Parcial 1 – Caso Viñedo de los Alpes

"Al entregar la solución de este parcial, yo, XXXX con código YYYY me comprometo a no conversar durante el desarrollo de este examen con ninguna persona que no sea el profesor del curso, sobre aspectos relacionados con el parcial; tampoco utilizaré algún medio de comunicación por voz, texto o intercambio de archivos, para consultar o compartir con otros, información sobre el tema del parcial. Soy consciente y acepto las consecuencias que acarreará para mi desempeño académico cometer fraude en este parcial"

Carga de los datos y librerías necesarias

In [1]:
```
!pip install imbalanced-learn
```

```
Requirement already satisfied: imbalanced-learn in /opt/anaconda3/lib/python3.8/site-
packages (0.8.0)
Requirement already satisfied: numpy>=1.13.3 in /opt/anaconda3/lib/python3.8/site-pac
kages (from imbalanced-learn) (1.20.1)
Requirement already satisfied: scikit-learn>=0.24 in /opt/anaconda3/lib/python3.8/sit
e-packages (from imbalanced-learn) (0.24.1)
Requirement already satisfied: scipy>=0.19.1 in /opt/anaconda3/lib/python3.8/site-pac
kages (from imbalanced-learn) (1.6.2)
Requirement already satisfied: joblib>=0.11 in /opt/anaconda3/lib/python3.8/site-pack
ages (from imbalanced-learn) (1.0.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in /opt/anaconda3/lib/python3.8/s
ite-packages (from scikit-learn>=0.24->imbalanced-learn) (2.1.0)
```

In [45]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn import tree

from sklearn.tree import DecisionTreeClassifier
from sklearn.preprocessing import LabelEncoder
from imblearn.over_sampling import SMOTE
# Para búsqueda de hiperparámetros
from sklearn.model_selection import GridSearchCV
# Para la validación cruzada
from sklearn.model_selection import KFold

# Metricas
from sklearn.metrics import confusion_matrix, classification_report, precision_score
from sklearn.metrics import plot_confusion_matrix

# q-q plots
import scipy.stats as stats
```

In [3]:
```
# Se cargan los datos.
df_original = pd.read_csv('vinosAlpes.csv', sep=";")
```

1. Contexto y objetivo

La eficiencia productiva es esencial para la competitividad de las empresas de fabricación de cualquier producto y, para conseguir buenos resultados, es imperativo contar con un sistema eficiente de control de calidad. Para el caso de la industria de vino, la certificación de calidad toma mucho tiempo, por lo que se ha planteado la idea de automatizar dicho proceso.

Con base en lo anterior, podemos establecer el siguiente objetivo: Desarrollar y programar un modelo de aprendizaje automático que sea capaz de predecir con alto nivel precisión la certificación de calidad de una producción de vino, tomando como entradas un conjunto de datos con información de viejos lotes de producción evaluados y certificados manualmente por expertos. Esto con el fin de que se pueda agilizar la tarea del control de calidad que actualmente es ineficiente.

Tarea y algoritmo

- Tarea seleccionada: Clasificación
- Algoritmo: para poder cumplir con el objetivo, se utilizará un algoritmo de Árbol de Clasificación, que se encargue de decidir si una producción de vino es de calidad o no
- Parámetros: los parámetros a considerar son el criterio de decisión en los nodos (gini o entropía), y la profundidad ma2xima del árbol

2. Perfilamiento de los datos

Revisamos la información sobre las columnas que existen, los tipos de datos y los valores nulos que contienen

In [4]:
```
print(df_original.shape)
df_original.info()
```

```
(2037, 14)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2037 entries, 0 to 2036
Data columns (total 14 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   acidezTotal           2037 non-null   float64
 1   acidezVolatil         2037 non-null   float64
 2   acidoCitrico          2037 non-null   float64
 3   azucaresResiduales    2037 non-null   float64
 4   cloruros              2037 non-null   float64
 5   dioxidoLibreSulfuro   2037 non-null   float64
 6   TotalDioxidoSulfurico 2037 non-null   float64
 7   densidad              2037 non-null   float64
 8   pH                    2037 non-null   float64
 9   sulfitos              2037 non-null   float64
 10  nivelCalidad          2037 non-null   int64  
 11  grdAlcohol            2035 non-null   float64
 12  tipoVino              1842 non-null   object 
 13  calificacionCalidad   2035 non-null   float64
dtypes: float64(12), int64(1), object(1)
memory usage: 222.9+ KB
```

Vemos un total de 2037 filas y 14 columnas. De estas, tenemos 13 numéricas y una categórica. La única columna categórica es la del tipo de vino. Es necesario codificar esta variable porque el algortimo solo toma valores numéricos.

Solo las últimas 3 columnas de la lista presentan algunos valores nulos. Es necesario quitarlos para podeer implementar correctamente el algortimo del árbol.

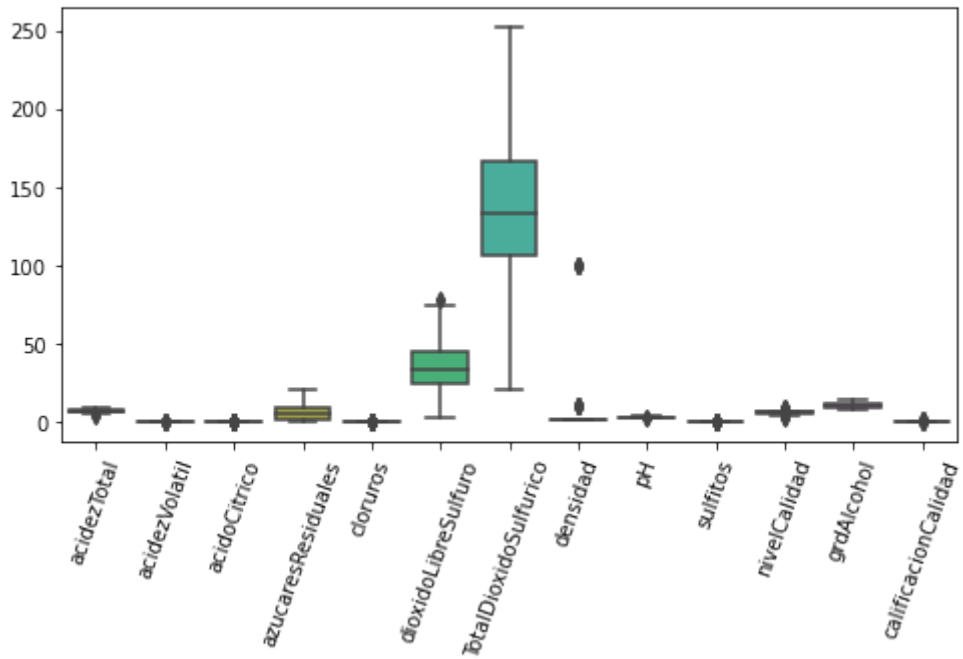
Ahora, obserrvamos la distribución de los datos:

In [5]:
```
df_original.describe()
```

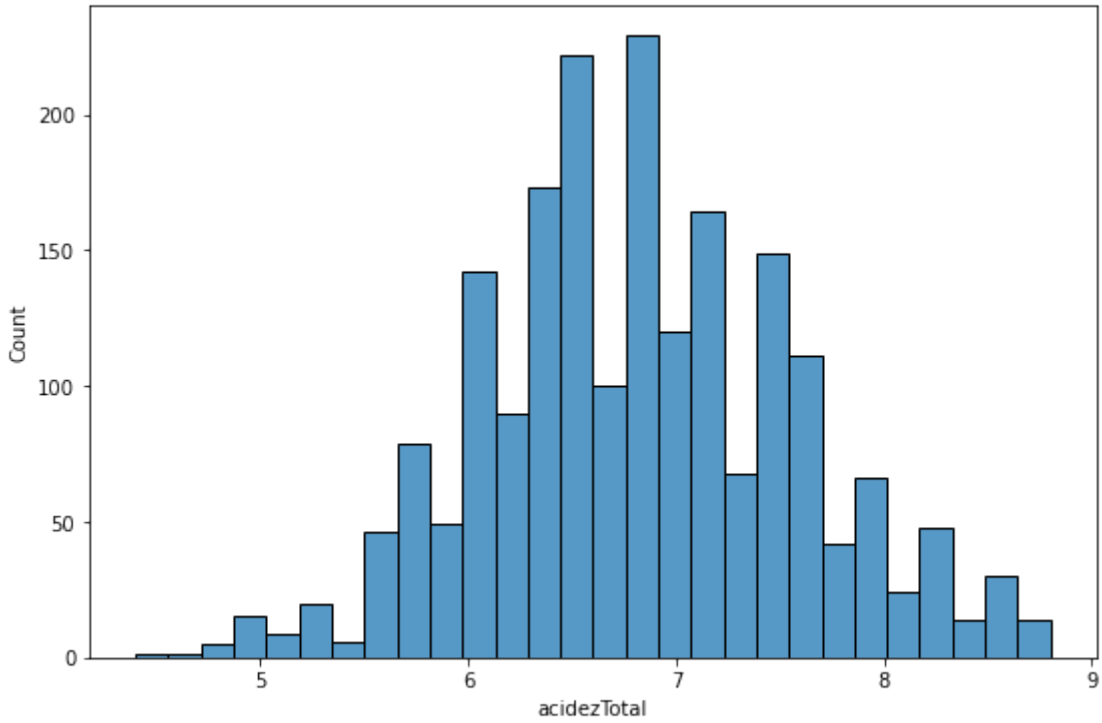
Out[5]:

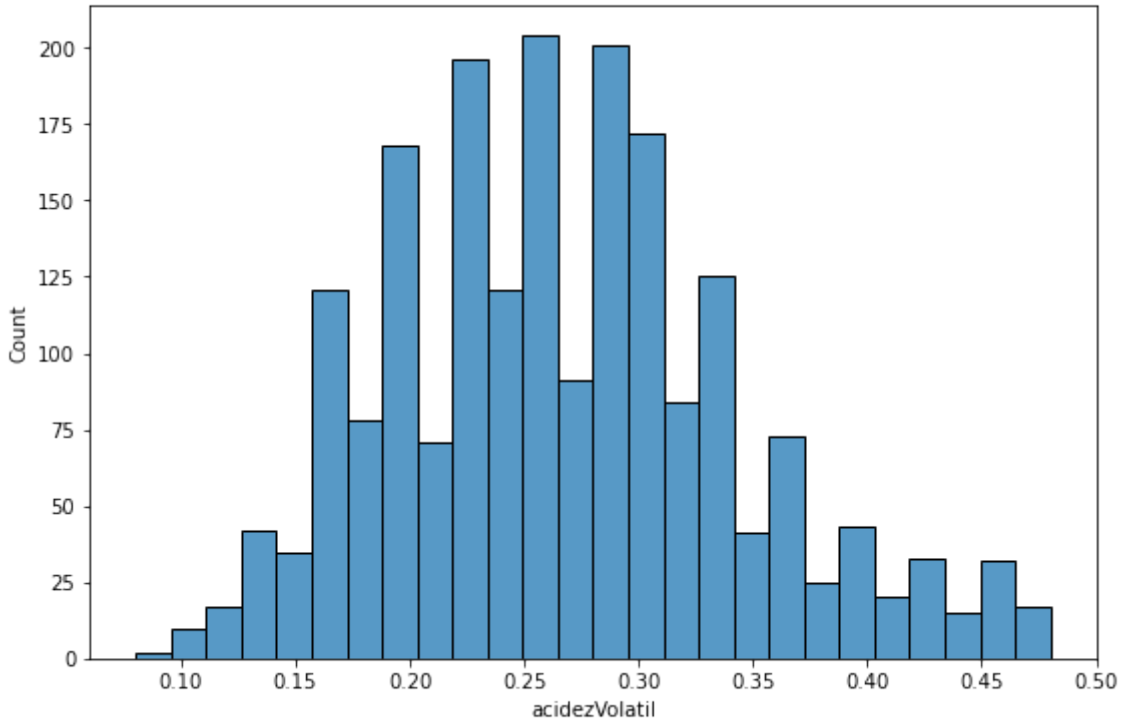
	acidezTotal	acidezVolatil	acidoCitrico	azucaresResiduales	cloruros	dioxidoLibreSulfuro
count	2037.000000	2037.000000	2037.000000	2037.000000	2037.000000	2037.000000
mean	6.825626	0.266564	0.323201	6.277590	0.042376	34.718949
std	0.753302	0.076768	0.094378	4.867284	0.010350	15.215444
min	4.400000	0.080000	0.000000	0.700000	0.010000	3.000000
25%	6.300000	0.210000	0.270000	1.700000	0.040000	24.000000
50%	6.800000	0.260000	0.310000	5.300000	0.040000	34.000000
75%	7.300000	0.310000	0.380000	9.400000	0.050000	45.000000
max	8.800000	0.480000	0.570000	20.800000	0.070000	78.000000

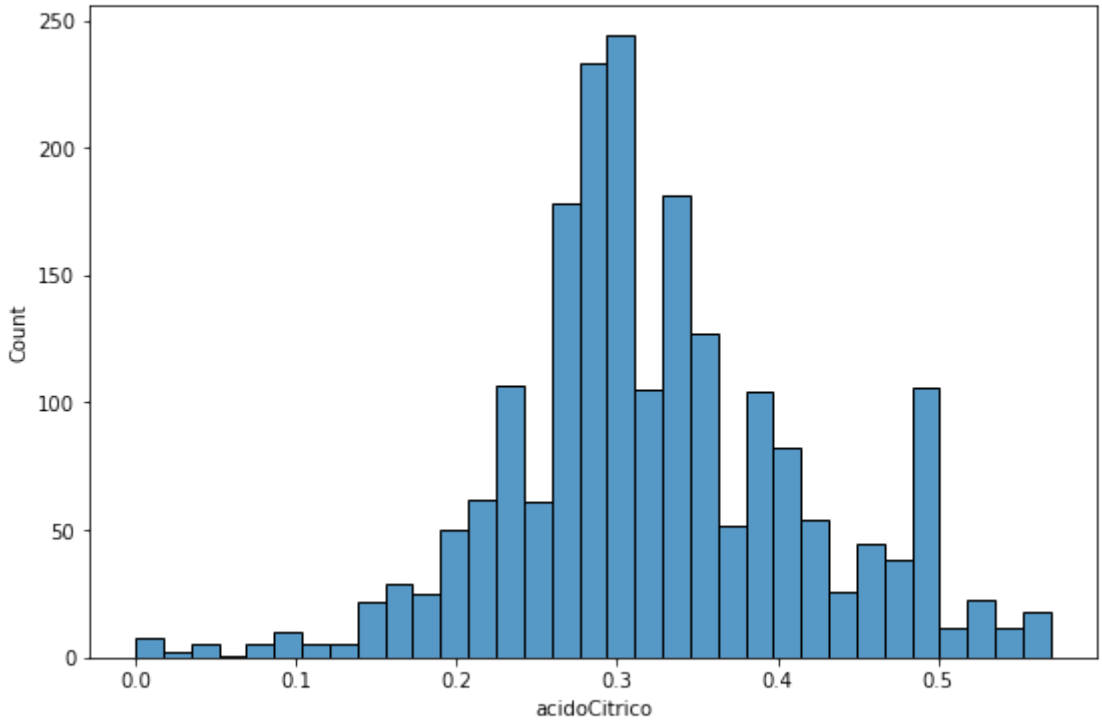
In [6]:
```
fig=plt.figure(figsize=(8,4))
ax = sns.boxplot(data=df_original[df_original.columns])
d = ax.set_xticklabels(ax.get_xticklabels(),rotation = 70)
```

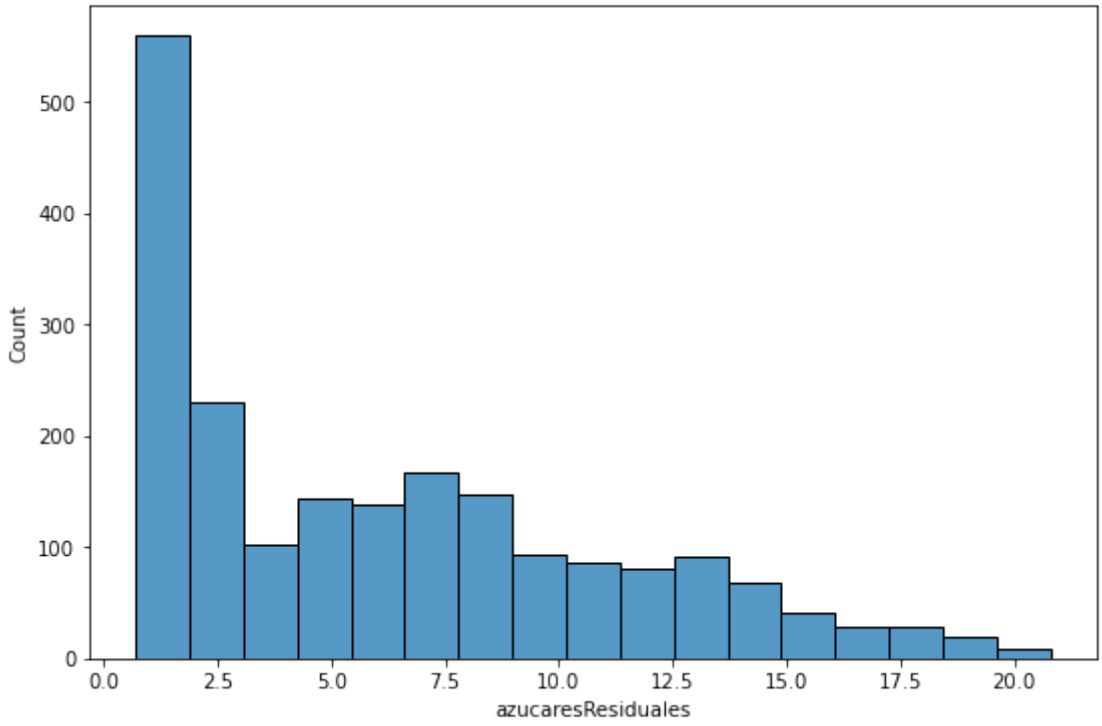


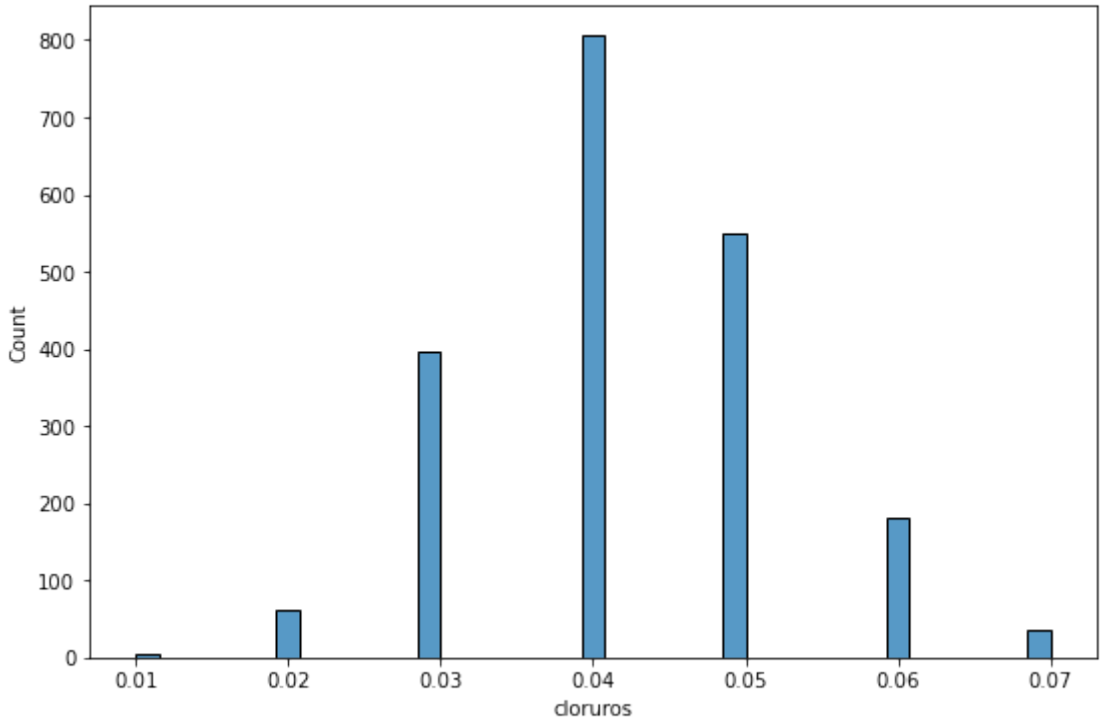
In [7]:
```
#Histogramas
df_num = df_original.select_dtypes(include = ['float','int'])
for col in df_num.columns:
    plt.figure(figsize=(9,6))
    plt.tight_layout()
    sns.histplot(df_num[col])
```

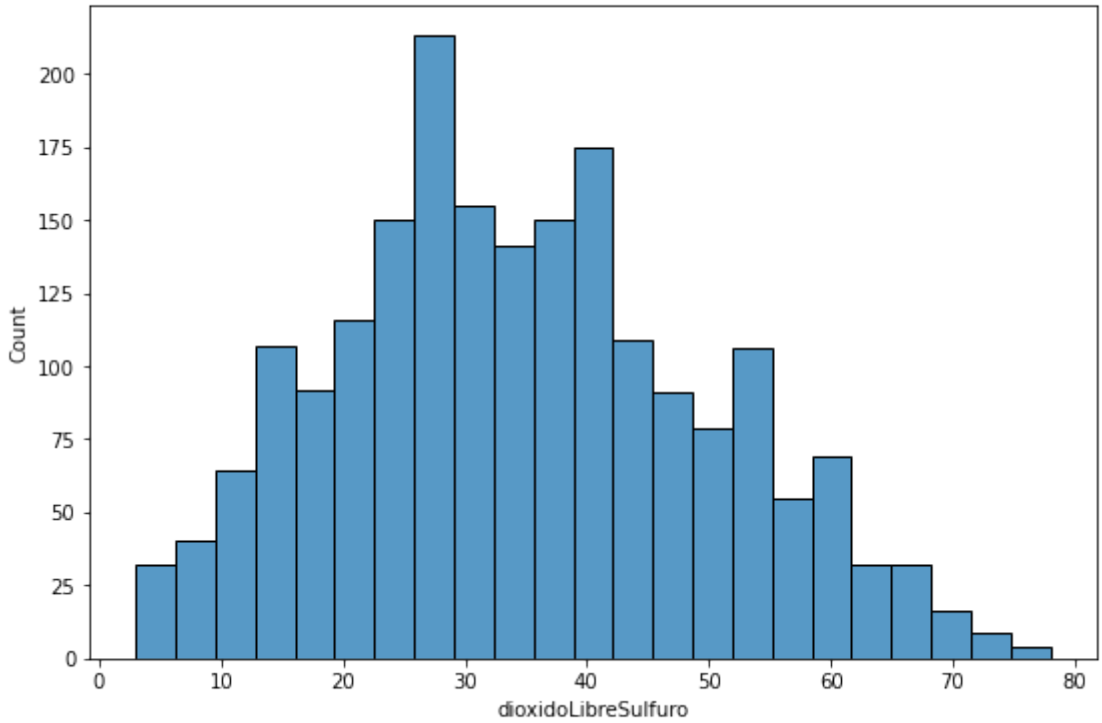

Count
200
150
100
50
0
5
6
7
8
9
acidezTotal

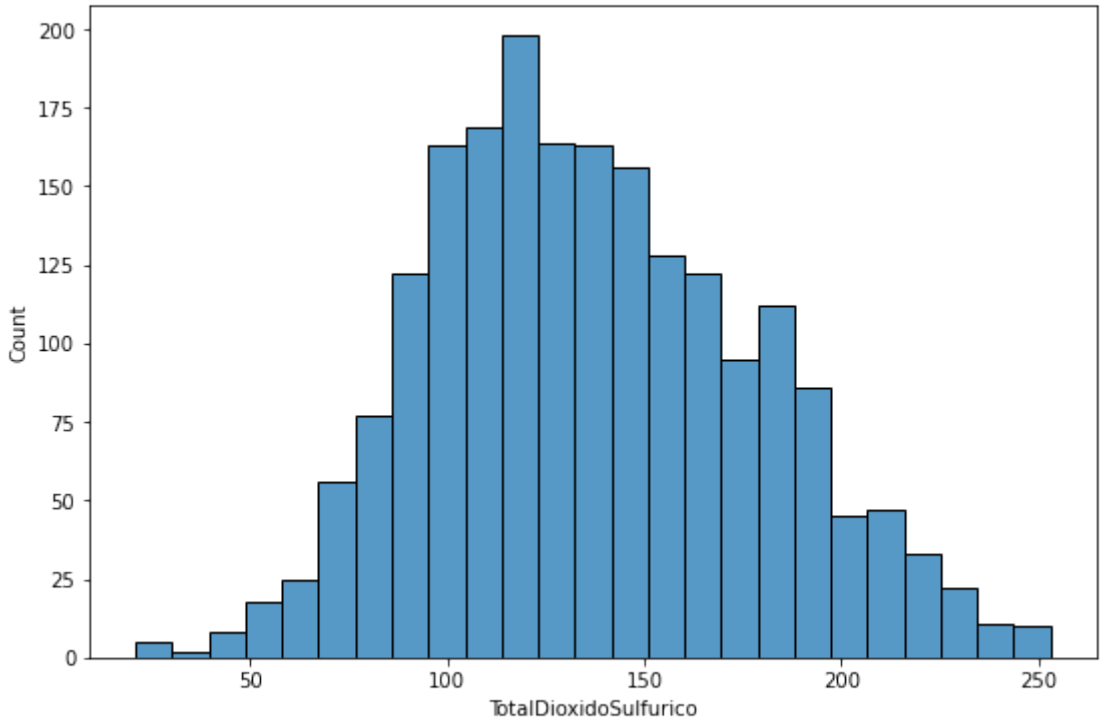

Count
200
175
150
125
100
75
50
25
0
0.10
0.15
0.20
0.25
0.30
0.35
0.40
0.45
0.50
acidezVolatil

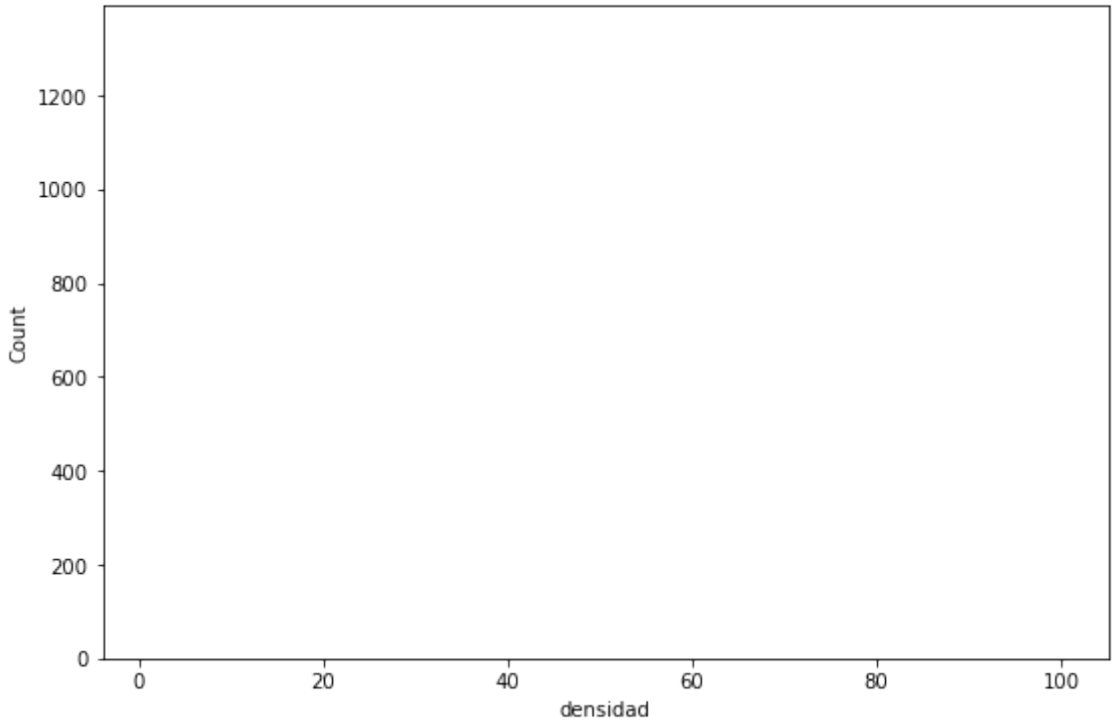

Count
250
200
150
100
50
0
0.0
0.1
0.2
0.3
0.4
0.5
acidoCitrico

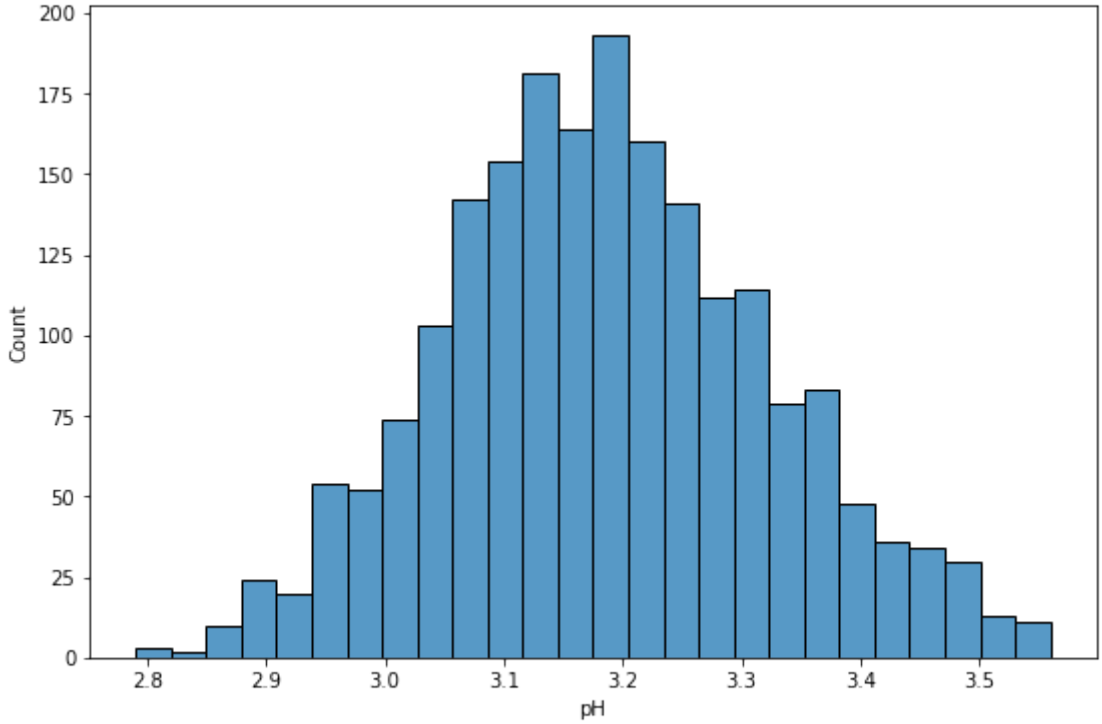

Count
500
400
300
200
100
0
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
azucaresResiduales

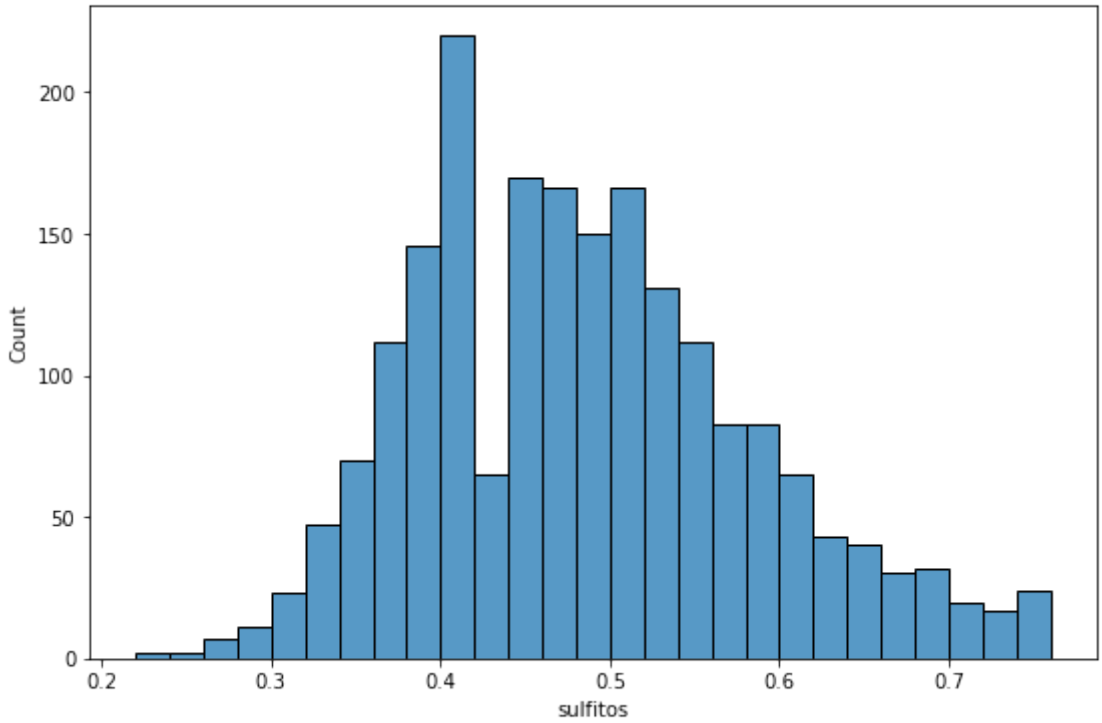

Count
800
700
600
500
400
300
200
100
0
0.01
0.02
0.03
0.04
0.05
0.06
0.07
cloruros

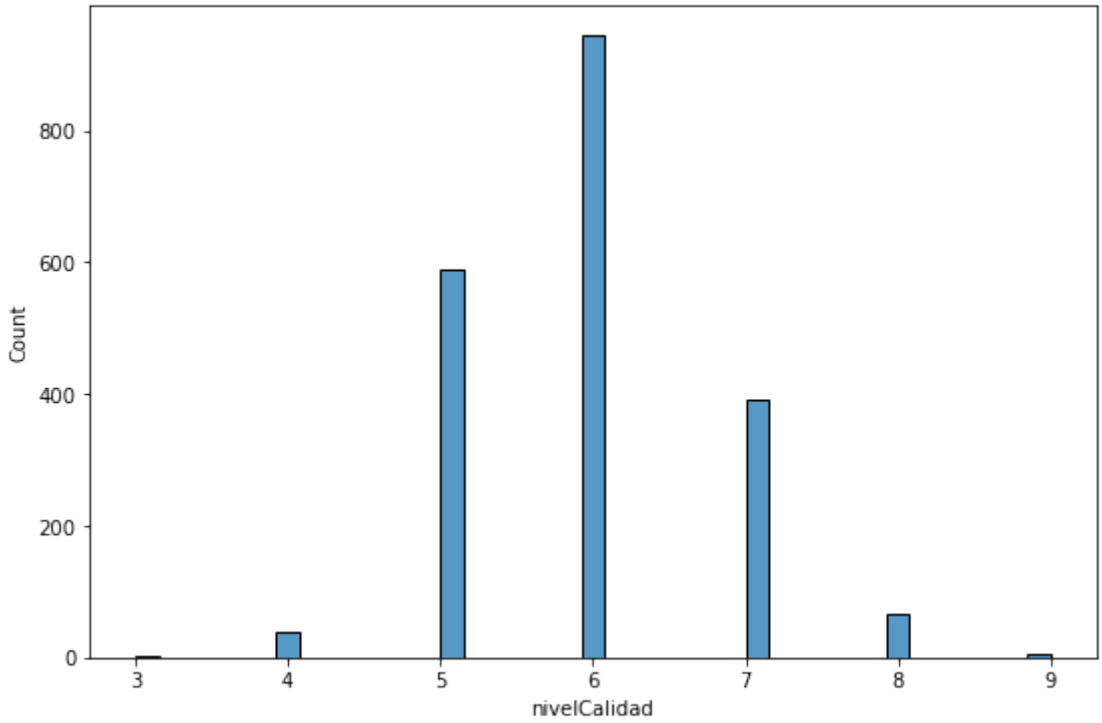

Count
200
175
150
125
100
75
50
25
0
0
10
20
30
40
50
60
70
80
dioxidoLibreSulfuro

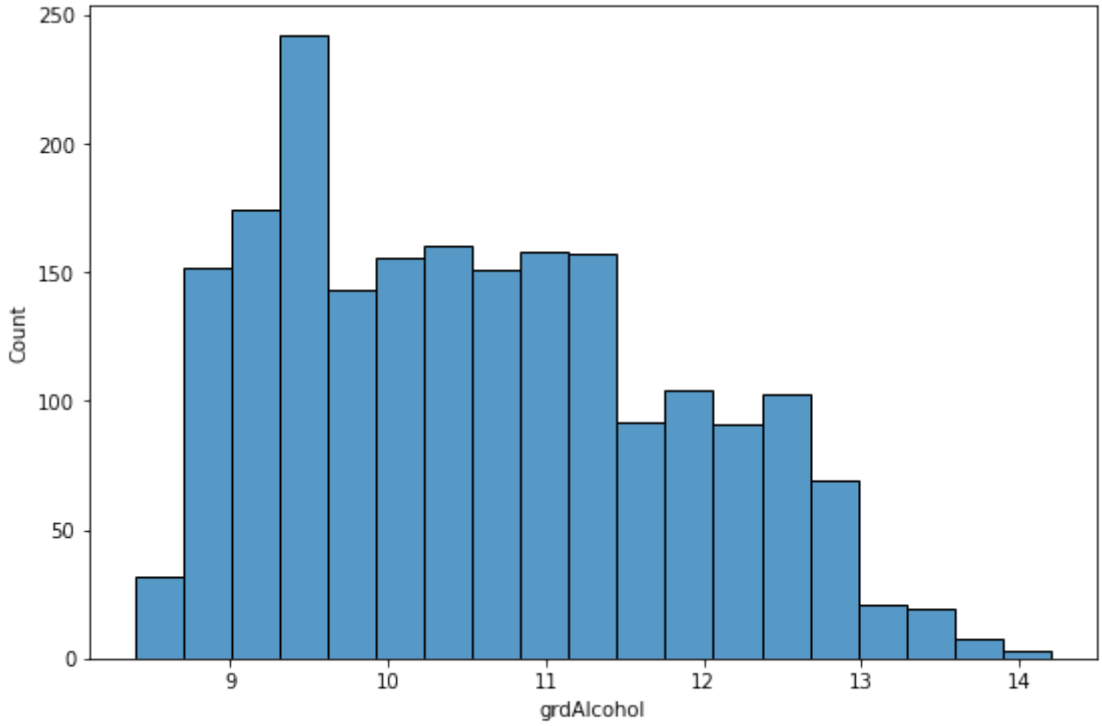

Count
TotalDioxidoSulfurico

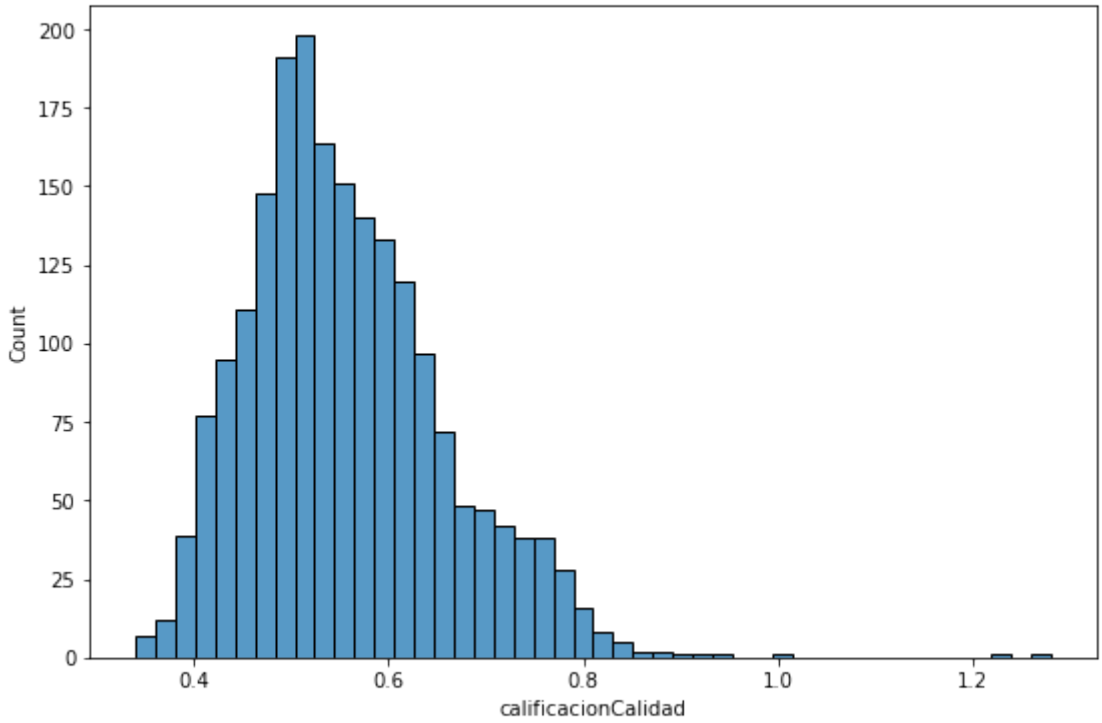

Count
densidad


Count
pH
0
25
50
75
100
125
150
175
200
2.8
2.9
3.0
3.1
3.2
3.3
3.4
3.5


Count
sulfitos
0
50
100
150
200
0.2
0.3
0.4
0.5
0.6
0.7


Count
800
600
400
200
0
3
4
5
6
7
8
9
nivelCalidad

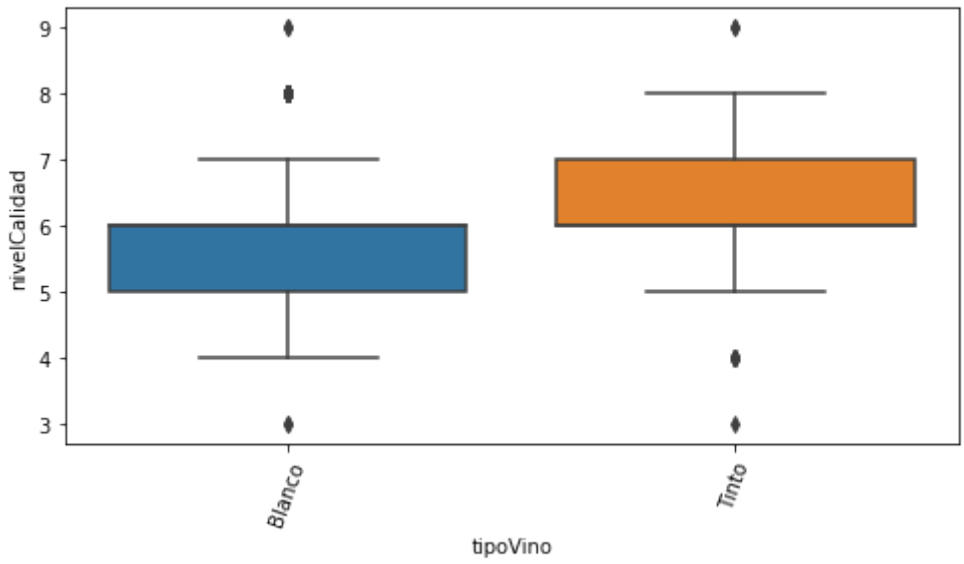

Count
250
200
150
100
50
0
9
10
11
12
13
14
grdAlcohol



La mayoría de columnas parece seguir una distribución normal, con exceepción de "azucaresResiduales". En general se puede ver que tienen una media similar, las únicas que tienen escalas más grandes son las relacionadas con el dioxido de sulfuro. Adicionalmente, encontramos una columna en los datos "densidad" que parece no aportar nada de información

Ahora revisamos la categórica:

In [8]:
```
fig=plt.figure(figsize=(8,4))
ax = sns.boxplot(x="tipoVino", y="nivelCalidad", data=df_original)
d = ax.set_xticklabels(ax.get_xticklabels(),rotation = 70)
```



Vemos que parece exisitir una diferencia en el nivel de calidad para cada tipo de vino. Por esto lo mantendremos en el modelo.

3. Limpieza y procesamiento de los datos

Es necesario dejar un conjunto de datos que sea utilizable por el algoritmo. En primer lugar seleccionamos las columnas a utilizar. Se decide sacar las columnas que aporten información redundnate o que no sea relevante para el caso de estudio. Entonces, se corta la columna "acidezTotal" ya es información similar a la dad por la columna "pH". De forma similar, quitamos la columna "calificacionCalidad" ya que existe la columna de "nivelCalidad" que nos ayudará a clasificar correctamente cada registro.

In [9]:
```
df_limpio = df_original.drop(["acidezTotal","calificacionCalidad"], axis=1)
```

Se eliminan los registros con valores faltantes/nulos

In [10]:
```
#Eliminar filas con valores nulos
df_limpio = df_limpio.dropna()
df_limpio
```

Out[10]:

	acidezVolatil	acidoCitrico	azucaresResiduales	cloruros	dioxidoLibreSulfuro	TotalDioxidoSulfu
0	0.33	0.32	11.1	0.04	25.0	11
1	0.27	0.29	12.2	0.04	59.0	19
2	0.30	0.51	13.6	0.05	40.0	16
3	0.38	0.27	7.5	0.04	24.0	16
5	0.20	0.38	7.9	0.05	30.0	14
...	...	...	...	...	...	
2031	0.33	0.20	1.8	0.03	49.0	15
2032	0.34	0.28	7.5	0.04	70.0	23
2033	0.19	0.31	14.5	0.04	39.0	19
2035	0.28	0.35	15.3	0.06	31.0	11
2036	0.22	0.28	1.3	0.03	26.0	10

1840 rows × 12 columns

In [11]:
```
df_limpio.reset_index(drop=True, inplace=True)
```

Codificando la variable categórica

Se codifica la variable tipoVino

In [12]:
```
#Se verifican errores de escritura en variables categóricas
df_limpio["tipoVino"].value_counts()
```

Out[12]:
```
Blanco    1392
Tinto      448
Name: tipoVino, dtype: int64
```

In [22]:
```
encoder = LabelEncoder()

data = df_limpio["tipoVino"]
```

```
encoded_vals = encoder.fit_transform(data)

df_enc = pd.DataFrame(encoded_vals, columns=["tipoVino"])

df_limpio = df_limpio.drop(["tipoVino"], axis=1)
df_limpio2 = pd.concat([df_limpio,df_enc], axis=1)
df_limpio2
```

Out[22]:

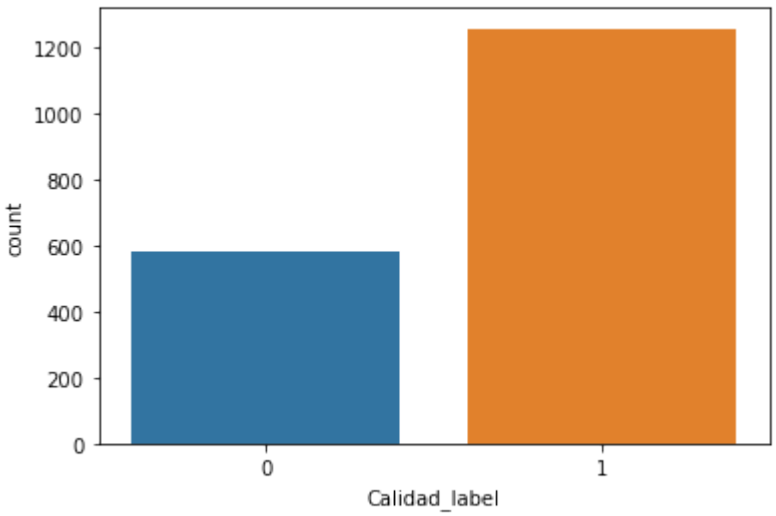
	acidezVolatil	acidoCitrico	azucaresResiduales	cloruros	dioxidoLibreSulfuro	TotalDioxidoSulfu
0	0.33	0.32	11.1	0.04	25.0	11
1	0.27	0.29	12.2	0.04	59.0	19
2	0.30	0.51	13.6	0.05	40.0	16
3	0.38	0.27	7.5	0.04	24.0	16
4	0.20	0.38	7.9	0.05	30.0	14
...	...	...	...	...	...	
1835	0.33	0.20	1.8	0.03	49.0	15
1836	0.34	0.28	7.5	0.04	70.0	23
1837	0.19	0.31	14.5	0.04	39.0	19
1838	0.28	0.35	15.3	0.06	31.0	11
1839	0.22	0.28	1.3	0.03	26.0	10

1840 rows × 12 columns

Como decidimos aplicar un clasificador, debemos crear una clase que defina si una producción de vino es de calidad o no. De acuerdo con el negocio, se clasifica como de calidad cuando el nivel de calidad es mayor a 5:

In [23]:

```
# Ahora definimos la función que nos va a permitir construir nuestra clase.
def label_calidad (row):
    if row['nivelCalidad'] > 5:
        return 1
    return 0
df_limpio2['Calidad_label']=df_limpio2.apply (lambda row: label_calidad(row), axis=1
ax = sns.countplot(x='Calidad_label', data=df_limpio2)
```



Finalmente, sacamos la columna nivelCalidad para que no afecte los criterios del algoritmo al decidir.

In [24]:
```
df_final = df_limpio2.drop(["nivelCalidad"], axis=1)
df_final.sample(6)
```

Out[24]:

	acidezVolatil	acidoCitrico	azucaresResiduales	cloruros	dioxidoLibreSulfuro	TotalDioxidoSulfu
583	0.37	0.30	6.20	0.04	49.0	13
919	0.26	0.34	6.40	0.05	36.0	16
313	0.24	0.33	1.10	0.05	28.0	15
1399	0.23	0.27	11.75	0.03	34.0	11
1325	0.17	0.34	2.00	0.04	38.0	11
113	0.23	0.30	14.90	0.05	33.0	11

4. Creación del modelo

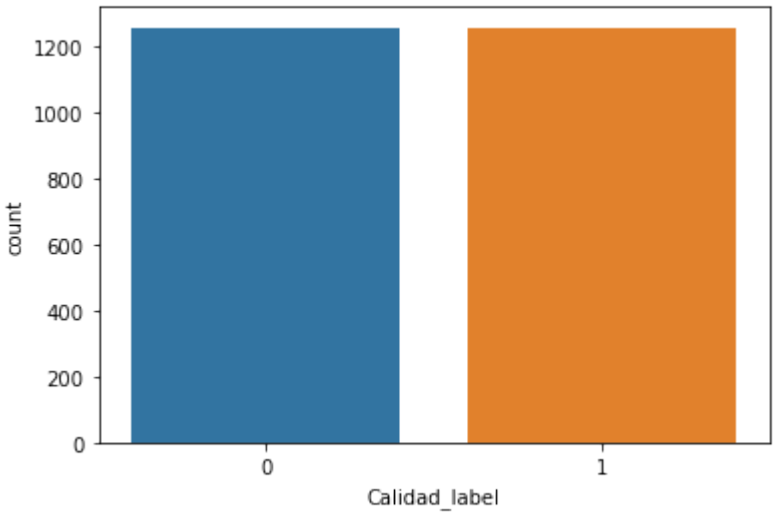
Se crea el modelo de árbol de decisión

In [25]:
```
X = df_final.drop(["Calidad_label"], axis=1)
y = df_final["Calidad_label"]
```

Se aplica la técnica SMOTE para balancear la clase objetivo

In [26]:
```
sm = SMOTE(random_state=0)

X_sm, Y_sm = sm.fit_resample(X,y)

print(f'''Shape of X before SMOTE: {X.shape}
Hape of X after SMOTE: {X_sm.shape}''')
print("\nBalance of positive and negative classes (%):")
Y_sm.value_counts(normalize=True)*100

ax = sns.countplot(x='Calidad_label', data=pd.DataFrame(Y_sm, columns=['Calidad_labe
```

```
Shape of X before SMOTE: (1840, 11)
Hape of X after SMOTE: (2514, 11)

Balance of positive and negative classes (%):
```



In [27]:
```
X_train, X_test, y_train, y_test = train_test_split(
                                       X,
                                       y.values.reshape(-1,1),
                                       train_size   = 0.8,
                                       random_state = 1234,
                                       shuffle      = True
                                   )
```

Búsqueda de hiperparámetros

In [28]:
```
# Fijemos el número de particiones. Utilizaremos K = 10.
particiones = KFold(n_splits=10, shuffle=True, random_state = 0)

# Establecemos el espacio de búsqueda para los hiperparámetros que deseamos ajustar.
param_grid = {'criterion':['gini', 'entropy'],'max_depth':[4,6,8,10,20],'min_samples

# Definimos el modelo sin ningún valor de estos hiperparámetros
arbol = DecisionTreeClassifier(random_state=0)
```

In [29]:
```
# Ahora utilizamos GridSearch sobre el grid definido y con 10 particiones en la vali
mejor_modelo = GridSearchCV(arbol, param_grid, cv=particiones)
# Ajuste del modelo
mejor_modelo.fit(X_train, y_train)
```

Out[29]:
```
GridSearchCV(cv=KFold(n_splits=10, random_state=0, shuffle=True),
             estimator=DecisionTreeClassifier(random_state=0),
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': [4, 6, 8, 10, 20],
                         'min_samples_split': [2, 3, 4, 5]})
```

In [30]:
```
# Podemos ver cuál fue el resultado de la búsqueda (mejores valores de hiperparámetr
mejor_modelo.best_params_
```

Out[30]:
```
{'criterion': 'entropy', 'max_depth': 6, 'min_samples_split': 2}
```

In [37]:
```
# Obtener el mejor modelo.
arbol_final = mejor_modelo.best_estimator_
arbol_final = arbol_final.fit(X_train,y_train)
```

5. Resultados

Observamos los resultados del algoritmo

Vemos el desempeño sobre el conjunto de entrenamiento:

In [41]:
```
y_pred = arbol_final.predict(X_train)
print('Exactitud: %.2f' % accuracy_score(y_train, y_pred))
print("Recall: {}".format(recall_score(y_train,y_pred)))
print("Precisión: {}".format(precision_score(y_train,y_pred)))
print("Puntuación F1: {}".format(f1_score(y_train,y_pred)))
```

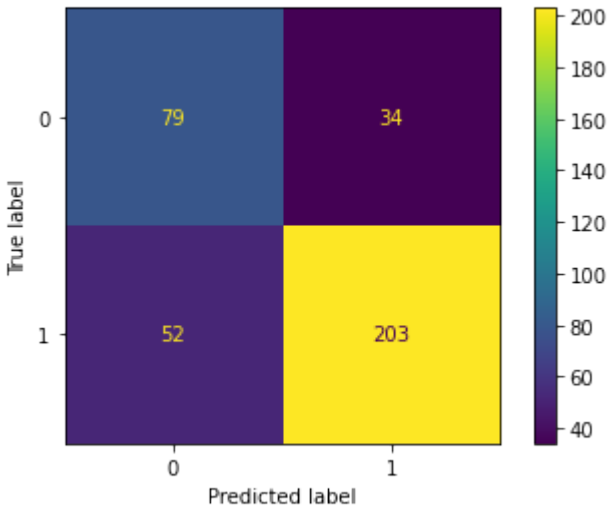
```
Exactitud: 0.80
Recall: 0.8502994011976048
Precisión: 0.852
Puntuación F1: 0.8511488511488512
```

Vemos el desempeño sobre el conjunto de prueba:

In [38]:
```
y_pred = arbol_final.predict(X_test)
print('Exactitud: %.2f' % accuracy_score(y_test, y_pred))
print("Recall: {}".format(recall_score(y_test,y_pred)))
print("Precisión: {}".format(precision_score(y_test,y_pred)))
print("Puntuación F1: {}".format(f1_score(y_test,y_pred)))
```

```
Exactitud: 0.77
Recall: 0.796078431372549
Precisión: 0.8565400843881856
Puntuación F1: 0.8252032520325203
```

In [43]:
```
# Se puede visualizar la matriz de confusión
plot_confusion_matrix(arbol_final, X_test, y_test)
plt.show()
```



Se revisa la importancia de las variables

In [42]:
```
importancia= arbol_final.feature_importances_
importancia_atributo = pd.DataFrame(data={"Atributo": X_train.columns,"Importancia":
importancia_atributo = importancia_atributo.sort_values(by='Importancia', ascending=
importancia_atributo
```

Out[42]:

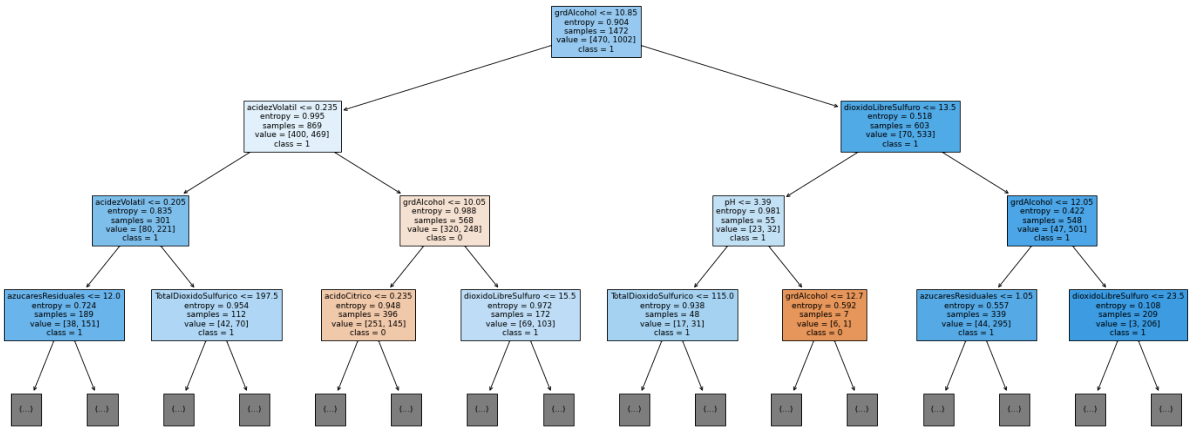
	Atributo	Importancia
0	grdAlcohol	0.404765
1	acidezVolatil	0.123860
2	dioxidoLibreSulfuro	0.120544
3	pH	0.082661
4	TotalDioxidoSulfurico	0.077672
5	acidoCitrico	0.061965
6	azucaresResiduales	0.057410
7	sulfitos	0.053397
8	cloruros	0.017725
9	densidad	0.000000
10	tipoVino	0.000000

Se puede observar que el modelo en general presenta unas métricas buenas. Si bien no soln las mejores, pueden ser aceptables para el negocio. De las métricas obtenidas, podemos ver que el modelo presenta un alto nivel de precisión, lo cual nos puede indicar que está en capacidad de generalizar. En cuanto a las variables, se identificó que las más representativas para definir la calidad son el grado de alcohol, la acidezVolatil, el dioxido y el pH. Finalmente el tipo de vino no aportaba mucha información.

Se logró crear este modelo que puede ayudar al negocio a realizar más eficientemente el proceso de certificación de calidad de sus productos. A continuación, se ve gráficamente:

In [46]:

```
fig = plt.figure(figsize=(25,10))
_ = tree.plot_tree(arbol_final, max_depth=3, feature_names=X.columns, class_names=["
```



Se puede observar que para la clase 1, tiene sentido hablar de vinos con grado de achl <=10.85, una ácidezVolatil mayor a 0.235, un grado de alch >10.5. Esta información puede ser útil para revisar la calidad en vinos con esas propiedades y compararlos con los que tienen condiciones contrarias.