

PREDICT ACCEPTIBILITY OF A CAR



DISEÑO DEL PROYECTO

Este conjunto de datos de nivel principiante se derivó de un modelo de toma de decisiones que se desarrolló originalmente para la investigación sobre la toma de decisiones de atributos múltiples. La toma de decisiones implica la selección entre alternativas aparentemente conflictivas.

El conjunto de datos tiene 1728 filas y 7 columnas en las que los atributos del automóvil, como el precio y la tecnología, se describen en 6 atributos como "Precio de compra", "Mantenimiento" y "Seguridad", etc. Hay varias alternativas en cada uno de los 6 atributos. . La aceptabilidad del automóvil, el séptimo atributo, es la variable de resultado.

OBJETIVO

Predecir la aceptabilidad de un coche.

Este conjunto de datos se recomienda para aprender y practicar sus habilidades en técnicas de modelado de clasificación .

ENTIDADES Y DATOS

Posición de la columna	Nombre de atributo	Definición	Tipo de datos	Ejemplo	% relaciones nulas
1	comprar	Precio de compra del automóvil (v-high, high, med, low)	Cualitativo	bajo, medio, alto	0
2	mantenimiento	Precio del mantenimiento del coche (v-high, high, med, low)	Cualitativo	bajo, medio, alto	0
3	puertas	Número de puertas (2, 3, 4, 5 y más)	Cualitativo	2, 3, 4	0
4	personas	Capacidad en términos de personas para llevar (2, 4, más)	Cualitativo	2, 4, más	0
5	lug_boot	El tamaño del maletero (pequeño, mediano, grande)	Cualitativo	pequeño, mediano, grande	0
6	seguridad	Seguridad estimada del coche (baja, media, alta)	Cualitativo	bajo, medio, alto	0
7	clase	Aceptabilidad del automóvil (unacc: inaceptable, acc: aceptable, buena: buena, v-buena: muy buena)	Cualitativo	unacc, acc, bueno	0

1- SET UP

IMPORTACIÓN DE PAQUETES

```
In [1]: import pandas as pd  
import numpy as np
```

CREAR LOS DATASETS INICIALES

CARGAR LOS DATOS

```
In [8]: df = pd.read_csv('../..../02_Datos/01_Originales/car.data', names = ['precio_compra'  
df.head()
```

```
Out[8]:
```

	precio_compra	precio_mto	n_puertas	n_personas	t_maletero	nivel_seguridad	aceptabilidad
0	vhigh	vhigh	2	2	small	low	unacc
1	vhigh	vhigh	2	2	small	med	unacc
2	vhigh	vhigh	2	2	small	high	unacc
3	vhigh	vhigh	2	2	med	low	unacc
4	vhigh	vhigh	2	2	med	med	unacc

EXTRAER Y RESERVAR EL DATASET DE PRUEBA

Los datos del dataset de PRUEBA el modelo no los verá hasta que no se cree el modelo de ejecución definitivo. Estos datos confirmaran que nuestro modelo es óptimo y viable tras evaluar los datos de train y test.

```
In [37]: prueba = df.sample(frac = 0.3)  
print(prueba.shape)
```

(518, 7)

```
In [38]: #Mostramos dataset  
prueba.head()
```

```
Out[38]:
```

	precio_compra	precio_mto	n_puertas	n_personas	t_maletero	nivel_seguridad	aceptabilidad
1134	med	med	4	2	small	low	una
880	med	vhigh	2	4	big	med	a
1578	low	med	4	4	med	low	una
496	high	vhigh	4	4	small	med	una
153	vhigh	high	3	more	small	low	una

```
In [39]: # Guardamos el dataset de prueba  
prueba.to_csv('.../02_Datos/02_Validator/prueba.csv')
```

EXTRAER Y RESERVAR EL DATASET DE TRABAJO

```
In [40]: trabajo = df[~df.index.isin(prueba.index)]  
print(trabajo.shape)  
(1210, 7)
```

```
In [41]: trabajo.head()
```

```
Out[41]:
```

	precio_compra	precio_mto	n_puertas	n_personas	t_maletero	nivel_seguridad	aceptabilidad
0	vhigh	vhigh	2	2	small	low	unacc
1	vhigh	vhigh	2	2	small	med	unacc
2	vhigh	vhigh	2	2	small	high	unacc
3	vhigh	vhigh	2	2	med	low	unacc
4	vhigh	vhigh	2	2	med	med	unacc

```
In [44]: #Guardamos el dataset de trabajo  
trabajo.to_csv('.../02_Datos/03_Trabajo/trabajo.csv')
```

2- CALIDAD DE DATOS Y EDA

IMPORTACIÓN DE PAQUETES

```
In [2]: import numpy as np  
import pandas as pd  
import matplotlib.pyplot as plt  
%matplotlib inline
```

IMPORTAR LOS DATOS

```
In [3]: df = pd.read_csv('../..../02_Datos/03_Trabajo/trabajo.csv', index_col=0)
```

VISIÓN GENERAL

```
In [4]: df.head()
```

```
Out[4]:    precio_compra  precio_mto  n_puertas  n_personas  t_maletero  nivel_seguridad  aceptabilidad  
0          vhigh        vhigh         2            2       small        low      unacc  
1          vhigh        vhigh         2            2       small       med      unacc  
2          vhigh        vhigh         2            2       small      high      unacc  
3          vhigh        vhigh         2            2        med       low      unacc  
4          vhigh        vhigh         2            2        med      med      unacc
```

```
In [5]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>  
Int64Index: 1210 entries, 0 to 1727  
Data columns (total 7 columns):  
 #   Column           Non-Null Count  Dtype     
---  --  
 0   precio_compra    1210 non-null   object    
 1   precio_mto       1210 non-null   object    
 2   n_puertas        1210 non-null   object    
 3   n_personas       1210 non-null   object    
 4   t_maletero        1210 non-null   object    
 5   nivel_seguridad  1210 non-null   object    
 6   aceptabilidad    1210 non-null   object    
dtypes: object(7)  
memory usage: 75.6+ KB
```

CORRECCIÓN DE NOMBRE DE VARIABLES

No aplica. La corrección de los nombre se realizó en el Sep-Up.

TIPOS DE DATOS

```
In [6]: df.dtypes
```

```
Out[6]: precio_compra    object
         precio_mto      object
         n_puertas       object
         n_personas      object
         t_maletero      object
         nivel_seguridad object
         aceptabilidad   object
dtype: object
```

- **Conclusión**

Aunque n_puertas y n_personas la mayor parte son numéricos hay una valore "more" que es texto, por ese motivo el tipo de datos en todas las variables es object.

No afecta debido a que se tendrá que realizar a todos las variables un Ordinal Encoder en la fase de transformación de variables.

BALANCEO DE LA TARGET

```
In [7]: df.aceptabilidad.value_counts(normalize = True)*100
```

```
Out[7]: unacc    70.909091
        acc     21.404959
        good    3.966942
        vgood   3.719008
Name: aceptabilidad, dtype: float64
```

VALORES ÚNICOS

```
In [8]: df.nunique()
```

```
Out[8]: precio_compra    4
         precio_mto      4
         n_puertas       4
         n_personas      3
         t_maletero      3
         nivel_seguridad 3
         aceptabilidad   4
dtype: int64
```

- **Conclusión:** No se aprecian valores únicos.

VALORES DUPLICADOS

```
In [9]: df.duplicated().sum()
```

```
Out[9]: 0
```

- **Conclusión:** No se aprecian valores duplicados.

VALORES NULOS

```
In [10]: df.isna().sum()
```

```
Out[10]: precio_compra      0
          precio_mto        0
          n_puertas         0
          n_personas        0
          t_maletero         0
          nivel_seguridad   0
          aceptabilidad     0
          dtype: int64
```

- **Conclusión:** No se aprecian valores nulos.

SEPARAR NUMÉRICAS Y CATEGÓRICAS

No hay valores numéricos por lo que se trabajará con el mismo dataset.

GESTIÓN DE CATEGÓRICAS

Estadísticos y gráficos de las variables categóricas

```
In [11]: def estadisticos_cont(df_cat):
    #Calculamos describe
    estadisticos = df_cat.describe().T
    return(estadisticos)

estadisticos_cont(df)
```

```
Out[11]:      count  unique    top  freq
precio_compra    1210      4  vhigh   309
precio_mto       1210      4    med   313
n_puertas        1210      4      3   320
n_personas       1210      3      2   417
t_maletero        1210      3   small   410
nivel_seguridad  1210      3    low   408
aceptabilidad     1210      4  unacc   858
```

```
In [12]: def graficos_eda_categoricos(df_cat):
```

```
    #Calculamos el número de filas que necesitamos
    from math import ceil
    filas = ceil(df_cat.shape[1] / 2)

    #Definimos el gráfico
    f, ax = plt.subplots(nrows = filas, ncols = 2, figsize = (16, filas * 6))

    #Aplanamos para iterar por el gráfico como si fuera de 1 dimensión en Lugar de
    ax = ax.flat

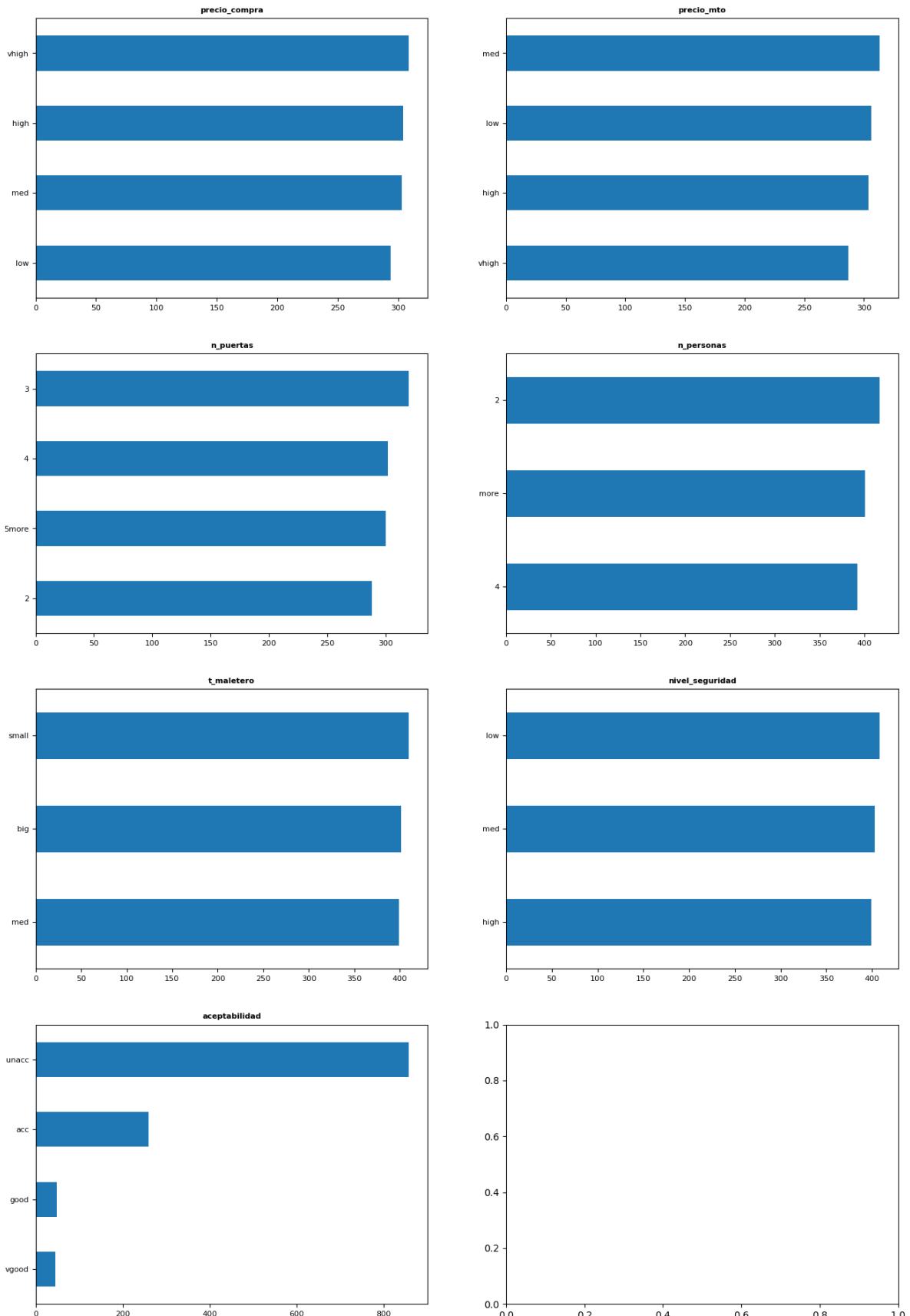
    #Creamos el bucle que va añadiendo gráficos
    for cada, variable in enumerate(df_cat):
        df_cat[variable].value_counts(ascending = True).plot.barh(ax = ax[cada])
```

```

        ax[cada].set_title(variable, fontsize = 8, fontweight = "bold")
        ax[cada].tick_params(labelsize = 8)

graficos_eda_categoricos(df)

```



GESTIÓN DE ATÍPICOS

No se detecta ninguna categórica con poco frecuencia (salvo la target)

GUARDAR DATASETS TRAS CALIDAD DE DATOS

In [16]:

```
#Nombre archivo y ruta
nombre_archivo = 'tablon_analitico.csv'
ruta = '../02_Datos/03_Trabajo/'
ruta_completa = ruta + nombre_archivo

df.to_csv(ruta_completa)
```

3 - TRANSFORMACIÓN DE DATOS

IMPORTACIÓN DE PAQUETES

```
In [23]: import numpy as np  
import pandas as pd  
  
from sklearn.preprocessing import OrdinalEncoder
```

IMPORTAR LOS DATOS

```
In [2]: df = pd.read_csv('../..../02_Datos/03_Trabajo/tablon_analitico.csv', index_col= 0)  
df.head()
```

```
Out[2]:    precio_compra  precio_mto  n_puertas  n_personas  t_maletero  nivel_seguridad  aceptabilidad  
0           vhigh       vhigh         2            2      small        low      unacc  
1           vhigh       vhigh         2            2      small       med      unacc  
2           vhigh       vhigh         2            2      small      high      unacc  
3           vhigh       vhigh         2            2       med        low      unacc  
4           vhigh       vhigh         2            2       med       med      unacc
```

TRANSFORMACIÓN DE CATEGÓRICAS

Variables a aplicar Ordinal Encoder

```
In [14]: var_oe = ['precio_compra', 'precio_mto', 'n_puertas', 'n_personas',  
           't_maletero', 'nivel_seguridad', 'aceptabilidad']  
var_oe
```

```
Out[14]: ['precio_compra',  
          'precio_mto',  
          'n_puertas',  
          'n_personas',  
          't_maletero',  
          'nivel_seguridad',  
          'aceptabilidad']
```

Orden de los valores de las variables

```
In [27]: df.nivel_seguridad.value_counts()
```

```
Out[27]: low      408  
med      403  
high     399  
Name: nivel_seguridad, dtype: int64
```

```
In [28]: orden_precio_compra = ['low', 'med', 'high', 'vhigh']  
orden_precio_mto = ['low', 'med', 'high', 'vhigh']
```

```

orden_n_puertas = ['2', '3', '4', '5more']
orden_n_personas = ['2', '4', 'more']
orden_t_maletero = ['small', 'med', 'big']
orden_nivel_seguridad = ['low', 'med', 'high']
orden_aceptabilidad = ['unacc', 'acc', 'good', 'vgood']

categorias = [orden_precio_compra,
              orden_precio_mto,
              orden_n_puertas,
              orden_n_personas,
              orden_t_maletero,
              orden_nivel_seguridad,
              orden_aceptabilidad]

```

Instanciar

```
In [29]: oe = OrdinalEncoder(categories = categorias,
                           handle_unknown = 'use_encoded_value',
                           unknown_value = 10)
```

Entrenar y aplicar

```
In [30]: df_oe = oe.fit_transform(df[var_oe])
```

Guardar como dataframe

```
In [31]: nombres_oe = [variable + '_oe' for variable in var_oe]
df_oe = pd.DataFrame(df_oe, columns = nombres_oe)
df_oe.head()
```

Out[31]:

	precio_compra_oe	precio_mto_oe	n_puertas_oe	n_personas_oe	t_maletero_oe	nivel_seguridad_oe
0	3.0	3.0	0.0	0.0	0.0	1.0
1	3.0	3.0	0.0	0.0	0.0	1.0
2	3.0	3.0	0.0	0.0	0.0	1.0
3	3.0	3.0	0.0	0.0	0.0	1.0
4	3.0	3.0	0.0	0.0	0.0	1.0

```
In [33]: df_oe.describe().T
```

Out[33]:

	count	mean	std	min	25%	50%	75%	max
precio_compra_oe	1210.0	1.519008	1.116854	0.0	1.0	2.0	3.0	3.0
precio_mto_oe	1210.0	1.472727	1.109251	0.0	0.0	1.0	2.0	3.0
n_puertas_oe	1210.0	1.507438	1.105828	0.0	1.0	1.0	2.0	3.0
n_personas_oe	1210.0	0.986777	0.822446	0.0	0.0	1.0	2.0	2.0
t_maletero_oe	1210.0	0.992562	0.818991	0.0	0.0	1.0	2.0	2.0
nivel_seguridad_oe	1210.0	0.992562	0.816969	0.0	0.0	1.0	2.0	2.0
aceptabilidad_oe	1210.0	0.404959	0.737493	0.0	0.0	0.0	1.0	3.0

El Ordinal Encoder está bien ejecutado al no haber ningún valor nulo(**10**)

GUARDAR DATASET TRAS TRANSFORMACIÓN DE DATOS

```
In [35]: df_oe.to_pickle('.../02_Datos/03_Trabajo/df_tablon.pickle')
```

05 - PRESELECCIÓN MEJOR MODELO SEGÚN LAZY PREDICT

IMPORTAR PAQUETES

```
In [2]: import numpy as np
import pandas as pd

from lazypredict.Supervised import LazyClassifier
from sklearn.model_selection import train_test_split
```

CARGAR DATOS

```
In [3]: df = pd.read_pickle('../02_Datos/03_Trabajo/df_tablon.pickle')
df.head()
```

```
Out[3]:   precio_compra_oe  precio_mto_oe  n_puertas_oe  n_personas_oe  t_maletero_oe  nivel_seguridad_
0           3.00            3.00          0.00          0.00          0.00          0.00          0.
1           3.00            3.00          0.00          0.00          0.00          0.00          1.
2           3.00            3.00          0.00          0.00          0.00          0.00          2.
3           3.00            3.00          0.00          0.00          0.00          1.00          0.
4           3.00            3.00          0.00          0.00          0.00          1.00          1.
```

SEPARAR PREDICTORAS Y TARGET

```
In [7]: x = df.drop(columns='aceptabilidad_oe').copy()
y = df.aceptabilidad_oe.copy()
```

DATASET DE TRAIN Y TEST

```
In [10]: train_x, val_x, train_y, val_y = train_test_split(x, y, test_size= 0.3, random_state= 42)
```

MODELIZAR SEGÚN LAZYPREDICT

```
In [11]: reg = LazyClassifier(verbose=0)
models, predictions = reg.fit(train_x, val_x, train_y, val_y)
models
```

100% |██████████| 29/29 [00:01<00:00, 18.69it/s]

Out[11]:

Accuracy Balanced Accuracy ROC AUC F1 Score Time Taken

Model	Accuracy	Balanced Accuracy	ROC AUC	F1 Score	Time Taken
LGBMClassifier	0.98	0.94	None	0.98	0.26
XGBClassifier	0.98	0.94	None	0.98	0.16
DecisionTreeClassifier	0.96	0.90	None	0.96	0.01
BaggingClassifier	0.96	0.86	None	0.96	0.04
ExtraTreesClassifier	0.96	0.83	None	0.96	0.13
NearestCentroid	0.75	0.83	None	0.78	0.01
RandomForestClassifier	0.95	0.81	None	0.95	0.23
KNeighborsClassifier	0.94	0.77	None	0.94	0.03
LabelSpreading	0.93	0.76	None	0.93	0.03
LabelPropagation	0.93	0.75	None	0.92	0.03
SVC	0.93	0.73	None	0.93	0.05
AdaBoostClassifier	0.88	0.73	None	0.87	0.14
ExtraTreeClassifier	0.88	0.69	None	0.89	0.01
LogisticRegression	0.83	0.59	None	0.82	0.03
QuadraticDiscriminantAnalysis	0.66	0.57	None	0.71	0.01
GaussianNB	0.74	0.57	None	0.74	0.01
Perceptron	0.78	0.54	None	0.77	0.01
LinearDiscriminantAnalysis	0.83	0.52	None	0.82	0.05
BernoulliNB	0.82	0.49	None	0.80	0.08
LinearSVC	0.82	0.47	None	0.80	0.02
SGDClassifier	0.79	0.46	None	0.78	0.02
CalibratedClassifierCV	0.81	0.43	None	0.79	0.10
PassiveAggressiveClassifier	0.74	0.39	None	0.70	0.01
RidgeClassifier	0.80	0.37	None	0.77	0.02
RidgeClassifierCV	0.80	0.37	None	0.76	0.02
DummyClassifier	0.74	0.25	None	0.62	0.01

CONCLUSIÓN:

Probaremos los siguientes modelos:

	Accuracy	Balanced Accuracy	ROC AUC	F1 Score	Time Taken
Model					
LGBMClassifier	0.98	0.94	None	0.98	0.26
XGBClassifier	0.98	0.94	None	0.98	0.16
DecisionTreeClassifier	0.96	0.90	None	0.96	0.01
BaggingClassifier	0.96	0.86	None	0.96	0.04
ExtraTreesClassifier	0.96	0.83	None	0.96	0.13

Realizaremos los siguientes procesos:

1. Entrenaremos un modelo sin parámetros y revisaremos su accuracy.
2. Si el accuracy es bajo en la validación, realizaremos una preselección de variables y entrenaremos un nuevo modelo sin parámetros.
3. Entrenaremos el mejor modelo e incluiremos parámetros para evitar sobreajuste.

In []:

In []:

05 - THE BEST ESTIMATOR

Modelo Base (CLASSIFIER)

Los modelos que vamos a poner a competir con mediante el GridSearch:

Model	Accuracy	Balanced Accuracy	ROC AUC	F1 Score	Time Taken
LGBMClassifier	0.98	0.94	None	0.98	0.26
XGBClassifier	0.98	0.94	None	0.98	0.16
DecisionTreeClassifier	0.96	0.90	None	0.96	0.01
BaggingClassifier	0.96	0.86	None	0.96	0.04
ExtraTreesClassifier	0.96	0.83	None	0.96	0.13

IMPORTACIÓN DE PAQUETES

In [85]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns

from sklearn.model_selection import train_test_split

# Modelos

from lightgbm import LGBMClassifier
from xgboost import XGBClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.ensemble import ExtraTreesClassifier

# Optimizar modelo

from sklearn.model_selection import GridSearchCV

#Métricas de evaluación

from sklearn.metrics import roc_curve, auc
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.preprocessing import label_binarize
from sklearn.metrics import classification_report

#Crear Pipeline
from sklearn.pipeline import Pipeline

#Guardar Modelo

import cloudpickle
```

IMPORTACIÓN DE DATOS

CARGAR LOS DATOS

```
In [86]: df_tablon = pd.read_pickle('../..../02_Datos/03_Trabajo/df_tablon.pickle')
df_tablon.head()
```

```
Out[86]:    precio_compra_oe  precio_mto_oe  n_puertas_oe  n_personas_oe  t_maletero_oe  nivel_seguridad_
0             3.0            3.0           0.0           0.0           0.0           0.0
1             3.0            3.0           0.0           0.0           0.0           0.0
2             3.0            3.0           0.0           0.0           0.0           0.0
3             3.0            3.0           0.0           0.0           0.0           1.0
4             3.0            3.0           0.0           0.0           0.0           1.0
```



SEPARAR PREDICTIVAS Y TARGET

```
In [87]: x = df_tablon.drop(columns= 'aceptabilidad_oe').copy()
y = df_tablon.aceptabilidad_oe.copy()
```

MODELIZAR

RESERVAR EL DATASET DE VALIDACIÓN

```
In [88]: train_x, val_x, train_y, val_y = train_test_split(x, y, test_size= 0.3, random_state= 42)
```

CREAR PIPE Y EL DICCIONARIO CON LOS ALGORITMOS, PARÁMETROS Y VALORES

```
In [89]: pipe = Pipeline([('algoritmo', XGBClassifier())])

grid = [
    {
        'algoritmo' : [LGBMClassifier()]
    },
    {
        'algoritmo' : [XGBClassifier()]
    },
    {
        'algoritmo' : [DecisionTreeClassifier()]
    },
    {
        'algoritmo' : [BaggingClassifier()]
    },
    {
```

```
        'algoritmo' : [ExtraTreesClassifier()]
    ]
]
```

OPTIMIZAR LOS PARÁMETROS CON GRIDSEARCH

```
In [90]: grid_search = GridSearchCV(estimator= pipe,
                                 param_grid= grid,
                                 cv=5,
                                 scoring= 'accuracy',
                                 verbose= 0,
                                 n_jobs= -1)

modelo = grid_search.fit(train_x, train_y)
pd.DataFrame(grid_search.cv_results_).sort_values(by = 'rank_test_score')
```

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_algoritmo	
1	0.561994	0.173694	0.003691	0.000398	XGBClassifier(base_score=None, booster=None, c...	XGI
0	1.525538	0.155298	0.006279	0.003482	LGBMClassifier()	{
3	0.028369	0.001661	0.003990	0.000001	BaggingClassifier()	{al
2	0.003211	0.000391	0.001796	0.000399	DecisionTreeClassifier()	
4	0.189318	0.003723	0.017688	0.000822	ExtraTreesClassifier()	

```
In [91]: modelo.best_estimator_
```

```
Out[91]: Pipeline
         XGBClassifier
```

```
In [92]: modelo.best_params_
```

```
Out[92]: {'algoritmo': XGBClassifier(base_score=None, booster=None, callbacks=None,
                                     colsample_bylevel=None, colsample_bynode=None,
                                     colsample_bytree=None, early_stopping_rounds=None,
                                     enable_categorical=False, eval_metric=None, feature_types=None,
                                     gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
                                     interaction_constraints=None, learning_rate=None, max_bin=None,
                                     max_cat_threshold=None, max_cat_to_onehot=None,
                                     max_delta_step=None, max_depth=None, max_leaves=None,
                                     min_child_weight=None, missing=nan, monotone_constraints=None,
                                     n_estimators=100, n_jobs=None, num_parallel_tree=None,
                                     predictor=None, random_state=None, ...)}
```

```
In [93]: modelo.best_score_
```

```
Out[93]: 0.9681239122868082
```

GUARDAR MODELO.BEST_ESTIMATOR, PARÁMETROS Y SCORE

```
In [94]: modelo_best_estimator = modelo
m_best_estimator = str(modelo.best_estimator_[0])
m_best_estimator = m_best_estimator.split('(')[0]
m_best_params = str(modelo.best_params_)
m_best_score = str(modelo.best_score_)
```

EVALUAR MODELO

PREDECIR Y EVALUAR SOBRE EL TRAIN

PREDECIR SOBRE EL TRAIN

```
In [95]: pred = modelo.best_estimator_.predict(train_x)
```

EVALUAR SOBRE EL TRAIN

```
In [96]: t_accuracy = accuracy_score(train_y, pred)
t_report = classification_report(train_y, pred)

print(f"Accuracy: {t_accuracy}")
print(f"Classification Report:\n{t_report}")
```

```
Accuracy: 1.0
Classification Report:
precision    recall   f1-score   support
          0.0      1.00      1.00      1.00      591
          1.0      1.00      1.00      1.00      191
          2.0      1.00      1.00      1.00       34
          3.0      1.00      1.00      1.00       31

accuracy                  1.00      847
macro avg      1.00      1.00      1.00      847
weighted avg     1.00      1.00      1.00      847
```

PREDECIR Y EVALUAR SOBRE LA VALIDACIÓN

PREDECIR SOBRE LA VALIDACIÓN

```
In [97]: pred = modelo.best_estimator_.predict(val_x)
```

EVALUAR SOBRE LA VALIDACIÓN

```
In [98]: v_accuracy = accuracy_score(val_y, pred)
v_report = classification_report(val_y, pred)

print(f"Accuracy: {v_accuracy}")
print(f"Classification Report:\n{v_report}")
```

```

Accuracy: 0.9834710743801653
Classification Report:
      precision    recall  f1-score   support

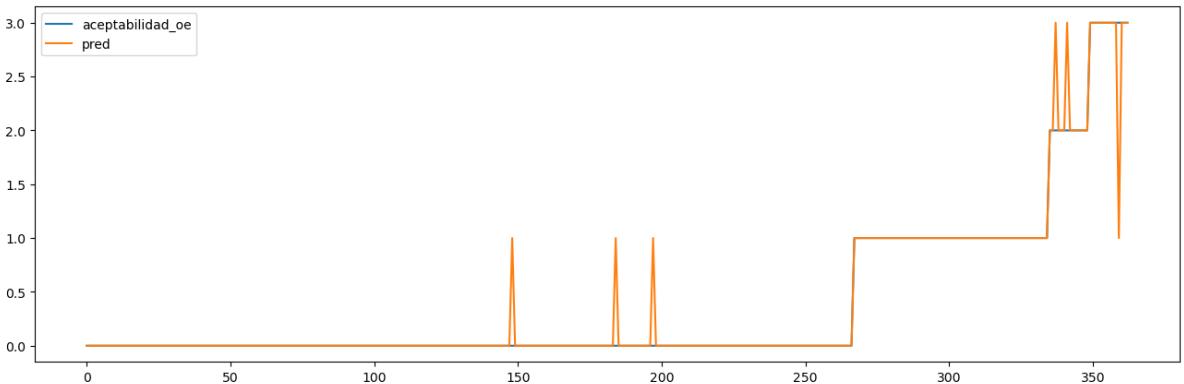
       0.0       1.00    0.99    0.99     267
       1.0       0.94    1.00    0.97     68
       2.0       1.00    0.86    0.92     14
       3.0       0.87    0.93    0.90     14

  accuracy                           0.98    363
 macro avg       0.95    0.94    0.95    363
weighted avg    0.98    0.98    0.98    363

```

Gráfico entre la diferencia entre la validación y el original

```
In [99]: test_y = val_y.reset_index().copy()
test_pred = pd.Series(pred).to_frame()
df_pred = pd.concat([test_y,test_pred], axis= 1).sort_values(by='aceptabilidad_oe')
df_pred.rename(columns = {0:'pred'}, inplace = True)
df_pred = df_pred.drop(columns= ['level_0'])
df_pred[['aceptabilidad_oe','pred']].plot(figsize=(16,5));
```



REPORTING DEL MODELO

Matrix de Confusión MultiClass y ROC CHART MULTICLASS

```
In [100...]:
# Crear subplots
f, ax = plt.subplots(1, 2, figsize=(16, 5))
ax = ax.flat

# Calcular la matriz de confusión
cm = confusion_matrix(val_y, pred)

# Crear un mapa de calor de la matriz de confusión
sns.heatmap(cm, annot=True, cmap="Blues", fmt="d", cbar=False, ax=ax[0])
ax[0].set_xlabel("Predicted Labels")
ax[0].set_ylabel("True Labels")
ax[0].set_title("Confusion Matrix")

# Binarizar las etiquetas verdaderas y las predicciones para cada clase
n_classes = len(np.unique(val_y))
binarized_val_y = label_binarize(val_y, classes=np.arange(n_classes))
binarized_pred = label_binarize(pred, classes=np.arange(n_classes))

# Calcular la curva ROC y el área bajo la curva (AUC) para cada clase
fpr = dict()
tpr = dict()
```

```

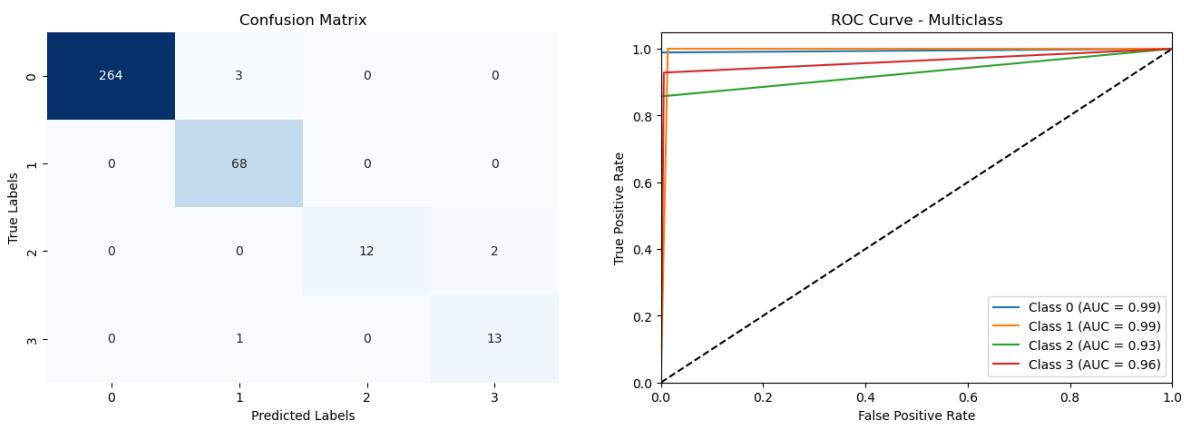
roc_auc = dict()
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(binarized_val_y[:, i], binarized_pred[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

for i in range(n_classes):
    ax[1].plot(fpr[i], tpr[i], label='Class {} (AUC = {:.2f})'.format(i, roc_auc[i]))

ax[1].plot([0, 1], [0, 1], 'k--')
ax[1].set_xlim([0.0, 1.0])
ax[1].set_ylim([0.0, 1.05])
ax[1].set_xlabel('False Positive Rate')
ax[1].set_ylabel('True Positive Rate')
ax[1].set_title('ROC Curve - Multiclass')
ax[1].legend(loc="lower right")

```

Out[100]: <matplotlib.legend.Legend at 0x2a723885110>



CONCLUSIÓN: El modelo ha dado muy buenos resultados con los datos de validación (accuracy = 98%), no se aprecia sobre-ajuste del modelo. De todas formas vamos a realizar un preselección de variables y generar un segundo modelo con las variables preseleccionadas sin parámetros.

GUARDAR BEST_ESTIMATOR, PARÁMETROS Y RESULTADOS DEL TEST Y LA VALIDACIÓN

Guardar el mejor estimador

```
In [101...]: version_estimator = '_v01'
nombre_best_estimator = m_best_estimator + version_estimator + '.pickle'
nombre_best_estimator
```

Out[101]: 'XGBClassifier_v01.pickle'

```
In [102...]: m_best_estimator
```

Out[102]: 'XGBClassifier'

```
In [103...]: ruta_pipe_entrenamiento = '../../../../../04_Modelos/Best_estimator/' + nombre_best_estimator

with open(ruta_pipe_entrenamiento, mode= 'wb') as file:
    cloudpickle.dump(modelo_best_estimator, file)
```

```
In [104...]: # Añadir comentarios sobre el modelo y definimos predictoras y target

comentarios = "Modelo Base"
x_columns = list(x.columns)
y_target = y.name
```

```
In [105...]: #Cargamos la lista con los resultados
resultado = {'m_Best_estimator': m_best_estimator,
             'm_Best_paramans' : m_best_params,
             'm_Best_Score': m_best_score,
             't_accuracy': t_accuracy,
             't_report': t_report,
             'v_accuracy': v_accuracy,
             'v_report': v_report,
             'comentarios': comentarios,
             'predictoras_X': x_columns,
             'target_y': y_target
            }
resultado= pd.Series(resultado,name=nombre_best_estimator)
resultado
```

```
Out[105]: m_Best_estimator                               XGBClassifier
m_Best_paramans      {'algoritmo': XGBClassifier(base_score=None, b...
m_Best_Score          0.9681239122868082
t_accuracy           1.0
t_report              precision    recall   f1-score   ...
v_accuracy            0.983471
v_report              precision    recall   f1-score   ...
comentarios           Modelo Base
predictoras_X         [precio_compra_oe, precio_mto_oe, n_puertas_oe...
target_y              aceptabilidad_oe
Name: XGBClassifier_v01.pickle, dtype: object
```

```
In [106...]: df_best = pd.read_excel('../04_Modelos/Best_estimator/Best_estimator.xlsx',index...
```

```
In [107...]: df_best = pd.concat([df_best, resultado], ignore_index= False, axis=1)
df_best.to_excel('../04_Modelos/Best_estimator/Best_estimator.xlsx')
```

6 - PRESELECCIÓN DE VARIABLES

IMPORTAR PAQUETES

```
In [28]: import os
import json

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns

from sklearn.feature_selection import mutual_info_classif
from sklearn.feature_selection import mutual_info_regression
from sklearn.feature_selection import RFE
from xgboost import XGBClassifier
from sklearn.inspection import permutation_importance

#Automcompletar rápido
%config IPCompleter.greedy=True
```

IMPORTAR LOS DATOS

Cargar los datos.

```
In [29]: df = pd.read_pickle('../..../02_Datos/03_Trabajo/df_tablon.pickle')
df.head()
```

```
Out[29]:   precio_compra_oe  precio_mto_oe  n_puertas_oe  n_personas_oe  t_maletero_oe  nivel_seguridad_
0             3.0            3.0           0.0           0.0           0.0           0.0
1             3.0            3.0           0.0           0.0           0.0           0.0
2             3.0            3.0           0.0           0.0           0.0           0.0
3             3.0            3.0           0.0           0.0           0.0           1.0
4             3.0            3.0           0.0           0.0           0.0           1.0
```

```
In [30]: df.corr()
```

Out[30]:

	precio_compra_oe	precio_mto_oe	n_puertas_oe	n_personas_oe	t_maletero_oe
precio_compra_oe	1.000000	-0.001918	-0.021880	-0.010532	-0.036468
precio_mto_oe	-0.001918	1.000000	-0.021075	-0.003116	-0.003410
n_puertas_oe	-0.021880	-0.021075	1.000000	-0.020809	0.017870
n_personas_oe	-0.010532	-0.003116	-0.020809	1.000000	-0.009970
t_maletero_oe	-0.036468	-0.003410	0.017870	-0.009970	1.000000
nivel_seguridad_oe	0.011486	-0.009808	0.027070	-0.007533	-0.008736
aceptabilidad_oe	-0.284503	-0.223081	0.079469	0.332025	0.141933

MÉTODOS SUPERVISADOS

Preparar x e y

```
In [31]: target = 'aceptabilidad_oe'  
x = df.drop(columns = target).copy()  
y = df[target].copy()
```

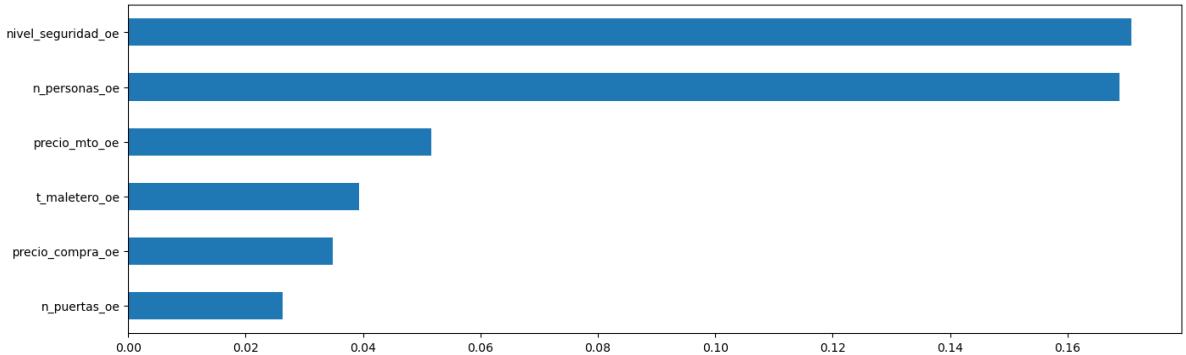
Mutual Information

Crear una función para mostrar el resultado

```
In [32]: def ranking_mi(mutual_selector, modo = 'tabla'):  
    #Maqueta el ranking  
    ranking_mi = pd.DataFrame(mutual_selector, index = x.columns).reset_index()  
    ranking_mi.columns = ['variable','importancia_mi']  
    ranking_mi = ranking_mi.sort_values(by = 'importancia_mi', ascending = False)  
    ranking_mi['ranking_mi'] = np.arange(0,ranking_mi.shape[0])  
    #Muestra la salida  
    if modo == 'tabla':  
        return(ranking_mi)  
    else:  
        g = ranking_mi.importancia_mi.sort_values().plot.barh(figsize = (16,5))  
        g.set_yticklabels(ranking_mi.sort_values(by = 'importancia_mi').variable)  
        return(g)
```

Calcular y revisar

```
In [33]: mutual_selector = mutual_info_classif(x,y)  
  
rank_mi = ranking_mi(mutual_selector, modo = 'grafico')
```



Seleccionar las variables que pasan

Definir la posición de la última variable que va a entrar

```
In [34]: posicion_variable_límite = 6
```

Extraer los nombres de las que entran

```
In [35]: entrant_mi = ranking_mi(mutual_selector).iloc[0:posicion_variable_límite].variable
```

Crear el dataframe con la selección

```
In [36]: x_mi = x[entrant_mi].copy()
```

Recursive Feature Elimination

Instanciar

```
In [37]: rfe = RFE(estimator = XGBClassifier(use_label_encoder=False, n_jobs = -1, eval_metric='mlogloss'), n_features_to_select=6, step=1)
```

c:\Users\ialca\anaconda3\envs\proyecto1\Lib\site-packages\xgboost\sklearn.py:1395:
UserWarning: `use_label_encoder` is deprecated in 1.7.0.
warnings.warn(``use_label_encoder` is deprecated in 1.7.0.")

Entrenar

```
In [38]: rfe.fit(x,y)
```

c:\Users\ialca\anaconda3\envs\proyecto1\Lib\site-packages\xgboost\sklearn.py:1395:
UserWarning: `use_label_encoder` is deprecated in 1.7.0.
warnings.warn(``use_label_encoder` is deprecated in 1.7.0.")
c:\Users\ialca\anaconda3\envs\proyecto1\Lib\site-packages\xgboost\sklearn.py:1395:
UserWarning: `use_label_encoder` is deprecated in 1.7.0.
warnings.warn(``use_label_encoder` is deprecated in 1.7.0.")
c:\Users\ialca\anaconda3\envs\proyecto1\Lib\site-packages\xgboost\sklearn.py:1395:
UserWarning: `use_label_encoder` is deprecated in 1.7.0.
warnings.warn(``use_label_encoder` is deprecated in 1.7.0.")
c:\Users\ialca\anaconda3\envs\proyecto1\Lib\site-packages\xgboost\sklearn.py:1395:
UserWarning: `use_label_encoder` is deprecated in 1.7.0.
warnings.warn(``use_label_encoder` is deprecated in 1.7.0.")

```
Out[38]:
```

► RFE

► estimator: XGBCClassifier

► XGBCClassifier

Extraer los nombres de las que entran

```
In [39]: entran_rfe = x.columns[rfe.support_]  
entran_rfe
```

```
Out[39]: Index(['precio_compra_oe', 'n_personas_oe', 'nivel_seguridad_oe'], dtype='object')
```

Crear el dataframe con la selección

```
In [40]: x_rfe = x[entran_rfe].copy()
```

Permutation Importance

Crear una función para mostrar el resultado

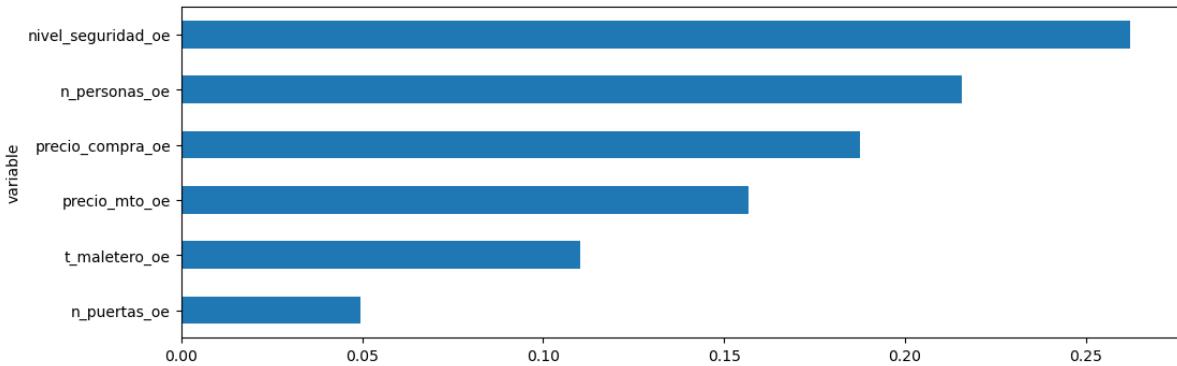
```
In [41]: def ranking_per(predictoras,permutacion):  
    ranking_per = pd.DataFrame({'variable': predictoras.columns, 'importancia_per':  
        ranking_per['ranking_per'] = np.arange(0,ranking_per.shape[0])  
    return(ranking_per)
```

Instanciar y entrenar

```
In [42]: import warnings  
warnings.filterwarnings(action="ignore", message=r'.*Use subset.*of np.ndarray is ')  
  
xgb = XGBCClassifier(use_label_encoder=False, n_jobs = -1, eval_metric='auc')  
  
xgb.fit(x,y)  
  
permutacion = permutation_importance(xgb,  
                                      x, y,  
                                      scoring = 'accuracy',  
                                      n_repeats=5, n_jobs = -1)  
  
c:\Users\ialca\anaconda3\envs\proyecto1\Lib\site-packages\xgboost\sklearn.py:1395:  
UserWarning: `use_label_encoder` is deprecated in 1.7.0.  
    warnings.warn("`use_label_encoder` is deprecated in 1.7.0.")
```

Revisar la salida

```
In [43]: rank_per = ranking_per(x,permutacion)  
  
rank_per.set_index('variable').importancia_per.sort_values().plot.barh(figsize = (:
```



Seleccionar las variables que pasan

Definir la posición de la última variable que va a entrar

```
In [44]: posicion_variable_límite = 6
```

Extraer los nombres de las que entran

```
In [45]: entrان_per = rank_per.iloc[0:posicion_variable_límite].variable
```

Crear el dataframe con la selección

```
In [46]: x_per = x[entrان_per].copy()
```

SELECCIONAR EL MÉTODO FINAL

Descomentar el método de preselección elegido y dejar comentados el resto.

```
In [47]: x_preseleccionado = x_mi
# x_preseleccionado = x_rfe
# x_preseleccionado = x_per
```

MÉTODOS NO SUPERVISADOS

Correlación

Crear una función para mostrar el resultado

```
In [48]: def correlaciones_fuertes(df, lim_inf = 0.3, lim_sup = 1, drop_duplicados=True):
    #Calcula la matriz de correlación
    c = df.corr().abs()
    #Lo pasa todo a filas
    c = c.unstack()
    #Pasa el índice a columnas y le pone nombres
    c = pd.DataFrame(c).reset_index()
    c.columns = ['var1', 'var2', 'corr']
    #A un dataframe, filtra límites y ordena en descendiente
    c = c.loc[(c['corr'] > lim_inf) & (c['corr'] < lim_sup), :].sort_values(by = 'corr', ascending=False)
    #Desduplica las correlaciones (o no si drop_duplicados es False)
    c = c if drop_duplicados == False else c.drop_duplicates(subset = ['corr'])
    #Devuelve la salida
    return(c)
```

Calcular y revisar

Calcular

```
In [49]: cor_finales = correlaciones_fuertes(x_preseleccionado)
```

Revisar agregado

```
In [50]: cor_finales.var1.value_counts()
```

```
Out[50]: Series([], Name: var1, dtype: int64)
```

Revisar detalle

```
In [51]: cor_finales.head(50)
```

```
Out[51]: var1  var2  corr
```

```
In [52]: x_preseleccionado.columns.to_list()
```

```
Out[52]: ['nivel_seguridad_oe',
          'n_personas_oe',
          'precio_mto_oe',
          't_maletero_oe',
          'precio_compra_oe']
```

CONCLUSIÓN: No se detectan correlaciones fuerte entre predictoras.

GUARDAR DATASETS TRAS PRESELECCION DE VARIABLES

```
In [53]: #Definir los nombres de los archivos
nombre_x_preseleccionado = '../../../../../02_Datos/03_Trabajo/' + 'x_preseleccionado.pickle'
nombre_y_preseleccionado = '../../../../../02_Datos/03_Trabajo/' + 'y_preseleccionado.pickle'
```

```
In [54]: #Guardar los archivos
x_preseleccionado.to_pickle(nombre_x_preseleccionado)

y_preseleccionado = y.copy()
y_preseleccionado.to_pickle(nombre_y_preseleccionado)
```

07 - THE BEST ESTIMATOR - V1

BaggingClassifier con preselección de variables

Los modelos que vamos a poner a competir con mediante el GridSearch:

Model	Accuracy	Balanced Accuracy	ROC AUC	F1 Score	Time Taken
LGBMClassifier	0.98	0.94	None	0.98	0.26
XGBClassifier	0.98	0.94	None	0.98	0.16
DecisionTreeClassifier	0.96	0.90	None	0.96	0.01
BaggingClassifier	0.96	0.86	None	0.96	0.04
ExtraTreesClassifier	0.96	0.83	None	0.96	0.13

IMPORTACIÓN DE PAQUETES

In [73]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns

from sklearn.model_selection import train_test_split

# Modelos

from lightgbm import LGBMClassifier
from xgboost import XGBClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.ensemble import ExtraTreesClassifier

# Optimizar modelo

from sklearn.model_selection import GridSearchCV

#Métricas de evaluación

from sklearn.metrics import roc_curve, auc
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.preprocessing import label_binarize
from sklearn.metrics import classification_report

#Crear Pipeline
from sklearn.pipeline import Pipeline

#Guardar Modelo

import cloudpickle
```

IMPORTACIÓN DE DATOS

CARGAR LOS DATOS

SEPARAR PREDICTIVAS Y TARGET

```
In [74]: x = pd.read_pickle('../..../02_Datos/03_Trabajo/x_preseleccionado.pickle')
y = pd.read_pickle('../..../02_Datos/03_Trabajo/y_preseleccionado.pickle')
```

MODELIZAR

RESERVAR EL DATASET DE VALIDACIÓN

```
In [75]: train_x, val_x, train_y, val_y = train_test_split(x, y, test_size= 0.3, random_st
```

CREAR PIPE Y EL DICCIONARIO CON LOS ALGORITMOS, PARÁMETROS Y VALORES

```
In [76]: pipe = Pipeline([('algoritmo', XGBClassifier())])

grid = [
    {
        'algoritmo' : [LGBMClassifier()]
    },
    {
        'algoritmo' : [XGBClassifier()]
    },
    {
        'algoritmo' : [DecisionTreeClassifier()]
    },
    {
        'algoritmo' : [BaggingClassifier()]
    },
    {
        'algoritmo' : [ExtraTreesClassifier()]
    }
]
```

OPTIMIZAR LOS PARÁMETROS CON GRIDSEARCH

```
In [77]: grid_search = GridSearchCV(estimator= pipe,
                                 param_grid= grid,
                                 cv=5,
                                 scoring= 'accuracy',
                                 verbose= 0,
                                 n_jobs= -1)

modelo = grid_search.fit(train_x, train_y)
pd.DataFrame(grid_search.cv_results_).sort_values(by = 'rank_test_score')
```

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_algoritmo	
3	0.045683	0.001690	0.006982	0.000631	BaggingClassifier() {al	
1	0.306680	0.018587	0.007579	0.001017	XGBClassifier(base_score=None, booster=None, c...	XGB
2	0.005187	0.000398	0.003591	0.000489	DecisionTreeClassifier()	
4	0.271894	0.004940	0.028168	0.001447	ExtraTreesClassifier()	
0	1.663857	0.084114	0.008466	0.003015	LGBMClassifier()	{

In [78]: `modelo.best_estimator_`

Out[78]:

```

▶ Pipeline
  ▶ BaggingClassifier
  
```

In [79]: `modelo.best_params_`

Out[79]: `{'algoritmo': BaggingClassifier()}`

In [80]: `modelo.best_score_`

Out[80]: `0.8075391576749043`

GUARDAR MODELO.BEST_ESTIMATOR, PARÁMETROS Y SCORE

In [81]:

```

modelo_best_estimator = modelo
m_best_estimator = str(modelo.best_estimator_[0])
m_best_estimator = m_best_estimator.split('(')[0]
m_best_params = str(modelo.best_params_)
m_best_score = str(modelo.best_score_)
  
```

EVALUAR MODELO

PREDECIR Y EVALUAR SOBRE EL TRAIN

PREDECIR SOBRE EL TRAIN

In [82]: `pred = modelo.best_estimator_.predict(train_x)`

EVALUAR SOBRE EL TRAIN

In [83]:

```

t_accuracy = accuracy_score(train_y, pred)
t_report = classification_report(train_y, pred)
  
```

```
print(f"Accuracy: {t_accuracy}")
print(f"Classification Report:\n{t_report}")
```

```
Accuracy: 0.8193624557260921
Classification Report:
precision    recall   f1-score   support
0.0          0.93     0.92      0.92      591
1.0          0.60     0.69      0.64      191
2.0          0.00     0.00      0.00      34
3.0          0.44     0.61      0.51      31

accuracy           0.82      847
macro avg       0.49     0.55      0.52      847
weighted avg    0.80     0.82      0.81      847
```

```
c:\Users\ialca\anaconda3\envs\proyecto1\Lib\site-packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
    _warn_prf(average, modifier, msg_start, len(result))
c:\Users\ialca\anaconda3\envs\proyecto1\Lib\site-packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
    _warn_prf(average, modifier, msg_start, len(result))
c:\Users\ialca\anaconda3\envs\proyecto1\Lib\site-packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
    _warn_prf(average, modifier, msg_start, len(result))
```

PREDECIR Y EVALUAR SOBRE LA VALIDACIÓN

PREDECIR SOBRE LA VALIDACIÓN

```
In [84]: pred = modelo.best_estimator_.predict(val_x)
```

EVALUAR SOBRE LA VALIDACIÓN

```
In [85]: v_accuracy = accuracy_score(val_y, pred)
v_report = classification_report(val_y, pred)

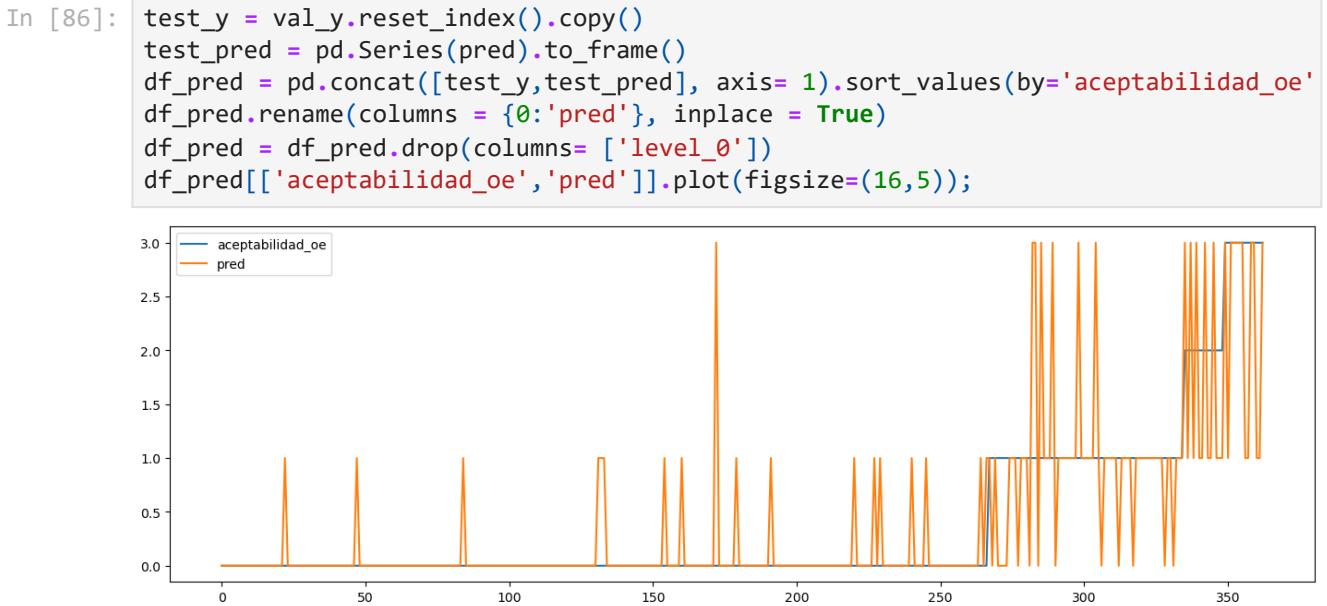
print(f"Accuracy: {v_accuracy}")
print(f"Classification Report:\n{v_report}")
```

```
Accuracy: 0.8429752066115702
Classification Report:
precision    recall   f1-score   support
0.0          0.95     0.93      0.94      267
1.0          0.61     0.71      0.65      68
2.0          0.00     0.00      0.00      14
3.0          0.43     0.64      0.51      14

accuracy           0.84      363
macro avg       0.50     0.57      0.53      363
weighted avg    0.83     0.84      0.83      363
```

```
c:\Users\ialca\anaconda3\envs\proyecto1\Lib\site-packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
    _warn_prf(average, modifier, msg_start, len(result))
c:\Users\ialca\anaconda3\envs\proyecto1\Lib\site-packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
    _warn_prf(average, modifier, msg_start, len(result))
c:\Users\ialca\anaconda3\envs\proyecto1\Lib\site-packages\sklearn\metrics\_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
    _warn_prf(average, modifier, msg_start, len(result))
```

Gráfico entre la diferencia entre la validación y el original



REPORTING DEL MODELO

Matrix de Confusión MultiClass y ROC CHART MULTICLASS

In [87]:

```
# Crear subplots
f, ax = plt.subplots(1, 2, figsize=(16, 5))
ax = ax.flat

# Calcular la matriz de confusión
cm = confusion_matrix(val_y, pred)

# Crear un mapa de calor de la matriz de confusión
sns.heatmap(cm, annot=True, cmap="Blues", fmt="d", cbar=False, ax=ax[0])
ax[0].set_xlabel("Predicted Labels")
ax[0].set_ylabel("True Labels")
ax[0].set_title("Confusion Matrix")

# Binarizar las etiquetas verdaderas y las predicciones para cada clase
n_classes = len(np.unique(val_y))
binarized_val_y = label_binarize(val_y, classes=np.arange(n_classes))
binarized_pred = label_binarize(pred, classes=np.arange(n_classes))

# Calcular la curva ROC y el área bajo la curva (AUC) para cada clase
```

```

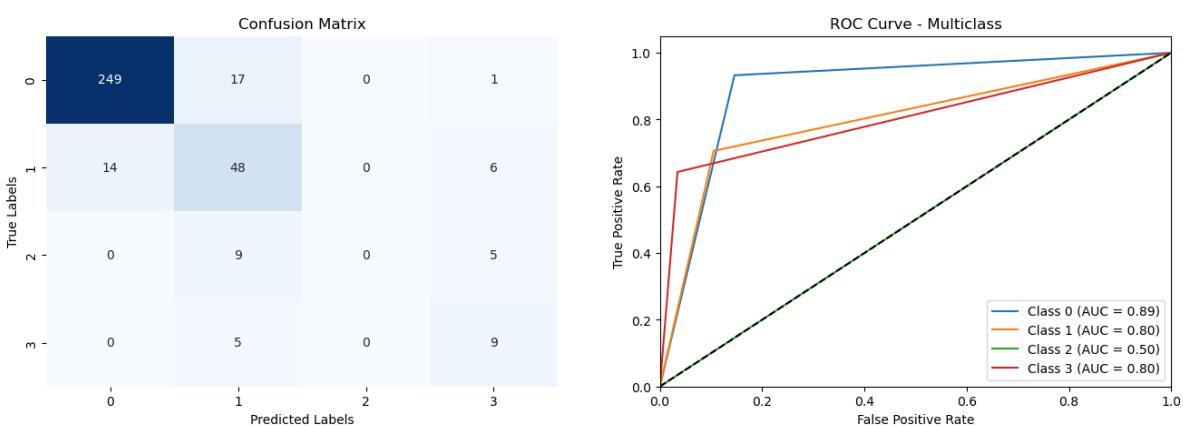
fpr = dict()
tpr = dict()
roc_auc = dict()
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(binarized_val_y[:, i], binarized_pred[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

for i in range(n_classes):
    ax[1].plot(fpr[i], tpr[i], label='Class {0} (AUC = {1:.2f})'.format(i, roc_auc[i]))

ax[1].plot([0, 1], [0, 1], 'k--')
ax[1].set_xlim([0.0, 1.0])
ax[1].set_ylim([0.0, 1.05])
ax[1].set_xlabel('False Positive Rate')
ax[1].set_ylabel('True Positive Rate')
ax[1].set_title('ROC Curve - Multiclass')
ax[1].legend(loc="lower right")

```

Out[87]: <matplotlib.legend.Legend at 0x1f8d1ea81d0>



CONCLUSIÓN: El modelo ha dado muy buenos resultados con los datos de validación (accuracy = 98%), no se aprecia sobre-ajuste del modelo. De todas formas vamos a realizar un preselección de variables y generar un segundo modelo con las variables preseleccionadas sin parámetros.

GUARDAR BEST_ESTIMATOR, PARÁMETROS Y RESULTADOS DEL TEST Y LA VALIDACIÓN

Guardar el mejor estimador

In [88]:

```

version_estimator = '_v02'
nombre_best_estimator = m_best_estimator + version_estimator + '.pickle'
nombre_best_estimator

```

Out[88]:

```
'BaggingClassifier_v02.pickle'
```

In [89]:

```
m_best_estimator
```

Out[89]:

```
'BaggingClassifier'
```

In [90]:

```

ruta_pipe_entrenamiento = '../../../../../04_Modelos/Best_estimator/' + nombre_best_estimator

with open(ruta_pipe_entrenamiento, mode='wb') as file:
    cloudpickle.dump(modelo_best_estimator, file)

```

```
In [91]: # Añadir comentarios sobre el modelo y definimos predictoras y target
```

```
comentarios = "Modelo Base con preselección de variables (3)"  
x_columns = list(x.columns)  
y_target = y.name
```

```
In [92]: #Cargamos la lista con los resultados
```

```
resultado = {'m_Best_estimator': m_best_estimator,  
            'm_Best_paramans' : m_best_params,  
            'm_Best_Score': m_best_score,  
            't_accuracy': t_accuracy,  
            't_report': t_report,  
            'v_accuracy': v_accuracy,  
            'v_report': v_report,  
            'comentarios': comentarios,  
            'predictoras_X': x_columns,  
            'target_y': y_target  
        }  
resultado= pd.Series(resultado,name=nombre_best_estimator)  
resultado
```

```
Out[92]: m_Best_estimator                               BaggingClassifier  
m_Best_paramans          ['algoritmo': BaggingClassifier()]  
m_Best_Score               0.8075391576749043  
t_accuracy                  0.819362  
t_report                    precision    recall   f1-score ...  
v_accuracy                  0.842975  
v_report                     precision    recall   f1-score ...  
comentarios      Modelo Base con preselección de variables (3)  
predictoras_X      [nivel_seguridad_oe, n_personas_oe, precio_com...  
target_y                           aceptabilidad_oe  
Name: BaggingClassifier_v02.pickle, dtype: object
```

```
In [93]: df_best = pd.read_excel('../04_Modelos/Best_estimator/Best_estimator.xlsx',index
```

```
In [94]: df_best = pd.concat([df_best, resultado], ignore_index= False, axis=1)  
df_best.to_excel('../04_Modelos/Best_estimator/Best_estimator.xlsx')
```

07 - THE BEST ESTIMATOR - V2 (CLASSIFIER)

XGBClassifier con preselección de variables (5)

Los modelos que vamos a poner a competir con mediante el GridSearch:

Model	Accuracy	Balanced Accuracy	ROC AUC	F1 Score	Time Taken
LGBMClassifier	0.98	0.94	None	0.98	0.26
XGBClassifier	0.98	0.94	None	0.98	0.16
DecisionTreeClassifier	0.96	0.90	None	0.96	0.01
BaggingClassifier	0.96	0.86	None	0.96	0.04
ExtraTreesClassifier	0.96	0.83	None	0.96	0.13

IMPORTACIÓN DE PAQUETES

```
In [1]: import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns

from sklearn.model_selection import train_test_split

# Modelos

from lightgbm import LGBMClassifier
from xgboost import XGBClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.ensemble import ExtraTreesClassifier

# Optimizar modelo

from sklearn.model_selection import GridSearchCV

#Métricas de evaluación

from sklearn.metrics import roc_curve, auc
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.preprocessing import label_binarize
from sklearn.metrics import classification_report

#Crear Pipeline
from sklearn.pipeline import Pipeline

#Guardar Modelo

import cloudpickle
```

IMPORTACIÓN DE DATOS

CARGAR LOS DATOS

SEPARAR PREDICTIVAS Y TARGET

```
In [2]: x = pd.read_pickle('../..../02_Datos/03_Trabajo/x_preseleccionado.pickle')
y = pd.read_pickle('../..../02_Datos/03_Trabajo/y_preseleccionado.pickle')
```

MODELIZAR

RESERVAR EL DATASET DE VALIDACIÓN

```
In [3]: train_x, val_x, train_y, val_y = train_test_split(x, y, test_size= 0.3, random_stan
```

CREAR PIPE Y EL DICCIONARIO CON LOS ALGORITMOS, PARÁMETROS Y VALORES

```
In [4]: pipe = Pipeline([('algoritmo', XGBClassifier())])

grid = [
    {
        'algoritmo' : [LGBMClassifier()]
    },
    {
        'algoritmo' : [XGBClassifier()]
    },
    {
        'algoritmo' : [DecisionTreeClassifier()]
    },
    {
        'algoritmo' : [BaggingClassifier()]
    },
    {
        'algoritmo' : [ExtraTreesClassifier()]
    }
]
```

OPTIMIZAR LOS PARÁMETROS CON GRIDSEARCH

```
In [5]: grid_search = GridSearchCV(estimator= pipe,
                                param_grid= grid,
                                cv=5,
                                scoring= 'accuracy',
                                verbose= 0,
                                n_jobs= -1)

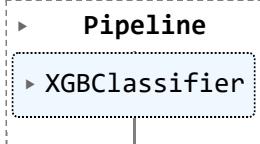
modelo = grid_search.fit(train_x, train_y)
pd.DataFrame(grid_search.cv_results_).sort_values(by = 'rank_test_score')
```

Out[5]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_algoritmo
1	0.430057	0.091237	0.007600	0.000898	XGBClassifier(base_score=None, booster=None, c...
0	2.576181	0.343715	0.008584	0.001238	LGBMClassifier()
3	0.047913	0.001084	0.007283	0.000401	BaggingClassifier()
2	0.005566	0.000380	0.003791	0.000397	DecisionTreeClassifier()
4	0.296542	0.006411	0.028733	0.000975	ExtraTreesClassifier()

In [6]: `modelo.best_estimator_`

Out[6]:



In [7]: `modelo.best_params_`

Out[7]:

```
{
  'algoritmo': XGBClassifier(base_score=None, booster=None, callbacks=None,
                             colsample_bylevel=None, colsample_bynode=None,
                             colsample_bytree=None, early_stopping_rounds=None,
                             enable_categorical=False, eval_metric=None, feature_types=None,
                             gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
                             interaction_constraints=None, learning_rate=None, max_bin=None,
                             max_cat_threshold=None, max_cat_to_onehot=None,
                             max_delta_step=None, max_depth=None, max_leaves=None,
                             min_child_weight=None, missing=nan, monotone_constraints=None,
                             n_estimators=100, n_jobs=None, num_parallel_tree=None,
                             predictor=None, random_state=None, ...)
}
```

In [8]: `modelo.best_score_`

Out[8]:

GUARDAR MODELO.BEST_ESTIMATOR, PARÁMETROS Y SCORE

In [9]:

```

m_best_estimator = str(modelo.best_estimator_[0])
m_best_estimator = m_best_estimator.split('(')[0]
m_best_params = str(modelo.best_params_)
m_best_score = str(modelo.best_score_)
  
```

EVALUAR MODELO

PREDECIR Y EVALUAR SOBRE EL TRAIN

PREDECIR SOBRE EL TRAIN

```
In [10]: pred = modelo.best_estimator_.predict(train_x)
```

EVALUAR SOBRE EL TRAIN

```
In [11]: t_accuracy = accuracy_score(train_y, pred)
t_report = classification_report(train_y, pred)

print(f"Accuracy: {t_accuracy}")
print(f"Classification Report:\n{t_report}")
```

```
Accuracy: 0.974025974025974
Classification Report:
precision    recall   f1-score   support
          0.0      1.00      0.98      0.99      591
          1.0      0.93      0.97      0.95      191
          2.0      0.86      0.94      0.90       34
          3.0      0.97      0.97      0.97       31

accuracy                   0.97      847
macro avg                  0.94      0.95      847
weighted avg                0.98      0.97      0.97      847
```

PREDECIR Y EVALUAR SOBRE LA VALIDACIÓN

PREDECIR SOBRE LA VALIDACIÓN

```
In [12]: pred = modelo.best_estimator_.predict(val_x)
```

EVALUAR SOBRE LA VALIDACIÓN

```
In [13]: v_accuracy = accuracy_score(val_y, pred)
v_report = classification_report(val_y, pred)

print(f"Accuracy: {v_accuracy}")
print(f"Classification Report:\n{v_report}")
```

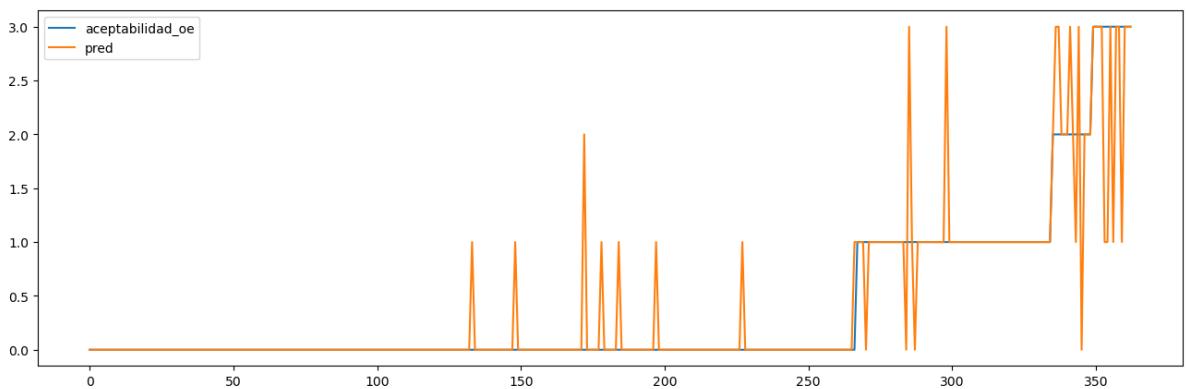
```
Accuracy: 0.9366391184573003
Classification Report:
precision    recall   f1-score   support
          0.0      0.98      0.97      0.98      267
          1.0      0.84      0.93      0.88       68
          2.0      0.89      0.57      0.70       14
          3.0      0.62      0.71      0.67       14

accuracy                   0.94      363
macro avg                  0.83      0.80      0.81      363
weighted avg                0.94      0.94      0.94      363
```

Gráfico entre la diferencia entre la validación y el original

```
In [14]: test_y = val_y.reset_index().copy()
test_pred = pd.Series(pred).to_frame()
df_pred = pd.concat([test_y,test_pred], axis= 1).sort_values(by='aceptabilidad_oe')
df_pred.rename(columns = {0:'pred'}, inplace = True)
```

```
df_pred = df_pred.drop(columns= ['level_0'])
df_pred[['aceptabilidad_oe','pred']].plot(figsize=(16,5));
```



REPORTING DEL MODELO

Matrix de Confusión MultiClass y ROC CHART MULTICLASS

```
In [15]: # Crear subplots
f, ax = plt.subplots(1, 2, figsize=(16, 5))
ax = ax.flat

# Calcular la matriz de confusión
cm = confusion_matrix(val_y, pred)

# Crear un mapa de calor de la matriz de confusión
sns.heatmap(cm, annot=True, cmap="Blues", fmt="d", cbar=False, ax=ax[0])
ax[0].set_xlabel("Predicted Labels")
ax[0].set_ylabel("True Labels")
ax[0].set_title("Confusion Matrix")

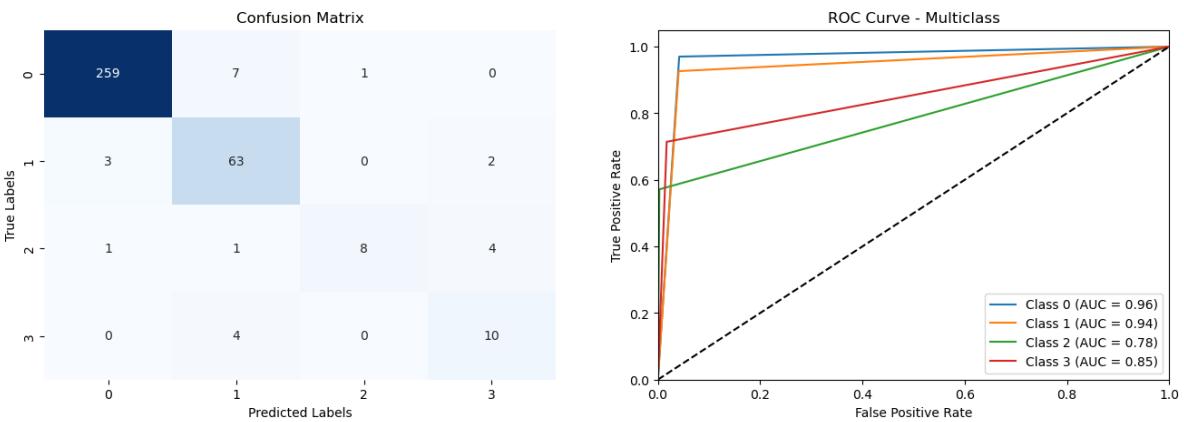
# Binarizar las etiquetas verdaderas y las predicciones para cada clase
n_classes = len(np.unique(val_y))
binarized_val_y = label_binarize(val_y, classes=np.arange(n_classes))
binarized_pred = label_binarize(pred, classes=np.arange(n_classes))

# Calcular la curva ROC y el área bajo la curva (AUC) para cada clase
fpr = dict()
tpr = dict()
roc_auc = dict()
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(binarized_val_y[:, i], binarized_pred[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

for i in range(n_classes):
    ax[1].plot(fpr[i], tpr[i], label='Class {} (AUC = {:.2f})'.format(i, roc_auc[i]))

ax[1].plot([0, 1], [0, 1], 'k--')
ax[1].set_xlim([0.0, 1.0])
ax[1].set_ylim([0.0, 1.05])
ax[1].set_xlabel('False Positive Rate')
ax[1].set_ylabel('True Positive Rate')
ax[1].set_title('ROC Curve - Multiclass')
ax[1].legend(loc="lower right")
```

Out[15]: <matplotlib.legend.Legend at 0x2cbc5250350>



CONCLUSIÓN: El modelo ha dado muy buenos resultados con los datos de validación (accuracy = 98%), no se aprecia sobre-ajuste del modelo. De todas formas vamos a realizar un preselección de variables y generar un segundo modelo con las variables preseleccionadas sin parámetros.

GUARDAR BEST_ESTIMATOR, PARÁMETROS Y RESULTADOS DEL TEST Y LA VALIDACIÓN

Guardar el mejor estimador

```
In [16]: version_estimator = '_v03'
nombre_best_estimator = m_best_estimator + version_estimator + '.pickle'
nombre_best_estimator
```

```
Out[16]: 'XGBClassifier_v03.pickle'
```

```
In [17]: m_best_estimator
```

```
Out[17]: 'XGBClassifier'
```

```
In [18]: ruta_pipe_entrenamiento = '../../../../../04_Modelos/Best_estimator/' + nombre_best_estimator

with open (ruta_pipe_entrenamiento, mode= 'wb') as file:
    cloudpickle.dump(modelo_best_estimator, file)
```

```
In [19]: # Añadir comentarios sobre el modelo y definimos predictoras y target

comentarios = "Modelo Base con preselección de variables (5)"
x_columns = list(x.columns)
y_target = y.name
```

```
In [20]: #Cargamos la lista con los resultados
resultado = {'m_Best_estimator': m_best_estimator,
             'm_Best_paramans' : m_best_params,
             'm_Best_Score': m_best_score,
             't_accuracy': t_accuracy,
             't_report': t_report,
             'v_accuracy': v_accuracy,
             'v_report': v_report,
             'comentarios': comentarios,
             'predictoras_X': x_columns,
             'target_y': y_target
            }
```

```
resultado= pd.Series(resultado,name=nombre_best_estimator)
resultado
```

```
Out[20]:
```

		XGBClassifier
m_Best_estimator	{'algoritmo': XGBClassifier(base_score=None, b...	0.942178907065785
m_Best_paramans		0.974026
m_Best_Score		
t_accuracy		0.936639
t_report	precision recall f1-score ...	
v_accuracy		
v_report	precision recall f1-score ...	
comentarios	Modelo Base con preselección de variables (5)	
predictoras_X	[nivel_seguridad_oe, n_personas_oe, precio_mto...	aceptabilidad_oe
target_y		
Name:	XGBClassifier_v03.pickle, dtype:	object

```
In [21]: df_best = pd.read_excel('../..../04_Modelos/Best_estimator/Best_estimator.xlsx',inde
```

```
In [22]: df_best = pd.concat([df_best, resultado], ignore_index= False, axis=1)
df_best.to_excel('../..../04_Modelos/Best_estimator/Best_estimator.xlsx')
```

CONCLUSIÓN: Realizar una preselección de variables no ha mejorado el accuracy. Así que realizaremos el modelo sobre el modelo base (XGBClassifier) con todos las variables (6) y realizaremos incluiremos parámetros para evitar el sobre-ajuste.

07 - THE BEST ESTIMATOR - V3

XGBCLASSIFIER con parámetros para evitar el sobreajuste

Los modelos que vamos a poner a competir con mediante el GridSearch:

Model	Accuracy	Balanced Accuracy	ROC AUC	F1 Score	Time Taken
LGBMClassifier	0.98	0.94	None	0.98	0.26
XGBClassifier	0.98	0.94	None	0.98	0.16
DecisionTreeClassifier	0.96	0.90	None	0.96	0.01
BaggingClassifier	0.96	0.86	None	0.96	0.04
ExtraTreesClassifier	0.96	0.83	None	0.96	0.13

IMPORTACIÓN DE PAQUETES

```
In [1]: import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns

from sklearn.model_selection import train_test_split

# Modelos

from lightgbm import LGBMClassifier
from xgboost import XGBClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.ensemble import ExtraTreesClassifier

# Optimizar modelo

from sklearn.model_selection import GridSearchCV

#Métricas de evaluación

from sklearn.metrics import roc_curve, auc
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.preprocessing import label_binarize
from sklearn.metrics import classification_report

#Crear Pipeline
from sklearn.pipeline import Pipeline

#Guardar Modelo

import cloudpickle
```

IMPORTACIÓN DE DATOS

CARGAR LOS DATOS

```
In [2]: df_tablon = pd.read_pickle('../..../02_Datos/03_Trabajo/df_tablon.pickle')
df_tablon.head()
```

```
Out[2]:   precio_compra_oe  precio_mto_oe  n_puertas_oe  n_personas_oe  t_maletero_oe  nivel_seguridad_
0             3.0            3.0           0.0           0.0           0.0           0.0
1             3.0            3.0           0.0           0.0           0.0           0.0
2             3.0            3.0           0.0           0.0           0.0           0.0
3             3.0            3.0           0.0           0.0           0.0           1.0
4             3.0            3.0           0.0           0.0           0.0           1.0
```



SEPARAR PREDICTIVAS Y TARGET

```
In [3]: x = df_tablon.drop(columns= 'aceptabilidad_oe').copy()
y = df_tablon.aceptabilidad_oe.copy()
```

MODELIZAR

RESERVAR EL DATASET DE VALIDACIÓN

```
In [4]: train_x, val_x, train_y, val_y = train_test_split(x, y, test_size= 0.3, random_state= 42)
```

CREAR PIPE Y EL DICCIONARIO CON LOS ALGORITMOS, PARÁMETROS Y VALORES

```
In [5]: pipe = Pipeline([('algoritmo', XGBClassifier())])

grid = [
    {'algoritmo': [XGBClassifier()],
     'algoritmo_n_jobs': [-1],
     'algoritmo_verbose': [0],#para que no salgan warnings
     'algoritmo_learning_rate': [0.01,0.025,0.05,0.1],
     'algoritmo_max_depth': [5,10,20],
     'algoritmo_reg_alpha': [0,0.1,0.5,1],
     'algoritmo_reg_lambda': [0.01,0.1,1],
     'algoritmo_n_estimators': [100, 200, 300]}
]
```

OPTIMIZAR LOS PARÁMETROS CON GRIDSEARCH

```
In [6]: grid_search = GridSearchCV(estimator= pipe,
                                param_grid= grid,
                                cv=5,
```

```

        scoring= 'accuracy',
        verbose= 0,
        n_jobs= -1)

modelo = grid_search.fit(train_x, train_y)
pd.DataFrame(grid_search.cv_results_).sort_values(by = 'rank_test_score')

```

Out[6]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_algoritmo
283	1.219078	0.044291	0.013964	0.008005	XGBClassifier(base_score=None, booster=None, c...
319	1.216785	0.158522	0.009800	0.000415	XGBClassifier(base_score=None, booster=None, c...
426	0.876023	0.020046	0.008776	0.000399	XGBClassifier(base_score=None, booster=None, c...
379	1.161144	0.336347	0.011238	0.003223	XGBClassifier(base_score=None, booster=None, c...
282	1.171225	0.043015	0.010459	0.001344	XGBClassifier(base_score=None, booster=None, c...
...
2	0.696133	0.135814	0.024234	0.022825	XGBClassifier(base_score=None, booster=None, c...
10	0.574567	0.040011	0.014562	0.010104	XGBClassifier(base_score=None, booster=None, c...
9	0.675687	0.082778	0.007493	0.002158	XGBClassifier(base_score=None, booster=None, c...
11	0.480320	0.061353	0.007397	0.001348	XGBClassifier(base_score=None, booster=None, c...
8	0.689216	0.031235	0.023054	0.017838	XGBClassifier(base_score=None, booster=None, c...

432 rows × 21 columns

In [7]: `modelo.best_estimator_`

Out[7]:

```

▶ Pipeline
  ▶ XGBClassifier

```

In [8]: `modelo.best_params_`

```
Out[8]: {'algoritmo': XGBClassifier(base_score=None, booster=None, callbacks=None,
                                     colsample_bylevel=None, colsample_bynode=None,
                                     colsample_bytree=None, early_stopping_rounds=None,
                                     enable_categorical=False, eval_metric=None, feature_types=None,
                                     gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
                                     interaction_constraints=None, learning_rate=0.05, max_bin=None,
                                     max_cat_threshold=None, max_cat_to_onehot=None,
                                     max_delta_step=None, max_depth=10, max_leaves=None,
                                     min_child_weight=None, missing=nan, monotone_constraints=None,
                                     n_estimators=300, n_jobs=-1, num_parallel_tree=None,
                                     predictor=None, random_state=None, ...),
          'algoritmo__learning_rate': 0.05,
          'algoritmo__max_depth': 10,
          'algoritmo__n_estimators': 300,
          'algoritmo__n_jobs': -1,
          'algoritmo__reg_alpha': 0.5,
          'algoritmo__reg_lambda': 0.1,
          'algoritmo__verbosity': 0}
```

```
In [9]: modelo.best_score_
```

```
Out[9]: 0.9764009745910199
```

GUARDAR MODELO.BEST_ESTIMATOR, PARÁMETROS Y SCORE

```
In [10]: m_best_estimator = modelo
m_best_estimator = str(modelo.best_estimator_[0])
m_best_estimator = m_best_estimator.split('(')[0]
m_best_params = str(modelo.best_params_)
m_best_score = str(modelo.best_score_)
```

EVALUAR MODELO

PREDECIR Y EVALUAR SOBRE EL TRAIN

PREDECIR SOBRE EL TRAIN

```
In [11]: pred = modelo.best_estimator_.predict(train_x)
```

EVALUAR SOBRE EL TRAIN

```
In [12]: t_accuracy = accuracy_score(train_y, pred)
t_report = classification_report(train_y, pred)

print(f"Accuracy: {t_accuracy}")
print(f"Classification Report:\n{t_report}")
```

```

Accuracy: 1.0
Classification Report:
precision    recall   f1-score   support
0.0          1.00    1.00    1.00      591
1.0          1.00    1.00    1.00      191
2.0          1.00    1.00    1.00       34
3.0          1.00    1.00    1.00       31

accuracy           1.00      847
macro avg        1.00    1.00    1.00      847
weighted avg     1.00    1.00    1.00      847

```

PREDECIR Y EVALUAR SOBRE LA VALIDACIÓN

PREDECIR SOBRE LA VALIDACIÓN

```
In [13]: pred = modelo.best_estimator_.predict(val_x)
```

EVALUAR SOBRE LA VALIDACIÓN

```
In [14]: v_accuracy = accuracy_score(val_y, pred)
v_report = classification_report(val_y, pred)

print(f"Accuracy: {v_accuracy}")
print(f"Classification Report:\n{v_report}")
```

```

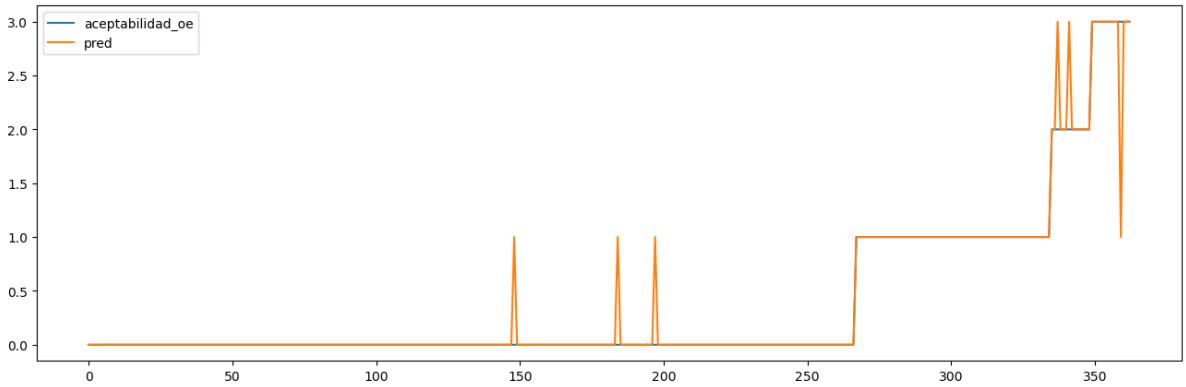
Accuracy: 0.9834710743801653
Classification Report:
precision    recall   f1-score   support
0.0          1.00    0.99    0.99      267
1.0          0.94    1.00    0.97       68
2.0          1.00    0.86    0.92       14
3.0          0.87    0.93    0.90       14

accuracy           0.98      363
macro avg        0.95    0.94    0.95      363
weighted avg     0.98    0.98    0.98      363

```

Gráfico entre la diferencia entre la validación y el original

```
In [15]: test_y = val_y.reset_index().copy()
test_pred = pd.Series(pred).to_frame()
df_pred = pd.concat([test_y,test_pred], axis= 1).sort_values(by='aceptabilidad_oe')
df_pred.rename(columns = {0:'pred'}, inplace = True)
df_pred = df_pred.drop(columns= ['level_0'])
df_pred[['aceptabilidad_oe','pred']].plot(figsize=(16,5));
```



REPORTING DEL MODELO

Matrix de Confusión MultiClass y ROC CHART MULTICLASS

```
In [16]: # Crear subplots
f, ax = plt.subplots(1, 2, figsize=(16, 5))
ax = ax.flat

# Calcular la matriz de confusión
cm = confusion_matrix(val_y, pred)

# Crear un mapa de calor de la matriz de confusión
sns.heatmap(cm, annot=True, cmap="Blues", fmt="d", cbar=False, ax=ax[0])
ax[0].set_xlabel("Predicted Labels")
ax[0].set_ylabel("True Labels")
ax[0].set_title("Confusion Matrix")

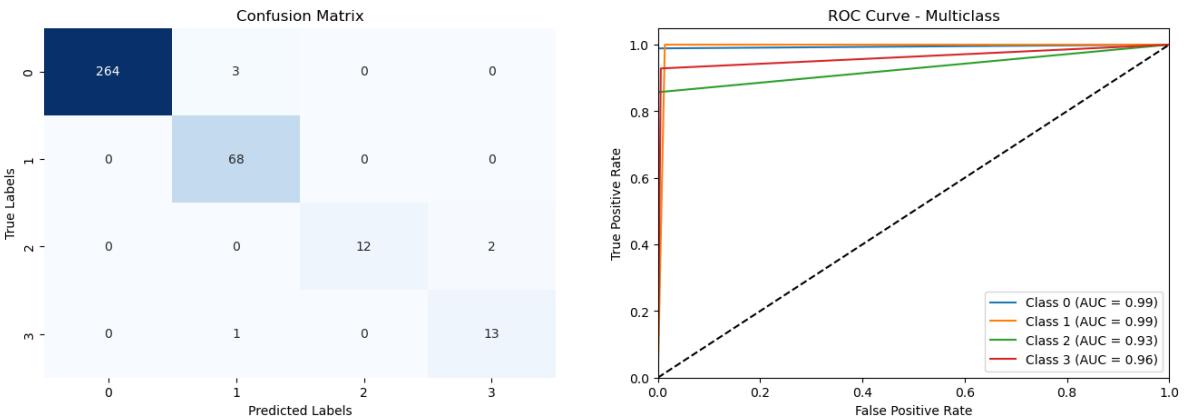
# Binarizar las etiquetas verdaderas y las predicciones para cada clase
n_classes = len(np.unique(val_y))
binarized_val_y = label_binarize(val_y, classes=np.arange(n_classes))
binarized_pred = label_binarize(pred, classes=np.arange(n_classes))

# Calcular la curva ROC y el área bajo la curva (AUC) para cada clase
fpr = dict()
tpr = dict()
roc_auc = dict()
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(binarized_val_y[:, i], binarized_pred[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

for i in range(n_classes):
    ax[1].plot(fpr[i], tpr[i], label='Class {0} (AUC = {1:.2f})'.format(i, roc_auc[i]))

ax[1].plot([0, 1], [0, 1], 'k--')
ax[1].set_xlim([0.0, 1.0])
ax[1].set_ylim([0.0, 1.05])
ax[1].set_xlabel('False Positive Rate')
ax[1].set_ylabel('True Positive Rate')
ax[1].set_title('ROC Curve - Multiclass')
ax[1].legend(loc="lower right")
```

Out[16]: <matplotlib.legend.Legend at 0x160850f5210>



CONCLUSIÓN: Parametrizando el XGBClassifier se obtiene el mismo accuracy.

GUARDAR BEST_ESTIMATOR, PARÁMETROS Y RESULTADOS DEL TEST Y LA VALIDACIÓN

Guardar el mejor estimador

```
In [17]: version_estimator = '_v03'
nombre_best_estimator = m_best_estimator + version_estimator + '.pickle'
nombre_best_estimator
```

```
Out[17]: 'XGBClassifier_v03.pickle'
```

```
In [18]: m_best_estimator
```

```
Out[18]: 'XGBClassifier'
```

```
In [19]: ruta_pipe_entrenamiento = '../../../../../04_Modelos/Best_estimator/' + nombre_best_estimator

with open (ruta_pipe_entrenamiento, mode= 'wb') as file:
    cloudpickle.dump(modelo_best_estimator, file)
```

```
In [20]: # Añadir comentarios sobre el modelo y definimos predictoras y target

comentarios = "XGBClassifier con hiper-parámetros para evitar el sobreajuste."
x_columns = list(x.columns)
y_target = y.name
```

```
In [21]: #Cargamos la lista con los resultados
resultado = {'m_Best_estimator': m_best_estimator,
             'm_Best_params' : m_best_params,
             'm_Best_Score': m_best_score,
             't_accuracy': t_accuracy,
             't_report': t_report,
             'v_accuracy': v_accuracy,
             'v_report': v_report,
             'comentarios': comentarios,
             'predictoras_X': x_columns,
             'target_y': y_target
            }
resultado= pd.Series(resultado, name=nombre_best_estimator)
resultado
```

```
Out[21]: m_Best_estimator                               XGBClassifier
m_Best_paramans      {'algoritmo': XGBClassifier(base_score=None, b...
m_Best_Score          0.9764009745910199
t_accuracy           1.0
t_report              precision    recall   f1-score ...
v_accuracy            0.983471
v_report               precision    recall   f1-score ...
comentarios           XGBClassifier con hiper-parámetros para evitar...
predictoras_X         [precio_compra_oe, precio_mto_oe, n_puertas_oe...
target_y              aceptabilidad_oe
Name: XGBClassifier_v03.pickle, dtype: object
```

```
In [22]: df_best = pd.read_excel('../04_Modelos/Best_estimator/Best_estimator.xlsx', index...
```

```
In [23]: df_best = pd.concat([df_best, resultado], ignore_index= False, axis=1)
df_best.to_excel('../04_Modelos/Best_estimator/Best_estimator.xlsx')
```

08 - MODELO CLASIFICACIÓN TRAS THE BEST ESTIMATOR Y PRESELECCIÓN DE VARIABLES

v3 - XGBCLASSIFIER con parámetros para evitar el sobreajuste

Hemos comparado varios modelos de clasificación con parámetros y preselección de variables. Pero el modelo que ha obtenido mejores resultados ha sido el modelo XGB Classifier (v3) con parámetros para evitar el sobre ajuste y todas las variables del dataset.

En este notebook entrenaremos el modelo definitivo (v3) con todos los datos de trabajo y crearemos el modelo de ejecucción para predecir con los datos de prueba (datos nuevos)

- 'algoritmo': XGBClassifier
- 'algoritmo_learning_rate': 0.05,
- 'algoritmo_max_depth': 10,
- 'algoritmo_n_estimators': 300,
- 'algoritmo_n_jobs': -1,
- 'algoritmo_reg_alpha': 0.5,
- 'algoritmo_reg_lambda': 0.1,
- 'algoritmo_verbose': 0

IMPORTACIÓN DE PAQUETES

```
In [1]: import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns

from xgboost import XGBClassifier

#metricas de evaluación
from sklearn.metrics import roc_curve, auc
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.preprocessing import label_binarize
from sklearn.metrics import classification_report

#Crear Pipeline
from sklearn.pipeline import make_pipeline

import cloudpickle
```

IMPORTAR LOS DATOS

La preselección de variables no ha dado mejor resultados. Así que utilizaremos todas las variables del df_tablon.pickle

CARGAR LOS DATOS

```
In [2]: df = pd.read_pickle('../..../02_Datos/03_Trabajo/df_tablon.pickle')  
df.head()
```

```
Out[2]:  precio_compra_oe  precio_mto_oe  n_puertas_oe  n_personas_oe  t_maletero_oe  nivel_seguridad_  
0          3.0            3.0           0.0            0.0            0.0            0.0  
1          3.0            3.0           0.0            0.0            0.0            0.0  
2          3.0            3.0           0.0            0.0            0.0            0.0  
3          3.0            3.0           0.0            0.0            0.0            1.0  
4          3.0            3.0           0.0            0.0            0.0            1.0
```

SEPARAR PREDICTORAS Y TARGET

```
In [3]: target = 'aceptabilidad_oe'  
x = df.drop(columns= target).copy()  
y = df[target].copy()
```

MODELIZAR

CARTGAR EL MEJOR MODELO CON EL ALGORITMO, PARÁMETROS Y VALORES

```
In [6]: modelo = pd.read_pickle('../..../04_Modelos/Best_Estimator/XGBClassifier_v03.pickle')
```

EXAMINAR MODELO

```
In [14]: modelo.best_estimator_
```

```
Out[14]: ▶ Pipeline  
         ▶ XGBClassifier
```

```
In [15]: modelo.best_params_
```

```
Out[15]: {'algoritmo': XGBClassifier(base_score=None, booster=None, callbacks=None,
                                         colsample_bylevel=None, colsample_bynode=None,
                                         colsample_bytree=None, early_stopping_rounds=None,
                                         enable_categorical=False, eval_metric=None, feature_types=None,
                                         gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
                                         interaction_constraints=None, learning_rate=0.05, max_bin=None,
                                         max_cat_threshold=None, max_cat_to_onehot=None,
                                         max_delta_step=None, max_depth=10, max_leaves=None,
                                         min_child_weight=None, missing=nan, monotone_constraints=None,
                                         n_estimators=300, n_jobs=-1, num_parallel_tree=None,
                                         predictor=None, random_state=None, ...),
            'algoritmo_learning_rate': 0.05,
            'algoritmo_max_depth': 10,
            'algoritmo_n_estimators': 300,
            'algoritmo_n_jobs': -1,
            'algoritmo_reg_alpha': 0.5,
            'algoritmo_reg_lambda': 0.1,
            'algoritmo_verbose': 0}
```

PREDECIR SOBRE LA VALIDACIÓN

```
In [16]: pred = modelo.best_estimator_.predict(x)
```

EVALUAR SOBRE LA VALIDACIÓN

```
In [17]: t_accuracy = accuracy_score(y, pred)
t_report = classification_report(y, pred)

print(f"Accuracy: {t_accuracy}\n")
print(f"Classification Report:\n{t_report}")
```

```
Accuracy: 0.9950413223140496
```

```
Classification Report:
      precision    recall   f1-score   support
      0.0       1.00     1.00     1.00      858
      1.0       0.98     1.00     0.99      259
      2.0       1.00     0.96     0.98      48
      3.0       0.96     0.98     0.97      45

      accuracy                           1.00      1210
     macro avg       0.99     0.98     0.98      1210
weighted avg       1.00     1.00     1.00      1210
```

REPORTING DEL MODELO

Matrix de Confusión MultiClass y ROC CHART MULTICLASS

```
In [19]: # Crear subplots
f, ax = plt.subplots(1, 2, figsize=(16, 5))
ax = ax.flat

# Calcular la matriz de confusión
cm = confusion_matrix(y, pred)

# Crear un mapa de calor de la matriz de confusión
sns.heatmap(cm, annot=True, cmap="Blues", fmt="d", cbar=False, ax=ax[0])
ax[0].set_xlabel("Predicted Labels")
```

```

ax[0].set_ylabel("True Labels")
ax[0].set_title("Confusion Matrix")

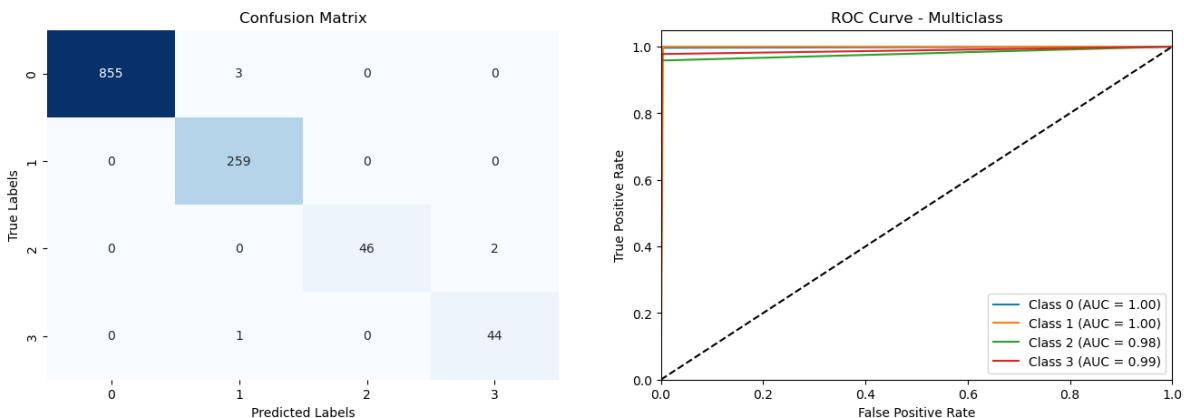
# Binarizar las etiquetas verdaderas y las predicciones para cada clase
n_classes = len(np.unique(y))
binarized_val_y = label_binarize(y, classes=np.arange(n_classes))
binarized_pred = label_binarize(pred, classes=np.arange(n_classes))

# Calcular la curva ROC y el área bajo la curva (AUC) para cada clase
fpr = dict()
tpr = dict()
roc_auc = dict()
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(binarized_val_y[:, i], binarized_pred[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

for i in range(n_classes):
    ax[1].plot(fpr[i], tpr[i], label='Class {} (AUC = {:.2f})'.format(i, roc_auc[i]))

ax[1].plot([0, 1], [0, 1], 'k--')
ax[1].set_xlim([0.0, 1.0])
ax[1].set_ylim([0.0, 1.05])
ax[1].set_xlabel('False Positive Rate')
ax[1].set_ylabel('True Positive Rate')
ax[1].set_title('ROC Curve - Multiclass')
ax[1].legend(loc="lower right");

```



CREAR PIPELINE DE ENTRENAMIENTO Y EJECUCIÓN

INSTANCIAR EL MODELO

```
In [20]: modelo = XGBClassifier(
    learning_rate = 0.05,
    max_depth = 10,
    n_estimators = 300,
    n_jobs = -1,
    reg_alpha = 0.5,
    reg_lambda = 0.1,
    verbosity = 0
)
```

CREAR Y GUARDAR EL PIPE FINAL DE ENTRENAMIENTO

```
In [21]: #crear pipe de entrenamiento  
pipe_entrenamiento = make_pipeline(modelo)  
  
In [22]: # Guardar pipe de entrenamiento  
nombre_pipe_entrenamiento = 'pipe_entrenamiento_v3.pickle'  
ruta_pipe_entrenamiento = '../..../04_Modelos/' + nombre_pipe_entrenamiento  
  
with open (ruta_pipe_entrenamiento, mode= 'wb') as file:  
    cloudpickle.dump(pipe_entrenamiento, file)
```

ENTRENAR Y GUARDAR EL PIPE DE EJECUCIÓN

```
In [24]: # Entrenar pipe de ejecución  
pipe_ejecucion = pipe_entrenamiento.fit(x,y)  
  
In [25]: nombre_pipe_ejecucion = 'pipe_ejecucion_v3.pickle'  
ruta_pipe_ejecucion = '../..../04_Modelos/' + nombre_pipe_ejecucion  
  
with open (ruta_pipe_ejecucion, mode= 'wb') as file:  
    cloudpickle.dump(pipe_ejecucion, file)
```

CONCLUSIÓN: El modelo ha dado muy buenos resultados incluyendo todos los datos del df_tablon, obteniendo un 99,5% de accuracy.

9 - MODELO DE EJECUCIÓN CON LOS DATOS DE PRUEBA

En este modelo vamos a cargar el dataset de prueba y realizaremos calidad de datos, transformación de variables, separaremos predictoras y target, predeciremos sobre los datos de prueba y evaluaremos el modelo.

IMPORTAR LOS PAQUETES

```
In [32]: import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns

#Transformación de variables
from sklearn.preprocessing import OrdinalEncoder

#metricas de evaluación
from sklearn.metrics import roc_curve, auc
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.preprocessing import label_binarize
from sklearn.metrics import classification_report
```

CARGAR LOS DATOS

```
In [5]: df = pd.read_csv('../02_Datos/02_Validacion/prueba.csv', index_col= 0)
df.head()
```

```
Out[5]:    precio_compra  precio_mto  n_puertas  n_personas  t_maletero  nivel_seguridad  aceptabilidad
1134        med         med          4           2      small        low        una
880        med        vhigh         2           4       big        med        a
1578       low         med          4           4      med        low        una
496        high        vhigh         4           4      small        med        una
153        vhigh        high         3        more      small        low        una
```

ESTRUCTURA DEL DATASET

TRANSFORMACIÓN DE VARIABLES

```
In [13]: # Variables a aplicar Ordinal Encoder
var_oe = ['precio_compra', 'precio_mto', 'n_puertas', 'n_personas',
          't_maletero', 'nivel_seguridad', 'aceptabilidad']

# Orden de los valores de las variables
```

```

orden_precio_compra = ['low','med','high','vhigh']
orden_precio_mto = ['low','med','high','vhigh']
orden_n_puertas = ['2','3','4','5more']
orden_n_personas = ['2','4','more']
orden_t_maletero = ['small','med','big']
orden_nivel_seguridad = ['low','med','high']
orden_aceptabilidad = ['unacc','acc','good','vgood']

categorias = [orden_precio_compra,
              orden_precio_mto,
              orden_n_puertas,
              orden_n_personas,
              orden_t_maletero,
              orden_nivel_seguridad,
              orden_aceptabilidad]

# Instanciar
oe = OrdinalEncoder(categories = categorias,
                     handle_unknown = 'use_encoded_value',
                     unknown_value = 10)
# Entrenar y aplicar
df_oe = oe.fit_transform(df[var_oe])

# Guardar como dataframe
nombres_oe = [variable + '_oe' for variable in var_oe]
df_oe = pd.DataFrame(df_oe, columns = nombres_oe)
df_oe.head()

```

Out[13]:

	precio_compra_oe	precio_mto_oe	n_puertas_oe	n_personas_oe	t_maletero_oe	nivel_seguridad_oe
0	1.0	1.0	2.0	0.0	0.0	1.0
1	1.0	3.0	0.0	1.0	2.0	1.0
2	0.0	1.0	2.0	1.0	1.0	0.0
3	2.0	3.0	2.0	1.0	0.0	1.0
4	3.0	2.0	1.0	2.0	0.0	1.0

In [17]: df_oe.shape

Out[17]: (518, 7)

SEPARAR PREDICTORAS Y TARGET

In [23]:

```
x = df_oe.drop(columns = 'aceptabilidad_oe').copy()
y = df_oe.aceptabilidad_oe.copy()
```

MODELIZAR CON EL PIPE DE EJECUCIÓN

CARGAMOS EL PIPE DE EJECUCIÓN

In [27]:

```
modelo = pd.read_pickle('../04_Modelos/pipe_ejecucion_v3.pickle')
```

PREDECIR Y EVALUAR CON DATASET DE PRUEBA

PREDECIR SOBRE LOS DATOS

```
In [30]: pred = modelo.predict(x)
```

EVALUAR SOBRE LOS DATOS

```
In [34]: v_accuracy = accuracy_score(y, pred)
v_report = classification_report(y, pred)

print(f"Accuracy: {v_accuracy}")
print(f"Classification Report:\n{v_report}")
```

```
Accuracy: 0.9864864864864865
Classification Report:
              precision    recall   f1-score   support

             0.0       0.99     1.00     1.00      352
             1.0       0.98     0.97     0.97      125
             2.0       0.90     0.90     0.90       21
             3.0       1.00     1.00     1.00       20

      accuracy                           0.99      518
   macro avg       0.97     0.97     0.97      518
weighted avg       0.99     0.99     0.99      518
```

REPORTING DEL MODELO

Matrix de Confusión MultiClass y ROC CHART MULTICLASS

```
In [42]: # Crear subplots
f, ax = plt.subplots(1, 2, figsize=(16,8))
ax = ax.flat

# Calcular la matriz de confusión
cm = confusion_matrix(y, pred)

# Crear un mapa de calor de la matriz de confusión
sns.heatmap(cm, annot=True, cmap="Blues", fmt="d", cbar=False, ax=ax[0])
ax[0].set_xlabel("Predicted Labels")
ax[0].set_ylabel("True Labels")
ax[0].set_title("Confusion Matrix")

# Binarizar las etiquetas verdaderas y las predicciones para cada clase
n_classes = len(np.unique(y))
binarized_val_y = label_binarize(y, classes=np.arange(n_classes))
binarized_pred = label_binarize(pred, classes=np.arange(n_classes))

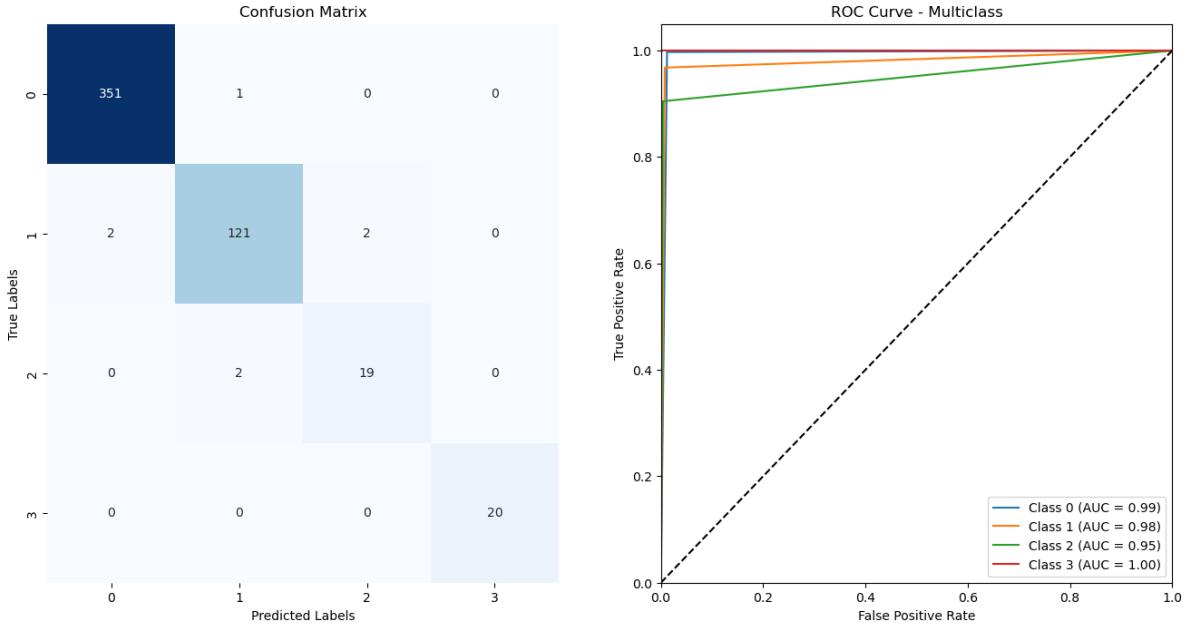
# Calcular la curva ROC y el área bajo la curva (AUC) para cada clase
fpr = dict()
tpr = dict()
roc_auc = dict()
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(binarized_val_y[:, i], binarized_pred[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

for i in range(n_classes):
    ax[1].plot(fpr[i], tpr[i], label='Class {} (AUC = {:.2f})'.format(i, roc_auc[i]))
```

```

ax[1].plot([0, 1], [0, 1], 'k--')
ax[1].set_xlim([0.0, 1.0])
ax[1].set_ylim([0.0, 1.05])
ax[1].set_xlabel('False Positive Rate')
ax[1].set_ylabel('True Positive Rate')
ax[1].set_title('ROC Curve - Multiclass')
ax[1].legend(loc="lower right");

```



CONCLUSIÓN:

El modelo ha funcionado muy bien con los datos de prueba, no se aprecia sobre ajuste del modelo y se ha obtenido tanto en train, validación y prueba (nuevos datos) un accuracy del 99%. Se revisamos cada clase, la predicción está por encima del 95%, siendo la clase 2 (good - buena) la clase con peor predicción pero el AUC está dentro de unos valores de predicción muy buenos.