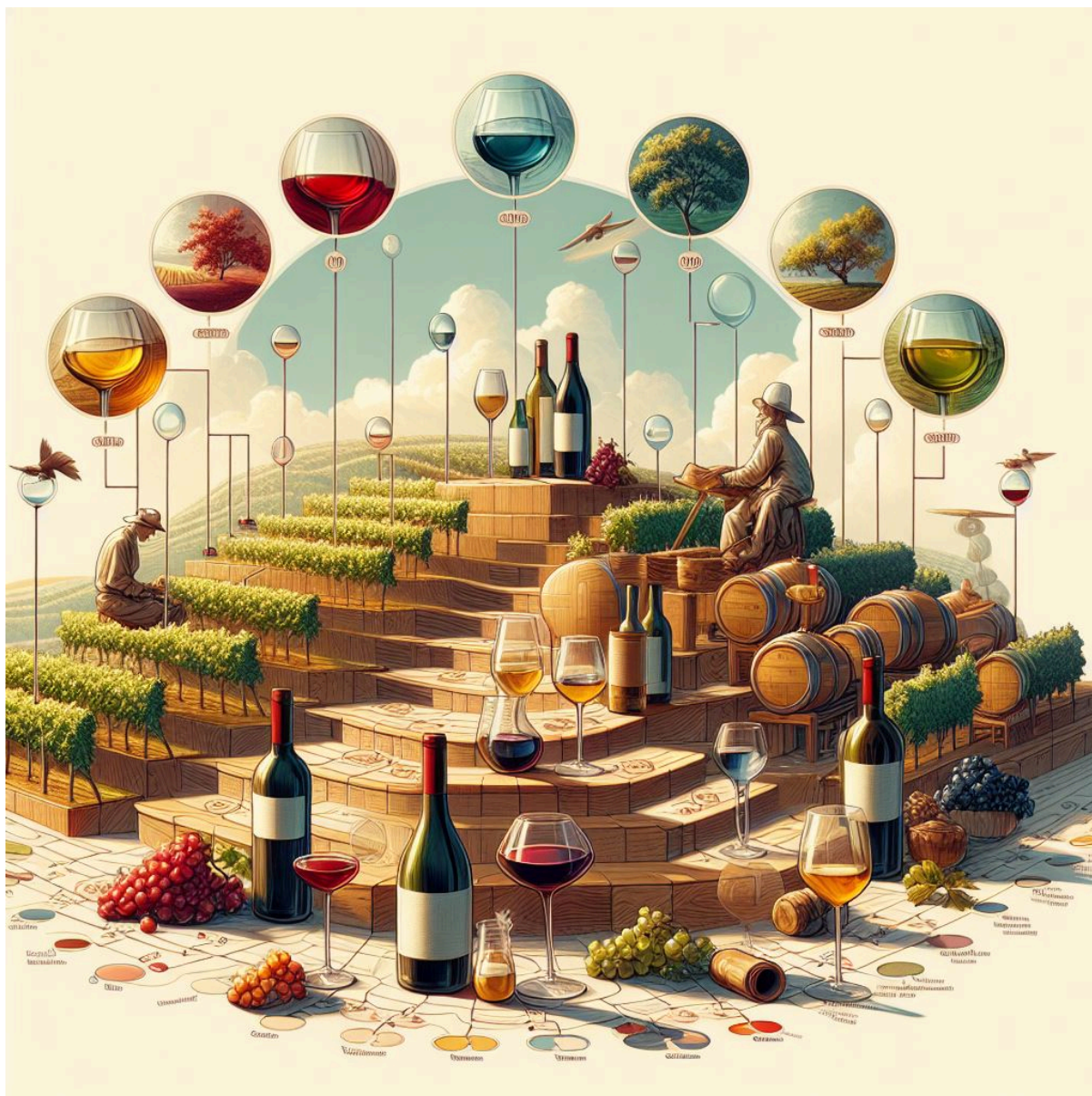


# CLASIFICACIÓN DE VINOS



AARÓN MEDINA MELIÁN

# ÍNDICE

Introducción.....	2
Importación de los datasets (utilizar el dataset winequality-red.csv).....	2
Mostrar la matriz de correlación de variables.....	3
Aplicar cualquier otra técnica de selección de características que consideres adecuados y justificar tu propuesta.....	4
Realizar una comparativa de la precisión en el entrenamiento de los diferentes modelos de árboles. Aplicando Cross Validation.....	9
Una vez decides el modelo que consideras mejor, entonces realizar las siguientes tareas:.....	10
Entrenarlo y obtener la matriz de confusión.....	13
Exportar a un fichero los parámetros del modelo entrenado.....	14
Importar los parámetros del modelo.....	14
Aplicar el modelo (predict) a todos los datos del dataset y obtener la matriz de confusión.....	15
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).....	16
Conclusiones.....	22

# Introducción

Esta tarea trata de predecir el valor de la calidad de un vino basándose en algunas de las características importantes que puede tener un vino.

La calidad se mide en un valor continuo entre 1 y 10, por lo tanto, estamos ante un problema de clasificación.

Enlace al github donde se realiza la tarea con el vino tinto:

[https://github.com/aaronmed/sns/blob/master/UT3%20-%20Algoritmos%20y%20herramientas%20para%20el%20aprendizaje%20supervisado/Actividad%203.5%20-%20Clasificación%20de%20vinos/SNS\\_3\\_5\\_Clasificacion\\_de\\_vinos\\_tintos\\_AaronMedinaMelian.ipynb](https://github.com/aaronmed/sns/blob/master/UT3%20-%20Algoritmos%20y%20herramientas%20para%20el%20aprendizaje%20supervisado/Actividad%203.5%20-%20Clasificación%20de%20vinos/SNS_3_5_Clasificacion_de_vinos_tintos_AaronMedinaMelian.ipynb)

## Importación de los datasets (utilizar el dataset winequality-red.csv)

Subimos el .csv a nuestro propio github para importarlo de una manera más fácil, rápida y cómoda.

Importamos el dataset desde Github y limpiamos los valores nulos

```
winequality_red_url = 'https://raw.githubusercontent.com/aaronmed/sns/master/repositories/winequality-red.csv'
data = pd.read_csv(winequality_red_url, sep=";")
data.dropna(inplace=True)
data
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	9.4	5
1	7.8	0.880	0.00	2.6	0.098	25.0	67.0	0.99680	3.20	0.68	9.8	5
2	7.8	0.760	0.04	2.3	0.092	15.0	54.0	0.99700	3.26	0.65	9.8	5
3	11.2	0.280	0.56	1.9	0.075	17.0	60.0	0.99800	3.16	0.58	9.8	6
4	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	9.4	5
...	...	...	...	...	...	...	...	...	...	...	...	...
1594	6.2	0.600	0.08	2.0	0.090	32.0	44.0	0.99490	3.45	0.58	10.5	5
1595	5.9	0.550	0.10	2.2	0.062	39.0	51.0	0.99512	3.52	0.76	11.2	6
1596	6.3	0.510	0.13	2.3	0.076	29.0	40.0	0.99574	3.42	0.75	11.0	6
1597	5.9	0.645	0.12	2.0	0.075	32.0	44.0	0.99547	3.57	0.71	10.2	5
1598	6.0	0.310	0.47	3.6	0.067	18.0	42.0	0.99549	3.39	0.66	11.0	6

1599 rows x 12 columns

# Mostrar la matriz de correlación de variables

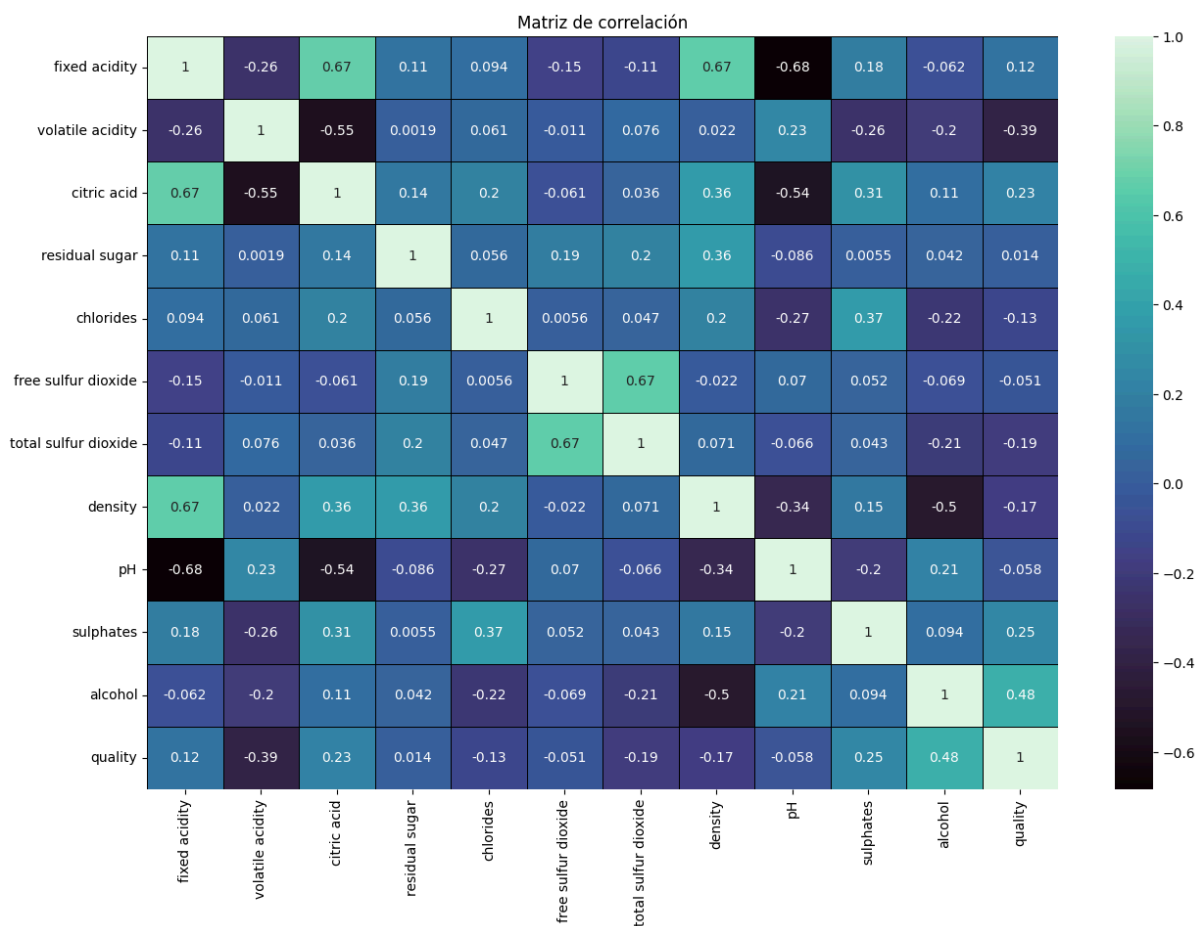
Aquí mostramos la matriz de correlación de variables donde ya podemos ir viendo la alta o baja relación que tienen con nuestro target.

## ✓ Matriz de correlación de variables

```
plt.figure(figsize=(15, 10))
sns.heatmap(data.corr(), annot=True, cmap='mako', linewidths=0.5, linecolor='black')

plt.title('Matriz de correlación')

plt.show()
```



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

Aplicando SelectKBest vamos a seleccionar las características con las que vamos a realizar el entrenamiento.

### ✓ Técnicas de selección de variables

```
▶ X = data.drop('quality', axis=1)
  y = data['quality']

# Especificar el número de características a seleccionar
k = 5 # Seleccionar las 5 mejores características

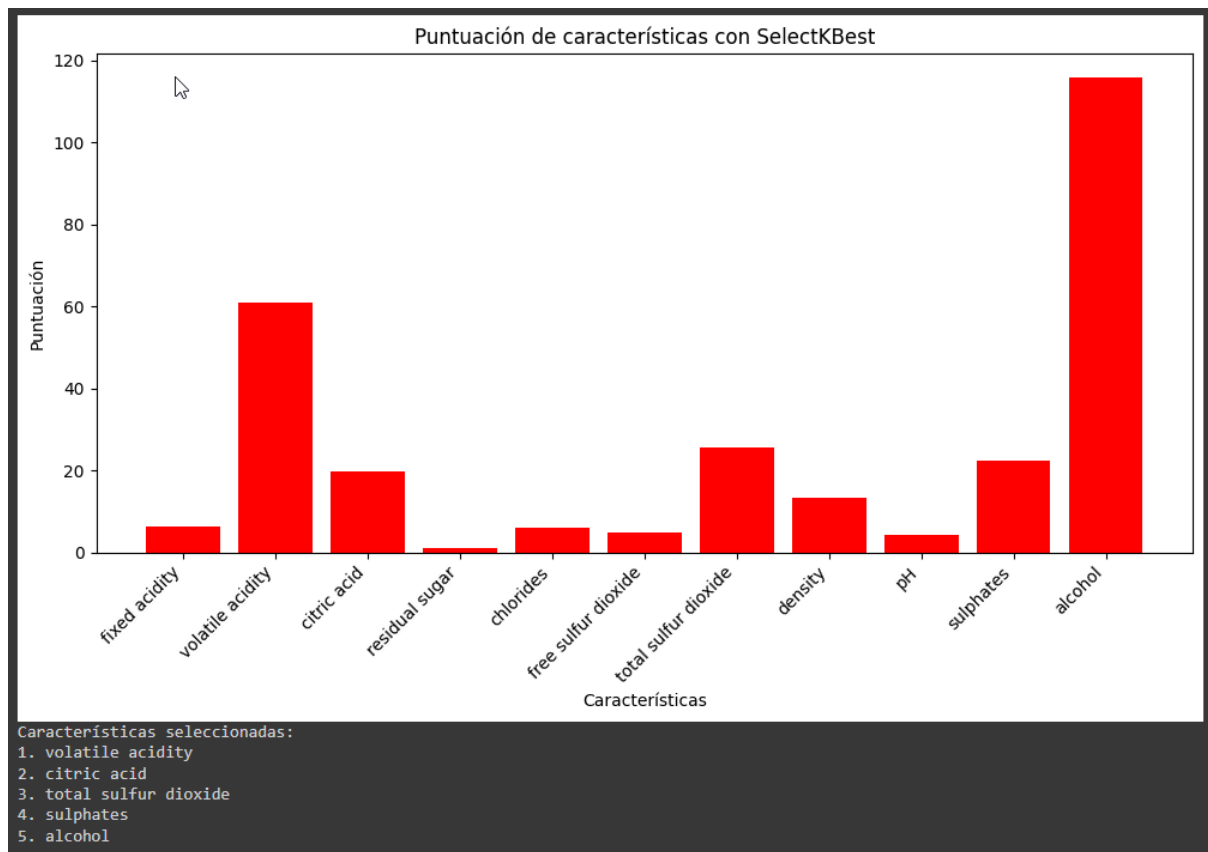
# Aplicar SelectKBest para la selección de características
selector = SelectKBest(score_func=f_classif, k=k)
selector.fit(X, y)

# Obtener las características seleccionadas
selected_features = selector.get_support(indices=True)

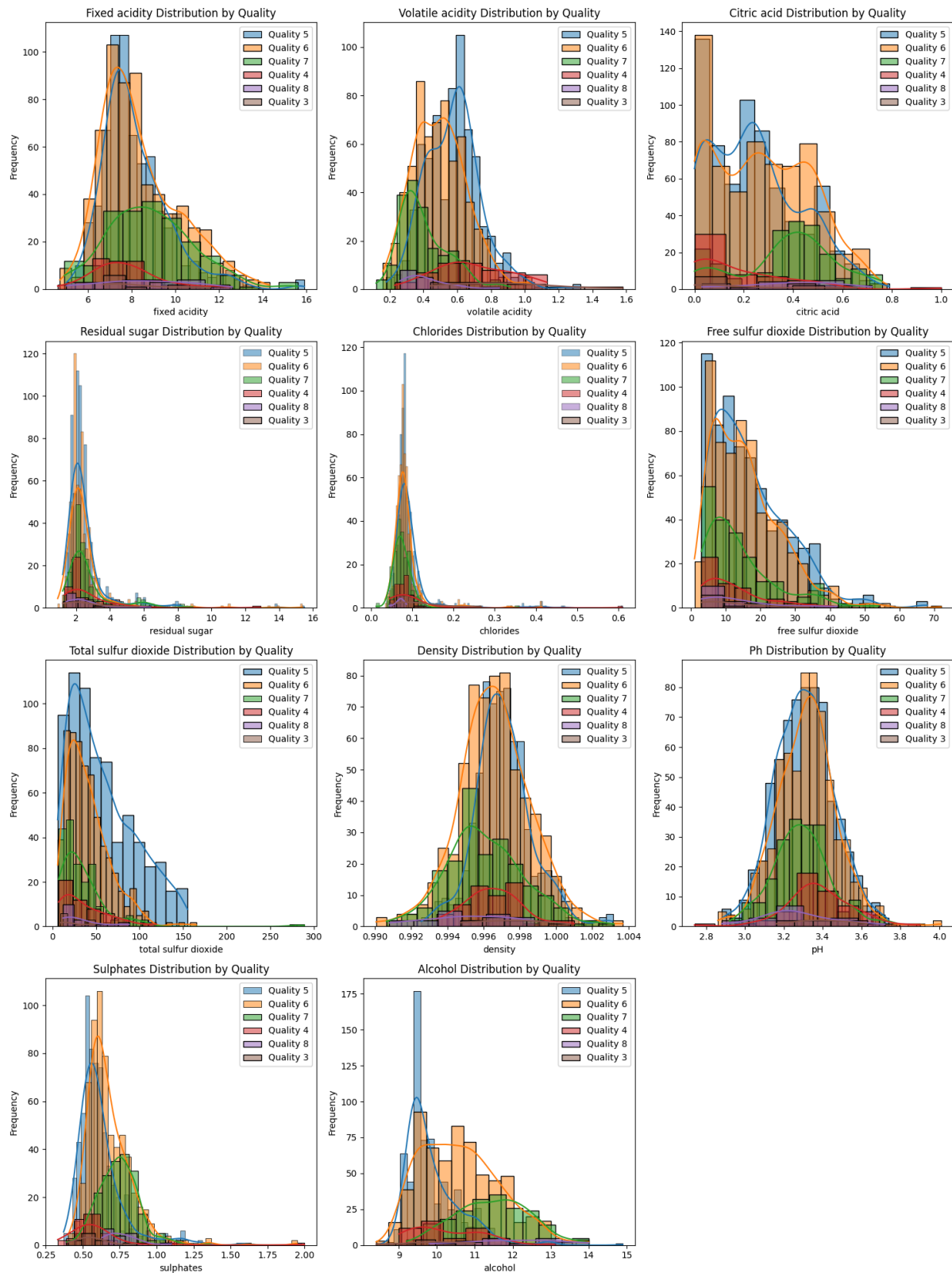
# Graficar la selección de características
plt.figure(figsize=(10, 6))
plt.bar(range(len(selector.scores_)), selector.scores_, tick_label=X.columns, color='red')
plt.title('Puntuación de características con SelectKBest')
plt.xlabel('Características')
plt.ylabel('Puntuación')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()

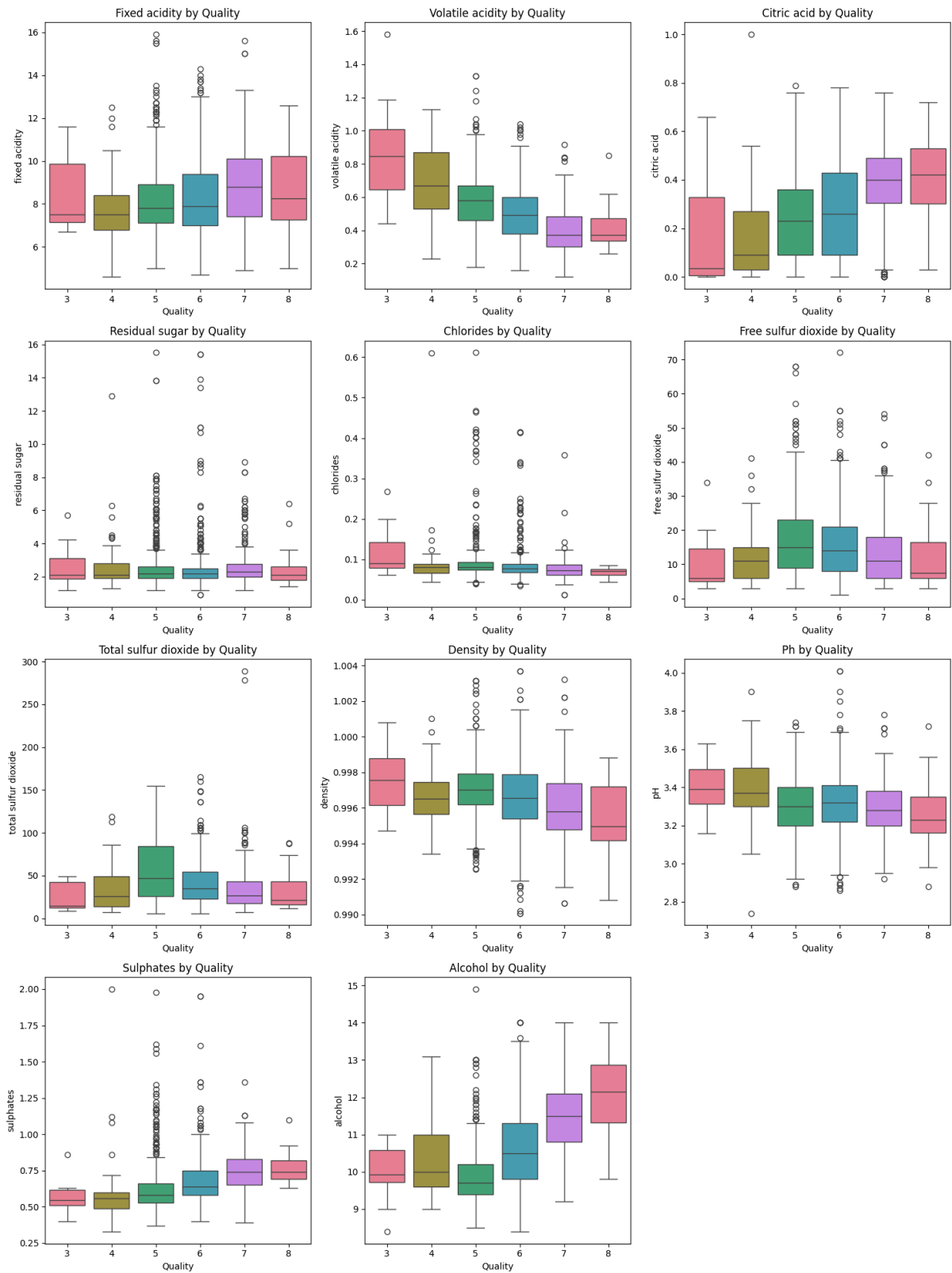
# Imprimir las características seleccionadas
print("Características seleccionadas:")
for i, feature_idx in enumerate(selected_features):
    print(f"{i+1}. {X.columns[feature_idx]}")
```

Apoyándonos en diferentes gráficas vamos a realizar la selección de características. En primer lugar, podemos ver la puntuación que SelectKBest le da a cada una de las características y podemos ver que las más importantes son el alcohol y *volatile acidity* por una gran diferencia del resto.



Vamos a utilizar histogramas para ver la distribución de las diferentes características para poder ver cuáles son las que más nos pueden aportar para la predicción de calidad. Con esta primera gráfica ya podemos darnos cuenta de que *Chlorides* y *Residual sugar* no nos serán útiles para ello.







## Selección de características

```
data = data[['quality', 'volatile acidity', 'citric acid', 'total sulfur dioxide', 'sulphates', 'alcohol']]
```

```
X = data.drop('quality', axis=1)
```

```
y = data['quality']
```

data



	quality	volatile acidity	citric acid	total sulfur dioxide	sulphates	alcohol
0	5	0.700	0.00	34.0	0.56	9.4
1	5	0.880	0.00	67.0	0.68	9.8
2	5	0.760	0.04	54.0	0.65	9.8
3	6	0.280	0.56	60.0	0.58	9.8
4	5	0.700	0.00	34.0	0.56	9.4
...	...	...	...	...	...	...
1594	5	0.600	0.08	44.0	0.58	10.5
1595	6	0.550	0.10	51.0	0.76	11.2
1596	6	0.510	0.13	40.0	0.75	11.0
1597	5	0.645	0.12	44.0	0.71	10.2
1598	6	0.310	0.47	42.0	0.66	11.0

1599 rows x 6 columns

Finalmente nos quedamos con las características que nos ha ofrecido SelectKBest después de apoyarnos en diferentes gráficas para afirmar nuestra decisión.

## Realizar una comparativa de la precisión en el entrenamiento de los diferentes modelos de árboles. Aplicando Cross Validation

```
# Modelos a entrenar
modelos = [
    DecisionTreeClassifier(),
    ExtraTreeClassifier(),
    RandomForestClassifier(),
    ExtraTreesClassifier(),
    GradientBoostingClassifier(),
    AdaBoostClassifier(),
    BaggingClassifier()
]

names = [
    "DecisionTreeClassifier",
    "ExtraTreeClassifier",
    "RandomForestClassifier",
    "ExtraTreesClassifier",
    "GradientBoostingClassifier",
    "AdaBoostClassifier",
    "BaggingClassifier"
]

cv = StratifiedKFold(n_splits = 5, shuffle = True) # shuffle = False si hay dimensión temporal
total_scores = []
for name, clf in zip(names, modelos):
    fold_accuracy = []
    for train_fold, test_fold in cv.split(X_train, y_train):
        # División train test aleatoria
        f_train_x = X_train.iloc[train_fold] # Extrae la información (iloc), atendiendo a los índices obtenidos por CrossValidation
        f_train_y = y_train.iloc[train_fold]
        # entrenamiento y ejecución 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:DecisionTreeClassifier = 0.61
Modelo:ExtraTreeClassifier = 0.60
Modelo:RandomForestClassifier = 0.68
Modelo:ExtraTreesClassifier = 0.67
Modelo:GradientBoostingClassifier = 0.61
Modelo:AdaBoostClassifier = 0.52
Modelo:BaggingClassifier = 0.66
```

Aquí podemos observar que el RandomForestClassifier es el mejor modelo con ligera diferencia del ExtraTreesClassifier, por lo tanto, vamos a seleccionar el RandomForestClassifier para este problema.

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

Antes de comenzar a predecir, vamos a intentar elegir cuáles serían los mejores parámetros para el RandomForestClassifier, para ello vamos a probar a aplicar los diferentes valores de criterion y de max\_depth, quedándonos con el que mayor precisión nos devuelva.

```
[8] classifiers = {
    "gini": RandomForestClassifier(criterion="gini", random_state=42),
    "entropy": RandomForestClassifier(criterion="entropy", random_state=42),
    "log_loss": RandomForestClassifier(criterion="log_loss", random_state=42)
}

best_accuracy = 0
best_criterion = "gini"

for criterion, clf in classifiers.items():
    acc_scores = []
    print({criterion})
    for fold_idx, (train_fold, test_fold) in enumerate(cv.split(X_train, y_train), 1):
        # División train test aleatoria
        f_train_x = X_train.iloc[train_fold] # Extrae la información (iloc), atendiendo a los índices obtenidos por CrossValidation
        f_train_y = y_train.iloc[train_fold]
        # entrenamiento
        clf.fit(f_train_x, f_train_y)
        y_pred = clf.predict(X_train.iloc[test_fold])
        acc = accuracy_score(y_train.iloc[test_fold], y_pred)
        acc_scores.append(acc)

    avg_acc = sum(acc_scores) / len(acc_scores)
    print(f"Accuracy: {avg_acc}\n")

    if avg_acc > best_accuracy:
        best_accuracy = avg_acc
        best_criterion = criterion

print(f"Mejor criterion: {best_criterion}, Precisión: {best_accuracy}")

{'gini'}
Accuracy: 0.6700582107843138

{'entropy'}
Accuracy: 0.6778860294117648

{'log_loss'}
Accuracy: 0.6700520833333333

Mejor criterion: entropy, Precisión: 0.6778860294117648
```

```
Buscamos el mejor valor para max_depth

best_accuracy = 0
best_max_depth = None

max_depths = []
accuracies = []

for max_depth in range(2, 30):
    print(f"Probando max_depth con {best_criterion}: {max_depth}")
    clf = RandomForestClassifier(max_depth=max_depth, criterion=best_criterion, random_state=42)
    acc_scores = []
    for train_fold, test_fold in cv.split(X_train, y_train):
        # División train-test aleatoria
        f_train = X_train.iloc[train_fold] # Extrae la información (iloc), atendiendo a los índices obtenidos por CrossValidation
        f_test = y_train.iloc[test_fold]
        # entrenamiento
        clf.fit(f_train, f_train.y)
        y_pred = clf.predict(f_test)
        acc = accuracy_score(y_test, y_pred)
        acc_scores.append(acc)



    avg_acc = sum(acc_scores) / len(acc_scores)
    print(f"Accuracy: {avg_acc}\n")

    max_depths.append(max_depth)
    accuracies.append(avg_acc)

    if avg_acc > best_accuracy:
        best_accuracy = avg_acc
        best_max_depth = max_depth

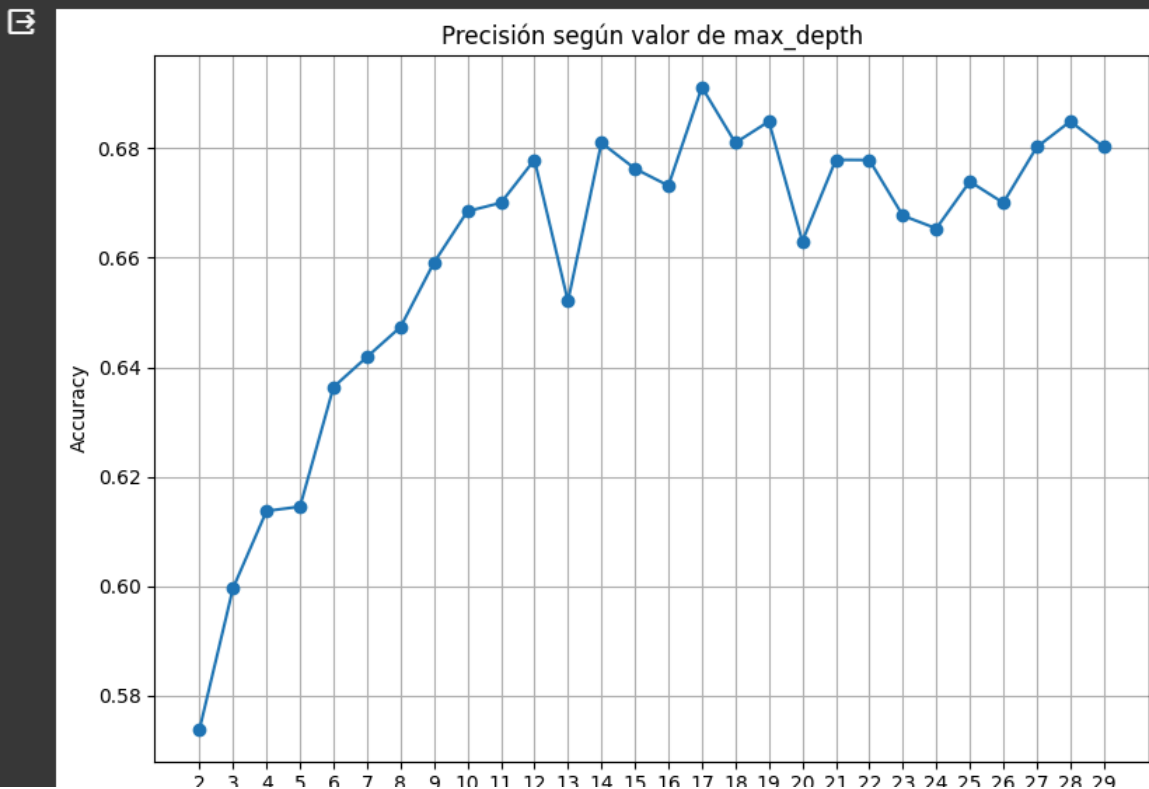
print(f"Mejor max_depth: {best_max_depth}, Precisión: {best_accuracy}")

Probando max_depth con entropy: 2
Accuracy: 0.5739093137254903
```

```
50 s  Probando max_depth con entropy: 15  
Accuracy: 0.6762867647058823  
 Probando max_depth con entropy: 16  
Accuracy: 0.6731740196078431  
  
Probando max_depth con entropy: 17  
Accuracy: 0.6911672794117647  
  
Probando max_depth con entropy: 18  
Accuracy: 0.6809926470588235  
  
Probando max_depth con entropy: 19  
Accuracy: 0.6849080882352941  
  
Probando max_depth con entropy: 20  
Accuracy: 0.6630177696078431  
  
Probando max_depth con entropy: 21  
Accuracy: 0.6778921568627452  
  
Probando max_depth con entropy: 22  
Accuracy: 0.6778584558823529  
  
Probando max_depth con entropy: 23  
Accuracy: 0.6676960784313726  
  
Probando max_depth con entropy: 24  
Accuracy: 0.6653615196078431  
  
Probando max_depth con entropy: 25  
Accuracy: 0.673921568627451  
  
Probando max_depth con entropy: 26  
Accuracy: 0.6700428921568627  
  
Probando max_depth con entropy: 27  
Accuracy: 0.680217524509804  
  
Probando max_depth con entropy: 28  
Accuracy: 0.6849142156862745  
  
Probando max_depth con entropy: 29  
Accuracy: 0.6802420343137255  
  
Mejor max_depth: 17, Precisión: 0.6911672794117647
```

Mostramos como va cambiando la precisión según cambia el valor de max\_depth

```
plt.figure(figsize=(8, 6))
plt.plot(max_depths, accuracies, marker='o')
plt.title('Precisión según valor de max_depth')
plt.xlabel('max_depth')
plt.ylabel('Accuracy')
plt.grid(True)
plt.xticks(range(2, 30))
plt.tight_layout()
plt.show()
```



Aquí podemos ver como va cambiando la precisión mientras cambia el max\_depth.

## Entrenarlo y obtener la matriz de confusión

En este caso nos hemos quedado con los parámetros:

criterion: entropy y max\_depth: 17

pero el código está preparado para si dependiendo del entrenamiento estos valores cambien, sean esos los mejores parámetros elegidos.

```
[28] model_selected = RandomForestClassifier(max_depth=best_max_depth, criterion=best_criterion, random_state=42)

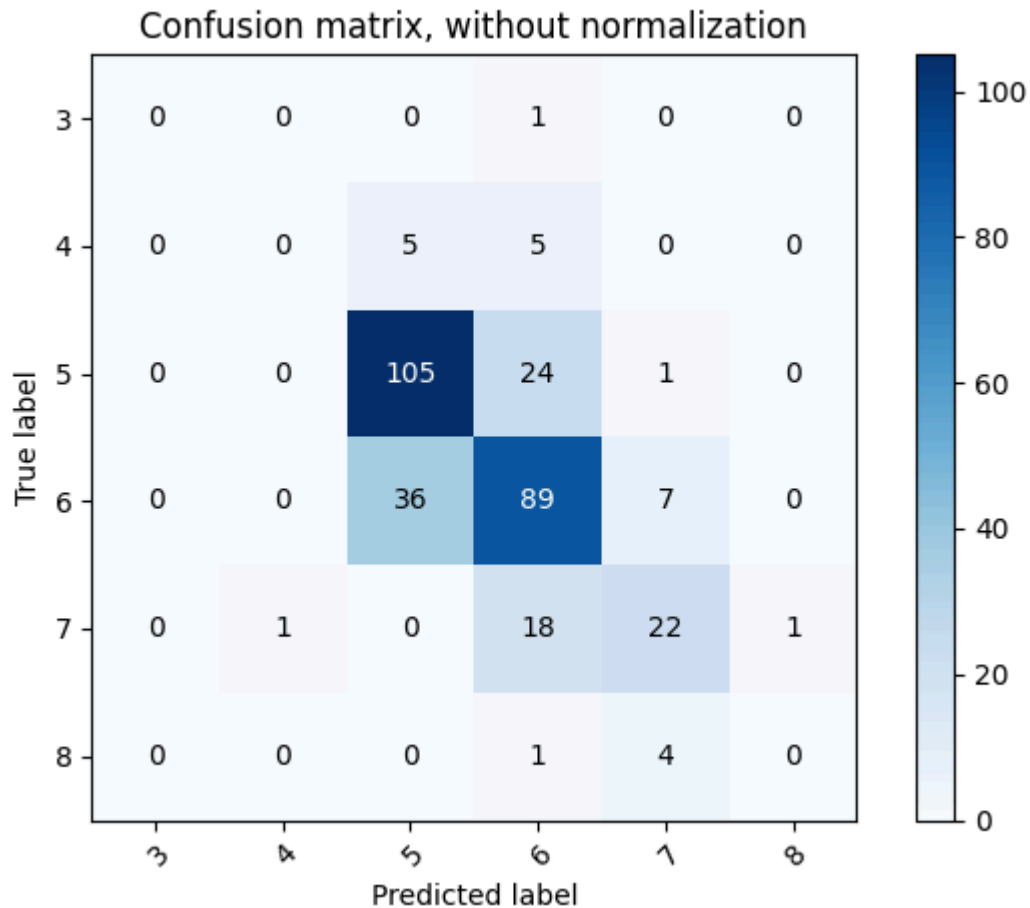
model_selected.fit(X_train, y_train)

y_pred = model_selected.predict(X_test)

acc = accuracy_score(y_test, y_pred)

print("Precisión:", acc)

Precisión: 0.675
```



Exportar a un fichero los parámetros del modelo entrenado  
 Importar los parámetros del modelo

✓ Exportamos, importamos el modelo y volvemos a predecir

```
0s [34] name_model = 'ModelSelectedVinosTintos.pkl'
    joblib.dump(model_selected, name_model)
    ['ModelSelectedVinosTintos.pkl']
```

```
0s [36] model_imported = joblib.load(name_model)
```

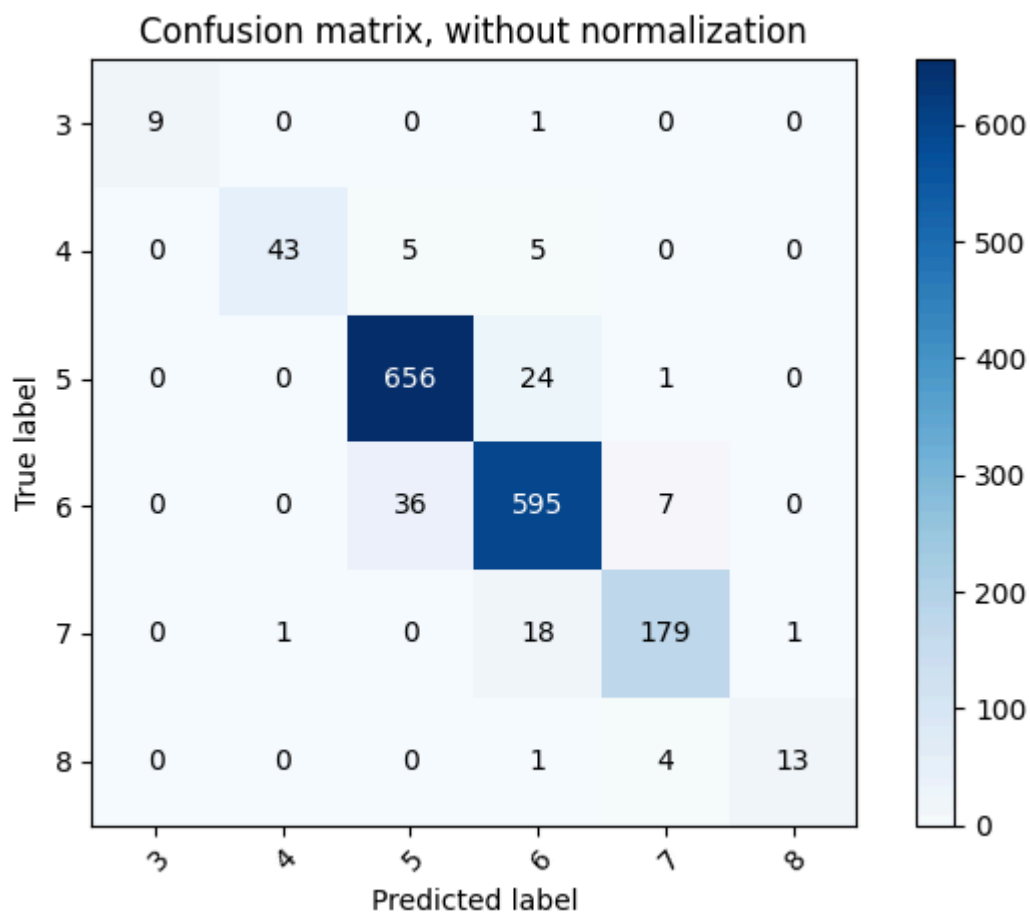
Aplicar el modelo (predict) a todos los datos del dataset y obtener la matriz de confusión

```
[41] y_pred_model_imported = model_imported.predict(X)
```

```
plot_confusion_matrix(y, y_pred_model_imported, classes = np.array(['0', '1', '2', '3', '4', '5', '6', '7', '8', '9', '10']), normalize=False)
```

Metrics

	precision	recall	f1-score	support
3	1.00	0.90	0.95	10
4	0.98	0.81	0.89	53
5	0.94	0.96	0.95	681
6	0.92	0.93	0.93	638
7	0.94	0.90	0.92	199
8	0.93	0.72	0.81	18
accuracy			0.93	1599
macro avg	0.95	0.87	0.91	1599
weighted avg	0.94	0.93	0.93	1599



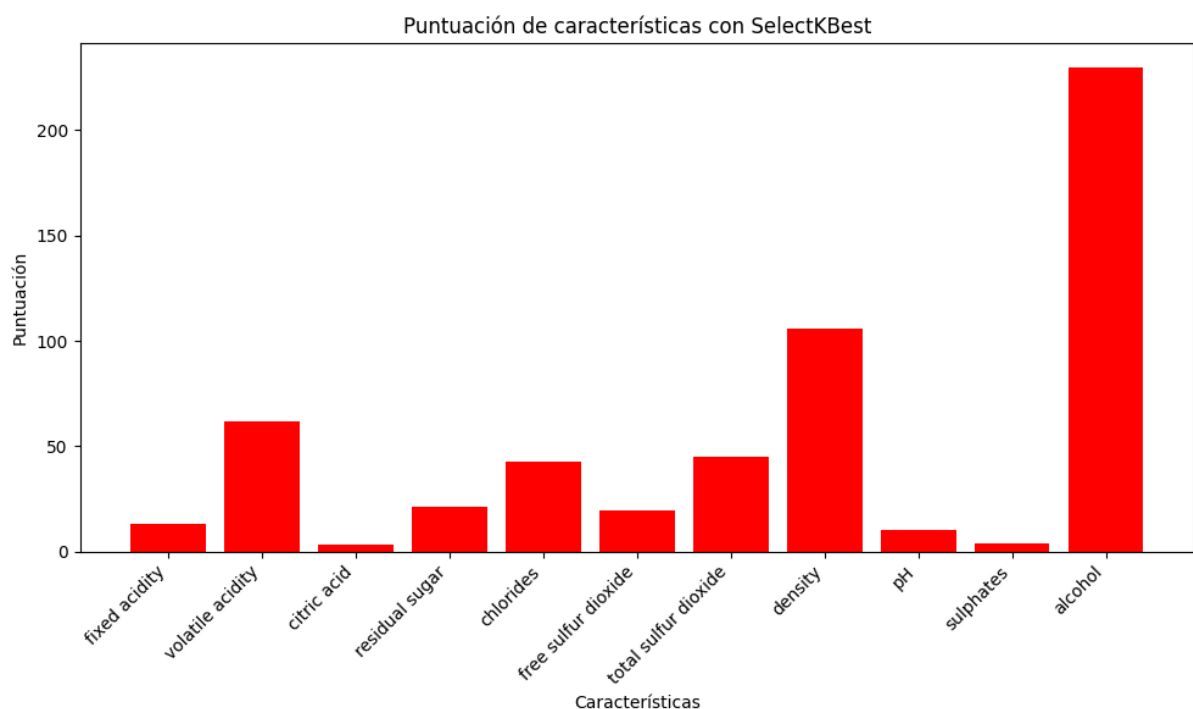


Probar a utilizar el cuaderno con el dataset de los vinos blancos y realizar captura de los resultados obtenidos. (utilizar el dataset winequality-white.csv)

Enlace al github donde se realiza la tarea de vinos blancos:

[https://github.com/aaronmed/sns/blob/master/UT3%20-%20Algoritmos%20y%20herramientas%20para%20el%20aprendizaje%20supervisado/Actividad%203.5%20-%200Clasificación%20de%20vinos/SNS\\_3\\_5\\_Clasificacion\\_de\\_vinos\\_blancos\\_AaronMedinaMelian.ipynb](https://github.com/aaronmed/sns/blob/master/UT3%20-%20Algoritmos%20y%20herramientas%20para%20el%20aprendizaje%20supervisado/Actividad%203.5%20-%200Clasificación%20de%20vinos/SNS_3_5_Clasificacion_de_vinos_blancos_AaronMedinaMelian.ipynb)

A la hora de seleccionar las mejores características vemos que hay diferencias entre las más importantes aunque el alcohol sigue siendo la más importante con diferencia.



Esto nos hace decantarnos por otras características diferentes al vino rojo

```
Características seleccionadas:  
1. volatile acidity  
2. chlorides  
3. total sulfur dioxide  
4. density  
5. alcohol
```

## ✓ Selección de características

```
data = data[['quality', 'volatile acidity', 'density', 'total sulfur dioxide', 'chlorides', 'alcohol']]  
  
X = data.drop('quality', axis=1)  
y = data['quality']  
  
data
```

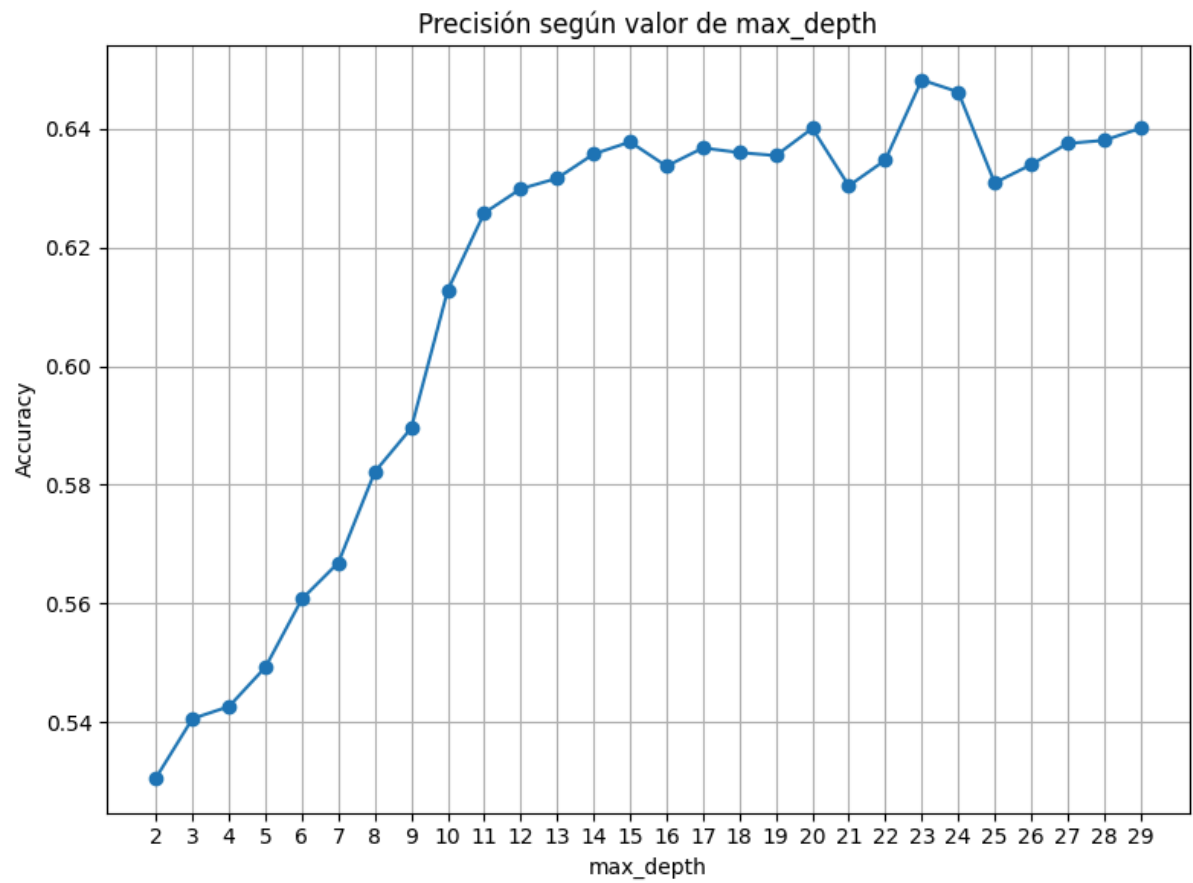
Al igual que con el de los vinos rojos, siguen siendo los mejores modelos los mismos

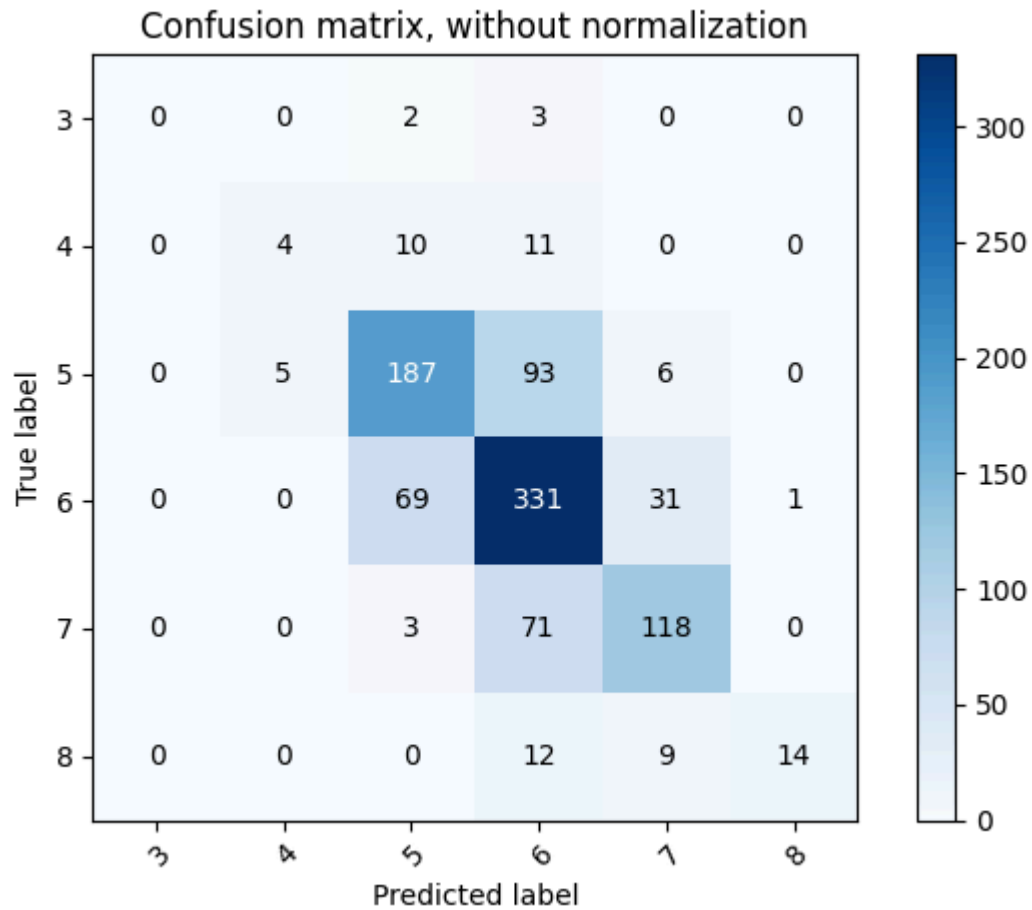
```
Modelo:DecisionTreeClassifier = 0.56  
Modelo:ExtraTreeClassifier = 0.55  
Modelo:RandomForestClassifier = 0.64  
Modelo:ExtraTreesClassifier = 0.64  
Modelo:GradientBoostingClassifier = 0.58  
Modelo:AdaBoostClassifier = 0.27  
Modelo:BaggingClassifier = 0.60
```

Nos decantamos por el RandomForestClassifier de nuevo.

```
{'gini'}  
Accuracy: 0.6431852242812834  
  
{'entropy'}  
Accuracy: 0.6340043540151692  
  
{'log_loss'}  
Accuracy: 0.6365437745979617  
  
Mejor criterion: gini, Precisión: 0.6431852242812834  
  
Mejor max_depth: 23, Precisión: 0.6482827038861522
```

Pero a diferencia de con el otro dataset, los mejores parámetros son diferentes.





Metrics				
	precision	recall	f1-score	support
3	0.00	0.00	0.00	5
4	0.44	0.16	0.24	25
5	0.69	0.64	0.67	291
6	0.64	0.77	0.69	432
7	0.72	0.61	0.66	192
8	0.93	0.40	0.56	35
accuracy			0.67	980
macro avg	0.57	0.43	0.47	980
weighted avg	0.67	0.67	0.66	980

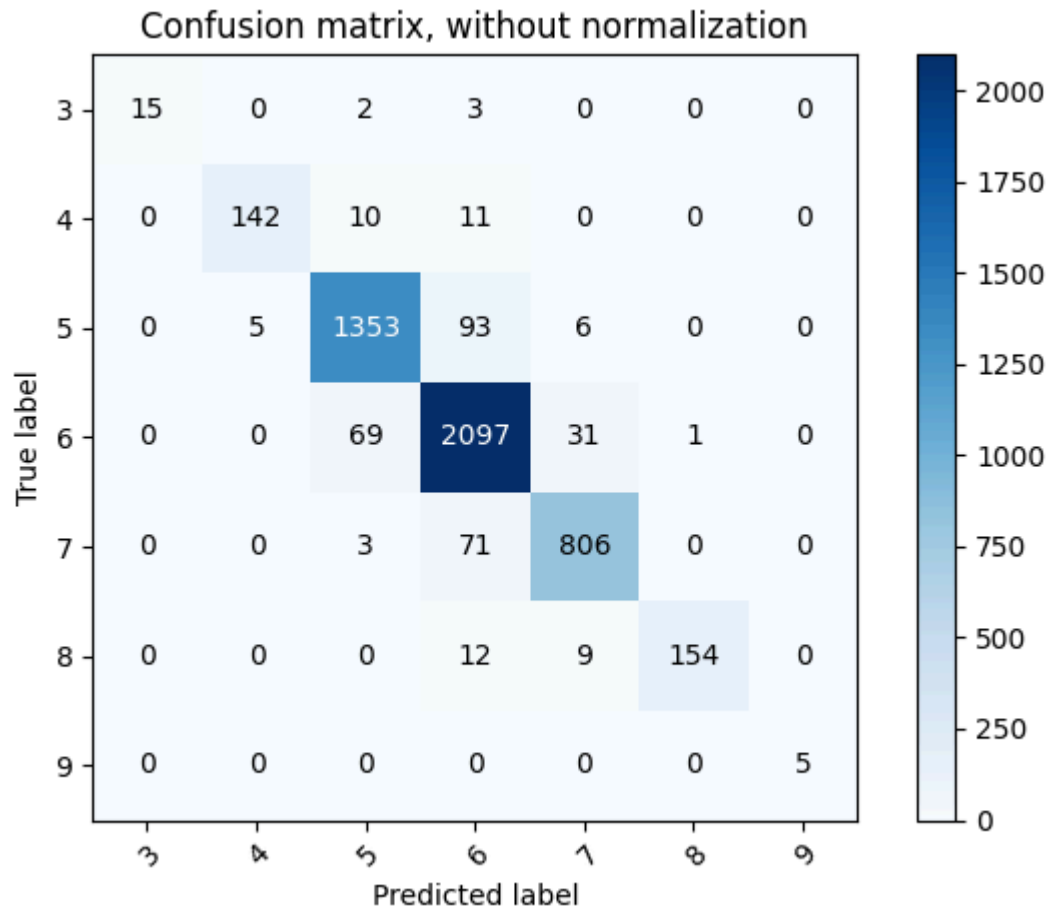
## ✓ Exportamos, importamos el modelo y volvemos a predecir

```
[20] name_model = 'ModelSelectedVinosBlancos.pkl'
      joblib.dump(model_selected, name_model)
```

```
↳ ['ModelSelectedVinosBlancos.pkl']
```

```
[21] model_imported = joblib.load(name_model)
```

```
[22] y_pred_model_imported = model_imported.predict(X)
```



Metrics				
	precision	recall	f1-score	support
3	1.00	0.75	0.86	20
4	0.97	0.87	0.92	163
5	0.94	0.93	0.94	1457
6	0.92	0.95	0.94	2198
7	0.95	0.92	0.93	880
8	0.99	0.88	0.93	175
9	1.00	1.00	1.00	5
accuracy			0.93	4898
macro avg	0.97	0.90	0.93	4898
weighted avg	0.93	0.93	0.93	4898

## Conclusiones

Finalmente, hemos visto que aunque sean problemas parecidos, se ha visto una diferencia de características seleccionados e incluso unos parámetros diferentes con un dataset y otro.

En el segundo podríamos haber realizado la predicción con el ExtraTressClassifier pero he decidido aprovechar la selección de parámetros que tenía preparada el RandomForestClassifier y me ha hecho mucho más fácil el trabajo para la elección de parámetros en esta segunda ocasión.