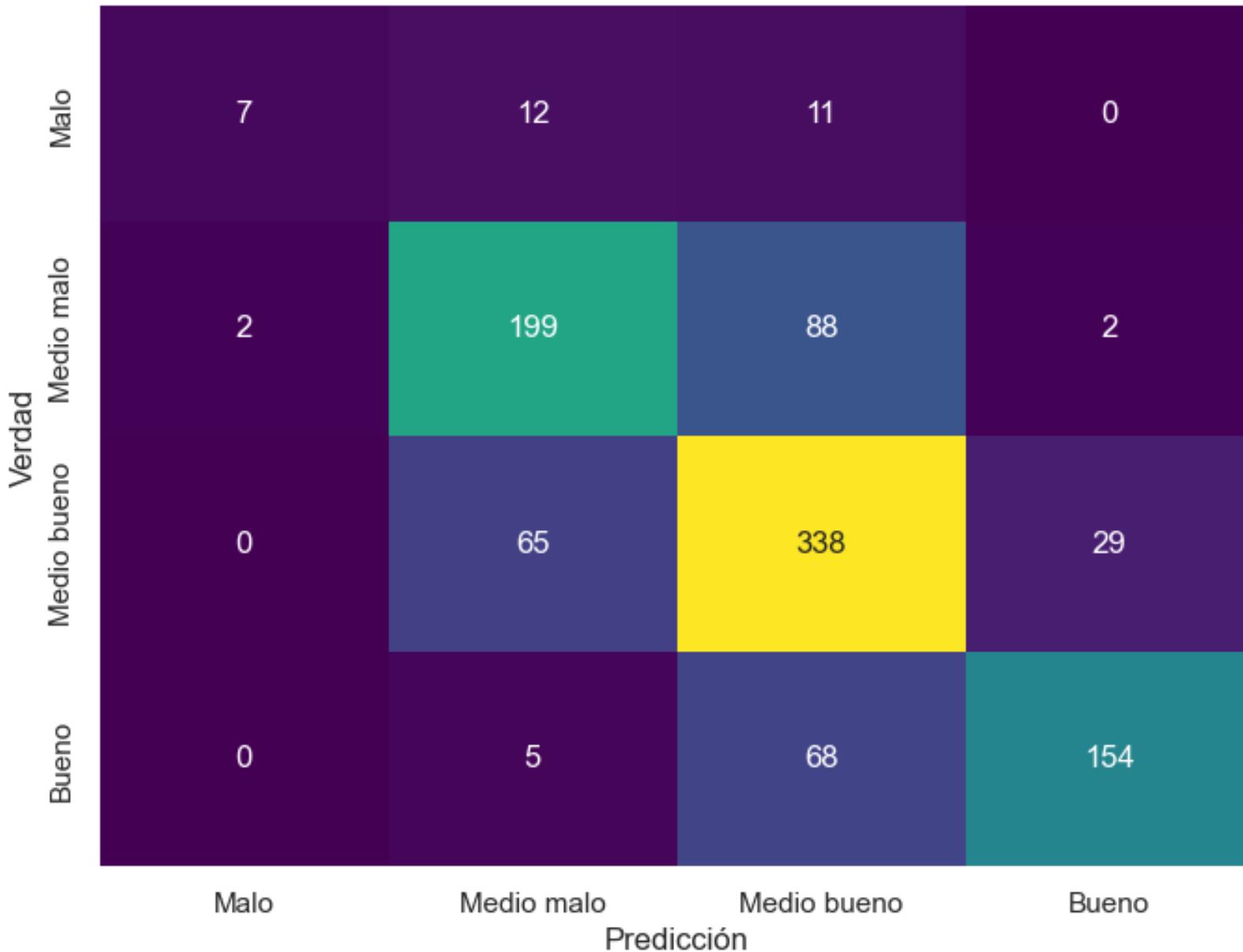
- **min\_samples\_leaf:**

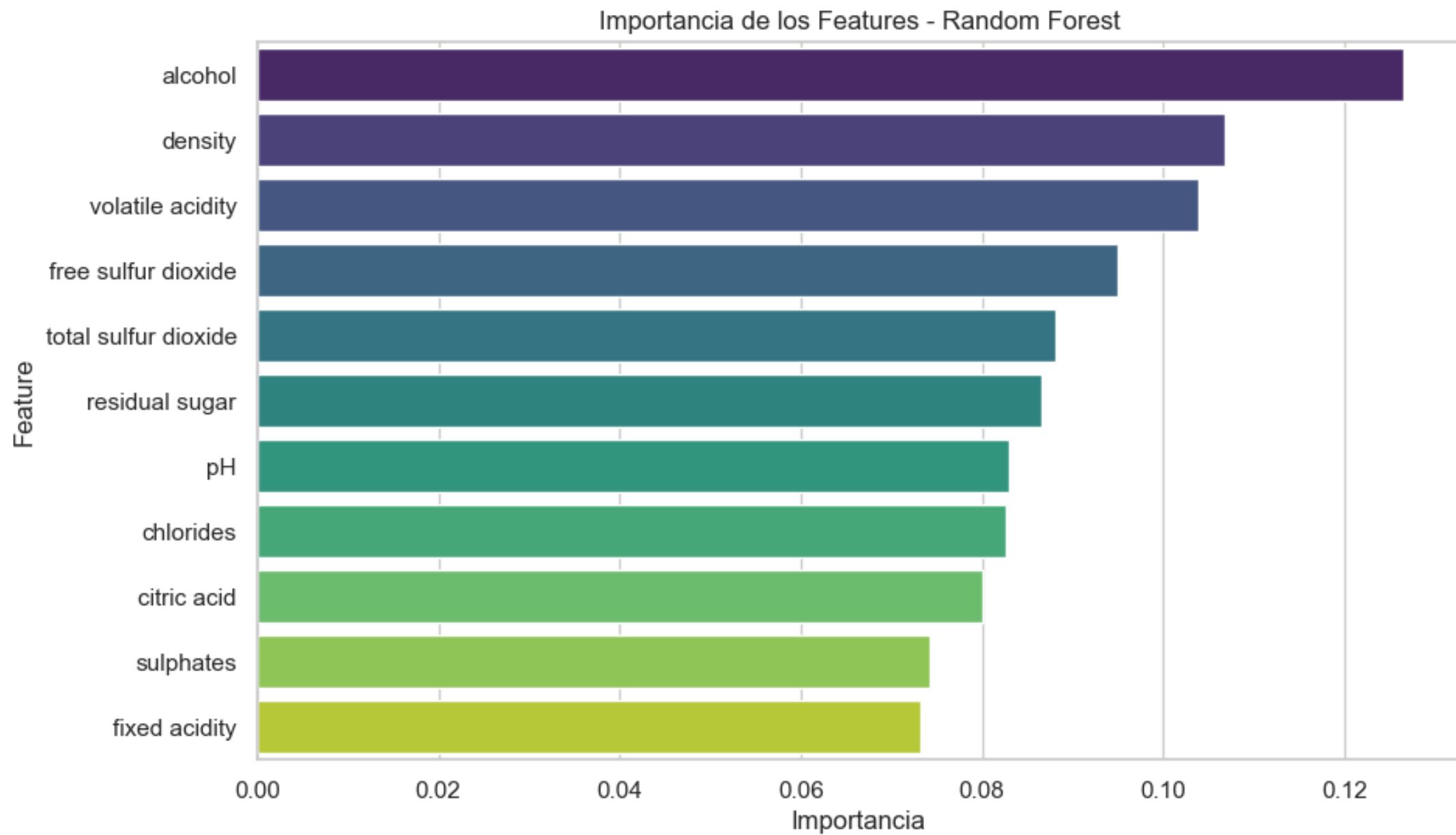
- Número mínimo de muestras necesarias en una hoja terminal.
- Un valor más alto (ej. 4) puede evitar que el modelo aprenda ruido al crear nodos con pocas muestras.

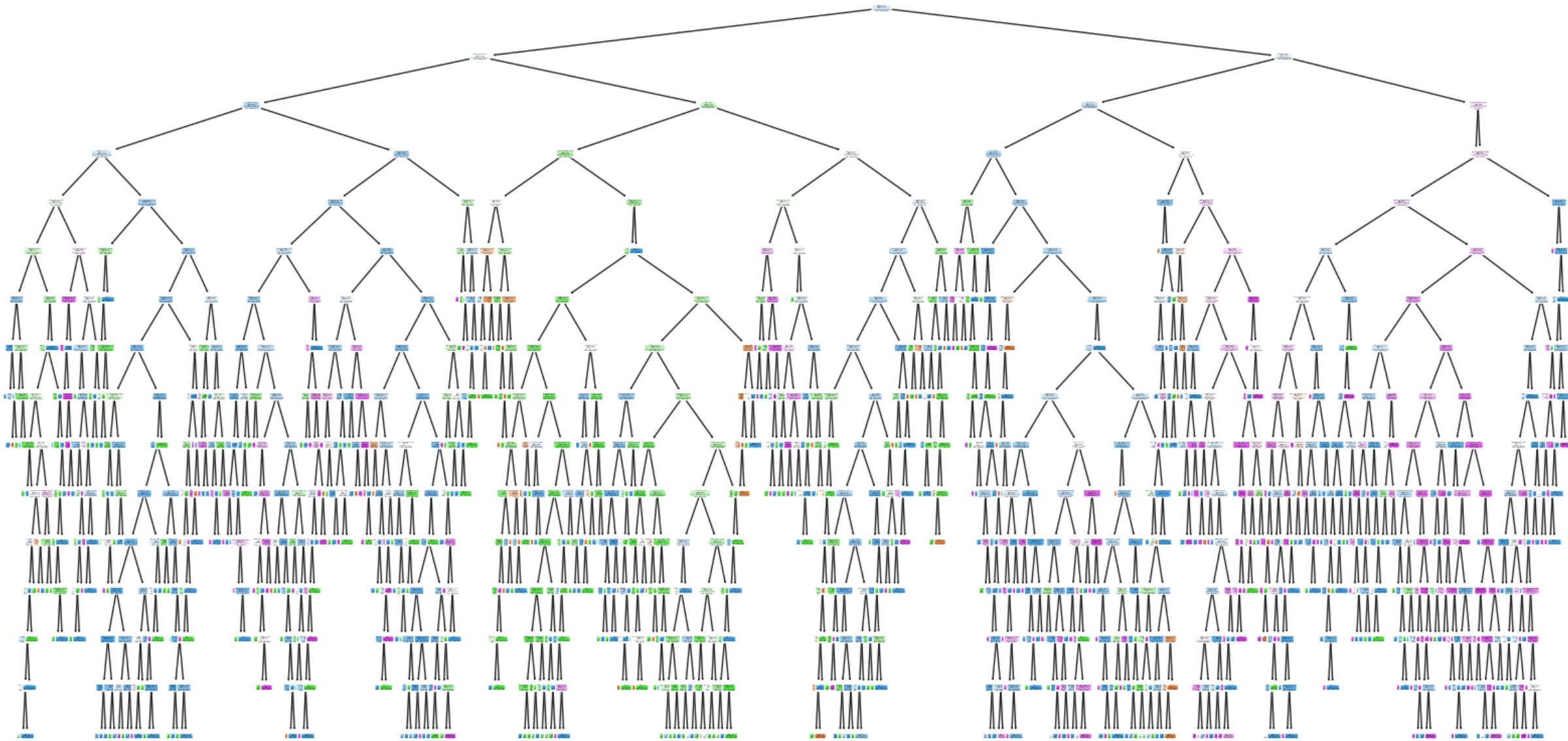
- **max\_features:**

- Número máximo de características consideradas para dividir un nodo.
- **sqrt**: Considera la raíz cuadrada del número total de características, lo cual es un valor predeterminado y común.
- **log2**: Usa el logaritmo base 2 del número total de características.
- **None**: Considera todas las características para cada división.

Matriz de Confusión - Random Forest en el Conjunto de Prueba







# **6.5) SVM (SVC-Support Vector Classifier)**



```
1 # Definir el modelo SVM
2 svm_model = SVC(random_state=seed)
3
4 # Definir el grid de hiperparámetros para GridSearchCV
5 param_grid = {
6     'C': [0.1, 1, 10, 100],
7     'gamma': [0.001, 0.01, 0.1, 1],
8     'kernel': ['linear', 'rbf', 'poly', 'sigmoid'],
9     'degree': [2, 3, 4] # Solo relevante si kernel='poly'
10 }
11
12 # Configurar GridSearchCV
13 grid_search = GridSearchCV(estimator=svm_model, param_grid=param_grid,
14
15 # Ajustar el modelo con GridSearchCV
16 grid_search.fit(X_train_white_scaled, y_train_white)
17
18 # Obtener el mejor modelo
19 best_svm = grid_search.best_estimator_
20
21 # Probar el modelo en el conjunto de test
22 y_pred_test = best_svm.predict(X_test_white_scaled)
```

```
Fitting 5 folds for each of 192 candidates, totalling 960 fits
Hiperparámetros óptimos encontrados por GridSearchCV:
{'C': 100, 'degree': 2, 'gamma': 1, 'kernel': 'rbf'}
```

Informe de clasificación en el conjunto de prueba (SVM):

	precision	recall	f1-score	support
--	-----------	--------	----------	---------

0	0.62	0.27	0.37	30
1	0.62	0.58	0.60	291
2	0.56	0.71	0.63	432
3	0.69	0.45	0.54	227

accuracy			0.60	980
macro avg	0.62	0.50	0.54	980
weighted avg	0.61	0.60	0.59	980

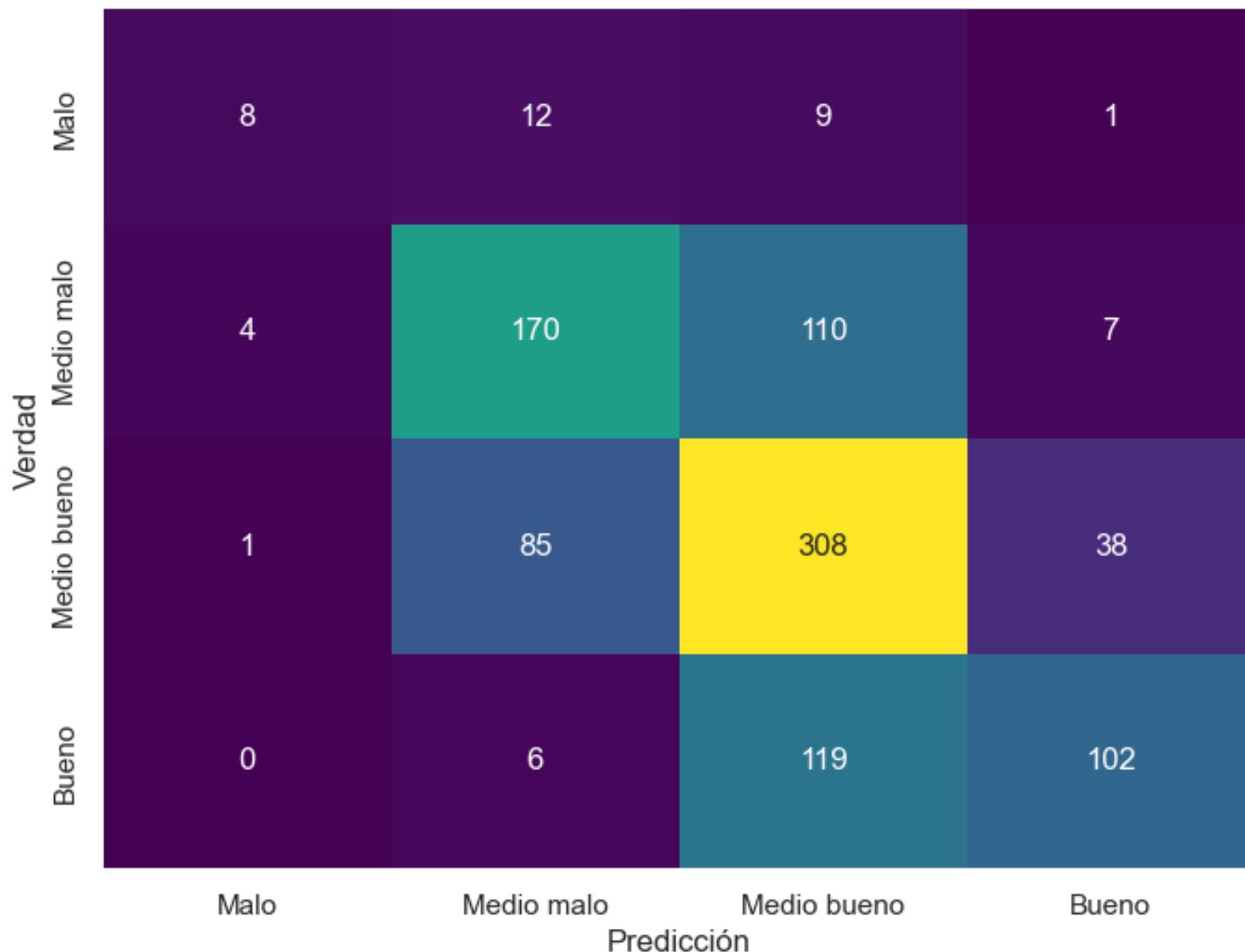
Precisión (accuracy) en el conjunto de prueba: 0.6000  
Precisión (precision) en el conjunto de prueba: 0.6120  
Recall (sensitividad) en el conjunto de prueba: 0.6000  
F1-Score en el conjunto de prueba: 0.5941

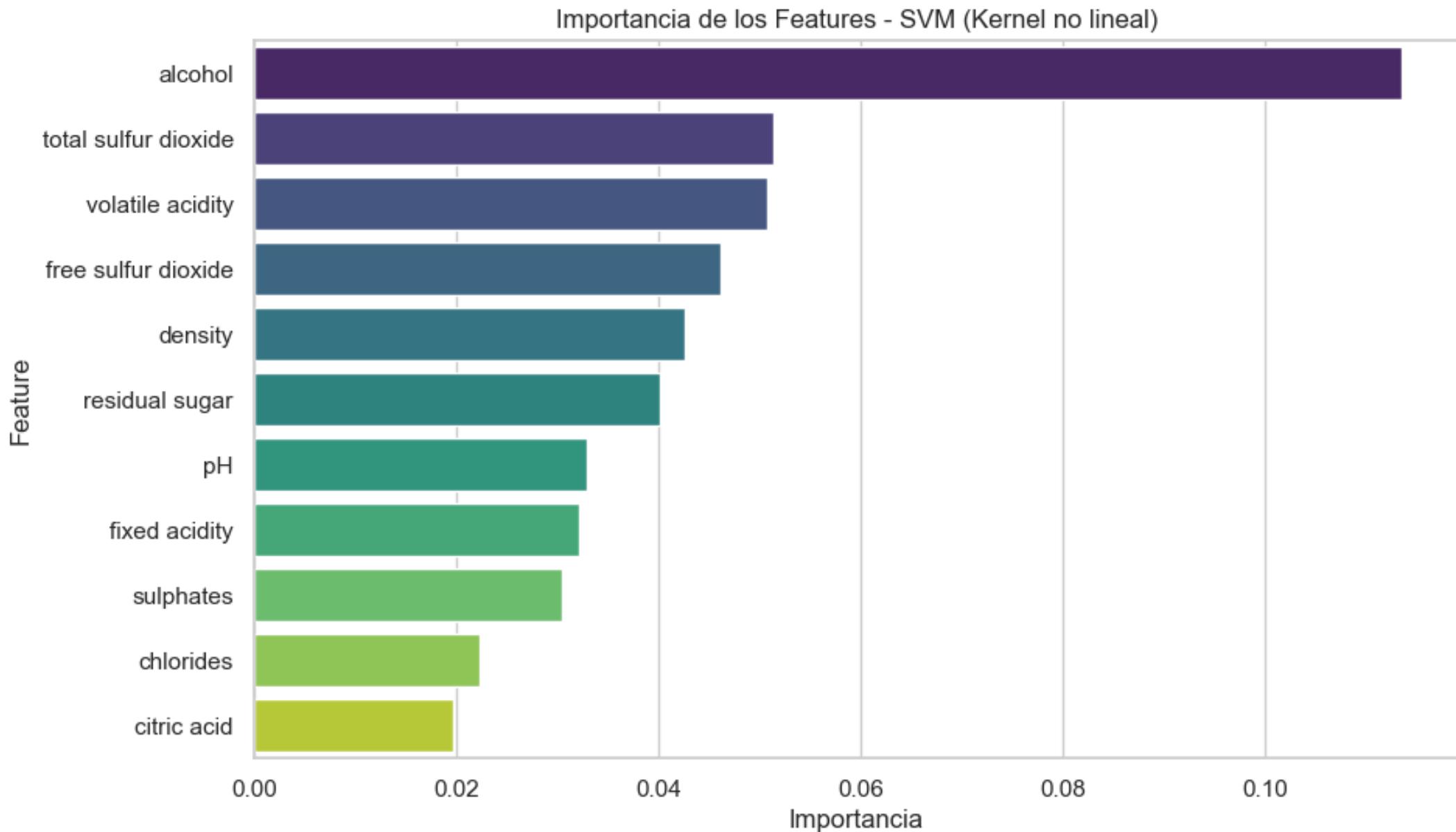
Hiperparámetros óptimos encontrados por GridSearchCV:  
{'C': 100, 'degree': 2, 'gamma': 1, 'kernel': 'rbf'}

#### Hiperparámetros de SVC (Support Vector Classifier)

- **C**:
  - Controla la penalización por error de clasificación en el conjunto de entrenamiento.
  - Valores bajos de **C** (ej. 0.1) permiten márgenes más amplios, pero pueden tolerar más errores de clasificación.
  - Valores altos de **C** (ej. 100) ajustan más estrictamente los datos, lo que podría conducir a un sobreajuste.
- **gamma**:
  - Controla la influencia de un solo punto de entrenamiento.
  - **gamma bajo** (ej. 0.001) significa que los puntos lejanos tienen una influencia más amplia, lo que genera un modelo más suave.
  - **gamma alto** (ej. 1) significa que cada punto solo tiene influencia local, lo que podría generar un sobreajuste si no se ajusta bien.
- **kernel**:
  - Especifica la función del kernel que transforma los datos a un espacio de mayor dimensionalidad.
    - **linear**: No transforma los datos, mantiene un espacio lineal.
    - **rbf (Radial Basis Function)**: Kernel no lineal que transforma los datos usando una función gaussiana. Ideal para datos no lineales.
    - **poly (Polynomial)**: Aplica un kernel polinómico, útil cuando la relación entre las clases es polinómica.
    - **sigmoid**: Similar a una función sigmoide utilizada en redes neuronales. Se utiliza menos frecuentemente.
- **degree** (solo relevante si `kernel='poly'`):
  - Define el grado del polinomio para el kernel polinómico.
  - Valores comunes: 2, 3, 4. A mayor grado, más complejo es el polinomio usado.

Matriz de Confusión - SVM en el Conjunto de Prueba





## **6.5) XGBoost**



```
1 # Definir el modelo de XGBoost
2 xgboost_model = xgb.XGBClassifier(random_state=seed)
3
4 # Definir el grid de hiperparámetros para GridSearchCV
5 param_grid = {
6     'n_estimators': [100, 200],
7     'max_depth': [3, 5, 7],
8     'learning_rate': [0.01, 0.1, 0.2],
9     'subsample': [0.8, 1],
10    'colsample_bytree': [0.8, 1]}
11 # Configurar GridSearchCV
12 grid_search = GridSearchCV(estimator=xgboost_model, param_grid=param_grid, cv=5, n_jobs=-1, verbose=2)
13
14 # Ajustar el modelo con GridSearchCV
15 grid_search.fit(X_train_white_scaled, y_train_white)
16
17 # Obtener el mejor modelo
18 best_xgb = grid_search.best_estimator_
19
20 # Probar el modelo en el conjunto de test
21 y_pred_test = best_xgb.predict(X_test_white_scaled)
```

Fitting 5 folds for each of 72 candidates, totalling 360 fits  
Hiperparámetros óptimos encontrados por GridSearchCV:  
{'colsample\_bytree': 1, 'learning\_rate': 0.1, 'max\_depth': 7, 'n\_estimators': 200, 'subsample': 0.8}

Informe de clasificación en el conjunto de prueba (XGBoost):				
	precision	recall	f1-score	support
0	0.65	0.43	0.52	30
1	0.71	0.69	0.70	291
2	0.68	0.76	0.72	432
3	0.78	0.67	0.73	227
accuracy			0.71	980
macro avg	0.71	0.64	0.67	980
weighted avg	0.71	0.71	0.71	980

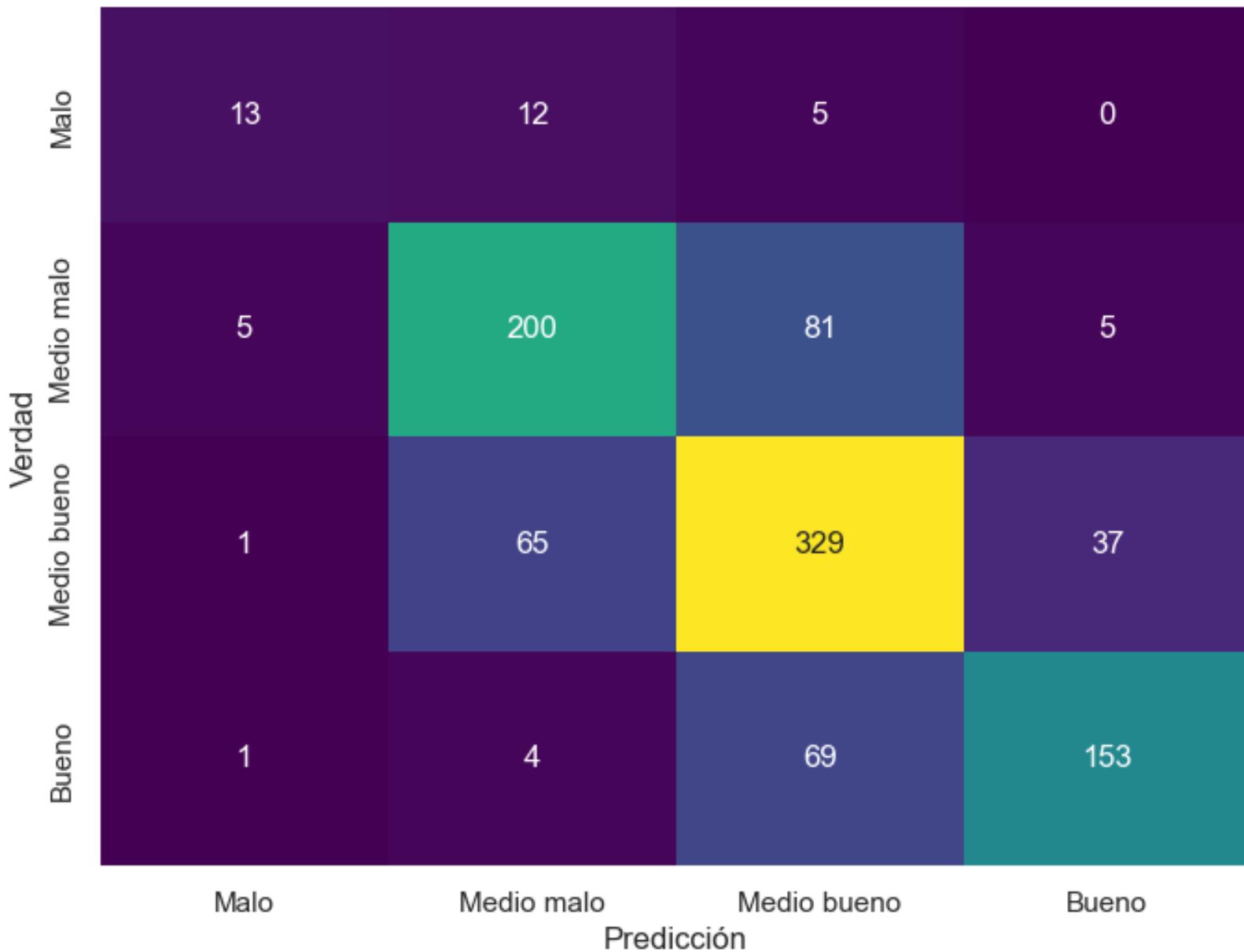
Precisión (accuracy) en el conjunto de prueba: 0.7092  
Precisión (precision) en el conjunto de prueba: 0.7126  
Recall (sensitividad) en el conjunto de prueba: 0.7092  
F1-Score en el conjunto de prueba: 0.7082

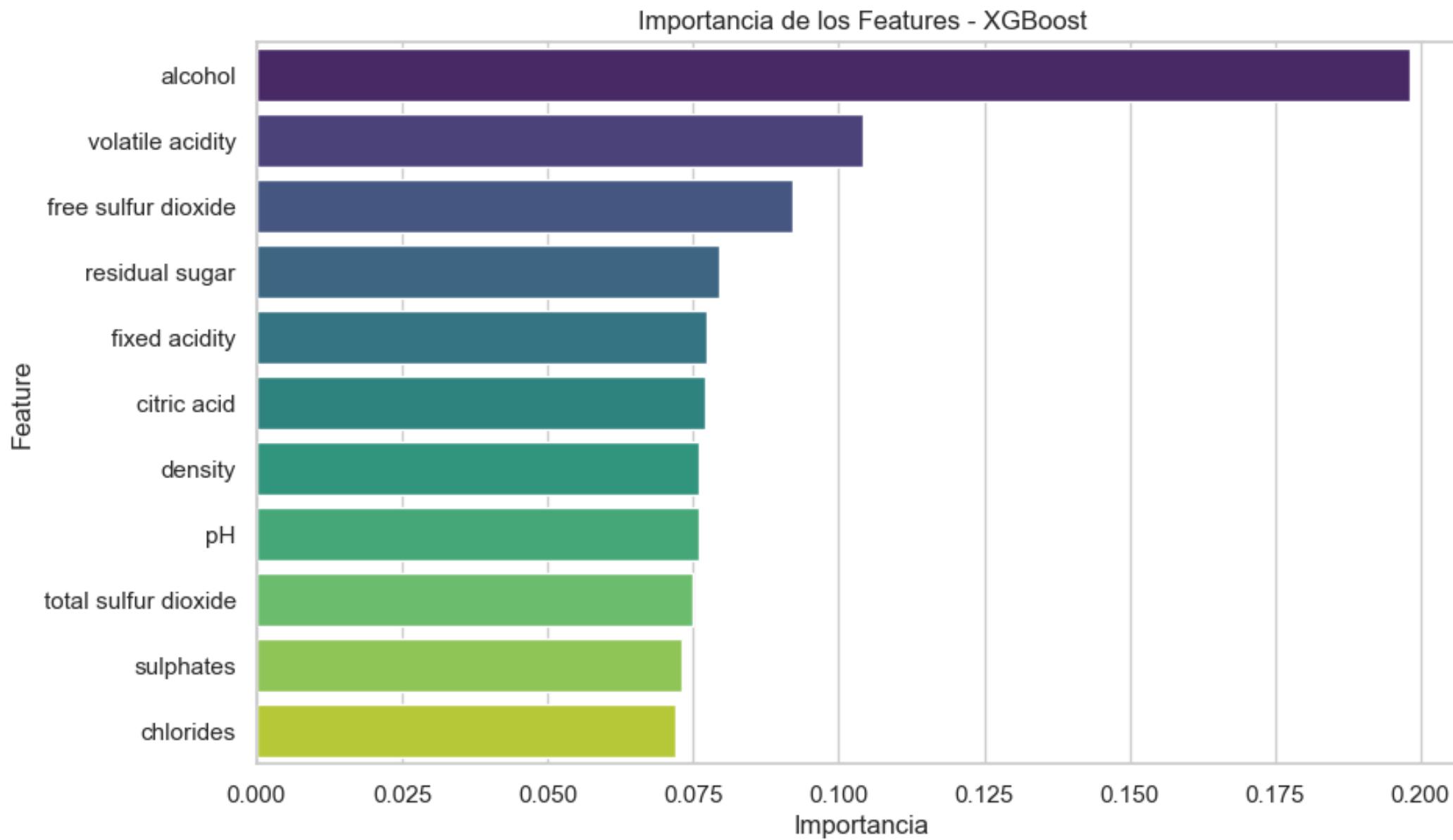
```
Fitting 5 folds for each of 72 candidates, totalling 360 fits
Hiperparámetros óptimos encontrados por GridSearchCV:
{'colsample_bytree': 1, 'learning_rate': 0.1, 'max_depth': 7, 'n_estimators': 200, 'subsample': 0.8}
```

## Hiperparámetros de XGBoost

- **n\_estimators:**
  - Número de árboles a entrenar en el modelo.
  - **Mayor número** de árboles puede aumentar la capacidad del modelo, pero también puede llevar al sobreajuste.
- **max\_depth:**
  - Profundidad máxima de cada árbol.
  - **Mayor profundidad** permite al modelo capturar patrones más complejos, pero también puede llevar al sobreajuste.
- **learning\_rate:**
  - Tasa de aprendizaje que controla el peso que se le da a cada árbol en el ensamblado.
  - **Tasas bajas** (ej. **0.01**) requieren más árboles (más iteraciones) para ajustar el modelo, pero pueden mejorar la generalización.
- **subsample:**
  - Proporción de los datos utilizados para entrenar cada árbol.
  - **Valores menores** a 1 (ej. **0.8**) pueden ayudar a reducir el sobreajuste al hacer el modelo menos dependiente de una única muestra.
- **colsample\_bytree:**
  - Fracción de características que se utilizan en cada árbol.
  - **Valores menores** a 1 pueden reducir el sobreajuste y acelerar el entrenamiento al utilizar solo una parte de las características en cada árbol.

Matriz de Confusión - XGBoost en el Conjunto de Prueba





## **6.6) Deep Learning**

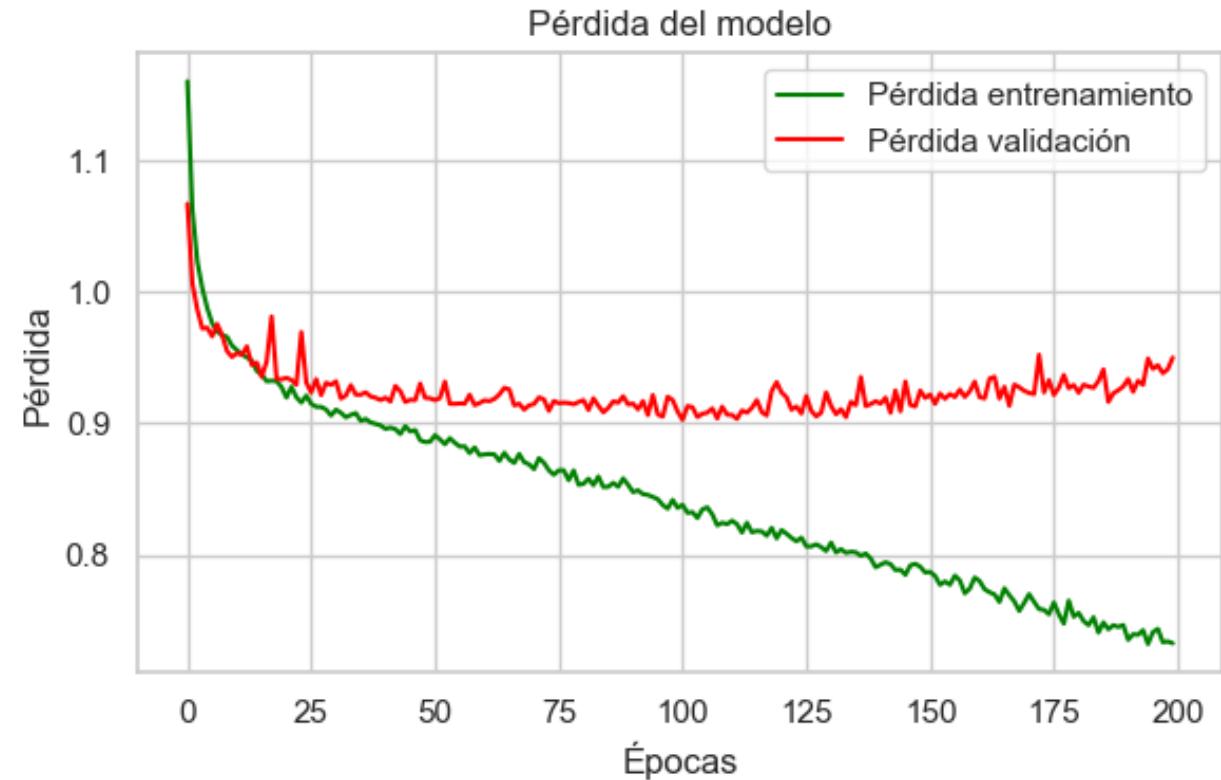
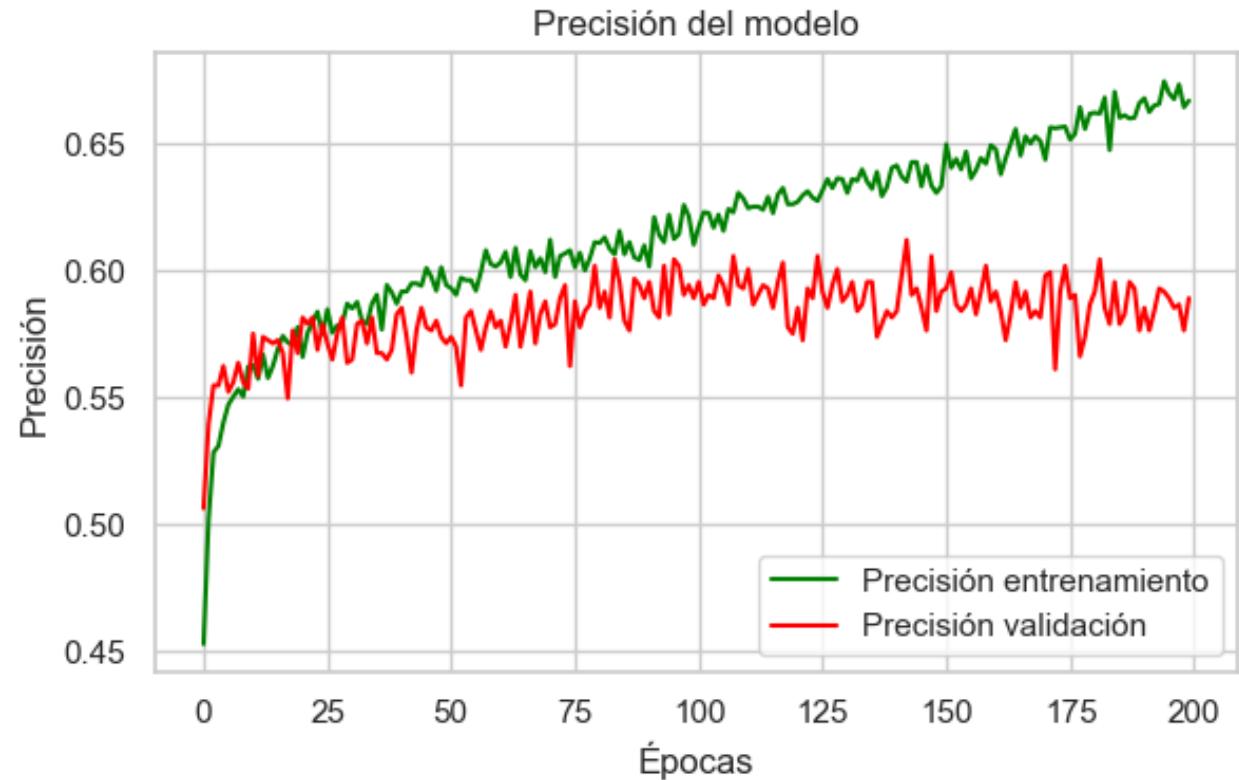


```
1 # Convertir las etiquetas en categorías (one-hot encoding) si es multiclase
2 y_train_cat = to_categorical(y_train_white)
3 y_test_cat = to_categorical(y_test_white)
4
5 tf.random.set_seed(seed)
6
7 # Definir el modelo de red neuronal
8 model = Sequential()
9
10 # Capa de entrada (input layer) usando Input en lugar de input_dim
11 model.add(Input(shape=(X_train_white_scaled.shape[1],)))
12 model.add(Dense(128, activation='relu'))
13
14 # Capa oculta (hidden layer)
15 model.add(Dense(64, activation='relu'))
16 model.add(Dropout(0.2)) # Agregar dropout para evitar sobreajuste
17
18 # Capa de salida (output layer), para clasificación multiclase
19 model.add(Dense(y_train_cat.shape[1], activation='softmax'))
20
21 # Compilar el modelo
22 model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
23
24 # Entrenar el modelo
25 history = model.fit(X_train_white_scaled, y_train_cat, epochs=200, batch_size=32, validation_split=0.2, verbose=1)
26
27 # Predecir sobre el conjunto de prueba
28 y_pred_test = model.predict(X_test_white_scaled)
29 y_pred_classes = np.argmax(y_pred_test, axis=1)
```

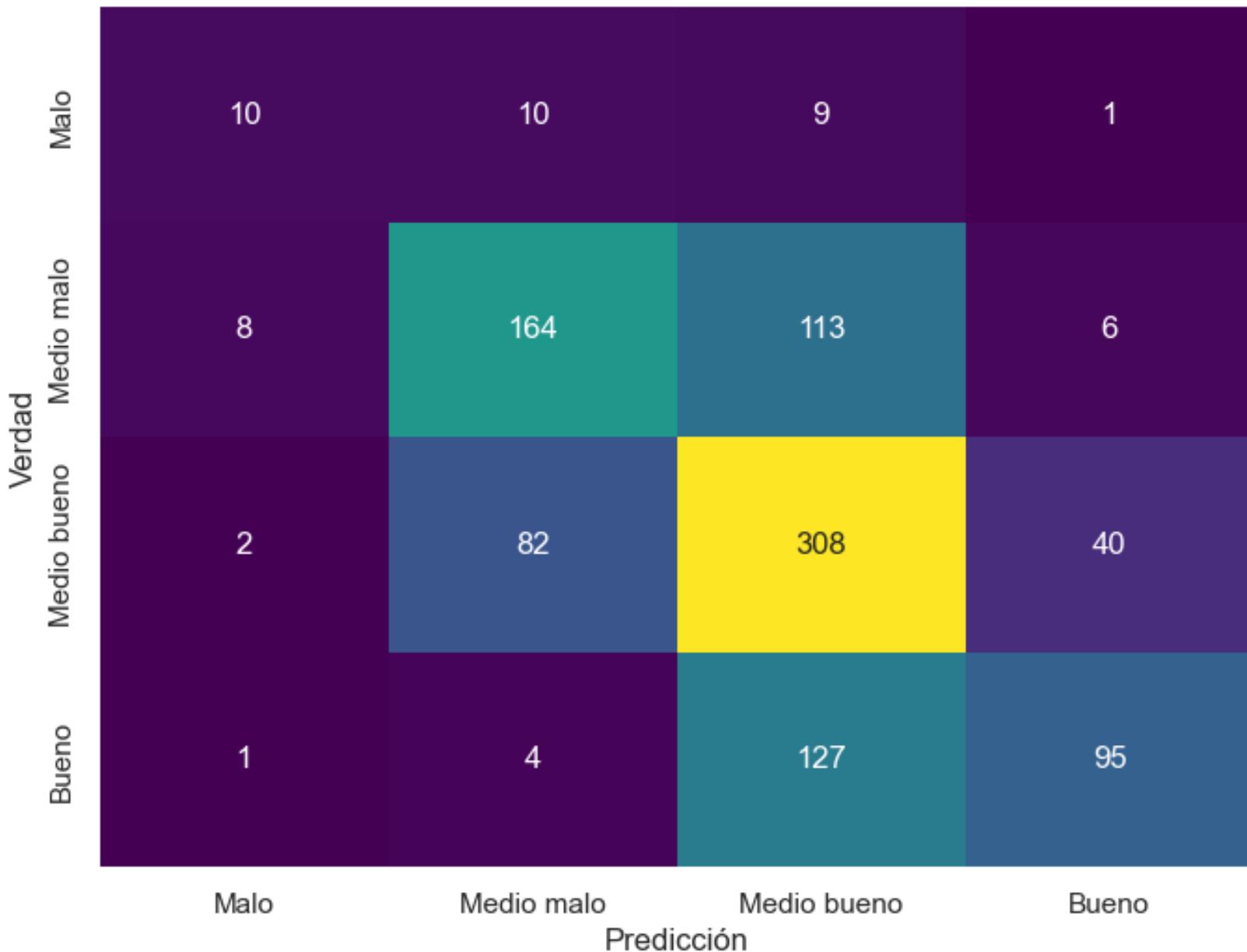
Informe de clasificación en el conjunto de prueba (Deep Learning):						
		precision	recall	f1-score	support	
	0	0.48	0.33	0.39	30	
	1	0.63	0.56	0.60	291	
	2	0.55	0.71	0.62	432	
	3	0.67	0.42	0.51	227	
		accuracy		0.59	980	
		macro avg	0.58	0.51	0.53	980
		weighted avg	0.60	0.59	0.58	980

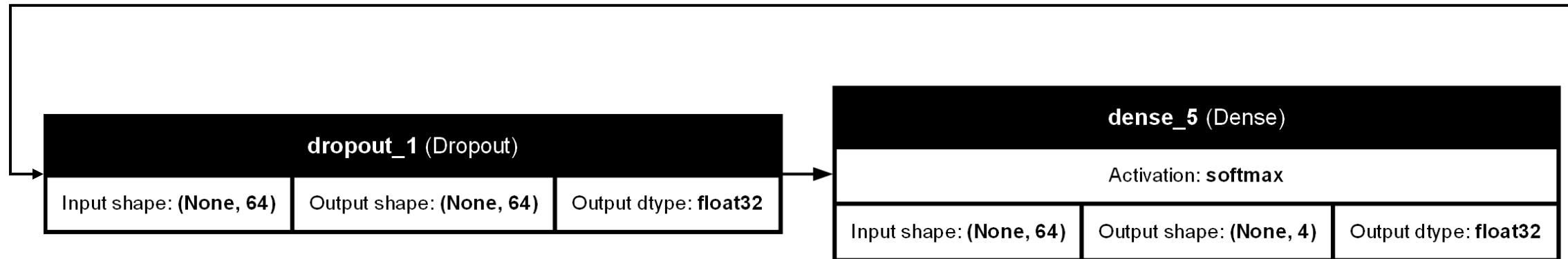
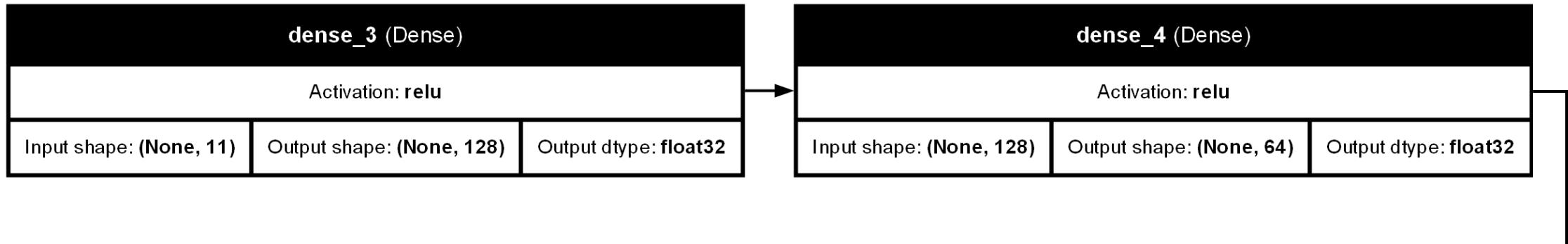
Precisión (accuracy) en el conjunto de prueba: 0.5888  
Precisión (precision) en el conjunto de prueba: 0.6006  
Recall (sensitividad) en el conjunto de prueba: 0.5888  
F1-Score en el conjunto de prueba: 0.5826

```
Epoch 1/200
98/98 6s 14ms/step - accuracy: 0.4514 - loss: 1.2079 - val_accuracy: 0.4668 - val_loss: 1.0972
Epoch 2/200
98/98 0s 2ms/step - accuracy: 0.4766 - loss: 1.0765 - val_accuracy: 0.5153 - val_loss: 1.0360
Epoch 3/200
98/98 0s 2ms/step - accuracy: 0.4923 - loss: 1.0554 - val_accuracy: 0.5255 - val_loss: 1.0154
Epoch 4/200
98/98 0s 2ms/step - accuracy: 0.5338 - loss: 0.9951 - val_accuracy: 0.5523 - val_loss: 1.0082
Epoch 5/200
98/98 0s 2ms/step - accuracy: 0.5401 - loss: 0.9820 - val_accuracy: 0.5344 - val_loss: 0.9879
Epoch 6/200
98/98 0s 2ms/step - accuracy: 0.5392 - loss: 0.9957 - val_accuracy: 0.5306 - val_loss: 0.9956
Epoch 7/200
98/98 0s 2ms/step - accuracy: 0.5434 - loss: 0.9942 - val_accuracy: 0.5497 - val_loss: 0.9726
Epoch 8/200
98/98 0s 2ms/step - accuracy: 0.5420 - loss: 0.9780 - val_accuracy: 0.5523 - val_loss: 0.9777
Epoch 9/200
98/98 0s 2ms/step - accuracy: 0.5486 - loss: 0.9581 - val_accuracy: 0.5485 - val_loss: 0.9701
Epoch 10/200
98/98 0s 2ms/step - accuracy: 0.5409 - loss: 0.9816 - val_accuracy: 0.5548 - val_loss: 0.9950
```



Matriz de Confusión - Deep Learning en el Conjunto de Prueba





Layer (type)	Output Shape	Param #
dense_3 (Dense)	(None, 128)	1,536
dense_4 (Dense)	(None, 64)	8,256
dropout_1 (Dropout)	(None, 64)	0
dense_5 (Dense)	(None, 4)	260

Total params: 30,158 (117.81 KB)  
Trainable params: 10,052 (39.27 KB)  
Non-trainable params: 0 (0.00 B)  
Optimizer params: 20,106 (78.54 KB)

# **7) Comparación de Modelos White y Red**

# Regresión Logística



White

Informe de clasificación en el conjunto de prueba:

	precision	recall	f1-score	support
0	0.00	0.00	0.00	30
1	0.55	0.49	0.52	291
2	0.50	0.70	0.58	432
3	0.58	0.30	0.39	227
accuracy			0.52	980
macro avg	0.41	0.37	0.37	980
weighted avg	0.52	0.52	0.50	980

Precisión (accuracy) en el conjunto de prueba: 0.5235

Precisión (precision) en el conjunto de prueba: 0.5178

Recall (sensitividad) en el conjunto de prueba: 0.5235

F1-Score en el conjunto de prueba: 0.5031

Red

Informe de clasificación en el conjunto de prueba:

	precision	recall	f1-score	support
0	0.00	0.00	0.00	11
1	0.62	0.77	0.68	130
2	0.53	0.58	0.55	132
3	0.64	0.19	0.30	47
accuracy			0.58	320
macro avg	0.45	0.38	0.38	320
weighted avg	0.56	0.58	0.55	320

Precisión (accuracy) en el conjunto de prueba: 0.5781

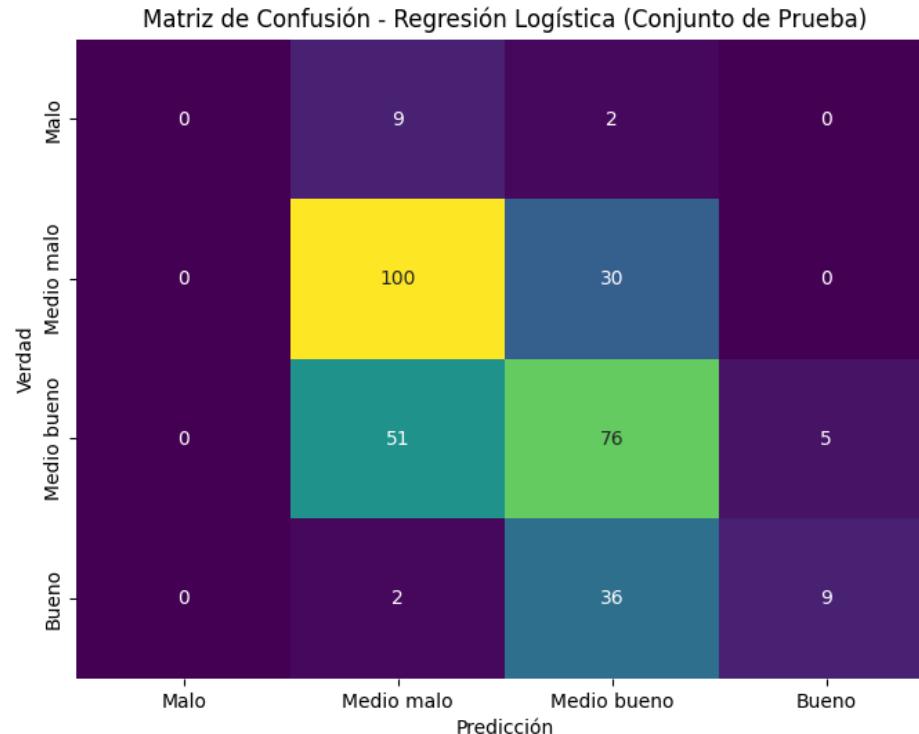
Precisión (precision) en el conjunto de prueba: 0.5629

Recall (sensitividad) en el conjunto de prueba: 0.5781

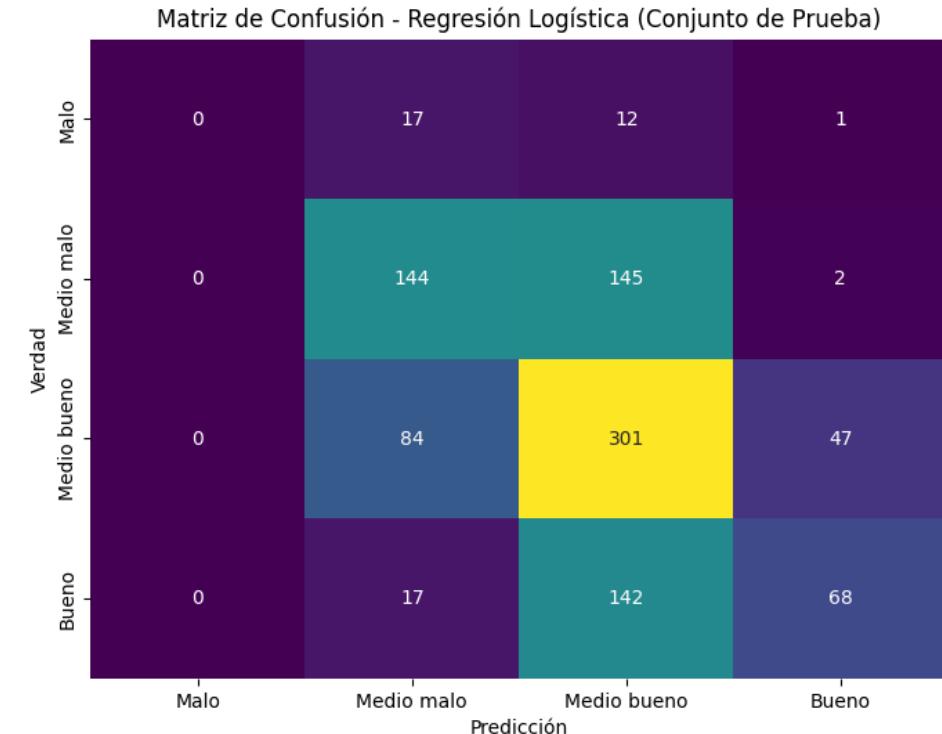
F1-Score en el conjunto de prueba: 0.5488

# Regresión Logística

White

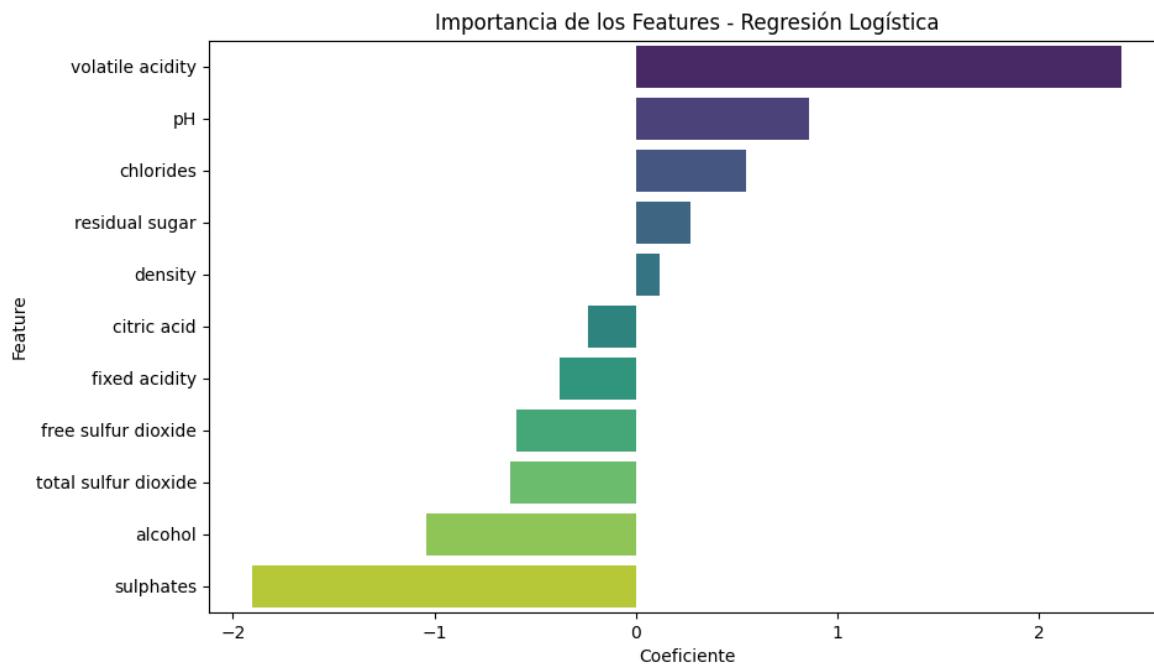


Red

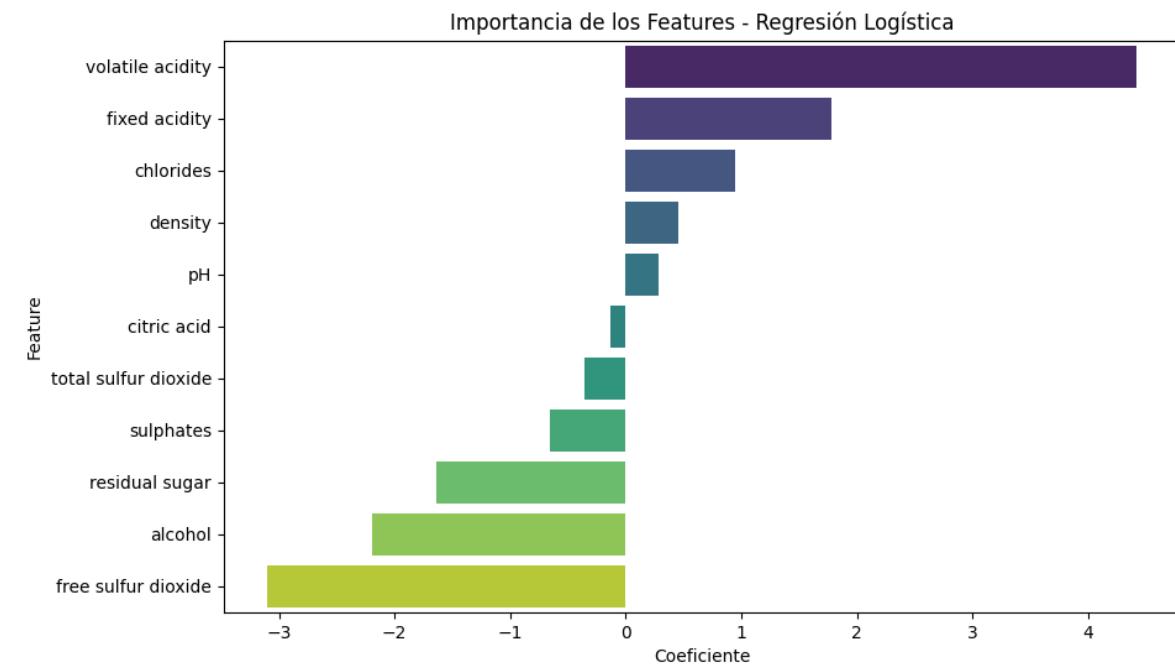


# Regresión Logística

White



Red



# ElasticNet



White

```
Fitting 5 folds for each of 25 candidates, totalling 125 fits
Mejores hiperparámetros encontrados: {'C': 10, 'l1_ratio': 0.5}
Informe de clasificación en el conjunto de prueba (Elastic Net):
```

	precision	recall	f1-score	support
0	0.33	0.03	0.06	30
1	0.57	0.52	0.54	291
2	0.51	0.70	0.59	432
3	0.60	0.32	0.42	227
accuracy			0.54	980
macro avg	0.50	0.39	0.40	980
weighted avg	0.54	0.54	0.52	980

Precisión (accuracy) en el conjunto de prueba: 0.5367  
Precisión (precision) en el conjunto de prueba: 0.5425  
Recall (sensitividad) en el conjunto de prueba: 0.5367  
F1-Score en el conjunto de prueba: 0.5195

Red

```
Fitting 5 folds for each of 25 candidates, totalling 125 fits
Mejores hiperparámetros encontrados: {'C': 10, 'l1_ratio': 0.25}
Informe de clasificación en el conjunto de prueba (Elastic Net):
```

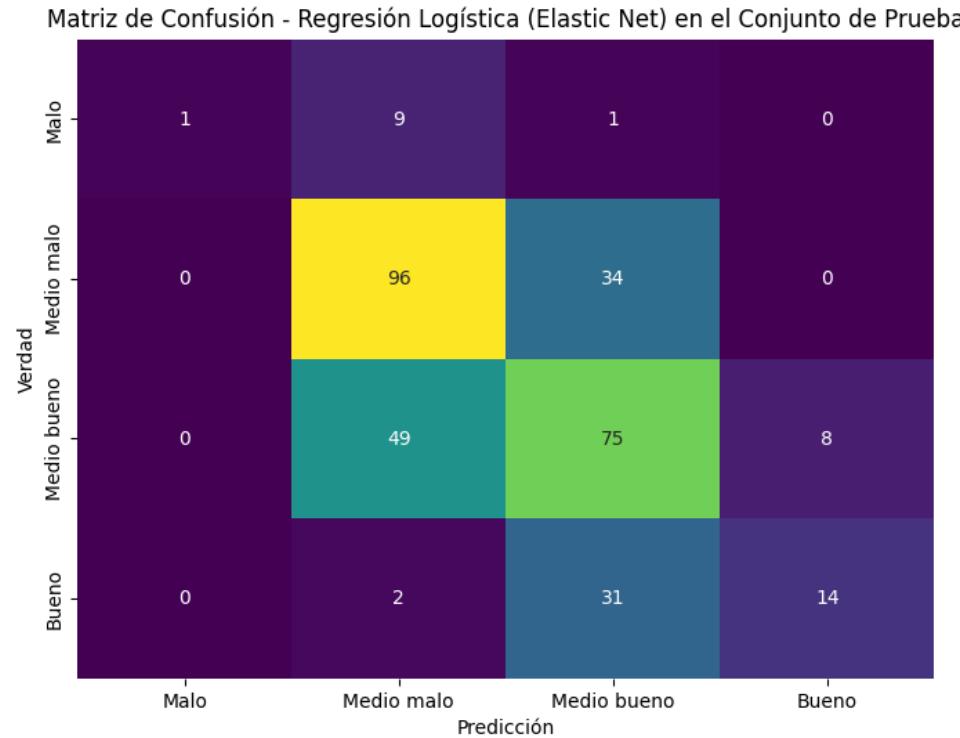
	precision	recall	f1-score	support
0	1.00	0.09	0.17	11
1	0.62	0.74	0.67	130
2	0.53	0.57	0.55	132
3	0.64	0.30	0.41	47
accuracy			0.58	320
macro avg	0.70	0.42	0.45	320
weighted avg	0.60	0.58	0.56	320

Precisión (accuracy) en el conjunto de prueba: 0.5813  
Precisión (precision) en el conjunto de prueba: 0.5973  
Recall (sensitividad) en el conjunto de prueba: 0.5813  
F1-Score en el conjunto de prueba: 0.5647

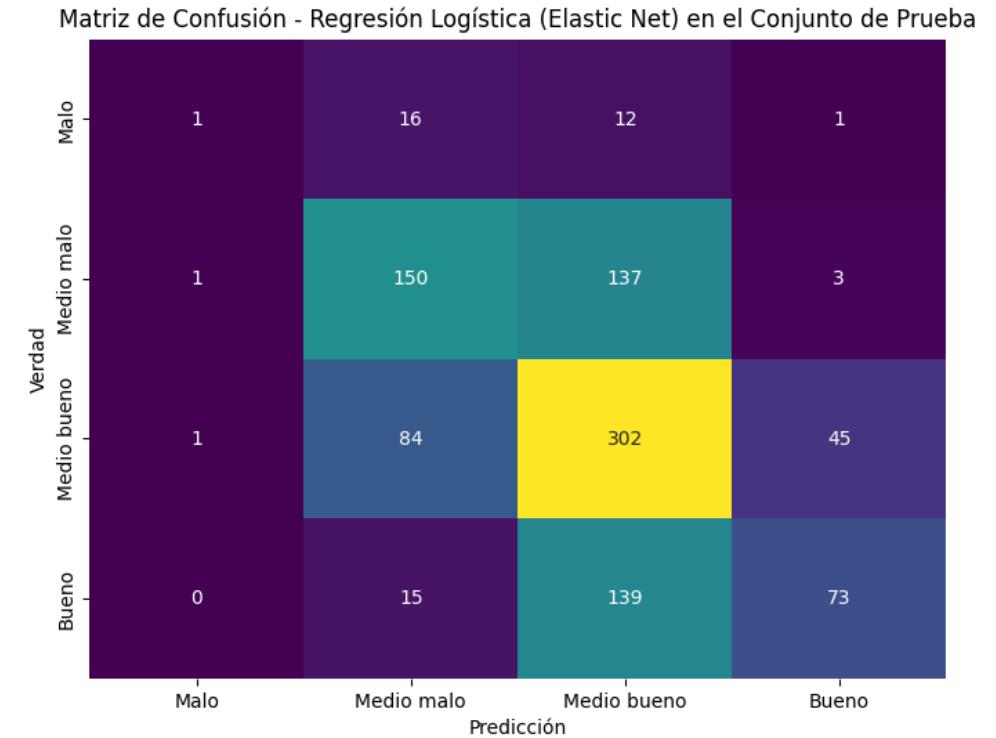
# ElasticNet



White

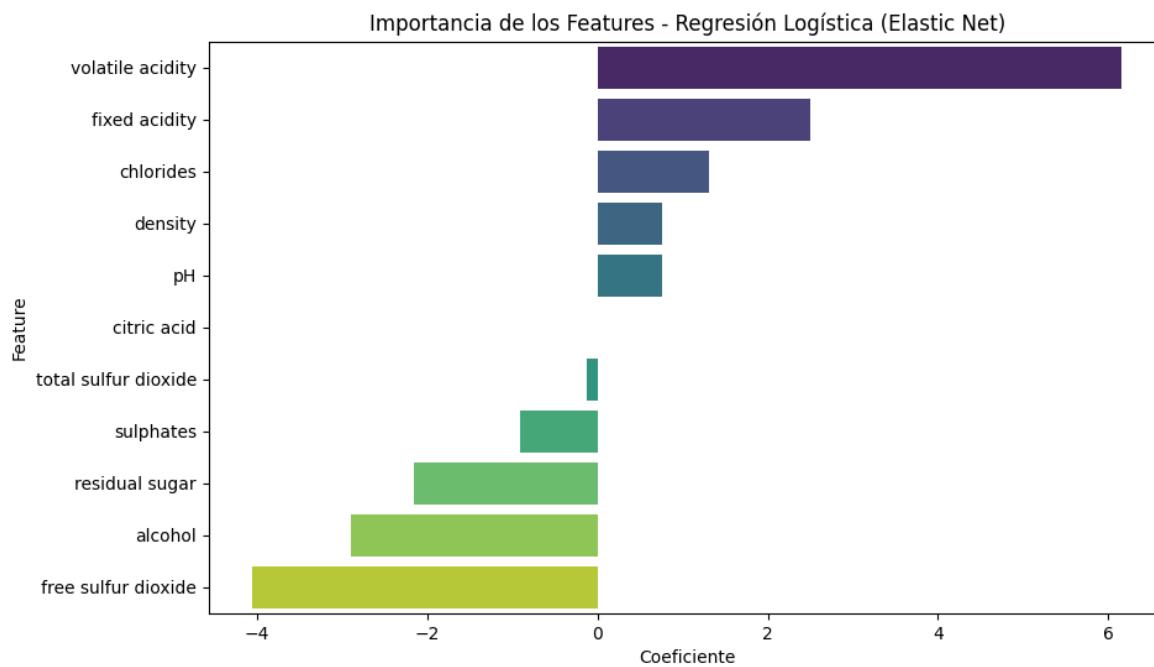


Red

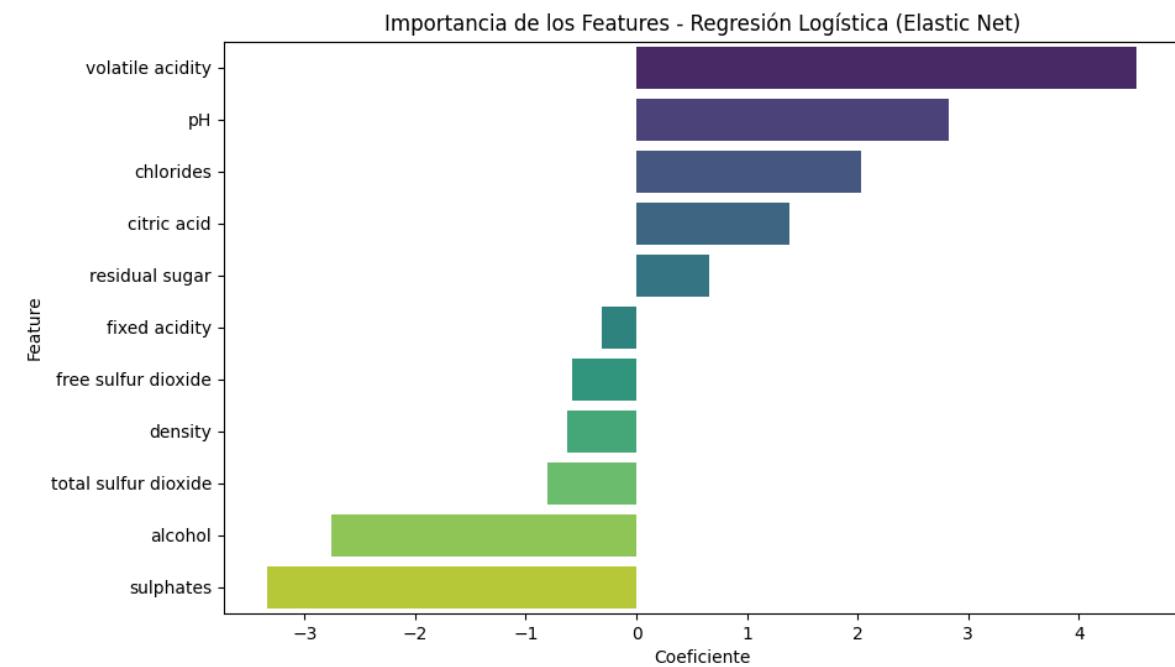


# ElasticNet

White



Red



# RandomForest

White

Red

Informe de clasificación en el conjunto de prueba (Random Forest):  
precision recall f1-score support

0	0.67	0.20	0.31	30
1	0.70	0.67	0.69	291
2	0.67	0.78	0.72	432
3	0.83	0.69	0.75	227
accuracy			0.71	980
macro avg	0.72	0.59	0.62	980
weighted avg	0.72	0.71	0.71	980

Precisión en el conjunto de prueba (accuracy): 0.7102  
Precisión en el conjunto de prueba (precision): 0.7165  
Recall (sensitividad) en el conjunto de prueba: 0.7102  
F1-Score en el conjunto de prueba: 0.7060

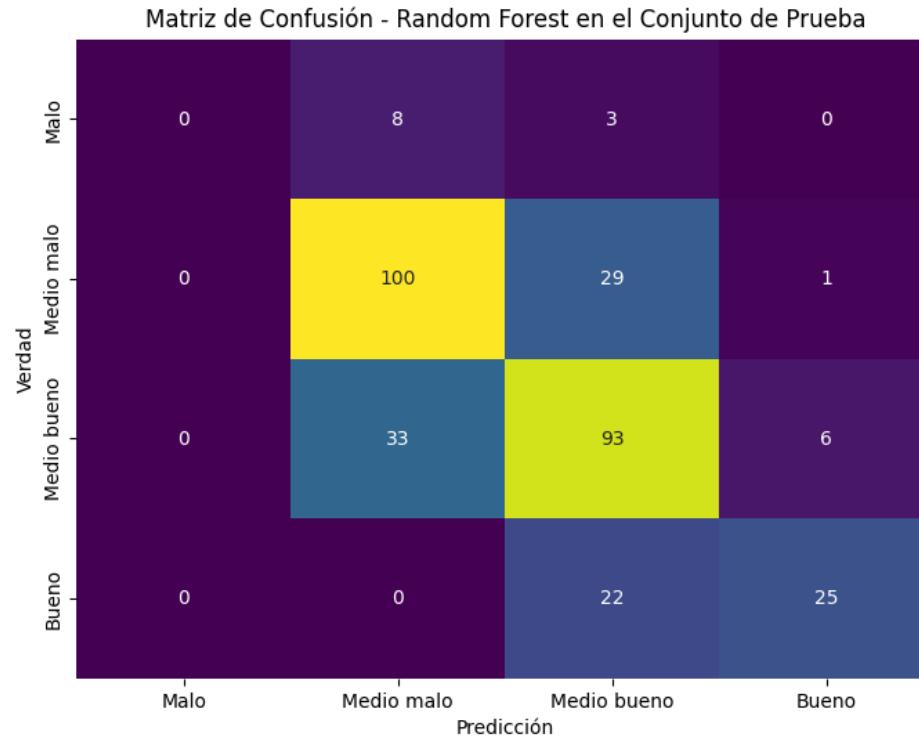
Informe de clasificación en el conjunto de prueba (Random Forest):  
precision recall f1-score support

0	0.00	0.00	0.00	11
1	0.70	0.77	0.74	130
2	0.62	0.68	0.65	132
3	0.79	0.57	0.67	47
accuracy			0.68	320
macro avg	0.53	0.51	0.51	320
weighted avg	0.66	0.68	0.67	320

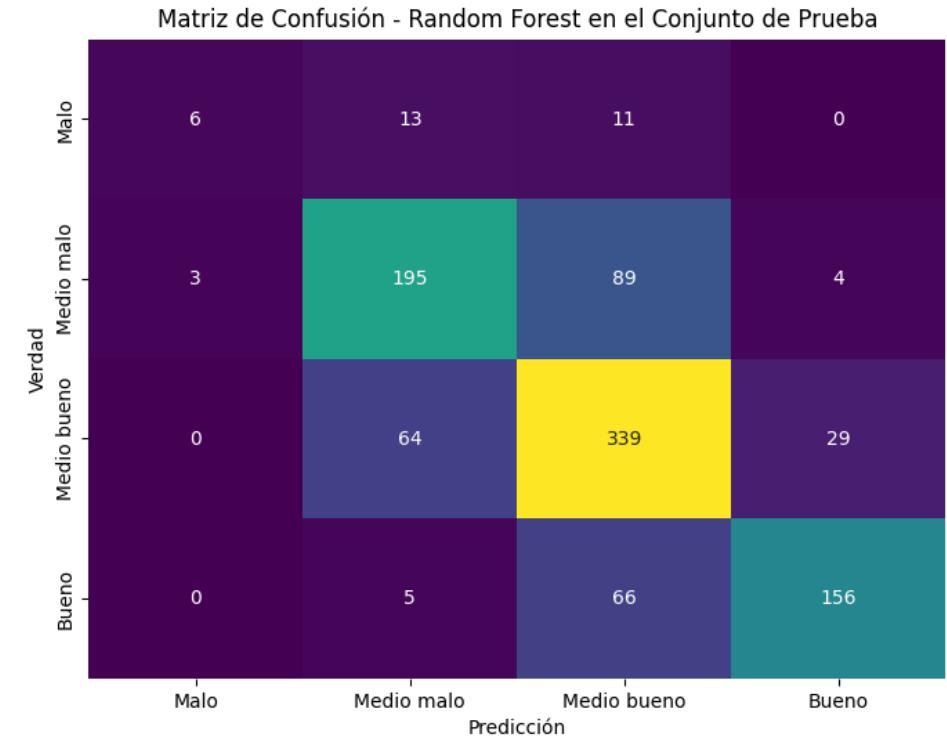
Precisión en el conjunto de prueba (accuracy): 0.6781  
Precisión en el conjunto de prueba (precision): 0.6605  
Recall (sensitividad) en el conjunto de prueba: 0.6781  
F1-Score en el conjunto de prueba: 0.6657

# RandomForest

White

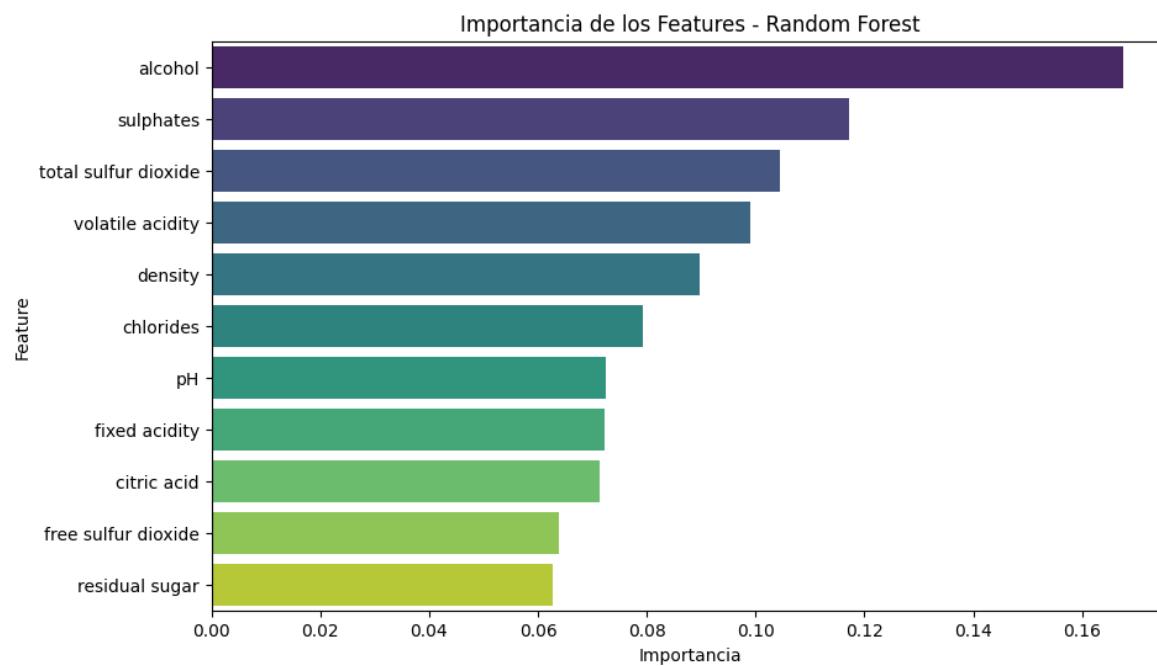


Red

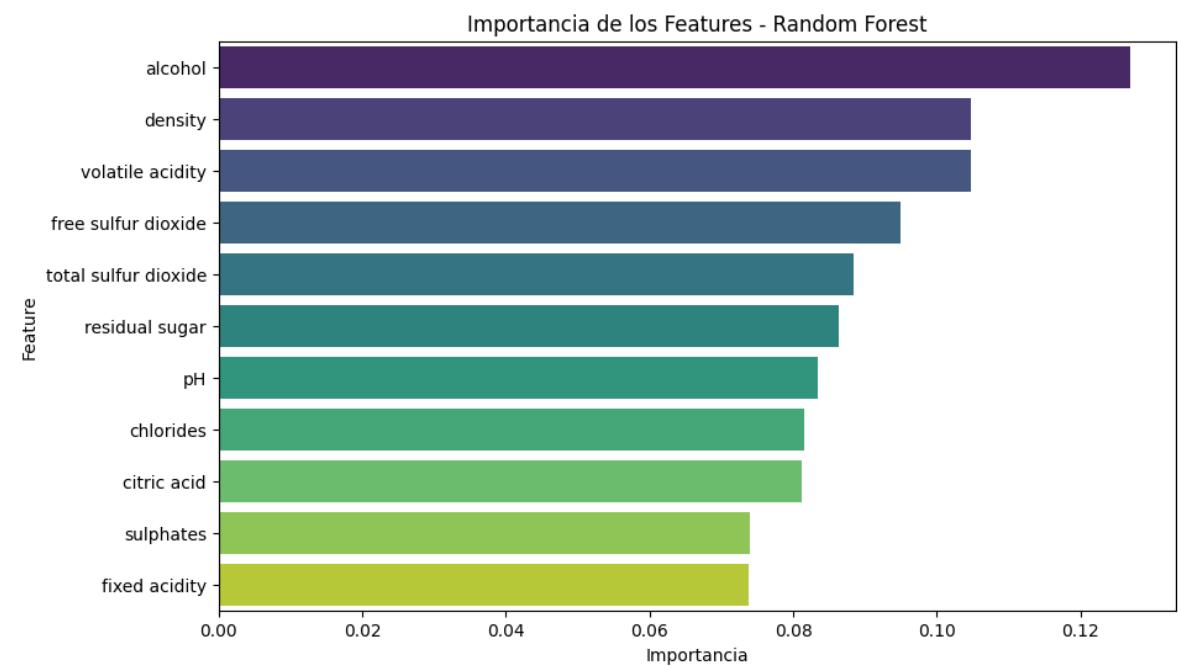


# RandomForest

White



Red



# SVM



White

```
Fitting 5 folds for each of 192 candidates, totalling 960 fits
Hiperparámetros óptimos encontrados por GridSearchCV:
{'C': 100, 'degree': 2, 'gamma': 1, 'kernel': 'rbf'}
```

Informe de clasificación en el conjunto de prueba (SVM):

	precision	recall	f1-score	support
0	0.62	0.27	0.37	30
1	0.62	0.58	0.60	291
2	0.56	0.71	0.63	432
3	0.69	0.45	0.54	227
accuracy			0.60	980
macro avg	0.62	0.50	0.54	980
weighted avg	0.61	0.60	0.59	980

Precisión (accuracy) en el conjunto de prueba: 0.6000  
Precisión (precision) en el conjunto de prueba: 0.6120  
Recall (sensitividad) en el conjunto de prueba: 0.6000  
F1-Score en el conjunto de prueba: 0.5941

Red

```
Fitting 5 folds for each of 192 candidates, totalling 960 fits
Hiperparámetros óptimos encontrados por GridSearchCV:
{'C': 100, 'degree': 2, 'gamma': 1, 'kernel': 'rbf'}
```

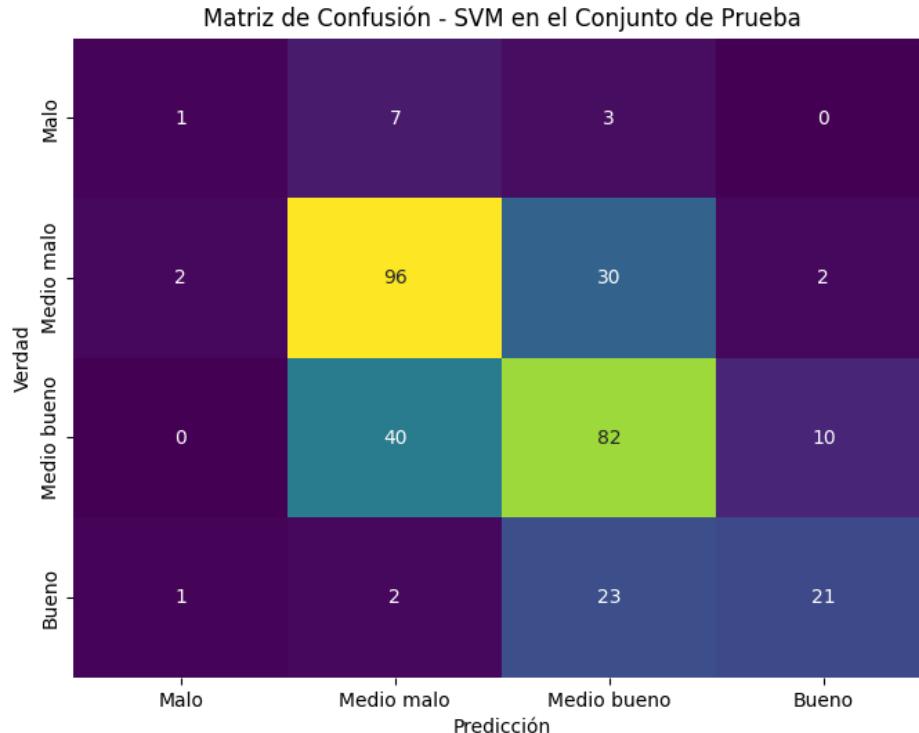
Informe de clasificación en el conjunto de prueba (SVM):

	precision	recall	f1-score	support
0	0.25	0.09	0.13	11
1	0.66	0.74	0.70	130
2	0.59	0.62	0.61	132
3	0.64	0.45	0.53	47
accuracy			0.62	320
macro avg	0.54	0.47	0.49	320
weighted avg	0.62	0.62	0.62	320

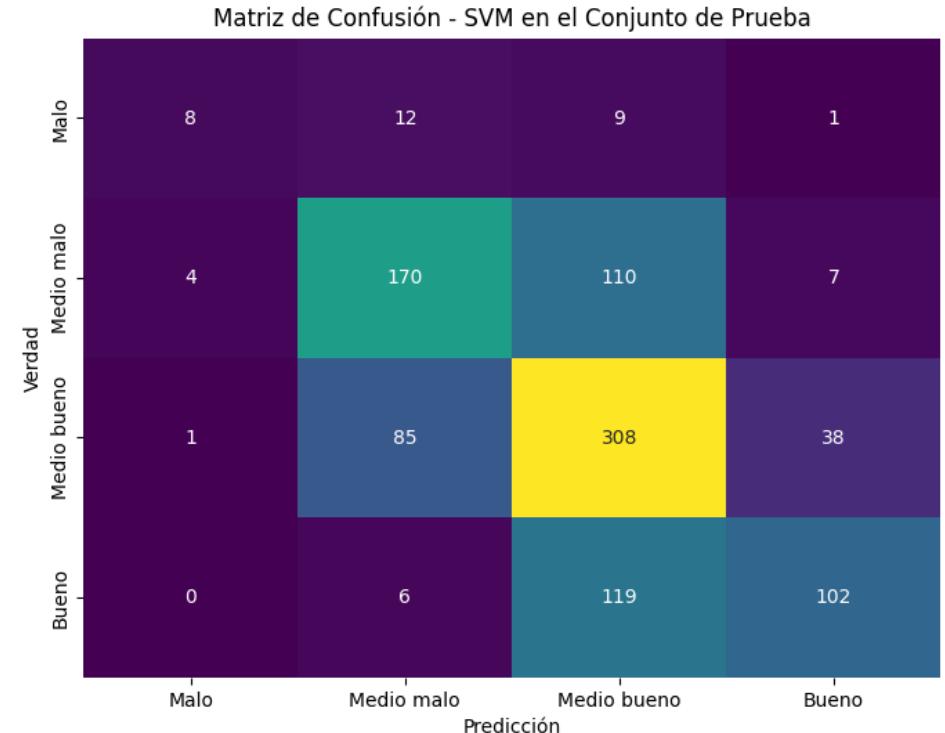
Precisión (accuracy) en el conjunto de prueba: 0.6250  
Precisión (precision) en el conjunto de prueba: 0.6161  
Recall (sensitividad) en el conjunto de prueba: 0.6250  
F1-Score en el conjunto de prueba: 0.6159

# SVM

White

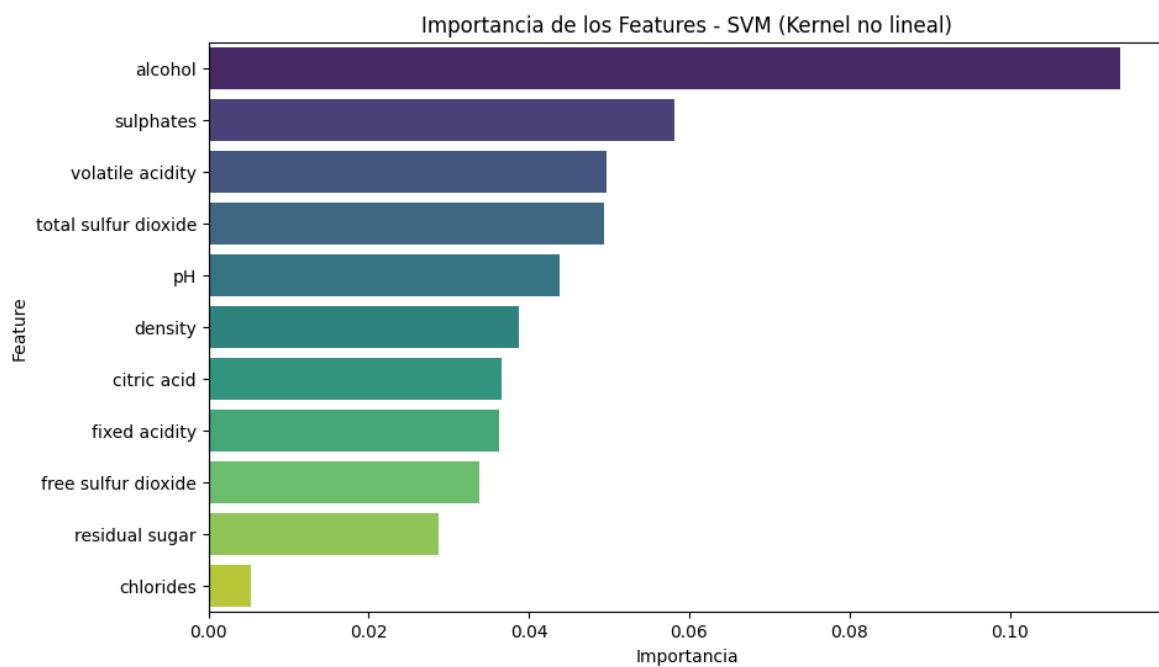


Red

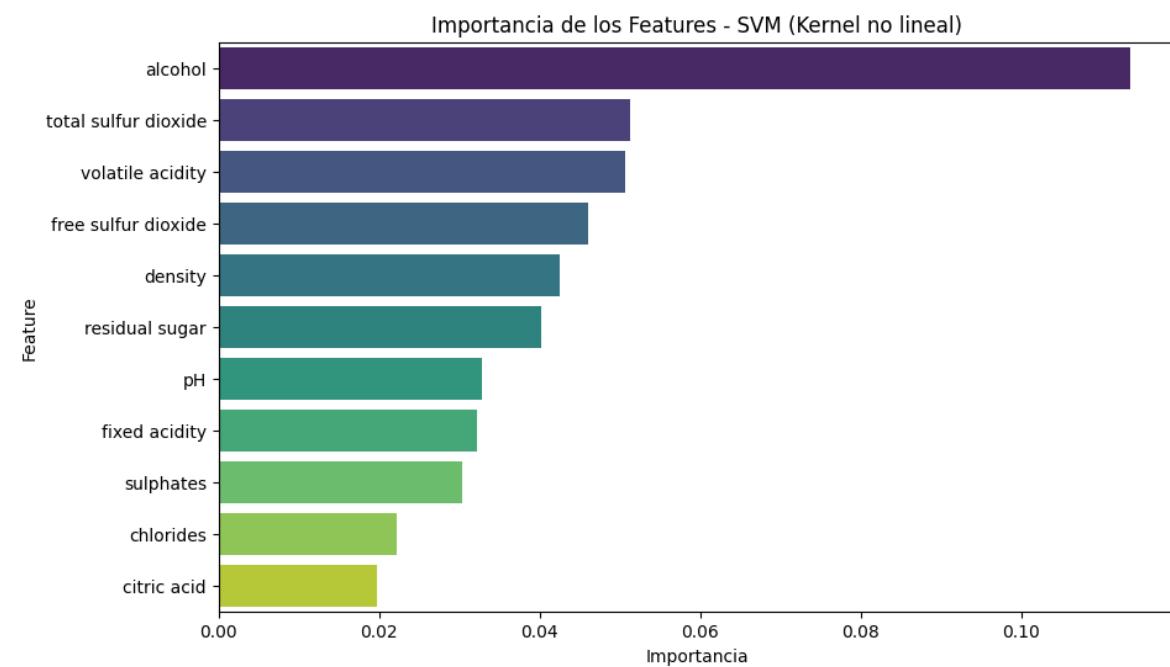


# SVM

White



Red



# XGBoost

White

Informe de clasificación en el conjunto de prueba (XGBoost):				
	precision	recall	f1-score	support
0	0.65	0.43	0.52	30
1	0.71	0.69	0.70	291
2	0.68	0.76	0.72	432
3	0.78	0.67	0.73	227
accuracy			0.71	980
macro avg	0.71	0.64	0.67	980
weighted avg	0.71	0.71	0.71	980

Precisión (accuracy) en el conjunto de prueba: 0.7092  
Precisión (precision) en el conjunto de prueba: 0.7126  
Recall (sensitividad) en el conjunto de prueba: 0.7092  
F1-Score en el conjunto de prueba: 0.7082

Red

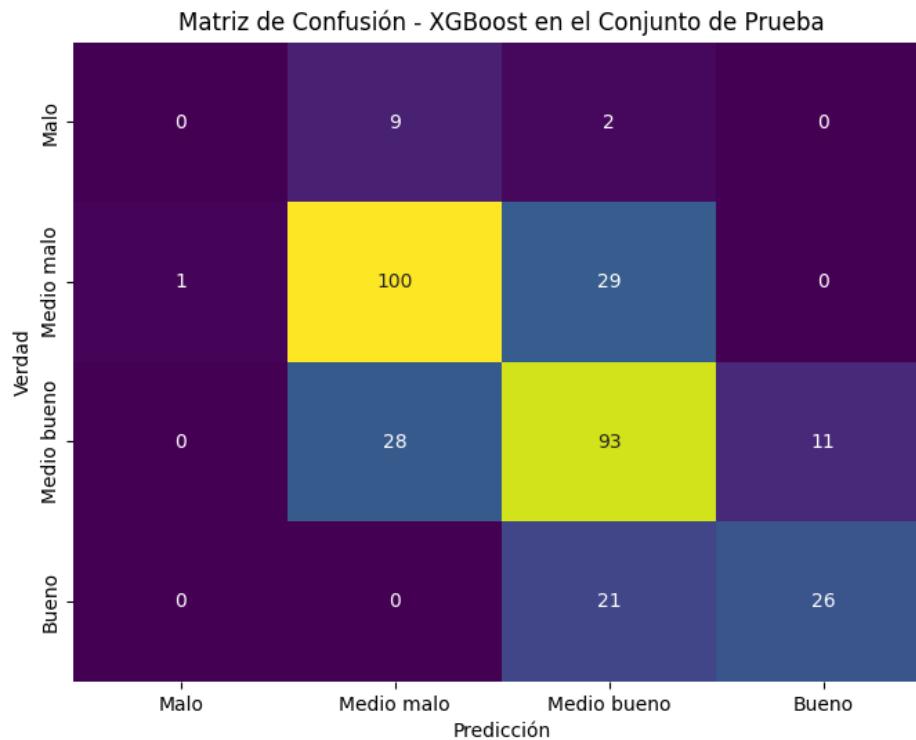
Informe de clasificación en el conjunto de prueba (XGBoost):				
	precision	recall	f1-score	support
0	0.00	0.00	0.00	11
1	0.73	0.77	0.75	130
2	0.64	0.70	0.67	132
3	0.70	0.55	0.62	47
accuracy			0.68	320
macro avg	0.52	0.51	0.51	320
weighted avg	0.66	0.68	0.67	320

Precisión (accuracy) en el conjunto de prueba: 0.6844  
Precisión (precision) en el conjunto de prueba: 0.6643  
Recall (sensitividad) en el conjunto de prueba: 0.6844  
F1-Score en el conjunto de prueba: 0.6722

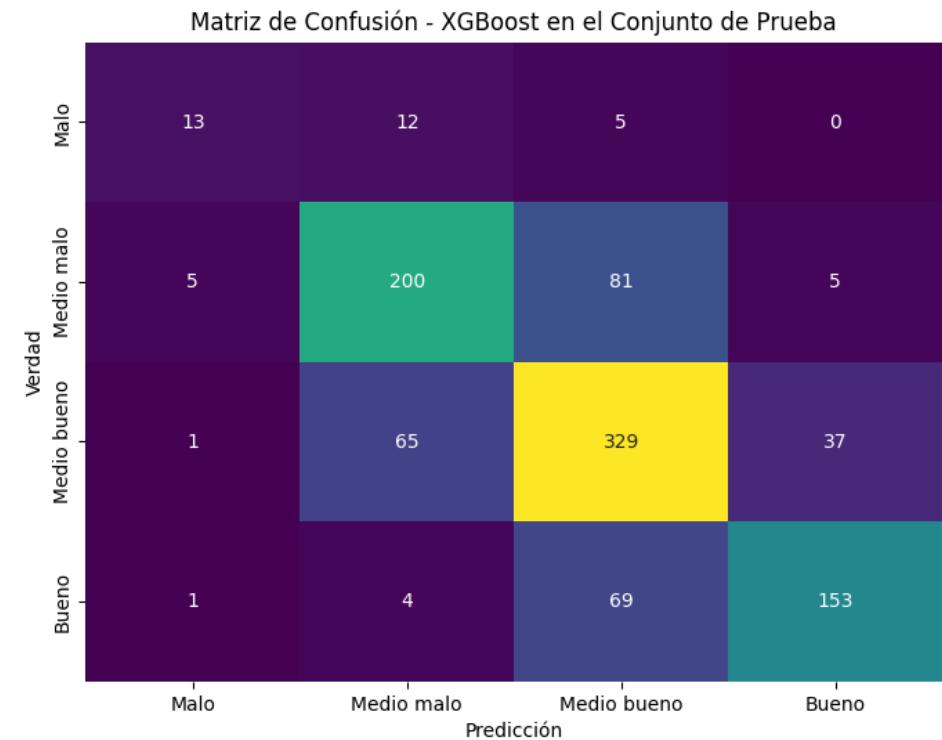
# XGBoost



White



Red

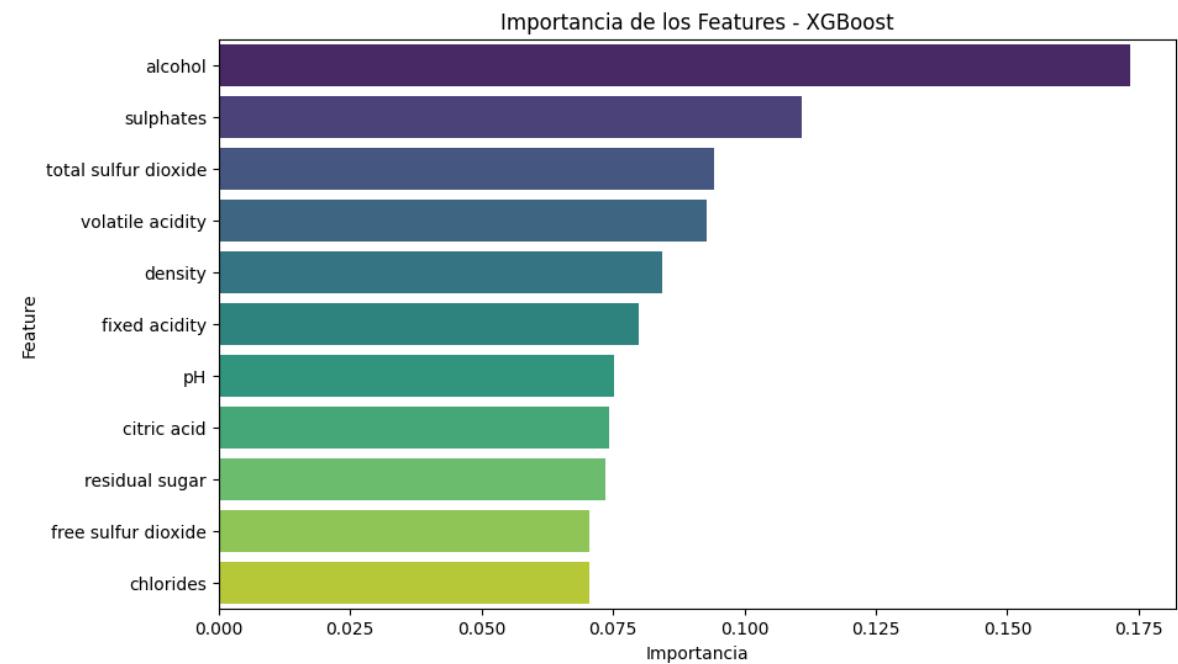
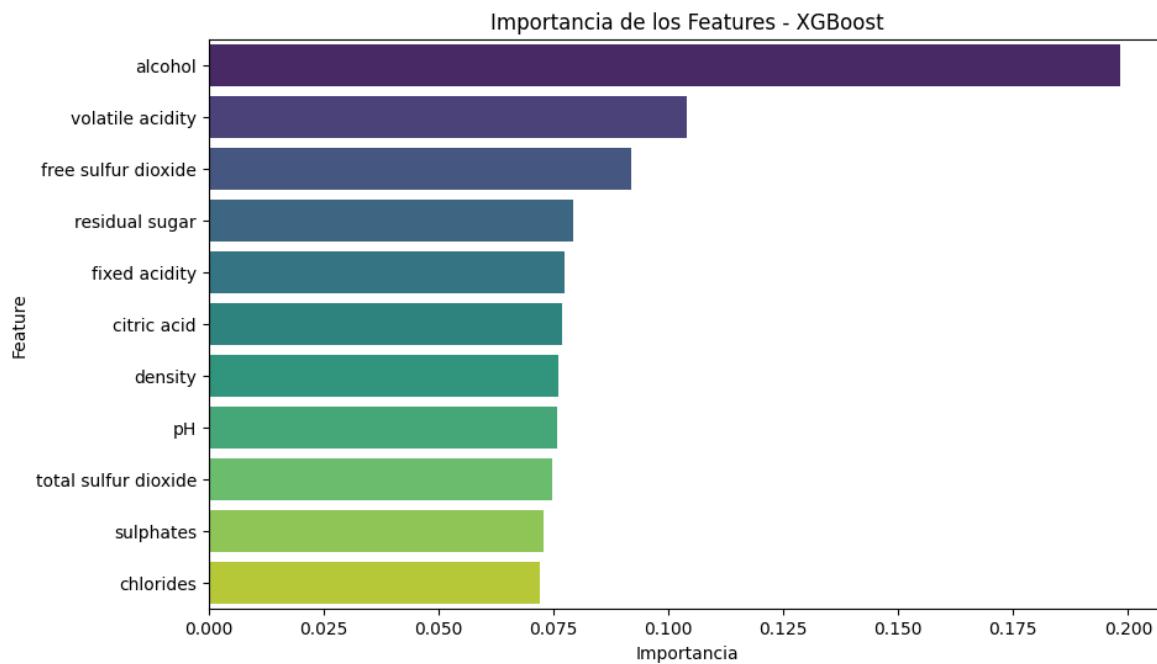


# XGBoost



White

Red



# Deep Learning



White

Informe de clasificación en el conjunto de prueba (Deep Learning):  
precision recall f1-score support

0	0.53	0.27	0.36	30
1	0.64	0.53	0.58	291
2	0.55	0.70	0.62	432
3	0.60	0.47	0.52	227

accuracy			0.58	980
macro avg	0.58	0.49	0.52	980
weighted avg	0.59	0.58	0.58	980

Precisión (accuracy) en el conjunto de prueba: 0.5816  
Precisión (precision) en el conjunto de prueba: 0.5878  
Recall (sensitividad) en el conjunto de prueba: 0.5816  
F1-Score en el conjunto de prueba: 0.5763

Red

Informe de clasificación en el conjunto de prueba (Deep Learning):  
precision recall f1-score support

0	0.00	0.00	0.00	11
1	0.58	0.78	0.66	130
2	0.53	0.46	0.49	132
3	0.56	0.32	0.41	47

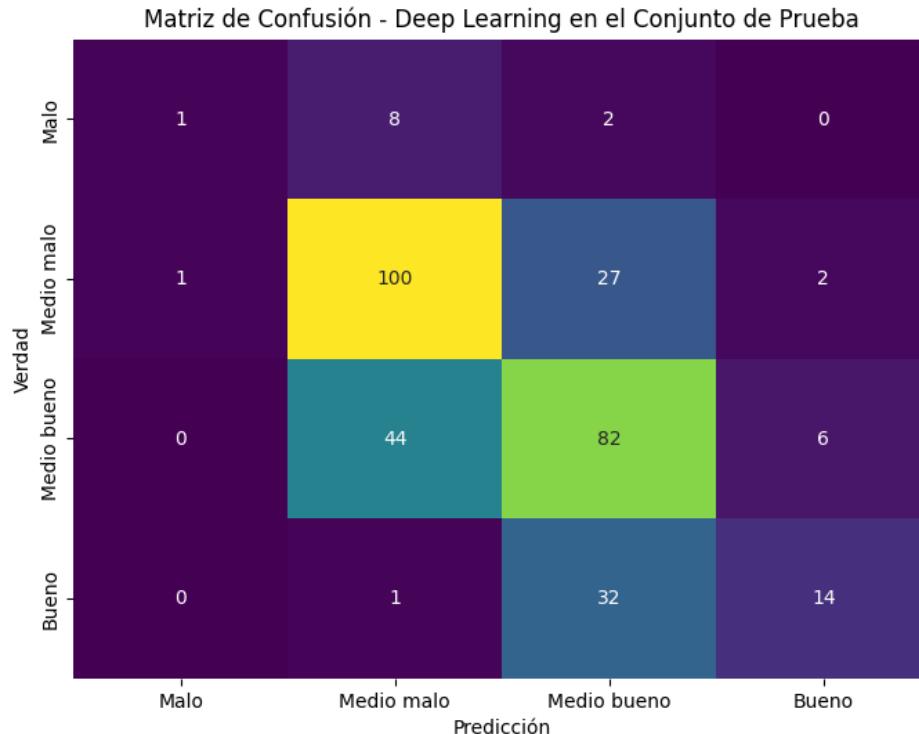
accuracy			0.56	320
macro avg	0.41	0.39	0.39	320
weighted avg	0.53	0.56	0.53	320

Precisión (accuracy) en el conjunto de prueba: 0.5563  
Precisión (precision) en el conjunto de prueba: 0.5326  
Recall (sensitividad) en el conjunto de prueba: 0.5563  
F1-Score en el conjunto de prueba: 0.5324

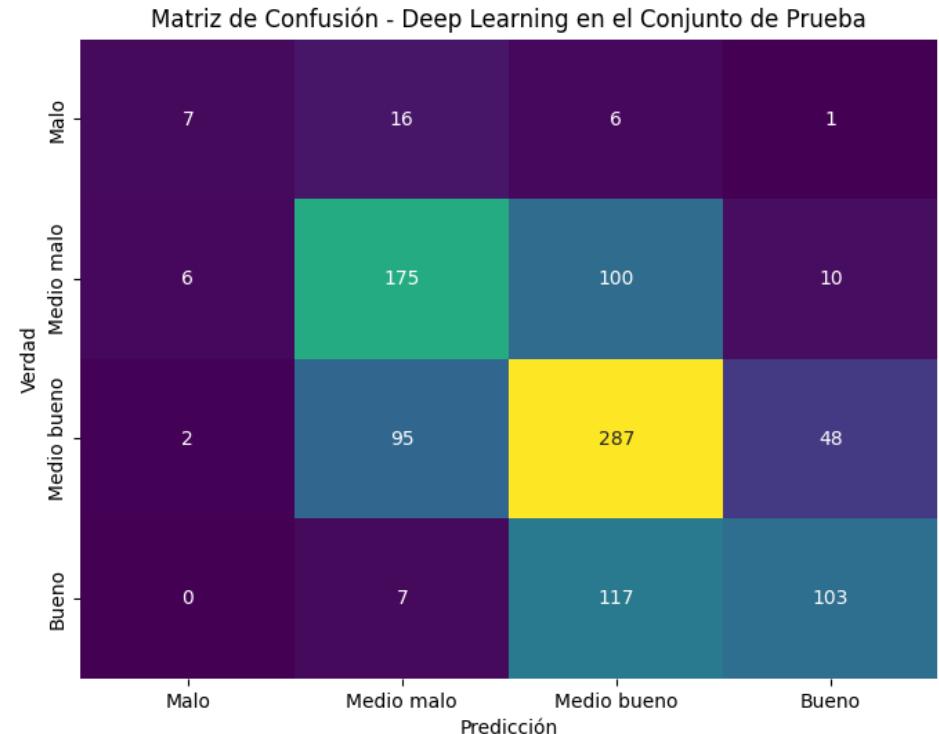
# Deep Learning



White

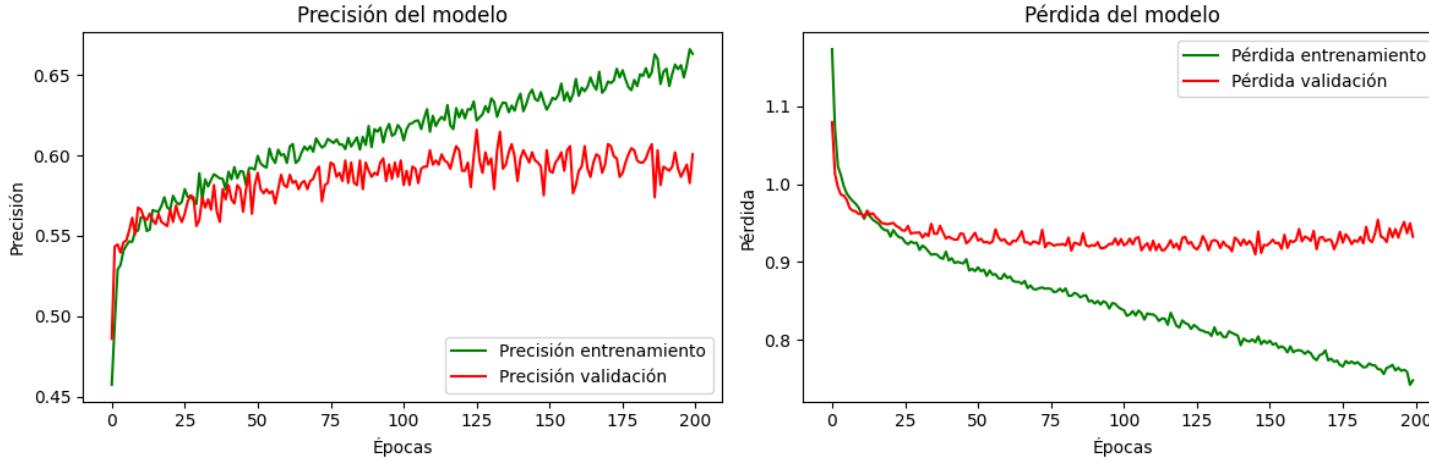


Red



# Deep Learning

White



Red

