Pipelines Sklearn

Rommel Lopez
Jose Luis Padilla
Rodrigo Álvarez
Toni Santacruz
Guillermo Castillón

Marzo 2025

Creación del entorno virtual

```
PS C:\Users\rodri\Git_Repositories\Fontaneros\src\result_notebooks> python -m venv fontaneros_entorno
```

```
PS C:\Users\rodri\Git_Repositories\Fontaneros\src\result_notebooks> Set-ExecutionPolicy Unrestricted -Scope Process
PS C:\Users\rodri\Git_Repositories\Fontaneros\src\result_notebooks> .\fontaneros_entorno\Scripts\Activate.ps1
(fontaneros_entorno) PS C:\Users\rodri\Git_Repositories\Fontaneros\src\result_notebooks>
```

```
(fontaneros_entorno) PS C:\Users\rodri\Git_Repositories\Fontaneros\src\result_notebooks> .\fontaneros_entorno\Scripts\python --version
Python 3.11.3
```

```
(fontaneros_entorno) PS C:\Users\rodri\Git_Repositories\Fontaneros\src\result_notebooks> pip freeze > requirements.txt
```

Creación del entorno virtual

```
1  !pip list
✓ 3.6s
```

Package	Version
absl-py	2.1.0
aiohappyeyeballs	2.4.6
aiohttp	3.11.12
aiosignal	1.3.2
anyio	4.6.2.post1
argon2-cffi	23.1.0
argon2-cffi-bindings	21.2.0
arrow	1.3.0
asttokens	2.4.1
astunparse	1.6.3
async-lru	2.0.4
attrs	24.2.0
babel	2.16.0
beautifulsoup4	4.12.3
bleach	6.1.0
catboost	1.2.7
category_encoders	2.8.0
certifi	2024.8.30
cffi	1.17.1
charset-normalizer	3.4.0
click	8.1.8
cloudpickle	3.1.1
colorama	0.4.6
comm	0.2.2
contourpy	1.3.0
cycler	0.12.1
Cython	3.0.12
datasets	3.2.0

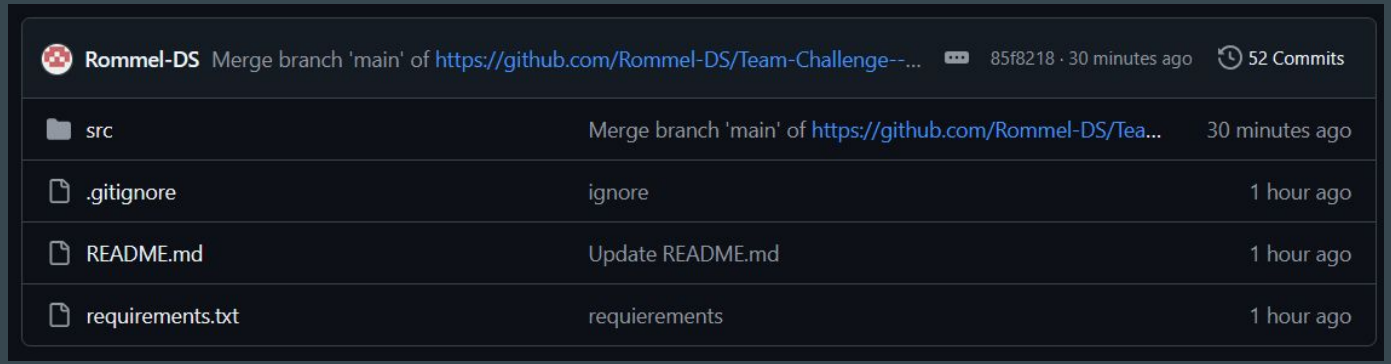
```
(fontaneros_entorno) PS C:\Users\rodri\Git_Repositories\Fontaneros\src\result_notebooks> pip install -r requirements.txt
```

Creación del entorno virtual

```
(fontaneros_entorno) PS C:\Users\rodri\Git_Repositories\Fontaneros\src\result_notebooks> New-Item .gitignore -ItemType File
```

.gitignore

```
# Ignorar el entorno virtual
fontaneros_entorno/

# Archivos de Jupyter Notebook checkpoints
.ipynb_checkpoints/

# Configuración de entornos virtuales en general
venv/
env/

# Ignorar archivos de configuración de entorno
.env
.envrc

```



Imports

```python
import pandas as pd
import numpy as np
import sys
sys.path.append("../utils/")
import Toolbox as tb
from Toolbox import *
import pickle # sirve para guardar cualquier objeto binario, inclusi el pipline
from sklearn.ensemble import RandomForestRegressor
from xgboost import XGBRegressor
from lightgbm import LGBMRegressor
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import FunctionTransformer
from sklearn.pipeline import Pipeline, make_pipeline
from sklearn.compose import ColumnTransformer, make_column_selector
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from lightgbm import LGBMClassifier
from sklearn.model_selection import train_test_split
import os
```

✓ 18.2s

Carga de los datos

```python
# Cargar el archivo CSV
data = pd.read_csv("../data/wines_dataset.csv", sep="|")

"""
División de datos en entrenamiento y prueba.

Se toma el 80% de los datos para entrenamiento y el 20% restante para prueba.

Parámetros:
- test_size (float): Proporción del conjunto de prueba (0.2 = 20% de los datos).
- random_state (int): Semilla para la reproducibilidad de la división.

Salida:
- train (DataFrame): Conjunto de entrenamiento.
- test (DataFrame): Conjunto de prueba.
"""
train, test = train_test_split(data, test_size=0.2, random_state=42)

"""
Guardado de los conjuntos de datos en archivos CSV.

Los archivos se almacenan en la carpeta "./data/" con los nombres:
- wines_train.csv → Contiene el 80% de los datos para entrenamiento.
- wines_test.csv → Contiene el 20% de los datos para prueba.

index=False evita que se guarde el índice en los archivos.
"""
# Crear el directorio si no existe
os.makedirs("./data/", exist_ok=True)

train.to_csv(os.path.join("./data/", 'wines_train.csv'), index=False)
test.to_csv(os.path.join("./data/", 'wines_test.csv'), index=False)
```

Train y Test

```python
# Cargar el conjunto de datos de prueba desde el archivo CSV
df_test = pd.read_csv("./data/wines_test.csv")

# Mostrar el DataFrame cargado
df_test
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality	class
0	7.4	0.320	0.27	1.4	0.049	38.0	173.0	0.99335	3.03	0.52	9.3	5	white
1	6.6	0.340	0.24	3.3	0.034	29.0	99.0	0.99031	3.10	0.40	12.3	7	white
2	6.4	0.320	0.35	4.8	0.030	34.0	101.0	0.99120	3.36	0.60	12.5	8	white
3	6.8	0.230	0.32	1.6	0.026	43.0	147.0	0.99040	3.29	0.54	12.5	6	white
4	6.7	0.340	0.26	1.9	0.038	58.0	138.0	0.98930	3.00	0.47	12.2	7	white
...	...	...	...	...	...	...	...	...	...	...	...	...	...
1295	7.6	0.285	0.32	14.6	0.063	32.0	201.0	0.99800	3.00	0.45	9.2	5	white
1296	11.6	0.470	0.44	1.6	0.147	36.0	51.0	0.99836	3.38	0.86	9.9	4	red
1297	10.2	0.340	0.48	2.1	0.052	5.0	9.0	0.99458	3.20	0.69	12.1	7	red
1298	6.2	0.460	0.17	1.6	0.073	7.0	11.0	0.99425	3.61	0.54	11.4	5	red
1299	6.2	0.360	0.14	8.9	0.036	38.0	155.0	0.99622	3.27	0.50	9.4	5	white

1300 rows × 13 columns

```python
# Cargar el conjunto de datos de entrenamiento desde el archivo CSV
df_train = pd.read_csv("./data/wines_train.csv")

# Mostrar el DataFrame cargado
df_train
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality	class
0	7.9	0.18	0.40	2.20	0.049	38.0	67.0	0.99600	3.33	0.93	11.3	5	red
1	7.1	0.18	0.74	15.60	0.044	44.0	176.0	0.99960	3.38	0.67	9.0	6	white
2	7.6	0.51	0.24	1.20	0.040	10.0	104.0	0.99200	3.05	0.29	10.8	6	white
3	6.0	0.25	0.28	7.70	0.053	37.0	132.0	0.99489	3.06	0.50	9.4	6	white
4	9.0	0.38	0.41	2.40	0.103	6.0	10.0	0.99604	3.13	0.58	11.9	7	red
...	...	...	...	...	...	...	...	...	...	...	...	...	...
5192	6.4	0.24	0.50	11.60	0.047	60.0	211.0	0.99660	3.18	0.57	9.3	5	white
5193	6.6	0.22	0.28	12.05	0.058	25.0	125.0	0.99856	3.45	0.45	9.4	5	white
5194	6.6	0.20	0.38	7.90	0.052	30.0	145.0	0.99470	3.32	0.56	11.0	7	white
5195	7.3	0.41	0.29	1.80	0.032	26.0	74.0	0.98889	2.96	0.35	13.0	8	white
5196	5.9	0.18	0.28	5.10	0.039	50.0	139.0	0.99165	3.16	0.44	11.3	6	white

Targets

```
# Definición de variables objetivo para el análisis:

# target_clf = "quality" → Problema de Clasificación
#     - La variable `quality` representa la calidad del vino.
#     - Es una variable categórica con 7 valores únicos (multiclase).
#     - Se utilizarán modelos de clasificación, como RandomForestClassifier o XGBoost.
#     - Es necesario aplicar OneHotEncoder a variables categóricas y StandardScaler a las numéricas.

# target_reg = "alcohol" → Problema de Regresión
#     - La variable `alcohol` indica el porcentaje de alcohol en el vino.
#     - Es una variable numérica continua.
#     - Se utilizarán modelos de regresión, como RandomForestRegressor o LinearRegression.
#     - Requiere escalado de las variables numéricas (StandardScaler) y codificación de las categóricas (OneHotEncoder).

# Ambos problemas requieren preprocesamiento adecuado según el tipo de modelo utilizado.

target_clf = "quality"
y_train_cat = df_train["quality"]
y_test_cat = df_test["quality"]
target_reg = "alcohol"
y_train_reg = df_train["alcohol"]
y_test_reg = df_test["alcohol"]

```

Descripción de los datos

```python
# Genera un resumen descriptivo del DataFrame utilizando la función tb.describe_df(df).
#
#  Muestra información sobre:
#    - DATA_TYPE: Tipo de dato de cada columna (numérico, categórico, etc.).
#    - MISSINGS (%): Porcentaje de valores nulos en cada columna.
#    - UNIQUE_VALUES: Cantidad de valores únicos en cada columna.
#    - CARDIN (%): Cardinalidad relativa (valores únicos / total de filas).
#
# Útil para comprender la estructura de los datos antes del preprocesamiento.

tb.describe_df(df_train)
```

COL_N	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality	class
DATA_TYPE	float64	float64	float64	float64	float64	float64	float64	float64	float64	float64	float64	int64	object
MISSINGS (%)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
UNIQUE_VALUES	100	177	87	308	194	132	274	951	106	106	102	7	2
CARDIN (%)	0.02	0.03	0.02	0.06	0.04	0.03	0.05	0.18	0.02	0.02	0.02	0.0	0.0

Analizar el problema

Clasificación

Problema negocio: Buscamos un modelo para clasificar vinos.

Enfoque técnico: Disponemos de un registro de 5.197 vinos con 12 variables y clasificados en 7 categorías para crear nuestro modelo de clasificación multiclase. Aplicaremos nuestro modelo para clasificar a 1.300 vinos nuevos y como disponemos de la clasificación real podremos evaluar su validez.

Regresión

Problema de negocio: Buscamos un modelo para predecir el porcentaje de alcohol en los vinos.

Enfoque técnico: Contamos con un conjunto de datos de vinos que incluye información como la acidez, la cantidad de azúcar y el nivel de pH. Con estos datos, entrenaremos un modelo que nos permitirá predecir el porcentaje de alcohol en los vinos. Luego, probaremos el modelo con vinos nuevos y compararemos los resultados con los valores reales para verificar qué tan preciso es.

Clasificación

En el análisis exploratorio hemos determinado nuestro target y las variables numéricas y categóricas relevantes para nuestro modelo:

```
# Definición del problema de clasificación.

# target_clf → Variable objetivo para clasificación.
#     - Se ha definido previamente como "quality".
#     - Representa la calidad del vino en una escala categórica.
#     - Es un problema de clasificación **multiclase** (7 categorías posibles).
target_clf
```

```
'quality'
```

```
# Definición de las características categóricas para el problema de clasificación.

# features_cat_clf → Variables categóricas a incluir en el modelo de clasificación.
#     - Contiene la variable "class" (tipo de vino: tinto o blanco).
#     - Se debe transformar con OneHotEncoder para convertirla en variables numéricas.
features_cat_clf
```

```
['class']
```

```
# Definición de características numéricas principales para clasificación.

# features_num_clf_1 → Variables numéricas seleccionadas para el modelo de clasificación.
#     - Incluye medidas químicas clave del vino:
#       - "volatile acidity", "citric acid", "chlorides", "free sulfur dioxide",
#       - "total sulfur dioxide", "density", "pH", "sulphates", "alcohol".
#     - Se recomienda aplicar StandardScaler para normalizar estas características antes del modelado.
features_num_clf_1
```

```
['volatile acidity',
 'citric acid',
 'chlorides',
 'free sulfur dioxide',
 'total sulfur dioxide',
 'density',
 'pH',
 'sulphates',
 'alcohol']
```

Clasificación

```
# Definición de columnas a incluir y excluir en el modelo de clasificación.

# columns_to_keep_clf → Columnas que se mantendrán en el modelo de clasificación.
#     - Incluye:
#       - features_num_clf_1 (variables numéricas relevantes).
#       - features_cat_clf (variables categóricas a transformar con OneHotEncoder).

# columns_to_exclude_clf → Columnas que se excluirán del modelo de clasificación.
#     - Se obtienen eliminando de df.columns las variables incluidas en columns_to_keep_clf.
#     - Estas columnas no serán utilizadas en el modelo.
```

```
columns_to_keep_clf =  features_num_clf_1 + features_cat_clf

columns_to_exclude_clf = [col for col in df_train.columns if col not in columns_to_keep_clf]

columns_to_exclude_clf
```

Clasificación: Preprocesamiento datos

```
# Definición de Pipelines para preprocesamiento de datos en clasificación.

# cat_pipeline → Preprocesamiento de variables categóricas.
#    - "Impute_Mode": Imputa valores faltantes con la moda (valor más frecuente).
#    - "OHEncoder": Aplica OneHotEncoder, ignorando categorías desconocidas en lugar que generar un error

# logaritmica → Transformación logarítmica de variables numéricas.
#    - Usa FunctionTransformer con np.log1p para estabilizar distribuciones sesgadas.
#    - feature_names_out="one-to-one" mantiene los nombres originales de las características.

# num_pipeline → Preprocesamiento de variables numéricas.
#    - "Impute_Mean": Imputa valores faltantes con la media.
#    - "logaritmo": Aplica la transformación logarítmica definida antes.
#    - "SScaler": Aplica StandardScaler para normalizar las variables numéricas.

# imputer_step_cat → ColumnTransformer para aplicar los Pipelines según el tipo de variable.
#    - "Process_Numeric": Aplica num_pipeline a features_num_clf_1 (variables numéricas).
#    - "Process_Categorical": Aplica cat_pipeline a features_cat_clf (variables categóricas).
#    - "Exclude": Elimina las columnas en columns_to_exclude_clf.
#    - remainder="passthrough": Mantiene cualquier otra columna sin modificar.

# pipe_missings_cat → Pipeline final que aplica el ColumnTransformer imputer_step_cat.
```

```
cat_pipeline = Pipeline(
    [("Impute_Mode", SimpleImputer(strategy="most_frequent")),
     ("OHEncoder", OneHotEncoder(handle_unknown='ignore'))
    ]
)

logaritmica = FunctionTransformer(np.log1p, feature_names_out="one-to-one")

num_pipeline = Pipeline(
    [("Impute_Mean", SimpleImputer(strategy = "mean")),
     ("logaritmo", logaritmica),
     ("SScaler", StandardScaler()),
    ]
)

imputer_step_cat = ColumnTransformer(
    [("Process_Numeric", num_pipeline,features_num_clf_1),
     ("Process_Categorical", cat_pipeline, features_cat_clf),
     ("Exclude", "drop", columns_to_exclude_clf)
    ], remainder = "passthrough"
    )

pipe_missings_cat = Pipeline([("first_stage", imputer_step_cat)])
```

Clasificación: evaluación modelos baseline

```
# Definición de Pipelines para modelos de clasificación.
#
# - Se crean pipelines que combinan preprocesamiento y modelos de clasificación.
# - Cada pipeline aplica primero `pipe_missings_cat`, que maneja datos faltantes y
# codificación de variables categóricas.
# - Luego, se entrena un modelo de clasificación diferente en cada pipeline:
#    - LogisticRegression (Regresión Logística)
#    - RandomForestClassifier (Bosques Aleatorios)
#    - LGBMClassifier (LightGBM)
#
# - Finalmente, se evalúan los modelos con validación cruzada usando "accuracy" como métrica.
```

```
logistic: 0.3323
[0.31826923 0.325      0.35226179 0.32435034 0.34167469]
randomF: 0.6644
[0.67019231 0.67115385 0.66313763 0.65928778 0.65832531]
LGBM: 0.6173
[0.61634615 0.59903846 0.62271415 0.62560154 0.62271415]
```

```python
# Pipeline con Regresión Logística
logistic_pipeline = Pipeline(
    [("Preprocesado", pipe_missings_cat),  # Paso de preprocesamiento
     ("Modelo", LogisticRegression(max_iter=10000, class_weight="balanced"))
    ])

# Pipeline con RandomForestClassifier
random_pipeline = Pipeline(
    [("Preprocesado", pipe_missings_cat),
     ("Modelo", RandomForestClassifier(class_weight="balanced"))
    ])


# Pipeline con LightGBMClassifier
LGBM_pipeline = Pipeline(
    [("Preprocesado", pipe_missings_cat),
     ("Modelo", LGBMClassifier(verbose=-1, class_weight="balanced"))
    ])

# Evaluación de los modelos con validación cruzada
for name, pipe in zip(["logistic","randomF","LGBM"],
                      [logistic_pipeline, random_pipeline, LGBM_pipeline]):
    resultado = cross_val_score(pipe, df_train, y_train_cat, cv=5, scoring="accuracy")

    # Mostrar el desempeño de cada modelo
    print(f"{name}: {np.mean(resultado):.4f}")
    print(resultado)
```

Clasificación: optimización hiper parámetros

```
# Evaluación de hiperparámetros para modelos de clasificación con GridSearchCV.
#
# - Se definen diferentes conjuntos de hiperparámetros para los modelos:
#   - Regresión Logística (LogisticRegression)
#   - Random Forest (RandomForestClassifier)
#   - LightGBM (LGBMClassifier)
#
# - Se aplica GridSearchCV para encontrar la mejor combinación de hiperparámetros.
# - Se usa validación cruzada con 5 particiones (cv=5) y "accuracy" como métrica de
# evaluación.
# - Se almacena cada búsqueda en un diccionario `pipe_grids_cat`.

# Definimos sus hiperparametros
reg_log_param = {"Modelo__penalty": [None,"l2"],
                 "Modelo__C": np.logspace(0, 4, 10),
                 "Modelo__class_weight": [None,"balanced"]}

rand_forest_param = {
    'Modelo__n_estimators': [10, 100, 200, 400],
    'Modelo__max_depth': [None,1,2,4,8],
    'Modelo__max_features': ['sqrt', 1, 2, 3],
    'Modelo__class_weight': [None,'balanced']}

param_grid_lgbm = {
    'Modelo__num_leaves': [15, 31, 50],
    'Modelo__learning_rate': [0.01, 0.05, 0.1],
    'Modelo__n_estimators': [50, 100, 200],
    'Modelo__max_depth': [-1,5, 10, 15],
    'Modelo__class_weight': [None,'balanced']}

cv = 5
```

```
gs_reg_log = GridSearchCV(logistic_pipeline,
                          reg_log_param,
                          cv=cv,
                          scoring="accuracy",
                          verbose=1,
                          n_jobs=-1)

gs_rand_forest = GridSearchCV(random_pipeline,
                              rand_forest_param,
                              cv=cv,
                              scoring="accuracy",
                              verbose=1,
                              n_jobs=-1)

gs_lgb = GridSearchCV(LGBM_pipeline,
                      param_grid_lgbm,
                      cv=cv,
                      scoring="accuracy",
                      verbose=1,
                      n_jobs=-1)

pipe_grids_cat = {"gs_reg_log":gs_reg_log,
                  "gs_rand_forest":gs_rand_forest,
                  "gs_lgb":gs_lgb}
```

Clasificación: optimización hiper parámetros II

```python
# Ejecución de GridSearchCV para cada modelo de clasificación.
#
# - Se entrena cada búsqueda de hiperparámetros con los datos de entrenamiento (df_train, y_train_cat).
# - GridSearchCV explorará todas las combinaciones de hiperparámetros definidas previamente.
# - Se utiliza validación cruzada para evaluar el rendimiento de cada combinación.
# - Los mejores modelos serán seleccionados para su uso posterior.

for nombre, grid_search in pipe_grids_cat.items():
    print(f"Entrenando GridSearch para {nombre}...")  # Mensaje informativo
    grid_search.fit(df_train, y_train_cat)  # Ajusta el modelo con la búsqueda de hiperparámetros
    print(f"Finalizado: {nombre}")  # Mensaje de finalización
    print(f"Mejores parámetros para {nombre}: {grid_search.best_params_}")  # Imprime los mejores hiperparámetros encontrados
    print(f"Mejor score para {nombre}: {grid_search.best_score_:.4f}")  # Muestra el mejor puntaje obtenido
    print("-" * 50)  # Separador para mayor claridad
```

```
Entrenando GridSearch para gs_reg_log...
Fitting 5 folds for each of 40 candidates, totalling 200 fits
Finalizado: gs_reg_log
Mejores parámetros para gs_reg_log: {'Modelo__C': 59.94842503189409, 'Modelo__class_weight': None, 'Modelo__penalty': 'l2'}
Mejor score para gs_reg_log: 0.5472
--------------------------------------------------
Entrenando GridSearch para gs_rand_forest...
Fitting 5 folds for each of 160 candidates, totalling 800 fits
Finalizado: gs_rand_forest
Mejores parámetros para gs_rand_forest: {'Modelo__class_weight': 'balanced', 'Modelo__max_depth': None, 'Modelo__max_features': 'sqrt', 'Modelo__n_estimators': 100}
Mejor score para gs_rand_forest: 0.6706
--------------------------------------------------
Entrenando GridSearch para gs_lgb...
Fitting 5 folds for each of 216 candidates, totalling 1080 fits
Finalizado: gs_lgb
Mejores parámetros para gs_lgb: {'Modelo__class_weight': None, 'Modelo__learning_rate': 0.1, 'Modelo__max_depth': 15, 'Modelo__n_estimators': 200, 'Modelo__num_leaves': 50}
Mejor score para gs_lgb: 0.6533
--------------------------------------------------
```

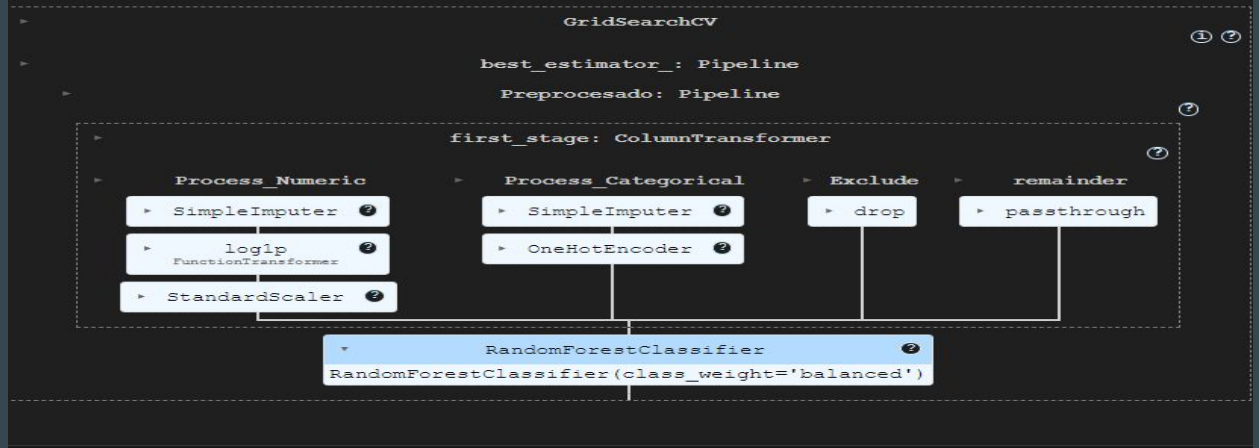
Clasificación: selección modelo

```python
# Evaluación de los mejores modelos de clasificación
#
# - Se extrae el mejor puntaje (best_score_) de cada modelo tras la búsqueda de hiperparámetros.
# - Se almacena en un DataFrame para facilitar la comparación.
# - Se ordenan los modelos en función de su desempeño, de mayor a menor.

best_grids_cat = [(i, j.best_score_) for i, j in pipe_grids_cat.items()]  # Extrae el mejor puntaje de cada modelo

best_grids_cat = pd.DataFrame(best_grids_cat, columns=["Grid", "Best score"]).sort_values(by="Best score", ascending=False)

best_grids_cat  # Muestra los resultados
```

	Grid	Best score
1	gs_rand_forest	0.670577
2	gs_lgb	0.653261
0	gs_reg_log	0.547240

Clasificación: guardamos el modelo elegido

```python
# Selección del mejor modelo de clasificación
#
# - Se extrae el modelo con mejor rendimiento según la evaluación de GridSearchCV.
# - Se utiliza la primera fila del DataFrame `best_grids_cat`, que ya está ordenado por desempeño.
# - Este modelo se almacenará en la variable `best_model_cat` para su posterior uso o guardado.
best_model_cat = pipe_grids_cat[best_grids_cat.iloc[0, 0]]  # Extrae el mejor modelo
best_model_cat  # Muestra el modelo seleccionado
```




```python
# Guardado del mejor modelo de clasificación en formato pickle
#
# - Se asegura de que el directorio `src/models` exista para almacenar el modelo.
# - Se guarda el modelo en un archivo .pkl para su uso posterior.
# - El modelo guardado podrá ser cargado y utilizado sin necesidad de volver a entrenarlo.

# Asegurar que el directorio de modelos exista
os.makedirs('src/models', exist_ok=True)

# Guardar el mejor modelo en formato pickle
with open('src/models/modelo_pipeline_cat.pkl', 'wb') as archivo:
    pickle.dump(best_model_cat, archivo)  # Guarda el modelo en el archivo
```

Clasificación: aplicación a test del modelo

```
target_clf = "quality"

y_test_clf = df_test["quality"]
```

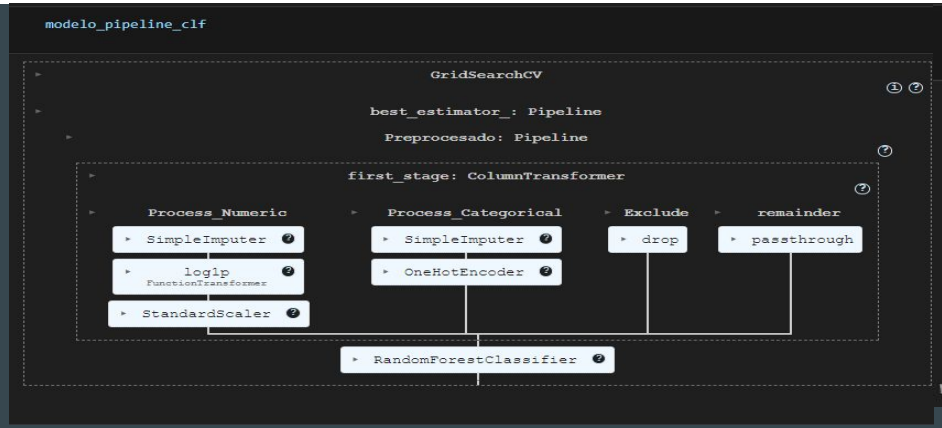
```
# Recuperamos el modelo de pipelines (version pickle)
with open('src/models/modelo_pipeline_cat.pkl', 'rb') as archivo: # ojo read binario
    modelo_pipeline_clf = pickle.load(archivo)
```

```
df_test
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality	class
0	7.4	0.320	0.27	1.4	0.049	38.0	173.0	0.99335	3.03	0.52	9.3	5	white
1	6.6	0.340	0.24	3.3	0.034	29.0	99.0	0.99031	3.10	0.40	12.3	7	white
2	6.4	0.320	0.35	4.8	0.030	34.0	101.0	0.99120	3.36	0.60	12.5	8	white
3	6.8	0.230	0.32	1.6	0.026	43.0	147.0	0.99040	3.29	0.54	12.5	6	white
4	6.7	0.340	0.26	1.9	0.038	58.0	138.0	0.98930	3.00	0.47	12.2	7	white
...	...	...	...	...	...	...	...	...	...	...	...	...	...
1295	7.6	0.285	0.32	14.6	0.063	32.0	201.0	0.99800	3.00	0.45	9.2	5	white
1296	11.6	0.470	0.44	1.6	0.147	36.0	51.0	0.99836	3.38	0.86	9.9	4	red
1297	10.2	0.340	0.48	2.1	0.052	5.0	9.0	0.99458	3.20	0.69	12.1	7	red
1298	6.2	0.460	0.17	1.6	0.073	7.0	11.0	0.99425	3.61	0.54	11.4	5	red
1299	6.2	0.360	0.14	8.9	0.036	38.0	155.0	0.99622	3.27	0.50	9.4	5	white

1300 rows × 13 columns

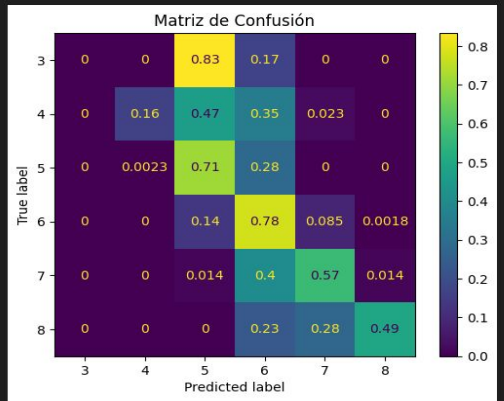
Clasificación: aplicación a test del modelo




```
print(classification_report(y_test_clf, modelo_pipeline_clf.predict(df_test)))
```

	precision	recall	f1-score	support
3	0.00	0.00	0.00	6
4	0.88	0.16	0.27	43
5	0.75	0.71	0.73	432
6	0.65	0.78	0.71	562
7	0.66	0.57	0.61	210
8	0.85	0.49	0.62	47
accuracy			0.69	1300
macro avg	0.63	0.45	0.49	1300
weighted avg	0.70	0.69	0.68	1300

```
ConfusionMatrixDisplay.from_predictions(y_test_clf, modelo_pipeline_clf.predict(df_test), normalize = "true")
# ConfusionMatrixDisplay.from_predictions(y_test_clf, y_pred)
# Configuración del gráfico
plt.title("Matriz de Confusión")
plt.show()
```



Evaluación general del modelo:
La precisión general del modelo es 0.69, lo que significa que el 69% de las predicciones fueron correctas.
El macro promedio (macro avg) de F1-score es 0.49, lo que indica un rendimiento bajo cuando se promedian todas las clases por igual.
El weighted avg (ponderado por la cantidad de ejemplos por clase) de F1-score es 0.68, reflejando un rendimiento más acorde con la
distribución de datos.
Posibles mejoras:
Balance de clases: La baja detección de la clase 3 indica que esta clase está subrepresentada.Sol: probar técnicas como oversampling (SMOTE).
Probar otro modelo: red neuronal podría mejorar los resultados.

Regresión

```python
# Definición del problema de regresión.

# target_reg → Variable objetivo para regresión.
#     - Se ha definido previamente como "alcohol".
#     - Representa el porcentaje de alcohol en el vino (variable continua).
#     - Es un problema de regresión, ya que el target es numérico y continuo.

target_reg
```

```
'alcohol'
```

```python
# Definición de las características categóricas para el problema de regresión.

# features_cat_reg → Variables categóricas a incluir en el modelo de regresión.
#     - Contiene:
#       - "class" (tipo de vino: tinto o blanco).
#       - "quality" (calidad del vino en una escala categórica).
#     - Ambas variables deben ser transformadas con OneHotEncoder para su uso en modelos numéricos.

features_cat_reg
```

```
['class', 'quality']
```

```python
# Definición de características numéricas principales para regresión.

# features_num_reg_1 → Variables numéricas con mayor correlación con el target de regresión ("alcohol").
#     - Se han seleccionado aquellas con correlación absoluta > r_min (0.05).
#     - Estas variables son las más relevantes para predecir el contenido de alcohol en el vino.
#     - Se recomienda aplicar StandardScaler para normalizar estas características antes del modelado.

features_num_reg_1
```

```
['density',
 'residual sugar',
 'total sulfur dioxide',
 'chlorides',
 'free sulfur dioxide',
 'pH',
 'fixed acidity']
```

Regresión

```
# Definición de columnas a incluir y excluir en el modelo de regresión.

# columns_to_keep_reg → Columnas que se mantendrán en el modelo de regresión.
#     - Incluye:
#       - features_num_reg_1 (variables numéricas más correlacionadas con el target).
#       - features_cat_reg (variables categóricas a transformar con OneHotEncoder).

# columns_to_exclude_reg → Columnas que se excluirán del modelo de regresión.
#     - Se obtienen eliminando de df.columns las variables incluidas en columns_to_keep_reg.
#     - Estas columnas no serán utilizadas en el modelo.

columns_to_keep_reg = features_num_reg_1 + features_cat_reg

columns_to_exclude_reg = [col for col in df_train.columns if col not in columns_to_keep_reg]

columns_to_exclude_reg
```

Regresión, procesamiento de datos

```
# Definición de Pipelines para preprocesamiento de datos en regresión.

# cat_pipeline → Preprocesamiento de variables categóricas.
#     - "Impute_Mode": Imputa valores faltantes con la moda (valor más frecuente).
#     - "OHEncoder": Aplica OneHotEncoder, ignorando categorías desconocidas.

# logaritmica → Transformación logarítmica de variables numéricas.
#     - Usa FunctionTransformer con np.log1p para estabilizar distribuciones sesgadas.
#     - feature_names_out="one-to-one" mantiene los nombres originales de las características.

# num_pipeline → Preprocesamiento de variables numéricas.
#     - "Impute_Mean": Imputa valores faltantes con la media.
#     - "logaritmo": Aplica la transformación logarítmica definida antes.
#     - "SScaler": Aplica StandardScaler para normalizar las variables numéricas.

# imputer_step_reg → ColumnTransformer para aplicar los Pipelines según el tipo de variable.
#     - "Process_Numeric": Aplica num_pipeline a features_num_reg_1 (variables numéricas seleccionadas para regresión).
#     - "Process_Categorical": Aplica cat_pipeline a features_cat_reg (variables categóricas seleccionadas para regresión).
#     - "Exclude": Elimina las columnas en columns_to_exclude_reg.
#     - remainder="passthrough": Mantiene cualquier otra columna sin modificar.

# pipe_missings_reg → Pipeline final que aplica el ColumnTransformer imputer_step_reg.
```

```
cat_pipeline = Pipeline(
    [("Impute_Mode", SimpleImputer(strategy="most_frequent")),  # Imputación con la moda
     ("OHEncoder", OneHotEncoder(handle_unknown='ignore'))  # Manejar categorías desconocidas
    ]
)

logaritmica = FunctionTransformer(np.log1p, feature_names_out="one-to-one")
# Esto le indica al Pipeline que el número de características no cambia y que puede usar los nombres originales.

num_pipeline = Pipeline(
    [("Impute_Mean", SimpleImputer(strategy = "mean")), # prevision que en el futuro lleguen datos faltantes
     ("logaritmo", logaritmica),
     ("SScaler", StandardScaler()),
    ]
)

imputer_step_reg = ColumnTransformer(
    [("Process_Numeric", num_pipeline,features_num_reg_1), # feature_numericas seleccionadas para clasificación
     ("Process_Categorical", cat_pipeline, features_cat_reg), # feature_categoriacas seleccionadas para regresión
     ("Exclude", "drop", columns_to_exclude_reg)
    ], remainder = "passthrough"
    )

pipe_missings_reg = Pipeline([("first_stage", imputer_step_reg)])
```

Regresión: evaluación modelos baseline

```
# Definición de Pipelines para modelos de regresión.

# randomreg_pipeline → Pipeline para RandomForestRegressor.
#     - "Preprocesado": Aplica el Pipeline de preprocesamiento pipe_missings_reg.
#     - "Modelo": Implementa RandomForestRegressor para realizar la regresión.

# XGB_pipeline → Pipeline para XGBRegressor.
#     - "Preprocesado": Aplica el Pipeline de preprocesamiento pipe_missings_reg.
#     - "Modelo": Implementa XGBRegressor, basado en árboles de decisión con boosting.

# LGBMreg_pipeline → Pipeline para LGBMRegressor.
#     - "Preprocesado": Aplica el Pipeline de preprocesamiento pipe_missings_reg.
#     - "Modelo": Implementa LGBMRegressor, optimizado para grandes volúmenes de datos.

# Validación cruzada con 5 folds.
#     - Se evalúan los tres modelos utilizando cross_val_score.
#     - La métrica utilizada es "neg_mean_absolute_percentage_error" (MAPE negativo).
#     - Se imprime el promedio del error absoluto porcentual y los valores individuales por fold.
```

```
randomreg_pipeline = Pipeline(
    [("Preprocesado", pipe_missings_reg),  # Aplicación del preprocesamiento definido
     ("Modelo", RandomForestRegressor())  # Modelo de regresión basado en árboles
    ])

XGB_pipeline = Pipeline(
    [("Preprocesado", pipe_missings_reg),
     ("Modelo", XGBRegressor())  # Modelo de boosting basado en XGBoost
    ])

LGBMreg_pipeline = Pipeline(
    [("Preprocesado", pipe_missings_reg),
     ("Modelo", LGBMRegressor(verbose=-1))  # Modelo optimizado basado en LightGBM
    ])

# Evaluación de modelos con validación cruzada
for name, pipe in zip(["RandomF", "XGB", "LGBM"], [randomreg_pipeline, XGB_pipeline, LGBMreg_pipeline]):
    resultado_reg = cross_val_score(pipe, df_train, y_train_reg, cv=5, scoring="neg_mean_absolute_percentage_error")
    print(f"{name}: {-np.mean(resultado_reg):.4f}")  # Se invierte el signo para interpretar el MAPE positivo
    print(resultado_reg)  # Resultados individuales de cada fold
```

```
RandomF: 0.0274
[-0.02855353 -0.02834254 -0.02683549 -0.02656188 -0.0267983 ]
XGB: 0.0271
[-0.02807295 -0.02796904 -0.02587827 -0.02658056 -0.02686444]
LGBM: 0.0287
[-0.02986407 -0.02965162 -0.02812008 -0.02791631 -0.02802684]
```

Regresión: optimización hiper parámetros

```python
# Búsqueda de hiperparámetros para los modelos de regresión mediante GridSearchCV.
# Se optimizan los siguientes modelos:
#   - RandomForestRegressor
#   - XGBRegressor
#   - LGBMRegressor
#
# Se definen los espacios de búsqueda para cada modelo.

# pipe_random_reg → Espacio de búsqueda para RandomForestRegressor.
#    - "Modelo__n_estimators": Número de árboles en el bosque.
#    - "Modelo__max_depth": Profundidad máxima de los árboles.
#    - "Modelo__max_features": Número de características a considerar en cada división.

pipe_random_reg = {
    "Modelo__n_estimators": [100, 200, 1000],  # Diferentes cantidades de árboles en el bosque
    "Modelo__max_depth": [1, 5, 10, 20],  # Variación en la profundidad máxima de los árboles
    "Modelo__max_features": ["log2", "sqrt", None]  # Diferentes métodos de selección de características
}

# pipe_LightGBM → Espacio de búsqueda para LightGBM.
#    - "Modelo__n_estimators": Número de árboles en el modelo.
#    - "Modelo__num_leaves": Número de hojas en cada árbol.
#    - "Modelo__learning_rate": Tasa de aprendizaje del modelo.

pipe_LightGBM = {
    "Modelo__n_estimators": [50, 100, 200],  # Variación en el número de árboles
    "Modelo__num_leaves": [31, 50, 100],  # Número de hojas en cada árbol
    "Modelo__learning_rate": [0.01, 0.1, 0.2]  # Tasa de aprendizaje del modelo
}

# pipe_xgb_param → Espacio de búsqueda para XGBoost.
#    - "Modelo__n_estimators": Número de árboles en el modelo.
#    - "Modelo__max_depth": Profundidad máxima de los árboles.
#    - "Modelo__learning_rate": Tasa de aprendizaje del modelo.

pipe_xgb_param = {
    'Modelo__n_estimators': [10, 100, 200, 400],  # Variación en la cantidad de árboles
    'Modelo__max_depth': [1, 2, 4, 8],  # Profundidad máxima de los árboles
    'Modelo__learning_rate': [0.1, 0.2, 0.5, 1.0]  # Tasa de aprendizaje
}

# Configuración de la validación cruzada con 5 folds.
cv = 5
```

```python
# GridSearchCV para RandomForestRegressor.
gs_random = GridSearchCV(
    randomreg_pipeline,
    pipe_random_reg,
    cv=cv,
    scoring="neg_mean_absolute_percentage_error",
    verbose=0,
    n_jobs=-1  # Utiliza todos los núcleos disponibles para acelerar la búsqueda
)

# GridSearchCV para LightGBMRegressor.
gs_LightGBM = GridSearchCV(
    LGBMreg_pipeline,
    pipe_LightGBM,
    cv=cv,
    scoring="neg_mean_absolute_percentage_error",
    verbose=0,
    n_jobs=-1
)

# GridSearchCV para XGBRegressor.
gs_xgb = GridSearchCV(
    XGB_pipeline,
    pipe_xgb_param,
    cv=cv,
    scoring="neg_mean_absolute_percentage_error",
    verbose=0,
    n_jobs=-1
)

# Diccionario con los GridSearch configurados para cada modelo.
pipe_grids_reg = {
    "gs_reg_log": gs_random,
    "gs_LightGBM": gs_LightGBM,
    "gs_xgb": gs_xgb
}
```

Regresión: optimización hiper parámetros

```python
# Ejecución de GridSearchCV para cada modelo en el diccionario pipe_grids_reg.
#
# - Itera sobre cada modelo y su respectiva búsqueda de hiperparámetros.
# - Ajusta el modelo con el conjunto de entrenamiento (df_train, y_train_reg).
# - GridSearchCV seleccionará automáticamente la mejor combinación de hiperparámetros según la métrica definida.
# - La métrica utilizada es "neg_mean_absolute_percentage_error" (MAPE negativo).
# - Los resultados se almacenan dentro de cada objeto GridSearchCV.

for nombre, grid_search in pipe_grids_reg.items():
    print(f"Entrenando GridSearch para {nombre}...")  # Mensaje informativo
    grid_search.fit(df_train, y_train_reg)  # Ajusta el modelo con la búsqueda de hiperparámetros
    print(f"Mejores parámetros para {nombre}: {grid_search.best_params_}")  # Imprime los mejores hiperparámetros encontrados
    print(f"Mejor score (neg-MAPE) para {nombre}: {grid_search.best_score_:.4f}")  # Muestra el mejor puntaje obtenido
    print("-" * 50)  # Separador para mayor claridad
```

```
Entrenando GridSearch para gs_reg_log...
Mejores parámetros para gs_reg_log: {'Modelo__max_depth': 20, 'Modelo__max_features': None, 'Modelo__n_estimators': 1000}
Mejor score (neg-MAPE) para gs_reg_log: -0.0273
--------------------------------------------------
Entrenando GridSearch para gs_LightGBM...
Mejores parámetros para gs_LightGBM: {'Modelo__learning_rate': 0.2, 'Modelo__n_estimators': 200, 'Modelo__num_leaves': 100}
Mejor score (neg-MAPE) para gs_LightGBM: -0.0253
--------------------------------------------------
Entrenando GridSearch para gs_xgb...
Mejores parámetros para gs_xgb: {'Modelo__learning_rate': 0.1, 'Modelo__max_depth': 8, 'Modelo__n_estimators': 400}
Mejor score (neg-MAPE) para gs_xgb: -0.0243
--------------------------------------------------
```

Regresión: selección modelo

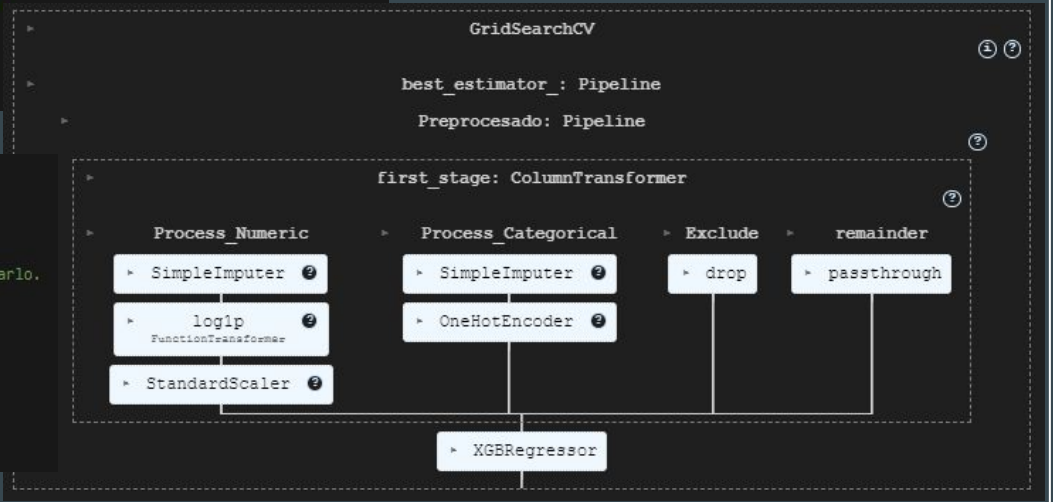
```python
# Evaluación de los mejores modelos tras la búsqueda de hiperparámetros con GridSearchCV.
#
# - Extrae el mejor score obtenido en validación cruzada para cada modelo en pipe_grids_reg.
# - Convierte los resultados en un DataFrame para facilitar su análisis y comparación.
# - Ordena los modelos en función del mejor score (neg-MAPE), de mayor a menor.
# - Permite identificar qué modelo tuvo mejor rendimiento tras la optimización.

# Extraer el mejor score de cada GridSearchCV y almacenarlo en una lista de tuplas.
best_grids_reg = [(i, j.best_score_) for i, j in pipe_grids_reg.items()]

# Convertir la lista en un DataFrame para facilitar su análisis.
best_grids_reg = pd.DataFrame(best_grids_reg, columns=["Grid", "Best score"]) \
                   .sort_values(by="Best score", ascending=False)  # Ordenar de mejor a peor score

# Mostrar los resultados
best_grids_reg
```

	Grid	Best score
2	gs_xgb	-0.024311
1	gs_LightGBM	-0.025330
0	gs_reg_log	-0.027334

Regresión: guardamos el modelo elegido

```python
# Selección del mejor modelo tras la búsqueda de hiperparámetros con GridSearchCV.
#
# - Se toma el modelo con el mejor score obtenido en la validación cruzada.
# - Se extrae el nombre del mejor modelo desde best_grids_reg.
# - Se usa ese nombre para acceder al mejor GridSearchCV almacenado en pipe_grids_reg.
# - El modelo seleccionado se almacenará en best_model_reg para futuras predicciones.

# Extraer el mejor modelo basado en la mejor puntuación obtenida.
best_model_reg = pipe_grids_reg[best_grids_reg.iloc[0, 0]]

# Mostrar el mejor modelo seleccionado
best_model_reg
```

```python
# Guardado del mejor modelo de regresión usando Pickle.
#
# - Se asegura de que el directorio "src/models" exista antes de guardar el modelo.
# - Serializa el mejor modelo encontrado (`best_model_reg`) en un archivo .pkl.
# - El modelo guardado podrá ser cargado posteriormente sin necesidad de volver a entrenarlo.

# Asegurar que el directorio "src/models" exista antes de guardar el modelo.
os.makedirs('src/models', exist_ok=True)

# Guardar el mejor modelo en un archivo .pkl dentro de "models".
with open('src/models/modelo_pipeline_reg.pkl', 'wb') as archivo:
    pickle.dump(best_model_reg, archivo)
```



Regresión: aplicación a test del modelo

```python
target_reg = "alcohol"

y_test_reg = df_test["alcohol"]
```

```python
# Recuperamos el modelo de pipelines (version pickle)
with open('src/models/modelo_pipeline_reg.pkl', 'rb') as archivo:
    modelo_pipeline_reg = pickle.load(archivo)
```

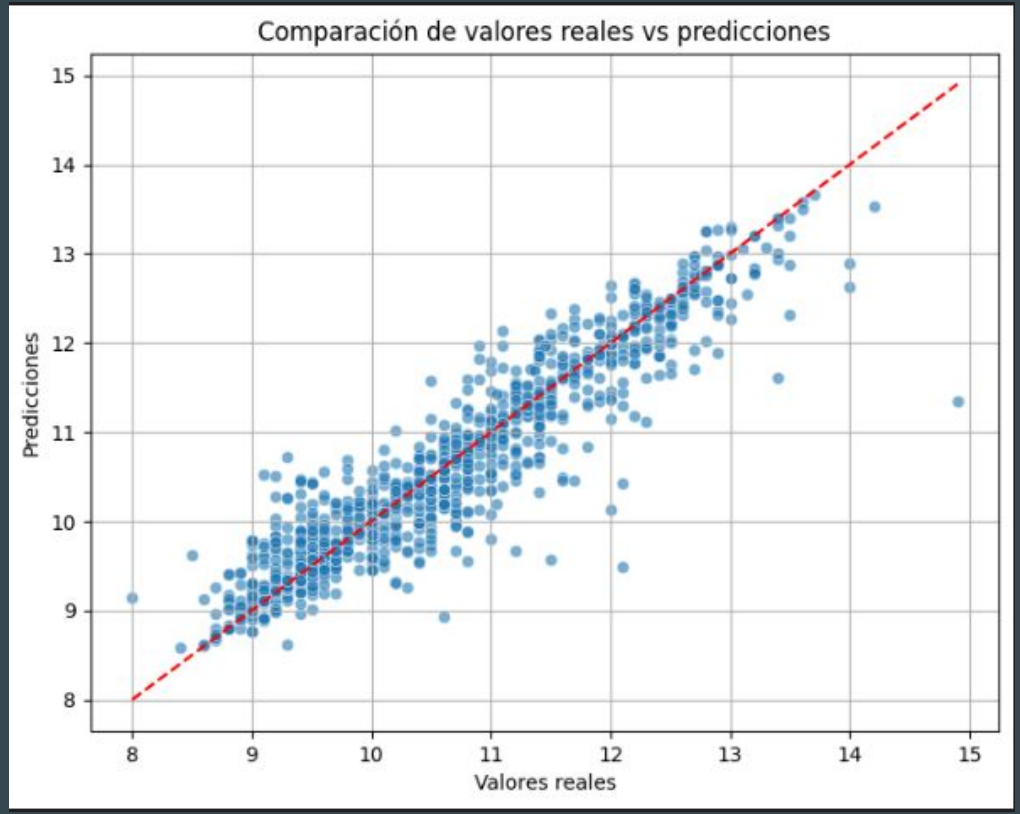
```python
# Evaluación del Modelo de Regresión

# modelo_pipeline_reg → Modelo entrenado con pipeline de regresión.
#     - Contiene los pasos de preprocesamiento y el modelo final entrenado.
#     - Se utiliza para generar predicciones en el conjunto de prueba.

# Cálculo de métricas → Se evalúa el desempeño del modelo con tres métricas clave.
#     - "RMSE (Root Mean Squared Error)": Mide el error cuadrático medio de las predicciones.
#     - "MAE (Mean Absolute Error)": Representa el error absoluto medio entre los valores reales y predichos.
#     - "R² (Coeficiente de Determinación)": Indica qué porcentaje de la variabilidad de los datos explica el modelo.

y_pred_reg = modelo_pipeline_reg.predict(df_test)  # Generar predicciones sobre el conjunto de prueba

rmse = np.sqrt(mean_squared_error(y_test_reg, y_pred_reg))  # Cálculo de RMSE
mae = mean_absolute_error(y_test_reg, y_pred_reg)  # Cálculo de MAE
r2 = r2_score(y_test_reg, y_pred_reg)  # Cálculo del coeficiente de determinación R²
```

```
Evaluación del Modelo de Regresión
RMSE (Error Cuadrático Medio)       0.3809
MAE (Error Absoluto Medio)          0.2367
R² (Coeficiente de Determinación)   0.8970
```

Regresión: aplicación a test del modelo

Evaluación del modelo de regresión XGBRegressor:

- El **RMSE** (Error Cuadrático Medio) es 0.3809, lo que indica una precisión moderada en las predicciones.
- El **MAE** (Error Absoluto Medio) es 0.2367, sugiriendo que, en promedio, las predicciones están desviadas por 0.2367 unidades del valor real.
- El **R^2** (Coeficiente de Determinación) es 0.8970, lo que muestra que el modelo explica el 89.70% de la variabilidad de los datos.



Balance Pipelines

Ventajas

Reproducibilidad: encapsulación de preprocesamiento y modelado, lo que permite la reutilización frente a nuevos datos.

Almacenamiento: Se puede guardar todo el proceso y exportarlo sencillamente.

Estructura: Construcción de modelos de manera modular y organizada.

Desventajas

Menor flexibilidad: rigidez ante necesidades especiales de especialización.

Complejidad para principiantes: se reduce la explicabilidad y puede ser más difícil de entender que cada función individualmente

Dificulta la depuración de errores: Si ocurre un error dentro del pipeline, puede ser difícil detectar en que paso falló el proceso

Comparativa Ilustrativa

Pipelines

REPRODUCIBILIDAD

Funciones

CONTROL