

✓ Prácticas

- **Asignatura:** Análisis Automático de Datos para las Ciencias Biomédicas, Medioambientales, Agroalimentarias
- **Estudiantes:** Mabrouka Salmi <z12salsm@uco.es>; Alba Marquez Rodriguez <z32maroa@uco.es>
- **Máster:** Inteligencia Computacional e Internet de las Cosas
- **Universidad:** Escuela Politécnica Superior de Córdoba -UCO
- **Curso:** 2023/2024

1.2 Elección caso práctico

Al ya conocer la base de datos *Wine* será esta la elegida para realizar los casos prácticos. Es una base de datos para la clasificación.

Wine: Uso del análisis químico para determinar el origen de los vinos. En esta base de datos el atributo de clase está en la primera columna.

✓ 1.3 Importar y preprocesar datos

```
!pip install ucimlrepo
```

```
Requirement already satisfied: ucimlrepo in /usr/local/lib/python3.10/dist-packages (0.0.7)
Requirement already satisfied: pandas>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from ucimlrepo) (2.0.3)
Requirement already satisfied: certifi>=2020.12.5 in /usr/local/lib/python3.10/dist-packages (from ucimlrepo) (2024.2.2)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.10/dist-packages (from pandas>=1.0.0->ucimlrepo) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=1.0.0->ucimlrepo) (2023.4)
Requirement already satisfied: tzdata>=2022.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=1.0.0->ucimlrepo) (2024.1)
Requirement already satisfied: numpy>=1.21.0 in /usr/local/lib/python3.10/dist-packages (from pandas>=1.0.0->ucimlrepo) (1.25.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.8.2->pandas>=1.0.0->ucimlrepo) (1.16.0)
```

```
from ucimlrepo import fetch_ucirepo
```

```
# fetch dataset
wine = fetch_ucirepo(id=109)
```

```
# data (as pandas dataframes)
X = wine.data.features
y = wine.data.targets
```

```
# metadata
print(wine.metadata)
```

```
# variable information
print(wine.variables)
```

```
{'uci_id': 109, 'name': 'Wine', 'repository_url': 'https://archive.ics.uci.edu/dataset/109/wine', 'data_url': 'https://archive.ics.uci.edu/dataset/109/wine'}
```

	name	role	type	demographic
0	class	Target	Categorical	None
1	Alcohol	Feature	Continuous	None
2	Malicacid	Feature	Continuous	None
3	Ash	Feature	Continuous	None
4	Alcalinity_of_ash	Feature	Continuous	None
5	Magnesium	Feature	Integer	None
6	Total_phenols	Feature	Continuous	None
7	Flavanoids	Feature	Continuous	None
8	Nonflavanoid_phenols	Feature	Continuous	None
9	Proanthocyanins	Feature	Continuous	None
10	Color_intensity	Feature	Continuous	None
11	Hue	Feature	Continuous	None
12	OD280_0D315_of_diluted_wines	Feature	Continuous	None
13	Proline	Feature	Integer	None

```
description units missing_values
```

0	None	None	no
1	None	None	no
2	None	None	no
3	None	None	no
4	None	None	no
5	None	None	no
6	None	None	no
7	None	None	no
8	None	None	no
9	None	None	no
10	None	None	no
11	None	None	no
12	None	None	no
13	None	None	no

✓ 1.4 Análisis preliminar

1. Para todas las variables cuantitativas, recoge el valor medio, máximo, mínimo y la desviación típica de la misma, indicando de dónde obtiene esa información en Weka (con poner un ejemplo de de dónde se obtiene es suficiente). Dado el caso, puede comentar algo que le parezca interesante sobre esos valores.
2. Para todas las variables cualitativas, recoge la frecuencia de cada categoría de la variable, indicando de dónde obtiene esa información en Weka (con poner un ejemplo de de dónde se obtiene es suficiente).

```
import pandas as pd
```

```
# Concatenar características y objetivos
data = pd.concat([X, y], axis=1)
```

```
# Estadísticas para variables cuantitativas
quantitative_stats = data.describe()
```

```
# Frecuencia para variables cualitativas
qualitative_freq = data.apply(pd.value_counts)
```

```
# Imprimir estadísticas para variables cuantitativas
print("Estadísticas para variables cuantitativas:")
print(quantitative_stats)
```

```
# Imprimir frecuencia para variables cualitativas
print("\nFrecuencia para variables cualitativas:")
print(qualitative_freq)
```

```
→ Estadísticas para variables cuantitativas:
```

	Alcohol	Malicacid	Ash	Alcalinity_of_ash	Magnesium \
count	178.000000	178.000000	178.000000	178.000000	178.000000
mean	13.000618	2.336348	2.366517	19.494944	99.741573
std	0.811827	1.117146	0.274344	3.339564	14.282484
min	11.030000	0.740000	1.360000	10.600000	70.000000
25%	12.362500	1.602500	2.210000	17.200000	88.000000
50%	13.050000	1.865000	2.360000	19.500000	98.000000
75%	13.677500	3.082500	2.557500	21.500000	107.000000
max	14.830000	5.800000	3.230000	30.000000	162.000000

	Total_phenols	Flavanoids	Nonflavanoid_phenols	Proanthocyanins \
count	178.000000	178.000000	178.000000	178.000000
mean	2.295112	2.029270	0.361854	1.590899
std	0.625851	0.998859	0.124453	0.572359
min	0.980000	0.340000	0.130000	0.410000
25%	1.742500	1.205000	0.270000	1.250000
50%	2.355000	2.135000	0.340000	1.555000
75%	2.800000	2.875000	0.437500	1.950000
max	3.880000	5.080000	0.660000	3.580000

	Color_intensity	Hue	0D280_0D315_of_diluted_wines	Proline \
count	178.000000	178.000000	178.000000	178.000000
mean	5.058090	0.957449	2.611685	746.893258
std	2.318286	0.228572	0.709990	314.907474
min	1.280000	0.480000	1.270000	278.000000
25%	3.220000	0.782500	1.937500	500.500000
50%	4.690000	0.965000	2.780000	673.500000
75%	6.200000	1.120000	3.170000	985.000000
max	13.000000	1.710000	4.000000	1680.000000

	class
count	178.000000
mean	1.938202
std	0.775035
min	1.000000
25%	1.000000
50%	2.000000
75%	3.000000
max	3.000000


```
Frecuencia para variables cualitativas:
```

	Alcohol	Malicacid	Ash	Alcalinity_of_ash	Magnesium	Total_phenols \
0.13	NaN	NaN	NaN	NaN	NaN	NaN
0.14	NaN	NaN	NaN	NaN	NaN	NaN
0.17	NaN	NaN	NaN	NaN	NaN	NaN
0.19	NaN	NaN	NaN	NaN	NaN	NaN
0.20	NaN	NaN	NaN	NaN	NaN	NaN
...
1480.00	NaN	NaN	NaN	NaN	NaN	NaN
1510.00	NaN	NaN	NaN	NaN	NaN	NaN
1515.00	NaN	NaN	NaN	NaN	NaN	NaN

1547.00	NaN	NaN	NaN	NaN	NaN	NaN
1680.00	NaN	NaN	NaN	NaN	NaN	NaN
	Flavanoids	Nonflavanoid_phenols	Proanthocyanins	Color_intensity		
0.13	NaN	1.0	NaN	NaN		
0.14	NaN	2.0	NaN	NaN		

Las estadísticas para las variables cuantitativas muestran valores medios, máximos, mínimos y desviaciones estándar para cada característica del conjunto de datos wine. Por ejemplo, el contenido medio de alcohol es de aproximadamente 13.0%, con un mínimo de 11.03% y un máximo de 14.83%. La desviación estándar para el contenido de alcohol es de aproximadamente 0.81%.

Sabiendo que, según los metadatos, nuestras variables son numéricas, ya sean continuas o discretas, excepto la variable de `class` que normalmente es categórica, contiene tres clases, pero se transforma a numérica {1, 2, 3}. Por eso, cuando extraemos las estadísticas descriptivas para las variables categóricas, obtenemos resultados en NAN como se describe a continuación:

En cuanto a las frecuencias para las variables cualitativas, los valores nulos indican que no se encuentran observaciones de esas categorías en el conjunto de datos. Por ejemplo, no hay ninguna observación con valores de alcohol de 0.13%, 0.14%, 0.17%, etc. Esto puede sugerir que las categorías ausentes podrían no ser relevantes en este contexto o podrían ser el resultado de una limitación en la recopilación de datos.

✓ Valores perdidos

¿Existen valores perdidos en la bases de datos? Si fuera así, analice en qué porcentaje y en qué variables (indique de dónde obtiene esa información). Sino fuera así, indique cómo podría analizarlo en Weka.

```
# Verificar si hay valores perdidos
missing_values = data.isnull().sum()

# Calcular el porcentaje de valores perdidos por variable
percentage_missing = (missing_values / len(data)) * 100

# Filtrar las variables que tienen valores perdidos
variables_with_missing_values = percentage_missing[percentage_missing > 0]

if variables_with_missing_values.empty:
    print("No hay valores perdidos en la base de datos.")
else:
    print("Variables con valores perdidos:")
    print(variables_with_missing_values)
```

➡ No hay valores perdidos en la base de datos.

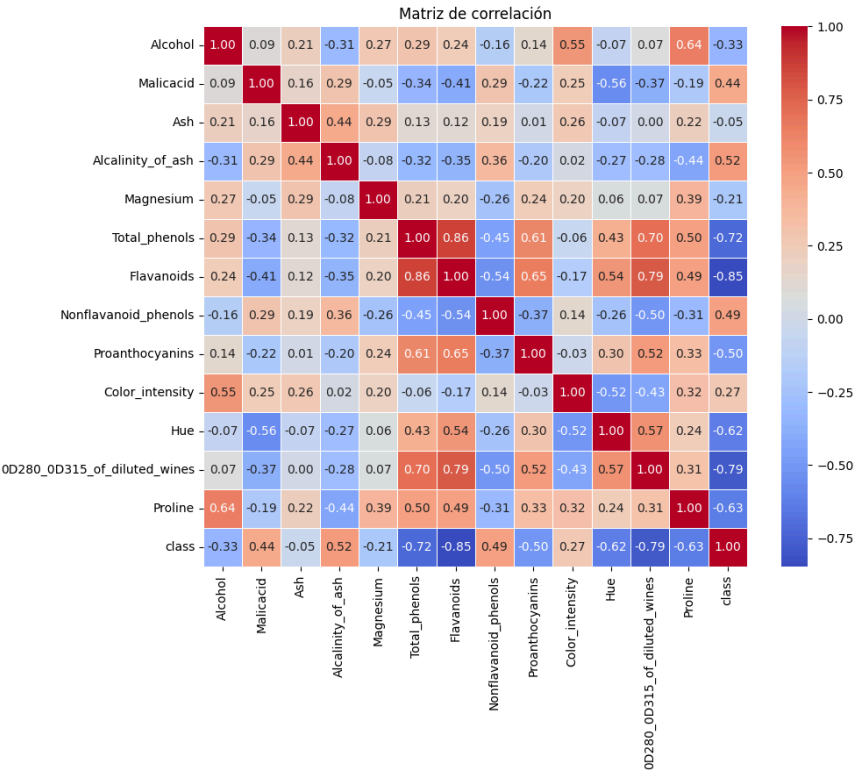
✓ Pares de atributo

¿Encuentra alguna relación entre pares de atributos, algunos valores de los mismos que permitan una cierta separación de clases, algún rango de valores donde se concentren datos?. Si localiza alguna relación significativa, explíquela (estudiando más a fondo el tipo de problema puede comprobar si puede tener o no sentido, por ejemplo, si es una base de datos de ILPD, puede mirar en la Web qué suele influir más en esa enfermedad). Si no ve relaciones indíquelo también.

Para saber si existe alguna relación se puede calcular la matriz de correlación entre atributos:

```
import matplotlib.pyplot as plt
import seaborn as sns

correlation_matrix = data.corr()
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f", linewidths=0.5)
plt.title('Matriz de correlación')
plt.show()
```



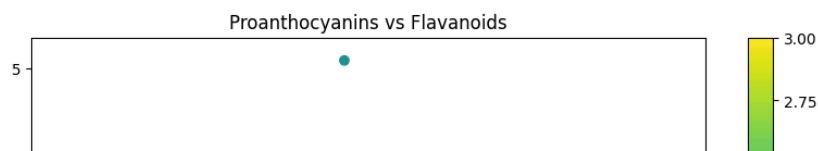
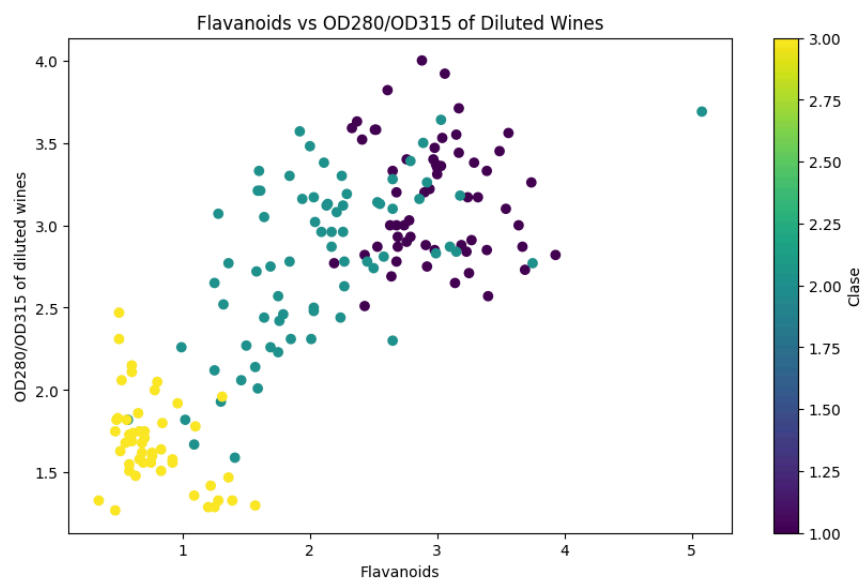
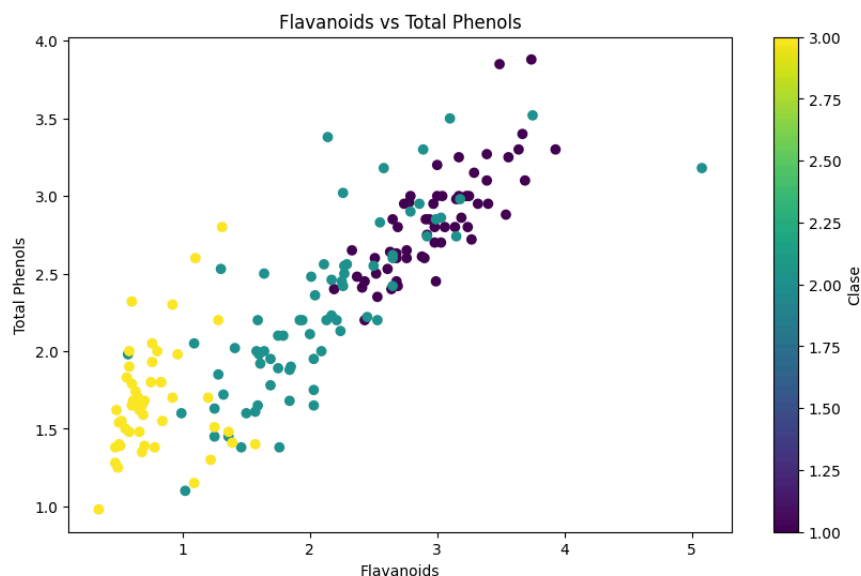
De esta matriz de correlación se puede sacar que los valores más cercanos al rojo son los que más correlacionados están. Por ejemplo, flavanoids y total_phenols parecen estar muy relacionados. Así como el compuesto de diluted_wines con flavanoids o con total_phenols. Phroanthocyanins y flavanoids también tienen un alto nivel de relación. Y Proline y alcohol. A continuación se muestra un gráfico de dispersión para visualizar en concreto algunas de estas relaciones.

```
# Scatter plot de flavanoids vs total_phenols
plt.figure(figsize=(10, 6))
plt.scatter(data['Flavanoids'], data['Total_phenols'], c=data['class'], cmap='viridis')
plt.xlabel('Flavanoids')
plt.ylabel('Total Phenols')
plt.title('Flavanoids vs Total Phenols')
plt.colorbar(label='Clase')
plt.show()

# Scatter plot de flavanoids vs OD280/OD315_of_diluted_wines
plt.figure(figsize=(10, 6))
plt.scatter(data['Flavanoids'], data['OD280_OD315_of_diluted_wines'], c=data['class'], cmap='viridis')
plt.xlabel('Flavanoids')
plt.ylabel('OD280/OD315 of diluted wines')
plt.title('Flavanoids vs OD280/OD315 of Diluted Wines')
plt.colorbar(label='Clase')
plt.show()

# Scatter plot de proanthocyanins vs flavanoids
plt.figure(figsize=(10, 6))
plt.scatter(data['Proanthocyanins'], data['Flavanoids'], c=data['class'], cmap='viridis')
plt.xlabel('Proanthocyanins')
plt.ylabel('Flavanoids')
plt.title('Proanthocyanins vs Flavanoids')
plt.colorbar(label='Clase')
plt.show()

# Scatter plot de proline vs alcohol
plt.figure(figsize=(10, 6))
plt.scatter(data['Proline'], data['Alcohol'], c=data['class'], cmap='viridis')
plt.xlabel('Proline')
plt.ylabel('Alcohol')
plt.title('Proline vs Alcohol')
plt.colorbar(label='Clase')
plt.show()
```



▼ Preprocesamiento

Una técnica común de preprocesamiento de datos es la estandarización de características (también conocida como normalización), que implica ajustar la escala de los atributos para que tengan una media de cero y una desviación estándar de uno. Esto es particularmente útil para algoritmos de aprendizaje automático que son sensibles a la escala de los atributos, como las máquinas de vectores de soporte (SVM) o los algoritmos de gradiente descendente.



X.head()



	Alcohol	Malicacid	Ash	Alcalinity_of_ash	Magnesium	Total_phenols	Flavanoids
0	14.23	1.71	2.43	15.6	127	2.80	3.06
1	13.20	1.78	2.14	11.2	100	2.65	2.76
2	13.16	2.36	2.67	18.6	101	2.80	3.24
3	14.37	1.95	2.50	16.8	113	3.85	3.49
4	13.24	2.59	2.87	21.0	118	2.80	2.69



Los datos parecen estar bien estandarizados y no tienen ningún valor que falte, por lo que en principio parece que no es necesario ningún preprocesado.



✓ 1.5 Análisis de correlación de atributos

Cargue la base de datos elegida y use el algoritmo CorrelationAttributeEval + Ranker (si es problema de regresión) o InfoGainAttributeEval + Ranker (si es problema de clasificación). Estos algoritmos indican un ranking de atributos con respecto a la variable de salida en función del coeficiente de correlación de Pearson o de la ganancia de información respectivamente. Use por defecto el conjunto de entrenamiento, que es la opción aplicada ("Use full training set"). ¿Qué atributos influyen más sobre la salida y cuáles menos? Se valorará si estudia el tipo de problema más a fondo y encuentra si tiene sentido o no la mayor o menor influencia de determinados atributos según muestra Weka.

```
from sklearn.datasets import load_wine
from sklearn.feature_selection import SelectKBest, mutual_info_classif
import numpy as np

# Seleccionar los K mejores atributos utilizando la ganancia de información
selector = SelectKBest(score_func=mutual_info_classif, k='all')
selector.fit(X, y)

# Obtener el ranking de atributos y los nombres de los atributos
scores = selector.scores_
feature_names = X.columns

# Crear un índice ordenado según los puntajes en orden descendente
sorted_index = np.argsort(scores)[::-1]

# Mostrar el ranking de atributos ordenados
print("Ranking de atributos (ordenado por score):")
for i, idx in enumerate(sorted_index):
    print(f"Atributo {i + 1} ({feature_names[idx]}): {scores[idx]}")
```

```
Ranking de atributos (ordenado por score):
Atributo 1 (Flavanoids): 0.6615335382491021
Atributo 2 (Proline): 0.5579224953256927
Atributo 3 (Color_intensity): 0.5519286728916415
Atributo 4 (OD280_OD315_of_diluted_wines): 0.5001644343819183
Atributo 5 (Alcohol): 0.47019369649692266
Atributo 6 (Hue): 0.45500137397121465
Atributo 7 (Total_phenols): 0.43260510526189466
Atributo 8 (Proanthocyanins): 0.3197170224999555
Atributo 9 (Malicacid): 0.2835455014881434
Atributo 10 (Alcalinity_of_ash): 0.2635283095630512
Atributo 11 (Magnesium): 0.21614974081596783
Atributo 12 (Nonflavanoid_phenols): 0.13762393886259838
Atributo 13 (Ash): 0.07709955186177297
/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:1143: DataConversionWarning: A column-vector y was passed when a
y = column_or_1d(y, warn=True)
```

El ranking de atributos muestra que los flavanoids son el atributo más importante con un puntaje de 0.67, seguido por el contenido de proline y la color_intensity. Estos tres atributos tienen puntajes significativamente más altos que los demás, lo que sugiere que tienen una fuerte influencia en la variable objetivo. Por otro lado, atributos como la ash y los nonflavanoid_phenols tienen puntajes mucho más bajos, lo que indica una menor importancia en la predicción del objetivo.

Estos resultados proporcionan información sobre qué atributos son más relevantes para la clasificación en este conjunto de datos específico.

✓ 1.6 Normalización o estandarización

Al examinar las estadísticas descriptivas de las variables, vemos que varían en diversos rangos de valores:

- Alcohol en [11.03, 14.83]
- Acido_Málico en [0.74, 5.8]
- Ceniza en [1.36, 3.23]
- Alcalinidad_de_la_ceniza en [10.6, 30.0]
- Magnesio en [70.0, 162.0]
- Fenoles_totales en [0.98, 3.88]
- Flavonoides en [0.34, 5.08]
- Fenoles_no_flavonoides en [0.13, 0.66]
- Proantocianinas en [0.41, 3.58]
- Intensidad_de_color en [1.28, 13.00]
- Matiz en [0.48, 1.71]
- OD280_OD315_de_vinos_diluidos en [1.27, 4.0]
- Prolina en [278.0, 1680.0]

Las escalas de tus características varían significativamente (por ejemplo, el Alcohol varía de 11.03 a 14.83, mientras que la Prolina varía de 278.0 a 1680.0). Para manejar diferentes escalas, elegimos aplicar la estandarización, que también se conoce como normalización Z-score, la cual transforma los datos para que tengan una media de 0 y una desviación estándar de 1. Considerando que vamos a utilizar una regresión logística para la tarea de clasificación más adelante, se selecciona la estandarización por tres razones: la regresión logística a menudo funciona mejor cuando las características están estandarizadas para seguir una distribución normal, aunque no requiere que las características sigan una distribución normal. La estandarización asegura que cada característica contribuya igualmente al modelo. Además, reduce el efecto de los valores atípicos en el modelo que se entrenará posteriormente.

```
# The initial features
X.head()
```

	Alcohol	Malicacid	Ash	Alcalinity_of_ash	Magnesium	Total_phenols	Flavanoids
0	14.23	1.71	2.43	15.6	127	2.80	3.06
1	13.20	1.78	2.14	11.2	100	2.65	2.76
2	13.16	2.36	2.67	18.6	101	2.80	3.24
3	14.37	1.95	2.50	16.8	113	3.85	3.49
4	13.24	2.59	2.87	21.0	118	2.80	2.69

```
cols=X.columns
print(cols)
```

```
Index(['Alcohol', 'Malicacid', 'Ash', 'Alcalinity_of_ash', 'Magnesium',
      'Total_phenols', 'Flavanoids', 'Nonflavanoid_phenols',
      'Proanthocyanins', 'Color_intensity', 'Hue',
      '0D280_0D315_of_diluted_wines', 'Proline'],
      dtype='object')
```

```
from sklearn.preprocessing import StandardScaler
# Standardizing the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
# Creating a DataFrame with the scaled features
scaled_X = pd.DataFrame(X_scaled, columns=X.columns)
# The features scaled
scaled_X.head()
```

	Alcohol	Malicacid	Ash	Alcalinity_of_ash	Magnesium	Total_phenols	Flavanoids
0	1.518613	-0.562250	0.232053	-1.169593	1.913905	0.808997	1.034
1	0.246290	-0.499413	-0.827996	-2.490847	0.018145	0.568648	0.733
2	0.196879	0.021231	1.109334	-0.268738	0.088358	0.808997	1.211
3	1.691550	-0.346811	0.487926	-0.809251	0.930918	2.491446	1.461
4	0.295700	0.227694	1.840403	0.451946	1.281985	0.808997	0.661

Observamos que los valores de las características son insignificativamente diferentes después de la estandarización.

✓ 1.7 Entrenamiento y prueba del modelo predictivo

Entrenamos un modelo de regresión logística utilizando una validación cruzada estratificada de 10 pliegues para asegurar una evaluación robusta. Luego, obtuvimos predicciones validadas cruzadamente y calculamos métricas de rendimiento, incluyendo la precisión (CCR), la sensibilidad (recall) para cada clase y el área bajo la curva ROC (ROC AUC). Finalmente, resumimos estas métricas para evaluar el rendimiento general del modelo.

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score, cross_val_predict, StratifiedKFold, cross_validate
from sklearn.metrics import confusion_matrix, roc_auc_score, roc_curve, classification_report
```



```
# Initialize the Logistic Regression model
model = LogisticRegression(random_state=42, multi_class='ovr', solver='liblinear')
y_r=y.values.ravel()

# Perform 10-fold cross-validation
cv = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)

# Get cross-validated predictions
y_pred = cross_val_predict(model, X_scaled, y, cv=cv, method="predict")
y_pred_prob = cross_val_predict(model, X_scaled, y_r, cv=cv, method="predict_proba")

# Calculate cross-validated scores
ccr_scores = cross_val_score(model, X_scaled, y, cv=cv, scoring='accuracy')
roc_auc = cross_val_score(model, X_scaled, y_r, cv=cv, scoring='roc_auc_ovr')

# Calculate confusion matrix
conf_matrix = confusion_matrix(y, y_pred)

# Calculate sensitivity (TP Rate) for each class
class_report = classification_report(y, y_pred, output_dict=True)
sensitivity = {cls: metrics['recall'] for cls, metrics in class_report.items() if cls.isdigit()}

# Calculate the overall Correctly Classified Instances (CCR)
ccr = np.mean(ccr_scores)
roc = np.mean(roc_auc)

# The results:
# The accuracy = Correctly Classified Instances
print(f'Correctly Classified Instances (CCR): {ccr:.2f}')
```

```
→ /usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:1143: DataConversionWarning: A column-vector y was passed when a
    y = column_or_1d(y, warn=True)
/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:1143: DataConversionWarning: A column-vector y was passed when a
    y = column_or_1d(y, warn=True)
/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:1143: DataConversionWarning: A column-vector y was passed when a
    y = column_or_1d(y, warn=True)
/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:1143: DataConversionWarning: A column-vector y was passed when a
    y = column_or_1d(y, warn=True)
/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:1143: DataConversionWarning: A column-vector y was passed when a
    y = column_or_1d(y, warn=True)
/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:1143: DataConversionWarning: A column-vector y was passed when a
    y = column_or_1d(y, warn=True)
/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:1143: DataConversionWarning: A column-vector y was passed when a
    y = column_or_1d(y, warn=True)
/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:1143: DataConversionWarning: A column-vector y was passed when a
    y = column_or_1d(y, warn=True)
/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:1143: DataConversionWarning: A column-vector y was passed when a
    y = column_or_1d(y, warn=True)
/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:1143: DataConversionWarning: A column-vector y was passed when a
    y = column_or_1d(y, warn=True)
/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:1143: DataConversionWarning: A column-vector y was passed when a
    y = column_or_1d(y, warn=True)
/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:1143: DataConversionWarning: A column-vector y was passed when a
    y = column_or_1d(y, warn=True)
/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:1143: DataConversionWarning: A column-vector y was passed when a
    y = column_or_1d(y, warn=True)
/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:1143: DataConversionWarning: A column-vector y was passed when a
    y = column_or_1d(y, warn=True)
/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:1143: DataConversionWarning: A column-vector y was passed when a
    y = column_or_1d(y, warn=True)
/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:1143: DataConversionWarning: A column-vector y was passed when a
    y = column_or_1d(y, warn=True)
/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:1143: DataConversionWarning: A column-vector y was passed when a
    y = column_or_1d(y, warn=True)
/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:1143: DataConversionWarning: A column-vector y was passed when a
    y = column_or_1d(y, warn=True)
/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:1143: DataConversionWarning: A column-vector y was passed when a
    y = column_or_1d(y, warn=True)
/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:1143: DataConversionWarning: A column-vector y was passed when a
    y = column_or_1d(y, warn=True)
Correctly Classified Instances (CCR): 0.99
Sensitivity (TP Rate) by Class: {'1': 1.0, '2': 0.971830985915493, '3': 1.0}
Area Under the ROC Curve (ROC AUC): 1.00
```

👉 El modelo logró una alta precisión con una tasa de Instancias Correctamente Clasificadas (CCR) del 99%, lo que indica que predijo correctamente las etiquetas de clase para el 99% de las instancias. Los valores de sensibilidad (recall) son casi perfectos para todas las clases, demostrando un excelente rendimiento en la identificación correcta de cada clase, con la Clase 2 teniendo un recall ligeramente más bajo de aproximadamente 97.2%. El Área Bajo la Curva ROC (ROC AUC) perfecta de 1.00 sugiere que el modelo tiene una capacidad sobresaliente para distinguir entre las clases.

```

# Import necessary libraries
from sklearn.ensemble import RandomForestClassifier
import xgboost as xgb

# Adjust labels to start from 0
y_adjusted = y - 1

# Define function to evaluate model
def evaluate_model(model, X_scaled, y):
    y_r = y.values.ravel()
    cv = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)

    y_pred = cross_val_predict(model, X_scaled, y, cv=cv, method="predict")
    ccr_scores = cross_val_score(model, X_scaled, y, cv=cv, scoring='accuracy')
    roc_auc = cross_val_score(model, X_scaled, y_r, cv=cv, scoring='roc_auc_ovr')

    conf_matrix = confusion_matrix(y, y_pred)
    class_report = classification_report(y, y_pred, output_dict=True)
    sensitivity = {cls: metrics['recall'] for cls, metrics in class_report.items() if cls.isdigit()}


    ccr = np.mean(ccr_scores)
    roc = np.mean(roc_auc)

    print(f'Correctly Classified Instances (CCR): {ccr:.2f}')
    print('Sensitivity (TP Rate) by Class:', sensitivity)
    print(f'Area Under the ROC Curve (ROC AUC): {roc:.2f}')
    return conf_matrix, sensitivity, ccr, roc

# Train and evaluate XGBoost
xgb_model = xgb.XGBClassifier(random_state=42, use_label_encoder=False, eval_metric='mlogloss')
print("XGBoost Results:")
xgb_conf_matrix, xgb_sensitivity, xgb_ccr, xgb_roc = evaluate_model(xgb_model, X_scaled, y_adjusted)

# Train and evaluate Random Forest
rf_model = RandomForestClassifier(random_state=42)
print("\nRandom Forest Results:")
rf_conf_matrix, rf_sensitivity, rf_ccr, rf_roc = evaluate_model(rf_model, X_scaled, y_adjusted)

```

 XGBoost Results:
 Correctly Classified Instances (CCR): 0.96
 Sensitivity (TP Rate) by Class: {'0': 0.9661016949152542, '1': 0.9436619718309859, '2': 0.9791666666666666}
 Area Under the ROC Curve (ROC AUC): 0.99

Random Forest Results:
 /usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_validation.py:1068: DataConversionWarning: A column-vector y was passed as a 1D array, which will be flattened before indexing. Please use y.reshape(-1) instead.
 estimator.fit(X_train, y_train, **fit_params)
 /usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_validation.py:1068: DataConversionWarning: A column-vector y was passed as a 1D array, which will be flattened before indexing. Please use y.reshape(-1) instead.
 estimator.fit(X_train, y_train, **fit_params)
 /usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_validation.py:1068: DataConversionWarning: A column-vector y was passed as a 1D array, which will be flattened before indexing. Please use y.reshape(-1) instead.
 estimator.fit(X_train, y_train, **fit_params)
 /usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_validation.py:1068: DataConversionWarning: A column-vector y was passed as a 1D array, which will be flattened before indexing. Please use y.reshape(-1) instead.
 estimator.fit(X_train, y_train, **fit_params)
 /usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_validation.py:1068: DataConversionWarning: A column-vector y was passed as a 1D array, which will be flattened before indexing. Please use y.reshape(-1) instead.
 estimator.fit(X_train, y_train, **fit_params)