



Challenge Analítica Avanzada e Inteligencia Artificial

Centro de Investigación Coppel

Elaborado por: Jhonathan Santacana

Descripción del problema

El objetivo de este problema es desarrollar un modelo de contactabilidad que asocie una probabilidad de contacto a cada cliente en función de ciertos segmentos horarios. Para resolver este problema, se utilizará el archivo 20210513_Challenge_AA.csv, que contiene información sociodemográfica y transaccional de los clientes.

I. Análisis preliminar y proceso ETL:

El análisis preliminar y el proceso ETL son pasos importantes en el manejo de datos. El proceso implica cargar los datos del archivo .csv, seleccionar y filtrar el conjunto de datos, y realizar cálculos de estadísticas básicas. También implica la visualización de los datos a través de gráficos y el análisis de las correlaciones entre las variables. Además, se establece un criterio para determinar los factores influyentes en relación con la variable objetivo. Este proceso ayuda a comprender la estructura de los datos, identificar patrones y relaciones, y preparar los datos para su análisis posterior.

Se procede a cargar las librerías a utilizar en la implementación de la solución.

In [139...]

```
# Librerias a utilizar, por defecto ya vienen instaladas en el entorno "pip install"
# para mejor visualizacion de los df y manejo de estadisticas:
# ! pip install pandasgui
# ! pip install pandas-profiling
# ! pip install seaborn
# ! pip install scipy
#! pip install optbinning
#! pip install ortools == 9.4.1874
#! pip install -U scikit-learn
```

```

import os # Librería para interactuar con el sistema operativo
import datetime # Librería para trabajar con fechas y horas
from IPython.display import clear_output # Librería para limpiar la salida en el navegador
import pandas as pd # Librería para manipulación y análisis de datos en formato tabular
import numpy as np # Librería para operaciones numéricas eficientes
import seaborn as sns # Librería para visualización de datos estadísticos
import matplotlib.pyplot as plt # Librería para visualización de gráficos
from scipy.stats import skew # Librería para cálculo de sesgo en distribuciones
from scipy.stats import kurtosis # Librería para cálculo de curtosis en distribuciones
import statsmodels.api as sm # Librería para modelado estadístico y análisis de regresión
from optbinning import OptimalBinning # Librería para el binning óptimo de variables
from optbinning import BinningProcess # Librería para el proceso de binning
import sklearn # Librería para aprendizaje automático (machine learning)
from sklearn import preprocessing # Submódulo de sklearn para preprocesamiento de datos
from sklearn import linear_model # Submódulo de sklearn para modelos lineales
from sklearn import model_selection # Submódulo de sklearn para selección de modelos
from sklearn.metrics import classification_report # Función para generar informe de clasificación
from sklearn.metrics import confusion_matrix # Función para generar matriz de confusión
from sklearn.metrics import accuracy_score # Función para calcular la precisión de un modelo
from sklearn.model_selection import train_test_split # Función para dividir datos en conjuntos de entrenamiento y prueba
from sklearn.preprocessing import OneHotEncoder # Codificación one-hot para variables categóricas
from sklearn.feature_selection import SelectKBest # Selección de características basada en la variancia
from sklearn.preprocessing import StandardScaler # Estandarización de variables numéricas
from sklearn.feature_selection import SelectFromModel # Selección de características basada en el rendimiento del modelo
from sklearn.svm import SVC # Máquina de vectores de soporte (SVM)
from sklearn.compose import ColumnTransformer # Transformador de columnas para el preprocesamiento
from sklearn.preprocessing import QuantileTransformer # Transformación cuantílica
from sklearn.preprocessing import MinMaxScaler # Escalado de variables al rango [0, 1]
from sklearn.preprocessing import Normalizer # Normalización de variables
from sklearn.pipeline import Pipeline # Construcción de pipelines de procesamiento
from sklearn.pipeline import make_pipeline # Construcción de pipelines de procesamiento
from sklearn.feature_selection import RFE # Eliminación recursiva de características
from sklearn.neighbors import KNeighborsClassifier # Clasificador k-NN
from sklearn.linear_model import LogisticRegression # Regresión logística
from sklearn.metrics import ConfusionMatrixDisplay # Visualización de matriz de confusión
from sklearn.decomposition import PCA # Análisis de componentes principales (PCA)
from sklearn.model_selection import GridSearchCV # Búsqueda de hiperparámetros con validación cruzada
from sklearn.metrics import precision_score, recall_score, f1_score, make_scorer # Funciones para medir el rendimiento
from sklearn.ensemble import RandomForestClassifier # Clasificador de bosques aleatorios
from sklearn.ensemble import VotingClassifier # Implementación de clasificador por votación
from sklearn.linear_model import SGDClassifier # Implementación del clasificador SGD
from sklearn.preprocessing import FunctionTransformer # aplicar transformaciones personalizadas
# from sklearn.datasets import make_classification

print("librerias cargadas correctamente")

```

librerias cargadas correctamente

0. Revisar Metadata

Se revisan las características del archivo, como el formato, el tipo de datos, los tamaños y las columnas. En la inspección visual del archivo, se detectaron separadores "|", y debido a su tamaño, se cargarán únicamente 5000 filas para revisar una parte del mismo. Esto nos permitirá visualizar rápidamente el número de columnas y los tipos de datos

```
In [6]: df = pd.read_csv("../\\datos_internos\\20210513_Challenge_AA.csv", sep="|", nrows=50)
```

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 21 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   ANIO              5000 non-null    int64  
 1   MES               5000 non-null    int64  
 2   CLIENTE           5000 non-null    int64  
 3   ESTADO             5000 non-null    object  
 4   INGRESO            4918 non-null    float64
 5   MORAS              5000 non-null    int64  
 6   SEXO               5000 non-null    object  
 7   ESTADOCIVIL        5000 non-null    object  
 8   FECHANACIMIENTO   5000 non-null    object  
 9   MARCACIONES         5000 non-null    int64  
 10  CONTACTOS          5000 non-null    int64  
 11  M1                5000 non-null    int64  
 12  C1                5000 non-null    int64  
 13  M2                5000 non-null    int64  
 14  C2                5000 non-null    int64  
 15  M3                5000 non-null    int64  
 16  C3                5000 non-null    int64  
 17  ANTIGUEDAD         5000 non-null    int64  
 18  EDAD               5000 non-null    int64  
 19  HORARIO             5000 non-null    object  
 20  TARGET              5000 non-null    int64  
dtypes: float64(1), int64(15), object(5)
memory usage: 820.4+ KB
```

```
In [3]: df.head()
```

Out[3]:

	ANIO	MES	CLIENTE	ESTADO	INGRESO	MORAS	SEXO	ESTADOCIVIL	FECHANACIMIENTO
0	2019	5	2019539059618	JALISCO	15000.0	3	M	S	C
1	2019	5	2019539059618	JALISCO	15000.0	3	M	S	C
2	2019	5	2019539059618	JALISCO	15000.0	3	M	S	C
3	2019	5	2019539059618	JALISCO	15000.0	3	M	S	C
4	2019	5	2019527392560	VERACRUZ	6000.0	6	M	C	C

5 rows × 21 columns

La visualización de los datos permite hacer varios tipos de clasificación preliminar:

```
categoricas = {'nominales': ['ANIO', 'MES', 'CLIENTE', 'ESTADO', 'INGRESO', 'MORAS', 'SEXO', 'ESTADOCIVIL', 'FECHANACIMIENTO', 'MARCACIONES', 'CONTACTOS3', 'ANTIGUEDAD', 'EDAD', 'TARGET'], 'ordinales': [], 'Desconocido': ['M1', 'C1', 'M2', 'C2', 'M3', 'C3']}
```

```
numerica = {'continuas': ['INGRESO'], 'discretas': ['ANIO', 'MES', 'MORAS', 'MARCACIONES', 'CONTACTOS', 'EDAD', 'ANTIGUEDAD'], }
```

```
cualidad = {'Cuantitativa Intervalo': ['ANIO', 'MES', 'INGRESO', 'MORAS', 'MARCACIONES', 'CONTACTOS', 'EDAD'], 'Cuantitativa Razon': ['ANTIGUEDAD'], 'Cualitativa Nominal': ['CLIENTE', 'ESTADO', 'SEXO', 'ESTADOCIVIL', 'FECHANACIMIENTO', 'HORARIO', 'TARGET'], 'Desconocido': ['M1', 'C1', 'M2', 'C2', 'M3', 'C3']}
```

La clasificación de las variables en categóricas y numéricas es importante en el modelado de machine learning porque los algoritmos y técnicas utilizados pueden variar según el tipo de variable. Por ejemplo, para las variables categóricas, se pueden aplicar técnicas como la codificación one-hot o la codificación ordinal para convertirlas en variables numéricas antes de utilizar ciertos algoritmos de machine learning. Por otro lado, las variables numéricas pueden requerir normalización o escalado antes de aplicar ciertos algoritmos para asegurar que todas las variables tengan el mismo rango o importancia relativa en el modelo.

Se asume que las variables desconocidas se comportan como categóricas ordinales, Cuantitativas de Intervalo.

In [9]: `print(f"Los horarios para clasificación son {list(df['HORARIO'].unique())}")`

Los horarios para clasificación son ['MEDIODIA', 'TARDE', 'NOCHE', 'MANANA']

1. Cargar los datos del archivo .csv

Se realizará la carga de los datos del archivo .csv en el entorno de trabajo. Para manejar eficientemente grandes volúmenes de datos, los datos del archivo .csv se cargan por partes utilizando el iterador chunksize, lo que permite procesarlos de manera incremental.

2. Seleccionar horario, aplicar filtro y trabajar con un dataframe más pequeño

Se construye la clase CargadorCSV que se utiliza para cargar archivos con formato .csv y realizar ciertas operaciones sobre ellos.

El constructor de la clase recibe la ruta del archivo a cargar.

La función "visualizar_avance" muestra el progreso de la carga del archivo en porcentaje.

La función "crear_csv_por_horarios" divide el archivo original en varios archivos CSV separados según ciertos horarios. Para cada chunk del archivo original, se filtran los datos por horario y se guardan en archivos CSV individuales. También se guarda un archivo adicional para los valores perdidos.

Este enfoque facilita el análisis específico para cada horario en particular y permite trabajar con los datos de manera más eficiente.

```
In [ ]: class CargadorCSV:
    """Clase para carga de archivos con formato .csv"""
    def __init__(self, file_path):
        """
        Constructor de la clase. Recibe la ruta del archivo.
        """
        self.file_path = file_path

    def visualizar_avance(self, bytes_avance, bytes_completo):
        """Visualiza el avance de la carga del archivo en porcentaje."""
        avance = min(round(bytes_avance / bytes_completo, 2) * 100, 100)
        clear_output(wait=True)
        print(f"Avance estimado: {avance}%")

    def crear_csv_por_horarios(self, chunk_size=100000, sep="|"):
        """Crea archivos CSV separados por horarios a partir del archivo original."""
        try:
            bytes_completo = os.path.getsize(self.file_path)
            encabezado = True
            horarios = ['MEDIODIA', 'TARDE', 'NOCHE', 'MANANA']

            for chunk in pd.read_csv(self.file_path, sep=sep, chunksize=chunk_size):
                bytes_avance = 0

                if encabezado:
                    for clasificador_horario in horarios:
                        df_filtered = chunk.loc[chunk["HORARIO"] == clasificador_horario]
                        file_name = f"{self.file_path[:-4]}_{clasificador_horario}.csv"
                        df_filtered.to_csv(file_name, index=False)
                else:
                    for clasificador_horario in horarios:
                        df_filtered = chunk.loc[chunk["HORARIO"] == clasificador_horario]
                        file_name = f"{self.file_path[:-4]}_{clasificador_horario}.csv"
                        df_filtered.to_csv(file_name, index=False)

                bytes_avance += len(chunk)

        except Exception as e:
            print(f"Error al procesar el archivo: {e}")

    def guardar_valores_perdidos(self):
        """Guarda los valores perdidos en un archivo CSV separado."""
        pass
```

```

        df_filtered.to_csv(file_name, index=False)
        bytes_avance += os.path.getsize(file_name)
    df_Nofiltered = chunk.loc[~chunk["HORARIO"].isin(horarios)]
    df_Nofiltered.to_csv(f"{self.file_path[:-4]}_ValoresPerdidos.csv", header=False)
    encabezado = False
else:
    for clasificador_horario in horarios:
        df_filtered = chunk.loc[chunk["HORARIO"] == clasificador_horario]
        file_name = f"{self.file_path[:-4]}_{clasificador_horario}.csv"
        df_filtered.to_csv(file_name, mode='a', index=False, header=encabezado)
        bytes_avance += os.path.getsize(file_name)
    df_Nofiltered = chunk.loc[~chunk["HORARIO"].isin(horarios)]
    df_Nofiltered.to_csv(f"{self.file_path[:-4]}_ValoresPerdidos.csv", header=False)

    self.visualizar_avance(bytes_avance, bytes_completo)
#clear_output(wait=True)
print(f"Proceso terminado")
except Exception as error:
    print("Ocurrió un error:", error)

```

In []: # Leer el archivo .csv original y guardar el archivo por partes según Los horarios
cargador = CargadorCSV("../datos_internos\\20210513_Challenge_AA.csv")
cargador.crear_csv_por_horarios()

De existir valores que no entren en la clasificación de horarios se almacenan en el archivo 20210513_Challenge_AA_ValoresPerdidos.csv, al revisarlo se puede confirmar que no existen valores perdidos.

In [11]: df_perdidos = pd.read_csv("../datos_internos\\20210513_Challenge_AA_ValoresPerdidos.csv")
df_perdidos.shape

Out[11]: (0, 21)

3. Indicar las dimensiones del nuevo archivo

Cargamos la data filtrada para analizar el número de observaciones, valores nulos, inconsistencias en tipo de dato y outliers, Indicando el número de filas y columnas en el nuevo archivo, así como los nombres de las variables y una descripción general de los datos.

La clase revisor_data_csv se utiliza para cargar y revisar archivos con formato .csv. Permite realizar diferentes operaciones y análisis en función de los datos cargados.

El constructor de la clase recibe la ruta del archivo y carga el archivo en un DataFrame utilizando la biblioteca pandas. También crea otro DataFrame para registrar los cambios realizados en los datos.

La función "mensaje" se utiliza para mostrar mensajes durante la carga del archivo, como "Cargando archivo..." y "Carga completa".

La función "analisis_nulos" realiza un análisis de los valores nulos en el DataFrame cargado y devuelve un nuevo DataFrame con la cantidad y el porcentaje de valores nulos para cada columna.

Las funciones "nulos_EliminacionSimple" y "nulos_MediaCondicional" se utilizan para manejar los valores nulos en el DataFrame. La primera elimina las filas que contienen valores nulos en las columnas indicadas por una lista. La segunda asigna el valor promedio de la columna variable a los valores nulos, agrupando por las columnas indicadas en otra lista.

La función "eliminar_por_condicion" elimina las filas que contienen un valor específico en una columna determinada.

```
In [5]: class revisor_data_csv:
    """Clase para carga y revisión de archivos con formato .csv
    Con esto podemos visualizar y manipular la data de forma controlada minimizando
    pudiendo posteriormente crear un pipeline del proceso ETL"""
    def __init__(self, file_path):
        """
        Constructor de la clase que recibe la ruta del archivo y carga el archivo
        Se asume archivos .csv estándar, sep = ","
        """
        self.file_path = file_path
        try:
            # Se carga el df
            self.mensaje("cargando", self.file_path)
            self.df = pd.read_csv(self.file_path)
            self.mensaje("cargado")
        except Exception as error:
            print("Ocurrió un error:", error)

    def mensaje(self, tipo, arg=None):
        tipos_mensajes = {"cargando": f"Cargando archivo {arg}",
                           "cargado": "Carga completa"}
        if tipo in tipos_mensajes:
            clear_output(wait=True)
            print(tipos_mensajes[tipo])
        else:
            clear_output(wait=True)
            print("Tipo de mensaje no válido")
            print(f"{list(tipos_mensajes.keys())}")

    def analisis_nulos(self):
        cuenta_nulos = self.df.isnull().sum()
        cuenta_nulos = cuenta_nulos[cuenta_nulos != 0]
        porcentaje_nulos = round((cuenta_nulos / self.df.shape[0]) * 100, 2)
        return pd.DataFrame({"cant_nulos": cuenta_nulos, "porcentaje_nulos": porcen

    def nulos_EliminacionSimple(self, lista):
        """Se eliminarán las filas que contengan valores nulos en las columnas indicadas"""
        try:
            self.df = self.df.dropna(subset=lista)
        except Exception as error:
            print("Ocurrió un error:", error)
```

```

def nulos_MediaCondicional(self, variable, lista):
    """Se asignará a los valores nulos de la columna variable,
    el valor promedio agrupando con las columnas indicadas por la lista"""
    try:
        self.df.loc[self.df[variable].isnull(), variable] = self.df.groupby(list)
    except Exception as error:
        print("Ocurrió un error:", error)

def eliminar_por_condicion(self, columna, condicion):
    """Se eliminarán las filas que contengan valores 'condicion' en la columna"""
    try:
        self.df = self.df.drop(self.df[self.df[columna] == condicion].index)
    except Exception as error:
        print("Ocurrió un error:", error)

```

Seleccionamos un clasificacion temporal para trabajar...

In [9]: `revision = revisor_data_csv("../\\datos_internos/20210513_Challenge_AA_MANANA.csv")`

3.1. Visualizacion dimensión del nuevo fichero y estructura de los datos.

In [10]: `revision.df.info()`

```

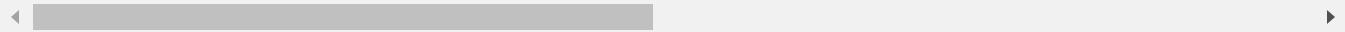
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9270223 entries, 0 to 9270222
Data columns (total 20 columns):
 #   Column            Dtype  
--- 
 0   ANIO              int64  
 1   MES               int64  
 2   CLIENTE           int64  
 3   ESTADO             object 
 4   INGRESO            float64
 5   MORAS              float64
 6   SEXO               object 
 7   ESTADOCIVIL        object 
 8   FECHANACIMIENTO  object 
 9   MARCACIONES        int64  
 10  CONTACTOS          int64  
 11  M1                int64  
 12  C1                int64  
 13  M2                int64  
 14  C2                int64  
 15  M3                int64  
 16  C3                int64  
 17  ANTIGUEDAD         float64
 18  EDAD               float64
 19  TARGET              int64  
dtypes: float64(4), int64(12), object(4)
memory usage: 1.4+ GB

```

In [11]: `revision.df.head()`

Out[11]:

	ANIO	MES	CLIENTE	ESTADO	INGRESO	MORAS	SEXO	ESTADOCIVIL	FECI
0	2019	5	2019539059618	JALISCO	15000.0	3.0	M	S	
1	2019	5	2019527392560	VERACRUZ	6000.0	6.0	M	C	
2	2019	5	201958823716	CHIAPAS	4000.0	1.0	F	S	
3	2019	5	201953294752	TAMAULIPAS	4000.0	5.0	M	U	
4	2019	5	2019568646789	ESTADO DE MEXICO	5000.0	3.0	F	U	



In [12]:

```
# Para revision posterior de total de registros perdidos en proceso ETL
registros_inicial = revision.df.shape[0]
print(f"Inicialmente el data set consta de {registros_inicial} registros")
```

Inicialmente el data set consta de 9270223 registros

3.2. Manejo valores nulos

Se procede a evaluar los valores nulos, porque estos pueden afectar al momento de querer modificar los tipos de dato, la mejor forma de Visualizar la cantidad de valores nulos es porcentual con respecto al tamaño de la base de datos.

In [13]:

```
revision.analisis_nulos()
```

Out[13]:

	cant_nulos	porcentaje_nulos
INGRESO	179892	1.94
MORAS	2815	0.03
SEXO	2815	0.03
ESTADOCIVIL	2815	0.03
FECHANACIMIENTO	2815	0.03
ANTIGUEDAD	2815	0.03
EDAD	2815	0.03

Existen diversas formas de abordar los valores nulos en los datos: Eliminacion simple, imputación por valor constante, por valor medio/mediana, regresión,media condicional o vecinos más cercanos. En este caso, se utilizarán dos métodos:

Para aquellas variables (MORAS, SEXO, ESTADOCIVIL, FECHANACIMIENTO, ANTIGUEDAD, EDAD) que representan el 0.03% de los datos, se procederá a aplicar eliminacion simple.

Dado que este porcentaje es muy pequeño en relación al análisis general, su exclusión no debería tener un impacto significativo en los resultados.

En el caso de la variable 'INGRESO', se realizará imputación por media condicional, reemplazar los valores nulos con la media de la columna condicionada a otras características del conjunto de datos. Es importante destacar que esta aproximación asume que el comportamiento de la variable 'INGRESO' sigue una distribución relativamente normal.

```
In [14]: # Eliminacion de valores nulos
revision.nulos_EliminacionSimple(['MORAS', 'SEXO', 'ESTADOCIVIL', 'FECHANACIMIENTO']
revision.analisis_nulos()
```

```
Out[14]:      cant_nulos  porcentaje_nulos
INGRESO        177077          1.91
```

```
In [15]: # Promedio de ajustado para valores nulos
revision.nulos_MediaCondicional('INGRESO',[ 'ESTADO','EDAD','ESTADOCIVIL','SEXO'])
revision.analisis_nulos()
```

```
Out[15]:      cant_nulos  porcentaje_nulos
INGRESO        606            0.01
```

Sin embargo, pueden quedar unos valores nulos que no tengan promedio, pero nuevamente al ser una cantidad muy pequeña, se puede eliminar esos registros.

```
In [16]: revision.nulos_EliminacionSimple(['INGRESO'])
revision.analisis_nulos()
```

```
Out[16]:      cant_nulos  porcentaje_nulos
```

3.3. Incosistencias de Tipo de dato

Una vez resuelto el problema de valores nulos se pueden identificar las siguientes inconsistencias de tipo:

Índice	Variable	Inconsistencia
0	ANIO	No hay inconsistencias. Valores válidos: [2019,2020]
1	MES	No hay inconsistencias. Valores válidos: enteros del 1 al 12
2	CLIENTE	No hay inconsistencias.
3	ESTADO	No hay inconsistencias. 32 estados coinciden con estados mexicanos.
4	INGRESO	No hay inconsistencias.

Índice	Variable	Inconsistencia
5	MORAS	Se entiende que está en unidades enteras, no puede haber moras fraccionarias. Debe ser de tipo entero (int).
6	SEXO	No hay inconsistencias.
7	ESTADOCIVIL	No hay inconsistencias.
8	FECHANACIMIENTO	No está reconocida como un datetime.
9	MARCACIONES	No hay inconsistencias.
10	CONTACTOS	No hay inconsistencias.
11	M1	No hay inconsistencias.
12	C1	No hay inconsistencias.
13	M2	No hay inconsistencias.
14	C2	No hay inconsistencias.
15	M3	No hay inconsistencias.
16	C3	No hay inconsistencias.
17	ANTIGUEDAD	Se asumirá que debe ser de tipo entero (int).
18	EDAD	Debe ser de tipo entero (int).
19	TARGET	No hay inconsistencias.

En resumen, se identificaron las inconsistencias de tipo en las variables y se propusieron las correcciones correspondientes en cuanto al tipo de dato esperado.

In [17]:

```
# correcciones de tipo en datos:
revision.df['MORAS'] = revision.df['MORAS'].astype(int)
revision.df['ANTIGUEDAD'] = revision.df['ANTIGUEDAD'].astype(int)
revision.df['EDAD'] = revision.df['EDAD'].astype(int)
revision.df['FECHANACIMIENTO'] = pd.to_datetime(revision.df['FECHANACIMIENTO']).dt.
revision.df['FECHANACIMIENTO'] = pd.to_datetime(revision.df['FECHANACIMIENTO'])
```

In [18]:

```
revision.df.dtypes
```

```
Out[18]: ANIO           int64
          MES            int64
          CLIENTE         int64
          ESTADO          object
          INGRESO         float64
          MORAS           int32
          SEXO            object
          ESTADOCIVIL     object
          FECHANACIMIENTO datetime64[ns]
          MARCACIONES      int64
          CONTACTOS        int64
          M1               int64
          C1               int64
          M2               int64
          C2               int64
          M3               int64
          C3               int64
          ANTIGUEDAD       int32
          EDAD             int32
          TARGET            int64
          dtype: object
```

```
In [19]: revision.df.head()
```

	ANIO	MES	CLIENTE	ESTADO	INGRESO	MORAS	SEXO	ESTADOCIVIL	FECI
0	2019	5	2019539059618	JALISCO	15000.0	3	M	S	
1	2019	5	2019527392560	VERACRUZ	6000.0	6	M	C	
2	2019	5	201958823716	CHIAPAS	4000.0	1	F	S	
3	2019	5	201953294752	TAMAULIPAS	4000.0	5	M	U	
4	2019	5	2019568646789	ESTADO DE MEXICO	5000.0	3	F	U	

4. Número de observaciones y valores perdidos

Para la revisión final de valores perdidos en proceso inicial de ETL se calcula el número de observaciones y valores faltantes (NA) en los datos.

```
In [20]: # Se realiza una verificación adicional de algunas columnas.
for variable in ['ANIO', 'MES', 'ESTADO', 'SEXO', 'ESTADOCIVIL']:
    if isinstance(variable, int) or isinstance(variable, float):
        print(f"{variable}: tiene {revision.df[variable].nunique()} valores únicos,
    else:
        print(f"{variable}: tiene {revision.df[variable].nunique()} valores únicos,
```

ANIO: tiene 2 valores únicos, [2019 2020]
 MES: tiene 12 valores únicos, [5 7 3 6 8 9 10 2 11 1 12 4]
 ESTADO: tiene 32 valores únicos, ['JALISCO' 'VERACRUZ' 'CHIAPAS' 'TAMAULIPAS' 'ESTADO DE MEXICO' 'PUEBLA'
 'SINALOA' 'YUCATAN' 'MICHOACAN' 'COLIMA' 'OAXACA' 'NUEVO LEON' 'MORELOS'
 'GUANAJUATO' 'TABASCO' 'CIUDAD DE MEXICO' 'SONORA' 'NAYARIT'
 'QUINTANA ROO' 'GUERRERO' 'AGUASCALIENTES' 'CHIHUAHUA' 'HIDALGO'
 'BAJA CALIFORNIA NORTE' 'COAHUILA' 'SAN LUIS POTOSI' 'ZACATECAS'
 'CAMPECHE' 'QUERETARO' 'BAJA CALIFORNIA SUR' 'DURANGO' 'TLAXCALA']
 SEXO: tiene 2 valores únicos, ['M' 'F']
 ESTADOCIVIL: tiene 6 valores únicos, ['S' 'C' 'U' 'D' 'V' ' ']

Se detecto que "ESTADOCIVIL" tiene el elemento '' que no estan definido. Dado que son pocos elementos con este problema se procedera a eliminar estas filas.

```
In [21]: print(f"Numero total de filas con 'ESTADOCIVIL' en blanco (' ') : {revision.df[revision.df['ESTADOCIVIL'].str.contains('')].shape[0]}")
```

Numero total de filas con 'ESTADOCIVIL' en blanco (' ') : 188

```
In [22]: revision.eliminar_por_condicion('ESTADOCIVIL', '')
```

```
In [23]: # revision final de valores perdidos en proceso inicial de ETL
print(f"Quedan un total de {revision.df.shape[0]} registros. Se perdieron en el proceso de limpieza {188} registros.")
```

5. Calculo de medidas estadísticas básicas

Se calcula medidas estadísticas básicas como el promedio, los percentiles, las desviaciones estándar, los valores mínimos y máximos, y la mediana.

```
In [2]: revision.df.describe().round(2)
```

	ANIO	MES	INGRESO	MORAS	MARCACIONES	CONTACTOS	
count	9266614.00	9266614.00	9266614.00	9266614.00	9266614.00	9266614.00	9266614.00
mean	2019.26	6.23	9092.36	2.65	52.66	1.50	
std	0.44	3.63	7474.53	2.10	73.77	3.29	
min	2019.00	1.00	0.00	1.00	1.00	0.00	
25%	2019.00	3.00	6000.00	1.00	7.00	0.00	
50%	2019.00	6.00	8000.00	2.00	26.00	0.00	
75%	2020.00	10.00	11000.00	4.00	67.00	2.00	
max	2020.00	12.00	2000013.00	36.00	2342.00	128.00	23

Se puede hacer una descripción mas profunda de cada una de las variables, como por ejemplo:

```
In [58]: lista = ['INGRESO', 'MORAS', 'MARCACIONES', 'CONTACTOS', 'M1', 'C1', 'M2', 'C2', 'M3', 'C3', 'ANTIGUEDAD', 'EDAD']
for columna in lista:
    data_set = revision.df[columna]
    maximo = data_set.max()
    minimo = data_set.min()
    media = data_set.mean()
    moda = data_set.mode().values[0]
    mediana = data_set.median()
    dev_std = data_set.std()
    simetria = skew(data_set)
    curtosis = kurtosis(data_set, fisher = True)
    #print(f'{columna}: simetria = {simetria}, curtosis = {curtosis}, media = {media:.2f}, Desviacion_std = {dev_std:.2f}')
    print(f'{columna}: maximo = {maximo:.2f}, minimo = {minimo:.2f}, media = {media:.2f}, Desviacion_std = {dev_std:.2f}, simetria = {simetria:.2f}, curtosis = {curtosis:.2f}')
    print("")
```

INGRESO: maximo = 2000013.00, minimo = 0.00, media = 9092.36, mediana = 8000.00, moda = 8000.00, Desviacion_std = 7474.53 ,simetria = 23.73, kurtosis = 4099.46

MORAS: maximo = 36.00, minimo = 1.00, media = 2.65, mediana = 2.00, moda = 1.00, Desviacion_std = 2.10 ,simetria = 1.42, kurtosis = 3.00

MARCACIONES: maximo = 2342.00, minimo = 1.00, media = 52.66, mediana = 26.00, moda = 1.00, Desviacion_std = 73.77 ,simetria = 3.34, kurtosis = 20.63

CONTACTOS: maximo = 128.00, minimo = 0.00, media = 1.50, mediana = 0.00, moda = 0.00, Desviacion_std = 3.29 ,simetria = 5.66, kurtosis = 54.17

M1: maximo = 2342.00, minimo = 0.00, media = 37.51, mediana = 6.00, moda = 0.00, Desviacion_std = 70.73 ,simetria = 3.82, kurtosis = 25.34

C1: maximo = 128.00, minimo = 0.00, media = 1.01, mediana = 0.00, moda = 0.00, Desviacion_std = 2.93 ,simetria = 6.87, kurtosis = 77.37

M2: maximo = 2342.00, minimo = 0.00, media = 27.30, mediana = 1.00, moda = 0.00, Desviacion_std = 62.46 ,simetria = 4.66, kurtosis = 37.04

C2: maximo = 125.00, minimo = 0.00, media = 0.79, mediana = 0.00, moda = 0.00, Desviacion_std = 2.56 ,simetria = 7.80, kurtosis = 101.08

M3: maximo = 2342.00, minimo = 0.00, media = 20.30, mediana = 0.00, moda = 0.00, Desviacion_std = 54.48 ,simetria = 5.55, kurtosis = 53.18

C3: maximo = 125.00, minimo = 0.00, media = 0.64, mediana = 0.00, moda = 0.00, Desviacion_std = 2.29 ,simetria = 8.75, kurtosis = 128.96

ANTIGUEDAD: maximo = 13.00, minimo = 0.00, media = 1.47, mediana = 1.00, moda = 0.00, Desviacion_std = 2.06 ,simetria = 2.04, kurtosis = 4.49

EDAD: maximo = 95.00, minimo = 18.00, media = 40.05, mediana = 38.00, moda = 27.00, Desviacion_std = 13.44 ,simetria = 0.49, kurtosis = -0.56

La variable "INGRESO" tiene un amplio rango de valores, desde 0 hasta 2,000,013. La media es 9,092 y la mediana es 8,000, lo que indica una ligera asimetría hacia la derecha. La desviación estándar es de 7,474, lo que indica una alta dispersión de los valores con respecto a la media. La simetría de 23.73 muestra una asimetría positiva pronunciada, con una cola larga hacia la derecha en la distribución. La kurtosis de 4,099 indica una alta concentración de valores en la cola de la distribución, lo que significa que hay colas pesadas. Esta variable puede ser útil para predecir o clasificar resultados basados en los ingresos de los individuos. Sin embargo, la asimetría y la alta kurtosis pueden afectar la capacidad de los modelos de machine learning para capturar patrones y hacer predicciones precisas. Se sugiere aplicar transformaciones para normalizar la distribución y mejorar la capacidad predictiva de los modelos. Algunas transformaciones comunes que se pueden considerar incluyen la transformación logarítmica, la raíz cuadrada y la transformación Box-Cox. Es importante realizar pruebas y evaluaciones para determinar qué transformación produce una distribución más normalizada y mejora la capacidad predictiva de los modelos. Se debe tener en cuenta que algunas transformaciones pueden requerir ajustes especiales si hay valores nulos o ceros en la variable.

La señal "MORAS" presenta un rango de valores entre 1 y 36. La media es de 2.65 y la mediana es de 2.00, indicando una distribución sesgada hacia la derecha. La moda se encuentra en 1.00, lo que sugiere que es el valor más común. La desviación estándar de 2.10 muestra una dispersión moderada alrededor de la media. La simetría de 1.42 indica una ligera asimetría positiva, mientras que la kurtosis de 3.00 sugiere una distribución leptocúrtica con una concentración moderada de valores en comparación con una distribución normal.

La señal "MARCACIONES" tiene un rango amplio de valores, desde 1 hasta 2342. La media es de 52.66 y la mediana es de 26.00, lo que indica una asimetría positiva debido a valores atípicos o extremos hacia la derecha. La moda se encuentra en 1.00, lo que indica que es el valor más frecuente. La desviación estándar de 73.77 muestra una alta dispersión de los datos con respecto a la media. La simetría de 3.34 y la kurtosis de 20.63 indican una distribución muy sesgada y con colas pesadas.

La señal "CONTACTOS" tiene un rango de valores entre 0 y 128. La media es de 1.50 y la mediana es de 0.00, lo que indica una distribución altamente sesgada hacia la derecha. La moda se encuentra en 0.00, lo que sugiere que es el valor más común. La desviación estándar de 3.29 muestra una dispersión moderada de los datos. La simetría de 5.66 y la kurtosis de 54.17 indican una asimetría positiva pronunciada y una distribución con colas extremadamente pesadas.

La señal "M1" tiene un rango de valores entre 0 y 2342. La media es de 37.51 y la mediana es de 6.00, lo que indica una distribución sesgada hacia la derecha debido a valores atípicos o extremos. La moda se encuentra en 0.00, lo que indica que es el valor más común. La desviación estándar de 70.73 muestra una alta dispersión de los datos. La simetría de 3.82 y la kurtosis de 25.34 sugieren una distribución con asimetría positiva y colas pesadas.

La señal "C1" tiene un rango de valores entre 0 y 128. La media es de 1.01 y la mediana es de 0.00, lo que indica una distribución altamente sesgada hacia la derecha. La moda se encuentra en 0.00, lo que sugiere que es el valor más común. La desviación estándar de 2.93 muestra una dispersión moderada de los datos. La simetría de 6.87 y la kurtosis de 77.37 indican una asimetría positiva pronunciada y una distribución con colas extremadamente pesadas.

Se puede hacer un análisis equivalente para el resto de las variables..

6. Elaborar graficos para visualizar estadistica

Para crear gráficos con una estadística general de los datos nos apoyaremos con la librería pandas-profiling, esta es una librería de análisis exploratorio de datos para pandas. Proporciona un informe detallado sobre la estructura y contenido de un DataFrame, incluyendo estadísticas descriptivas, correlaciones, distribuciones, valores perdidos y mucho más. Una ventaja de esta librería es que puede generar el informe en formato HTML para una fácil visualización y análisis. nota: https://ydata-profiling.ydata.ai/docs/master/pages/getting_started/quickstart.html

```
In [7]: profile = ProfileReport(revision.df, title="20210513_Challenge_AA_MANANA Pandas Profiling Report")

In [8]: profile.to_file("20210513_Challenge_AA_MANANA_report.html")
```

Summarize dataset: 0% | 0/5 [00:00<?, ?it/s]
Generate report structure: 0% | 0/1 [00:00<?, ?it/s]
Render HTML: 0% | 0/1 [00:00<?, ?it/s]
Export report to file: 0% | 0/1 [00:00<?, ?it/s]

Sin embargo se puede profundizar un poco más en el análisis, incluso hacerlo más dinámico, se puede elaborar un programa para crear un gráfico con una estadística general o específica para los datos...

```
In [93]: class revisor_grafico:
    """Clase para contener todos las formas de revisión gráfica de un dataframe que
    Con esto podemos crear y agregar diferentes métodos en función de lo que queramos"""
    def __init__(self, df):
        """
        Constructor de la clase que recibe el df
        """
        try:
            # Se carga el df
            self.df = df
        except Exception as error:
            print("Ocurrió un error:", error)

    def histograma_general(self, columnas=None, filas=None):
        """Grafica los histogramas de las variables en el df:
            - si no se indica ningún parámetro grafica todas las columnas y filas po
```

```

        - Si se introduce una lista con las columnas a graficar se grafican sol
        - Si se introduce un integer con la cantidad de filas a utilizar se hace
try:
    if columnas:
        # Se confirma que las columnas existan, se obvian columnas que no e
        columnas = [ col for col in columnas if col in self.df.columns]
    else:
        columnas = df.columns
    if filas:
        self.df[columnas].sample(filas).hist(figsize=(10, 10))
    else:
        self.df[columnas].hist(figsize=(10, 10))
    plt.show()
except Exception as error:
    print("Ocurrió un error:", error)

def boxplot_general(self,columnas=None,filas=None):
    """Grafica Box plots de las variables en el df:
    -si no se indica ningun parametro grafica todas las columnas y filas po
    - Si se introduce una lista con las columnas a graficar se grafican sol
    - Si se introduce un integer con la cantidad de filas a utilizar se hace
try:
    if columnas:
        # Se confirma que las columnas existan, se obvian columnas que no e
        columnas = [ col for col in columnas if col in self.df.columns]
    else:
        columnas = df.columns
    if filas:
        self.df[columnas].sample(filas).plot(kind='box', subplots=True, lay
    else:
        self.df[columnas].plot(kind='box', subplots=True, layout=(3, 4), sh
    plt.show()
except Exception as error:
    print("Ocurrió un error:", error)

def histograma_frecuencia(self, columna, n):
    """Crea un histograma de frecuencia para una columna del dataframe utilizan
try:
    dataset = self.df.sample(n)
    if dataset[columna].dtype == 'object':
        conteo = dataset[columna].value_counts().reset_index()
        conteo.columns = [columna, 'conteo']
        conteo['conteo_%'] = round(conteo['conteo'] / sum(conteo['conteo']), 2)
        plt.bar(columna, conteo['conteo_%'])
        plt.xlabel(columna)
        plt.ylabel('Frecuencia %')
        plt.title(f'Frecuencia de variable {columna} para {n} registros')
    else:
        data_set = dataset[columna]
        plt.figure(figsize=(10, 6))
        sns.histplot(data=data_set, kde=True, bins=20, alpha=0.7)
        media = data_set.mean()
        moda = data_set.mode().values[0]
        mediana = data_set.median()
        plt.axvline(media, color='red', linestyle='--', label=f'Media: {med

```

```

        plt.axvline(modal, color='green', linestyle='--', label=f'Moda: {modal}')
        plt.axvline(mediana, color='blue', linestyle='--', label=f'Mediana: {mediana}')
        plt.legend(loc='upper right')
        plt.title(f'Histograma de {columna} para {n} registros')
        plt.xlabel(f'Valores para {columna}')
        plt.ylabel(f'Frecuencias')
    plt.show()
except Exception as error:
    print("Ocurrió un error:", error)

def histograma_dinamico(self, columna, max_registros):
    """Crea histogramas de frecuencia dinámicos para una columna del dataframe"""
    try:
        cantidades = [int(np.exp(i)) for i in np.linspace(np.log(100), np.log(max_registros), 10)]
        for cant in cantidades:
            clear_output(wait=True)
            self.histograma_frecuencia(columna, cant)
    except Exception as error:
        print("Ocurrió un error:", error)

def variables_numericas(self):
    """Devuelve una lista con las variables numericas de un df"""
    return [column_name for column_name, data_type in zip(self.df.columns, self.df.dtypes) if data_type in [np.int64, np.float64]]


def pairplot(self, columnas=None, filas=None):
    """Generará una matriz de gráficos de dispersión, donde cada gráfico muestra la relación entre dos columnas. La diagonal principal de la matriz mostrará un histograma de cada columna, -si no se indica ningún parámetro grafica todas las columnas y filas posibles. - Si se introduce una lista con las columnas a graficar se grafican solo esas columnas. - Si se introduce un entero con la cantidad de filas a utilizar se hace una matriz cuadrada de esa tamaño.
    """
    try:
        if columnas:
            # Se confirma que las columnas existan y sean numéricas
            columnas = [col for col in columnas if col in self.variables_numericas()]
        else:
            columnas = self.variables_numericas()
        if filas:
            data_set = self.df[columnas].sample(filas)
            data_set.reset_index(inplace=True, drop=True)
        else:
            data_set = self.df[columnas]
        sns.pairplot(data_set)
    except Exception as error:
        print("Ocurrió un error:", error)

def detalle_scatterplot(self, x, y, filas=None, hue=None, style=None, size=None):
    try:
        if filas:
            data_set = self.df.sample(filas)
            data_set.reset_index(inplace=True, drop=True)
        else:
            data_set = self.df
        sns.scatterplot(data=data_set, x=x, y=y, hue=hue, style=style, size=size)
    plt.show()

```

```

        except Exception as error:
            print("Ocurrió un error:", error)

    def detalle_boxplot(self, columna,filas=None, categoria1=None, categoria2=None):
        if filas:
            data_set = self.df.sample(filas)
            data_set.reset_index(inplace=True,drop=True)
        else:
            data_set = self.df

        if categoria1:
            if categoria2:
                sns.boxplot(x=categoria1, y=columna, hue=categoria2, data=data_set,
            else:
                sns.boxplot(x=categoria1, y=columna, data=data_set)
        else:
            sns.boxplot(y=columna, data=data_set, width=0.2, showmeans=True, linewidth=1)
        plt.show()

    def correlacion(self, columnas=None,filas=None):
        try:
            if columnas:
                # Se confirma que las columnas existan y sean numericas
                columnas = [ col for col in columnas if col in self.variables_numericas()]
            else:
                columnas = self.variables_numericas()
            if filas:
                data_set = self.df[columnas].sample(filas)
                data_set.reset_index(inplace=True,drop=True)
            else:
                data_set = self.df[columnas]

            correlacion = data_set.corr(method='pearson', numeric_only=True)
            plt.figure(figsize=(9,9))
            sns.heatmap(correlacion,cmap = 'RdBu',vmin=-1, vmax=1, square=True, annot=True)

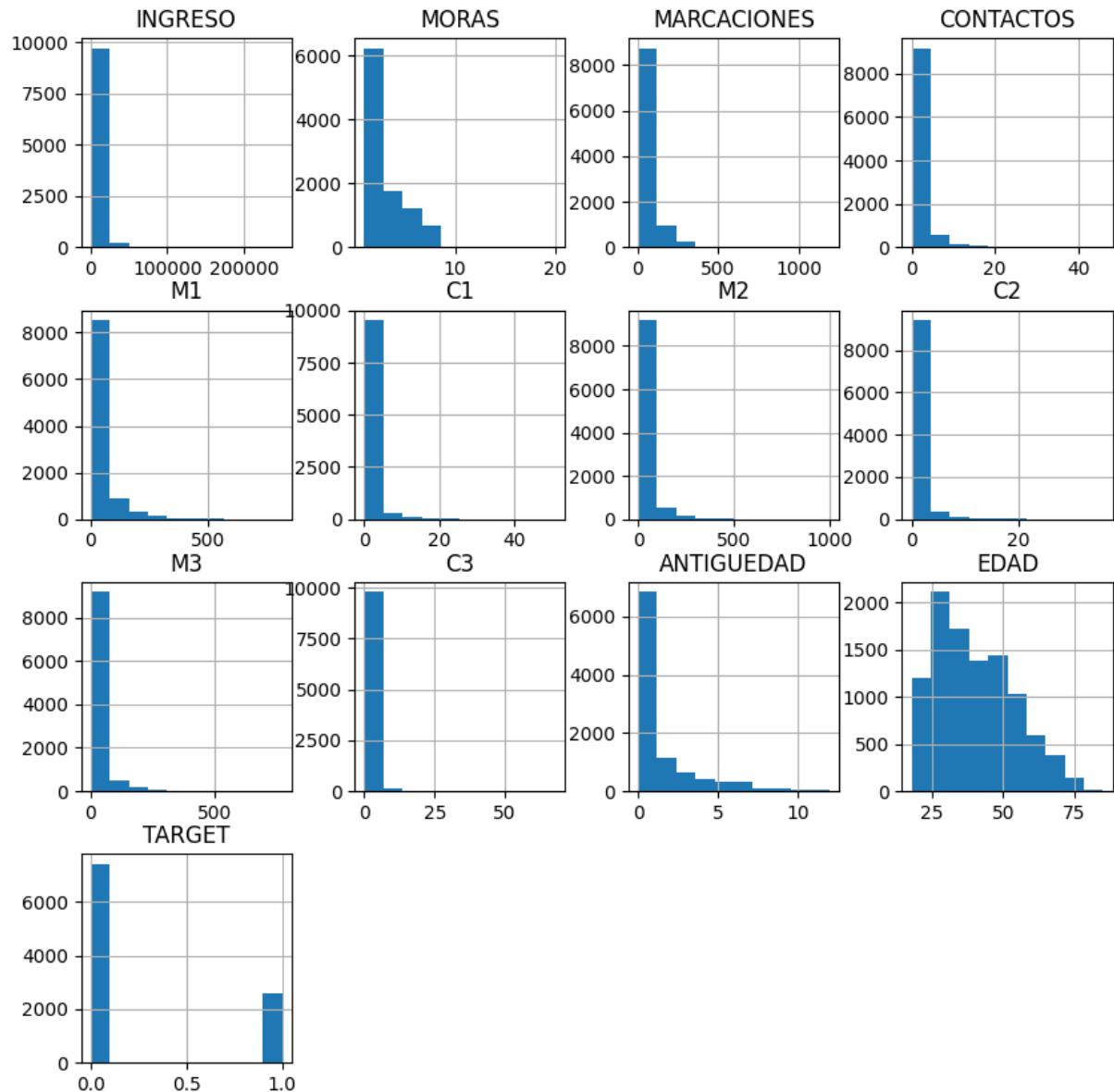
        except Exception as error:
            print("Ocurrió un error:", error)

```

In [94]: `graficas = revisor_grafico(revision.df)`

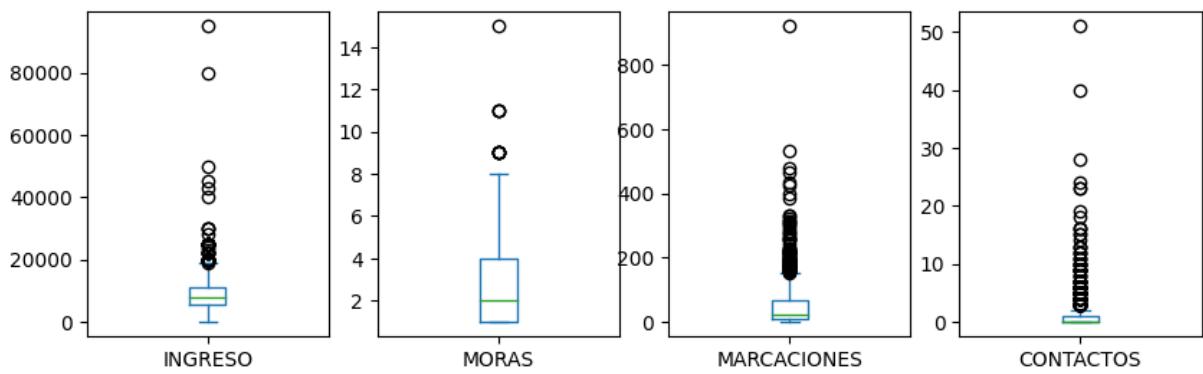
6.1. Histogramas y Barchars generales

In [78]: `graficas.histograma_general(['columna_error','INGRESO', 'MORAS', 'MARCACIONES', 'COMUNICACIONES', 'CREDITOS', 'EDAD', 'TARGET'],100)`



6.1. Boxplot generales

```
In [52]: graficas.boxplot_general(['INGRESO', 'MORAS', 'MARCACIONES', 'CONTACTOS'], 1000)
```

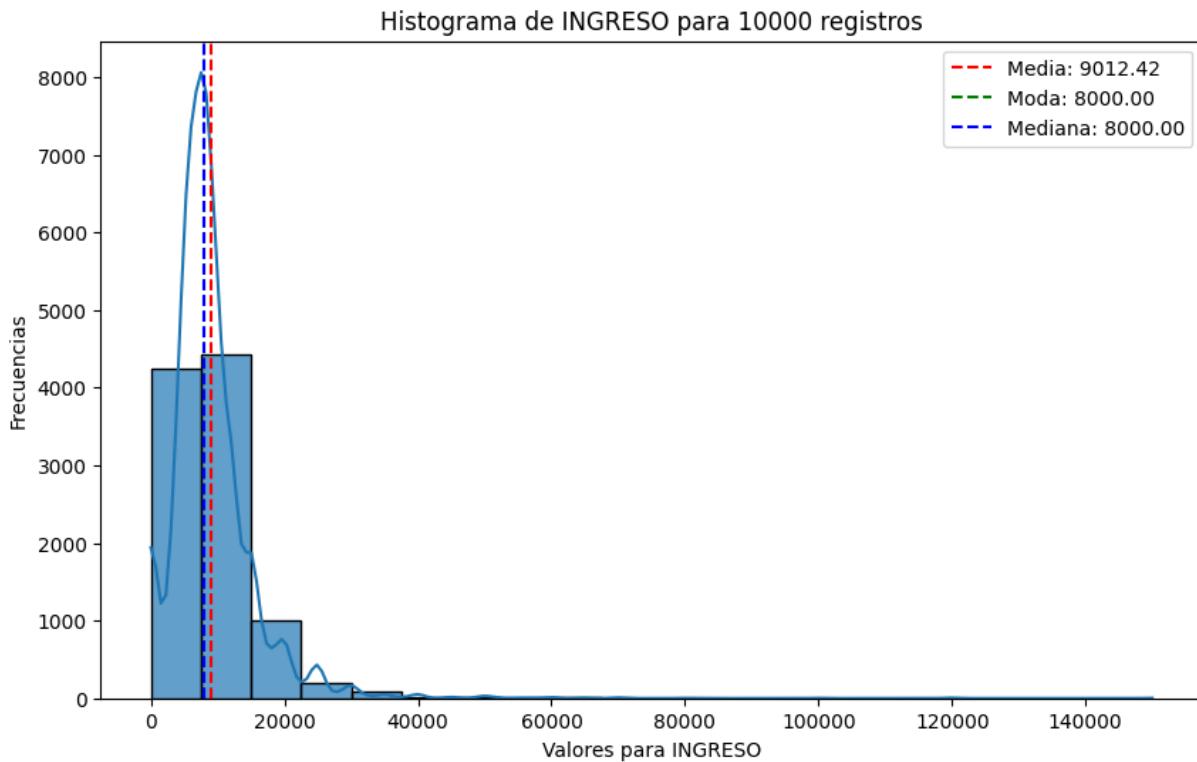


7. Crear un gráfico de barras específicos

Se puede crear un gráfico de barras que muestre la frecuencia de las categorías (discretas) y otro gráfico de barras que muestre la frecuencia de las variables continuas, histogramas mas detallados para obtener la frecuencia de las categorías de los datos. Tanto numericas como no numericas**

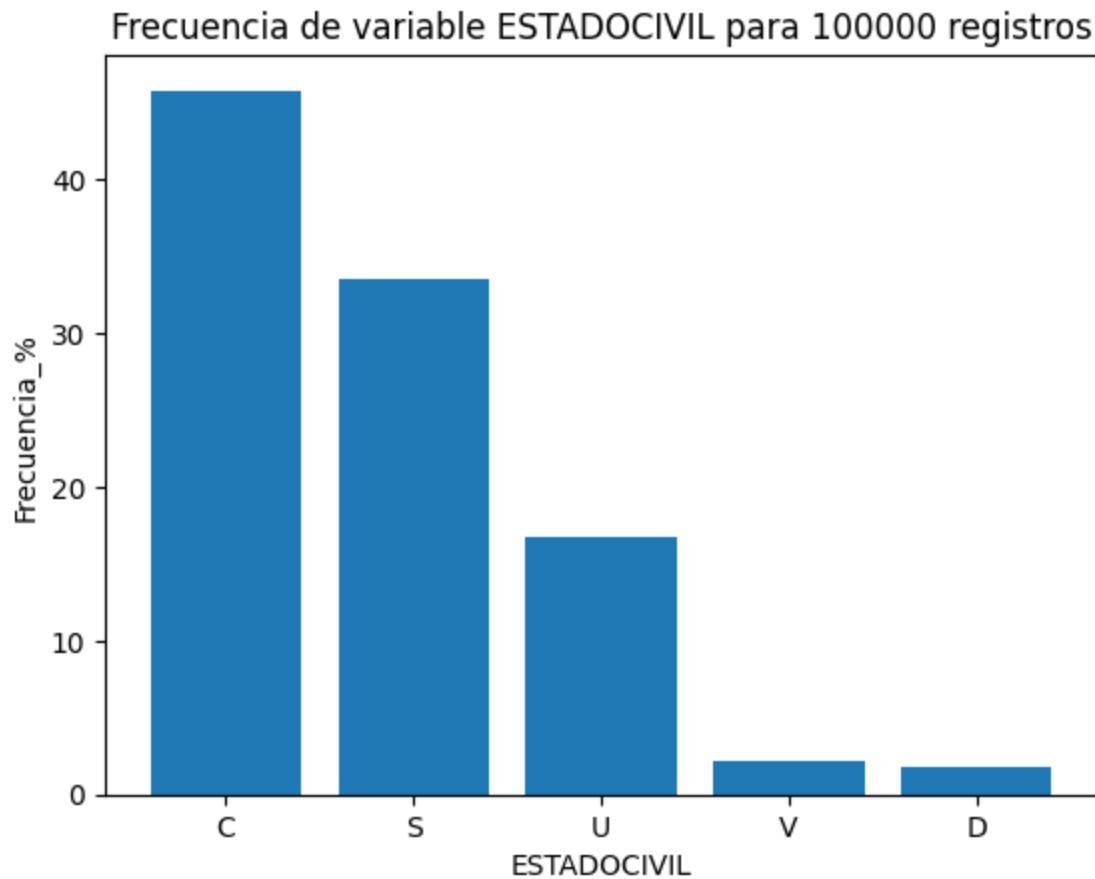
In [81]:

```
n = 10000  
graficas.histograma_frecuencia('INGRESO', n)  
graficas.histograma_dinamico('INGRESO', n)
```



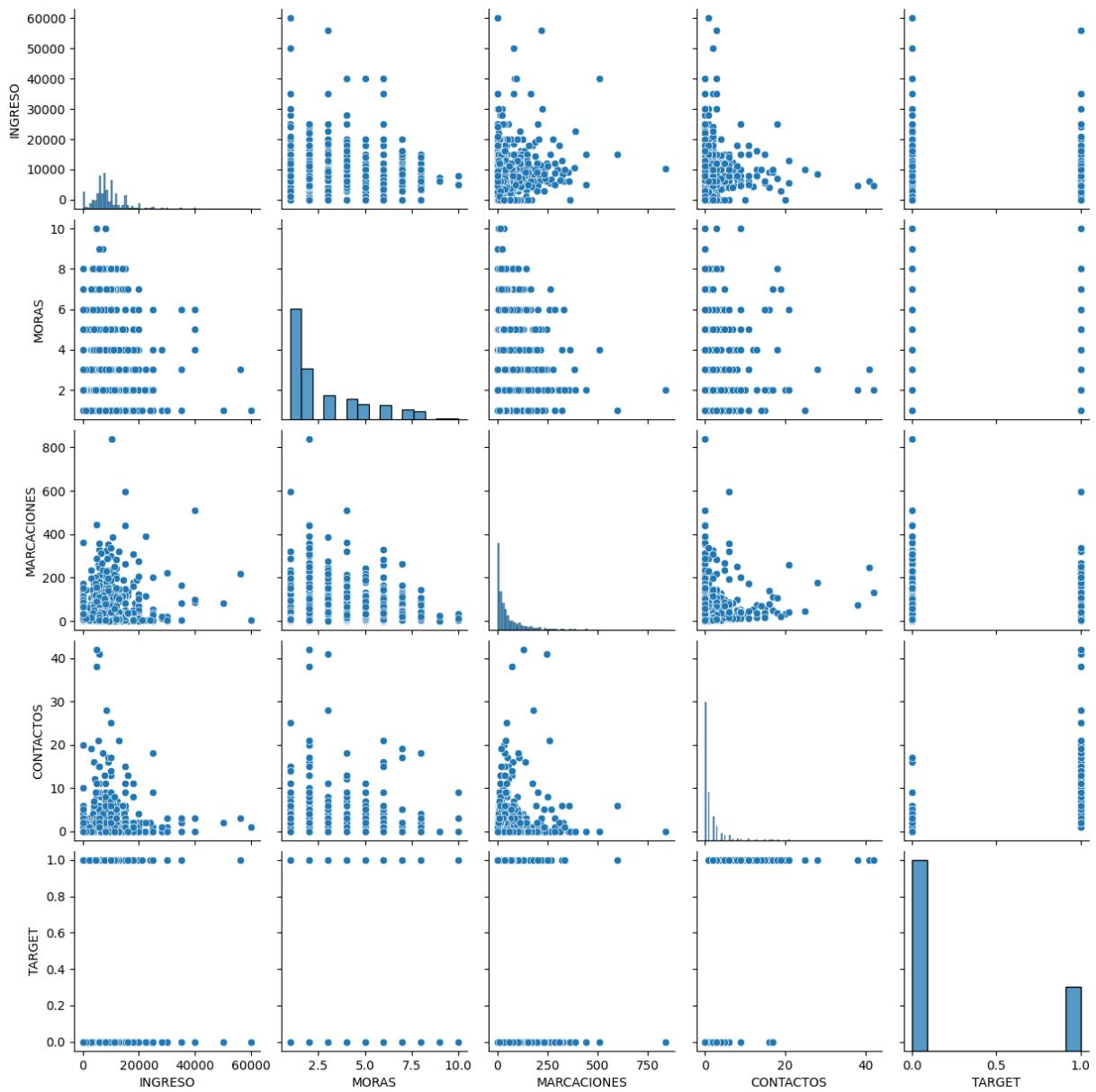
In [82]:

```
n = 100000  
graficas.histograma_frecuencia('MORAS', n)  
graficas.histograma_dinamico('ESTADOCIVIL', n)
```

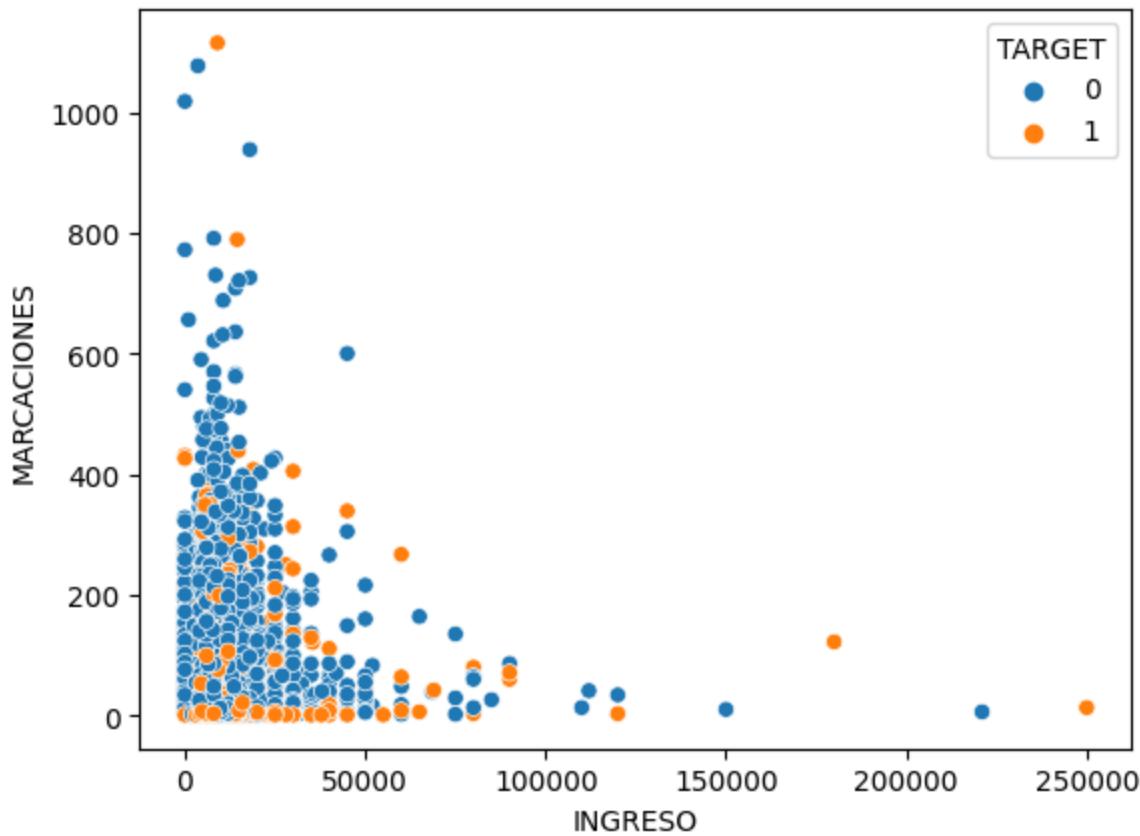


7.1 Otras graficas

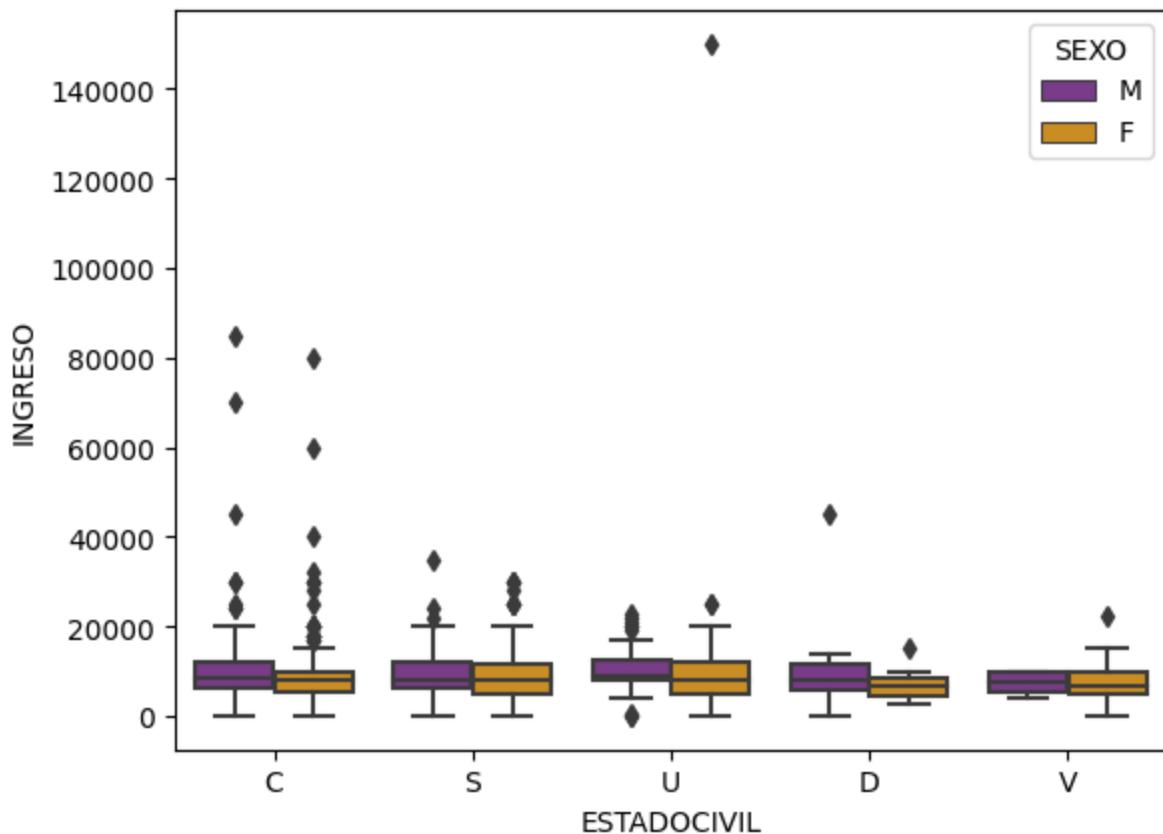
```
In [86]: graficas.pairplot(['INGRESO', 'ESTADOCIVIL','MORAS', 'MARCACIONES', 'CONTACTOS', 'T
```



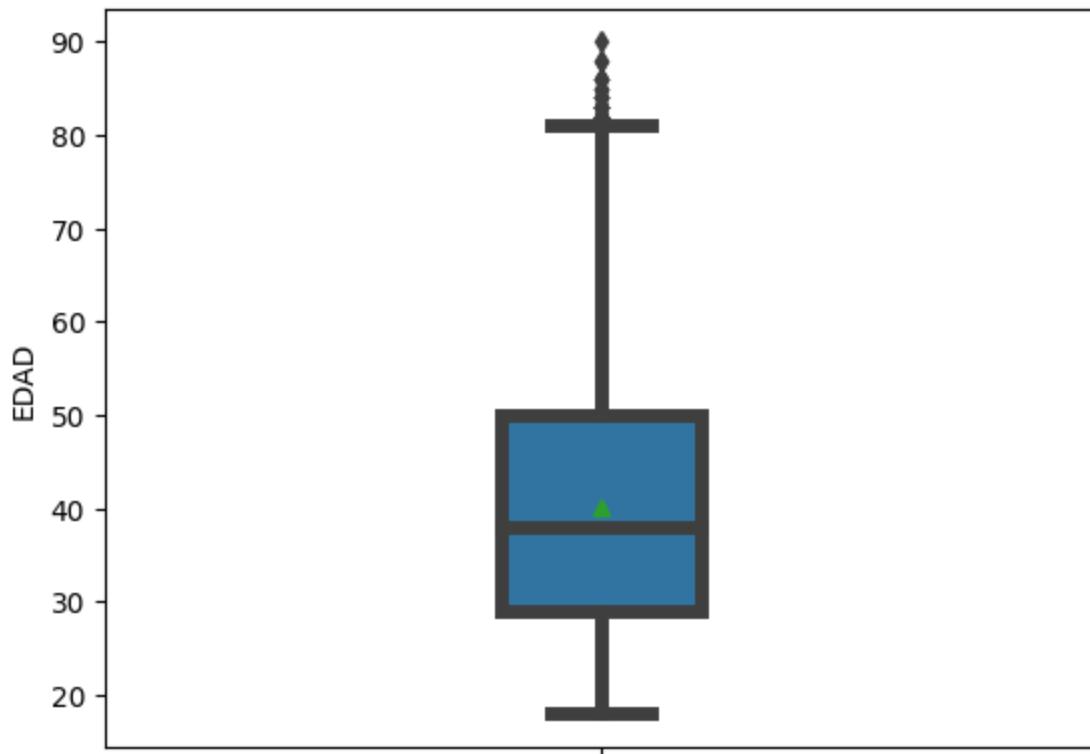
```
In [111...]: graficas.detalle_scatterplot('INGRESO', 'MARCACIONES', 10000, 'TARGET')
```



```
In [120...]: graficas.detalle_boxplot('INGRESO',1000, 'ESTADOCIVIL ', 'SEXO' )
```



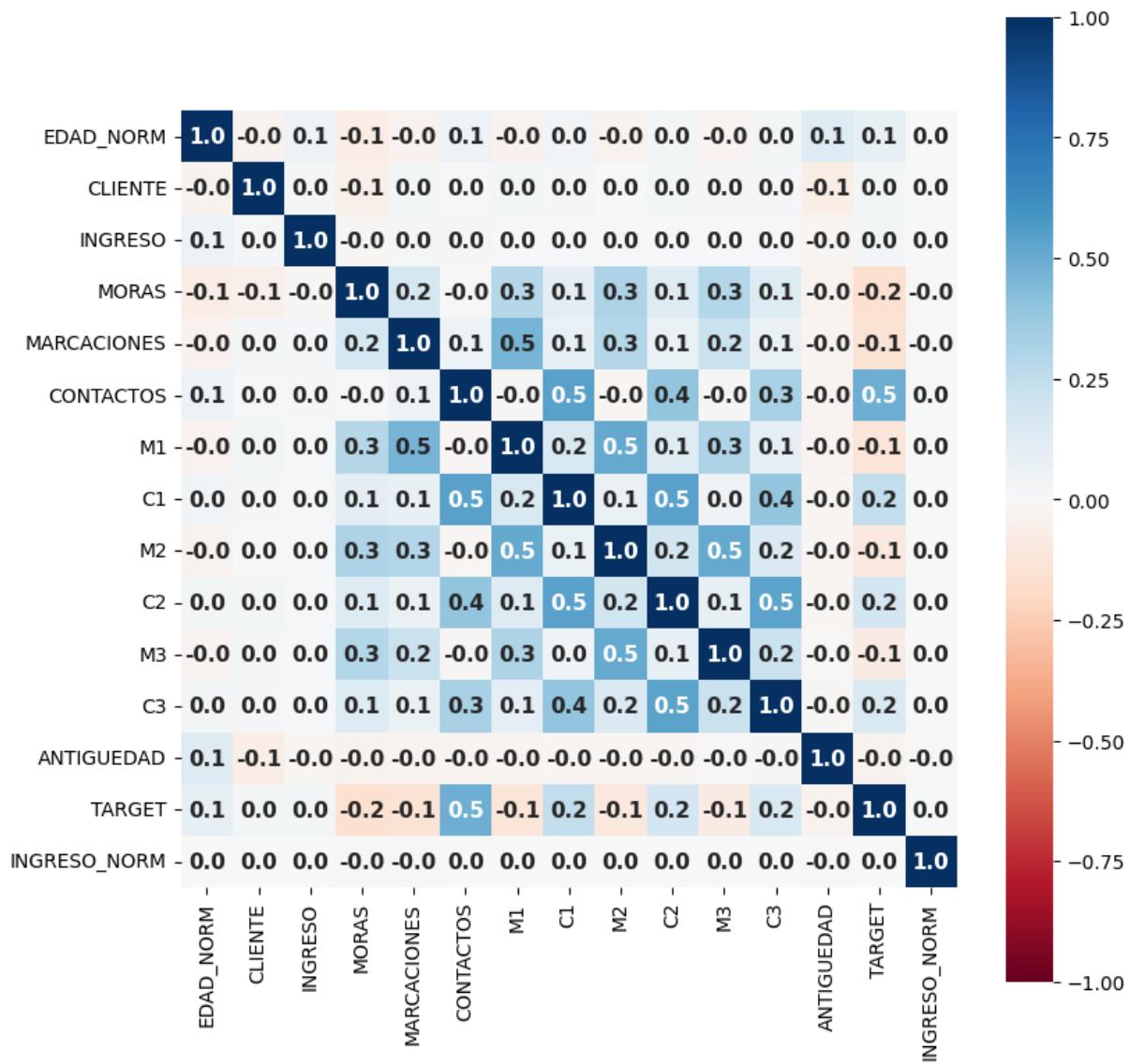
```
In [122...]: graficas.detalle_boxplot('EDAD',100000)
```



8. Analizar las correlaciones entre las variables

Se analiza las correlaciones entre las variables para identificar aquellas que sean linealmente dependientes.

```
In [31]: graficas.correlacion(list(revision.df.columns))
```



8.1 Criterio para interpretar la fuerza de la correlación

Dado que se está utilizando el coeficiente de correlación de Pearson (r), se puede utilizar el siguiente criterio para clasificar:

- Fuerte correlación positiva: $r > 0.7$
- Moderada correlación positiva: $0.3 < r < 0.7$
- Débil correlación positiva: $0.1 < r < 0.3$
- No correlación: $-0.1 < r < 0.1$
- Débil correlación negativa: $-0.3 < r < -0.1$
- Moderada correlación negativa: $-0.7 < r < -0.3$
- Fuerte correlación negativa: $r < -0.7$

```
In [11]: def clasificacion_correlacion(matriz):
    signals = []
    for i in range(len(matriz)):
        for j in range(i+1, len(matriz)):
            signal_1 = matriz.columns[i]
```

```

        signal_2 = matriz.columns[j]
        correlation = matriz.iloc[i, j]
        classification = ""

        if correlation > 0.7:
            classification = "Fuerte correlación positiva"
        elif 0.3 < correlation <= 0.7:
            classification = "Moderada correlación positiva"
        elif 0.1 < correlation <= 0.3:
            classification = "Débil correlación positiva"
        elif -0.1 <= correlation <= 0.1:
            classification = "No correlación"
        elif -0.3 <= correlation < -0.1:
            classification = "Débil correlación negativa"
        elif -0.7 <= correlation < -0.3:
            classification = "Moderada correlación negativa"
        elif correlation < -0.7:
            classification = "Fuerte correlación negativa"

        signals.append([signal_1, signal_2, correlation, classification])

    df = pd.DataFrame(signals, columns=['sig_1', 'sig_2', 'nivel_correlacion', 'clasificacion'])
    return df

correlacion = revision.df.corr(method='pearson', numeric_only=True)
df_correlacion_clasificada = clasificacion_correlacion(correlacion)

print("Variables con Moderada correlación o mayor")
df_correlacion_clasificada[abs(df_correlacion_clasificada['nivel_correlacion']) >=

```

VARIABLES CON MODERADA CORRELACIÓN O MAYOR

Out[11]:

	sig_1	sig_2	nivel_correlacion	clasificacion
15	MES	CLIENTE	0.708841	Fuerte correlación positiva
93	C1	C2	0.546740	Moderada correlación positiva
76	CONTACTOS	C1	0.541238	Moderada correlación positiva
106	C2	C3	0.531372	Moderada correlación positiva
85	M1	M2	0.513041	Moderada correlación positiva
100	M2	M3	0.510563	Moderada correlación positiva
83	CONTACTOS	TARGET	0.486836	Moderada correlación positiva
66	MARCACIONES	M1	0.464187	Moderada correlación positiva
78	CONTACTOS	C2	0.394086	Moderada correlación positiva
95	C1	C3	0.393286	Moderada correlación positiva
80	CONTACTOS	C3	0.326370	Moderada correlación positiva
87	M1	M3	0.312086	Moderada correlación positiva
58	MORAS	M2	0.309503	Moderada correlación positiva

Conocer las correlaciones entre las características (features) en un problema de clasificación utilizando machine learning es importante por varias razones:

Reducción de dimensiones: Identificar características altamente correlacionadas puede ayudar a eliminar redundancias en los datos. Si dos características están altamente correlacionadas entre sí, significa que aportan información similar al modelo. En lugar de utilizar ambas características, se puede elegir una de ellas para reducir la dimensionalidad del conjunto de datos y evitar problemas de multicolinealidad.

Selección de características: La correlación entre las características y la variable objetivo (target) proporciona información sobre la relevancia de cada característica para el problema de clasificación. Características altamente correlacionadas con el target suelen tener un mayor impacto en el modelo y pueden ser consideradas como buenos predictores. Esto puede ayudar en la selección de características, ya que se pueden priorizar aquellas con una correlación más fuerte.

Interpretación de resultados: Comprender las correlaciones entre características puede brindar información sobre las relaciones internas dentro de los datos. Esto puede ayudar a interpretar los resultados del modelo y a comprender qué características son más influyentes en las predicciones. s finales.

II. Ingeniería de atributos/características

La ingeniería de características implica analizar los datos existentes y extraer información relevante que pueda ayudar a los algoritmos de aprendizaje automático a comprender mejor el problema y tomar decisiones más precisas. Esto puede incluir la creación de nuevas características a partir de las existentes, la normalización o escala de características, la codificación de variables categóricas, el manejo de valores faltantes, la reducción de dimensionalidad, entre otras técnicas.

9. Criterio para determinar los factores influyentes con respecto al objetivo

El proceso de determinar los factores influyentes con respecto al objetivo se conoce como "Selección de características" (Feature Selection) y se puede realizar de dos formas principalmente:

1. Utilizando métodos de filtro: En este enfoque, las características se clasifican según puntajes estadísticos que determinan su relación con la variable objetivo. Un método común es utilizar la correlación, que proporciona una primera aproximación. Sin embargo, también se puede utilizar el Valor de Información (IV) y el método de Chi-cuadrado para obtener diferentes perspectivas sobre las características.

2. Utilizando métodos de envoltura: En este enfoque, se emplean algoritmos de Machine Learning y se evalúa su rendimiento como criterio de selección de características. El objetivo es encontrar las características más adecuadas para el algoritmo, mejorando así su rendimiento. A medida que se iteran los modelos, se van tomando decisiones sobre la adición o eliminación de características del subconjunto. Este tipo de métodos suele ser computacionalmente exigente y se recomienda aplicarlo en muestras de datos.

Durante este proceso de verificación, también se realiza la preparación de los datos o preprocesamiento (feature engineering) para adaptar los datos a los modelos de machine learning que se utilizarán. Una de las variables que claramente no aporta información con respecto al objetivo es la columna "CLIENTE", por lo que se puede descartar esta columna en cualquier análisis.

```
In [180...]: revision.df = revision.df.drop(['CLIENTE'], axis=1)
```

Para poder utilizar la correlación y otros métodos de clasificación, es recomendable realizar la estandarización y normalización de los datos. Esto es un requisito común para muchos estimadores de Machine Learning, ya que los algoritmos pueden comportarse de manera deficiente si las características individuales no se parecen más o menos a datos estándar normalmente distribuidos. Para facilitar los cambios necesarios en el DataFrame, se crea la clase `Preprocesamiento`.

A continuación, se detalla la funcionalidad de cada uno de los métodos de la clase:

- `__init__(self, revisor_data)`: Constructor de la clase que recibe un objeto de la clase `RevisorDataCSV` como parámetro. Inicializa las variables `revisor_data`, `df`, `matriz_correlacion` y `transformaciones` con los valores correspondientes.
- `variables_numericas(self)`: Devuelve una lista con las variables numéricas del DataFrame.
- `detection_outlier(self, nombre_columna, q=0.1)`: Detecta los valores atípicos (outliers) de una columna específica utilizando cuantiles. Calcula los cuartiles, el rango intercuartílico y los límites inferior y superior para identificar los outliers. Devuelve una lista con los índices de los valores atípicos.
- `eliminacion_outlier(self, nombre_columna, q=0.1)`: Elimina los valores atípicos (outliers) de una columna específica utilizando cuantiles. Los cambios se aplican directamente al DataFrame.
- `guardar_transformador(self, nombre_columna, transformador)`: Guarda un transformador aplicado a una columna en un diccionario. Si ya existe un transformador para esa columna, se agrega al final de la lista de transformadores.
- `Transf_MinMaxScaler(self, nombre_columna)`: Crea un objeto que transforma los valores de una columna utilizando el escalador `MinMaxScaler` de scikit-learn.

Devuelve el objeto escalador para transformar futuros valores.

- `Transf_Quantile(self, nombre_columna)` : Crea un objeto para transformar los valores de una columna utilizando el transformador `QuantileTransformer` de scikit-learn.
- `Transf_OneHot_binario(self, nombre_columna)` : Crea un objeto para realizar la transformación One-Hot cuando la variable es binaria. Devuelve el transformador ya entrenado.
- `Transf_OneHot(self, nombre_columna)` : Crea un objeto para transformar los valores de una columna utilizando el codificador `OneHotEncoder` de scikit-learn.
- `Transf_woe(self, nombre_columna, nombre_target)` : Crea un objeto para transformar los valores de una columna utilizando el Weight of Evidence (WOE) de OptimalBinning. Este método distingue entre variables numéricas y categóricas y devuelve el objeto transformador correspondiente.
- `calc_matriz_correlacion(self, columnas=None, filas=None)` : Método para la creación de la matriz de correlación. Permite especificar el número de filas y columnas a considerar para el cálculo de la correlación. Calcula la matriz de correlación utilizando el método de correlación de Pearson y la almacena en la variable `matriz_correlacion`.
- `clasificacion_correlacion(self, target_name, nivel_correlacion_target=0.3, nivel_correlacion_inter_variables=0.3)` : Clasifica las variables en función del nivel de correlación. Identifica las variables con una correlación moderada con el objetivo (target) y las variables con correlación moderada entre ellas para reducir la multicolinealidad. Devuelve una lista de las variables seleccionadas.
- `crea_EDAD_NORM(self, nombre_columna='EDAD')` : Este método crea una variable llamada "EDAD" normalizada. Utiliza la fecha actual como referencia y calcula la diferencia entre la fecha de nacimiento proporcionada y la fecha actual. Luego, aplica una eliminación de outliers y una transformación estándar, normalizando los valores entre el valor mínimo y máximo. Utiliza el objeto `Transf_MinMaxScaler` para realizar la transformación y guarda el objeto transformador para su uso posterior. Finalmente, elimina las columnas 'ANIO', 'MES' y 'FECHANACIMIENTO' del DataFrame.
- `mod_columna_Transf_Quantile(self, nombre_columna)` : Este método modifica la columna indicada utilizando `Transf_Quantile`. Aplica la transformación de cuantiles a la columna, utilizando el objeto `Transf_Quantile`, y guarda el objeto transformador para su uso posterior.
- `mod_columna_OneHot_binario(self, nombre_columna)` : Este método modifica la columna indicada utilizando la codificación One-Hot binaria. Utiliza el objeto `Transf_OneHot_binario` para realizar la codificación. Convierte la columna en un

arreglo binario y reemplaza la columna original en el DataFrame por las columnas codificadas. Guarda el objeto transformador para su uso posterior.

- `mod_columna_OneHot(self, nombre_columna)` : Este método modifica la columna indicada utilizando la codificación One-Hot. Utiliza el objeto `Transf_OneHot` para realizar la codificación. Convierte la columna en un arreglo de variables binarias y agrega estas columnas al DataFrame. Luego, elimina la columna original del DataFrame. Guarda el objeto transformador para su uso posterior.
- `mod_woe(self, nombre_columna, nombre_target, metrica="woe")` : Este método transforma los elementos de una columna utilizando la técnica Weight of Evidence (WOE) de OptimalBinning. El WOE se utiliza para medir la relación entre una variable predictora y una variable objetivo. Se puede especificar la métrica a utilizar, que puede ser "event_rate", "woe", "indices" o "bins". Utiliza el objeto `Transf_woe` para realizar la transformación. Calcula el WOE de la columna indicada en relación con la variable objetivo especificada. Reemplaza los valores de la columna con los valores WOE calculados. Guarda el objeto transformador para su uso posterior.

Estos métodos realizan diferentes transformaciones en las columnas del DataFrame utilizando diversas técnicas de transformación o codificación. También guardan los objetos transformadores utilizados, lo que permite aplicar las mismas transformaciones a nuevos datos en el futuro.

In [7]:

```

class preprocesamiento:
    def __init__(self,revisor_data):
        """
        Constructor de la clase recibe el objeto instanciado de la clase revisor_da
        seria mejor usar un patron de diseño DataFrameSingleton pero esta division
        para fines demostrativos,en codigo final todos los metodos deben
        pertenecer a una sola clase.
        """
        try:
            # Se carga el df
            self.revisor_data = revisor_data
            self.df = self.revisor_data.df
            self.matriz_correlacion = None
            self.transformaciones = {}
        except Exception as error:
            print("Ocurrió un error:", error)

#Este metodo esta repetido en la clase graficos, al final se debe mejorar el co
    def variables_numericas(self):
        """Devuelve una lista con las variables numericas de un df"""
        return [column_name for column_name, data_type in zip(self.df.columns, self

    def detencion_outlier(self,nombre_columna,q=.1):
        """
        Funcion para detectar valores atípicos (utiliers) de una columna especifica
        Por defecto se usa Deciles / dividiendo la distribucion en 10 partes, pero
        q = 0.25 se trabajaria con cuartiles.
        La funcion devuelve una "lista/pandas.core.indexes.numeric.Int64Index" con

```

```

valores atípicos del df
"""
# try:
# calculo de cuantiles
Q1 = self.df[nombre_columna].quantile(q)
Q3 = self.df[nombre_columna].quantile(1-q)
IQR = Q3-Q1
limite_inferior = Q1 - 1.5 * IQR
limite_superior = Q3 + 1.5 * IQR
indice_filas_eliminar = self.df.index[(self.df[nombre_columna] < limite_inf
return indice_filas_eliminar

def eliminacion_outlier(self,nombre_columna,q=0.1):
"""
Funcion para eliminar valores atípicos (outliers) de una columna especifica
La funcion no devuelve nada porque los cambios se hacen en el df que se pasa
"""
try:
    if nombre_columna in self.variables_numericas():
        indices = self.detection_outlier(nombre_columna,q)
        self.df = self.df.drop(indices)
        self.df.reset_index(inplace=True,drop=True)
    else:
        print("La variable no es numerica")
except Exception as error:
    print("Ocurrió un error:", error)

def guardar_transformador(self,nombre_columna,transformador):
"""
Metodo para guardar transformador aplicado, se almacenan en lista, si se aplican dos o mas se almacenaran en la secuencia aplicada"""
if nombre_columna in list(self.transformaciones.keys()):
    self.transformaciones[nombre_columna] = self.transformaciones[nombre_co
else:
    self.transformaciones[nombre_columna] = [transformador]

def Transf_MinMaxScaler(self,nombre_columna):
"""
crea objeto que Transforma los valores de la columna indicada
usando MinMaxScaler de sklear, devuelve el objeto para transformar futuros
try:
    if nombre_columna in self.variables_numericas():
        scaler = preprocessing.MinMaxScaler()
        escalador = scaler.fit(self.df[nombre_columna].values.reshape(-1, 1
        return escalador
    else:
        print("La variable no es numerica")
except Exception as error:
    print("Ocurrió un error:", error)

def Transf_Quantile(self,nombre_columna):
"""
crea objeto para Transformar los valores de la columna indicada
usando QuantileTransformer de sklearn"""
try:
    if nombre_columna in self.variables_numericas():
        scaler = preprocessing.QuantileTransformer()
        escalador = scaler.fit(self.df[nombre_columna].values.reshape(-1, 1
        return escalador

```

```

        else:
            print("La variable no es numerica")
    except Exception as error:
        print("Ocurrió un error:", error)

    def Transf_OneHot_binario(self, nombre_columna):
        """Crea objeto para transformacion OneHot cuando la variable es binaria, de
        el transformador ya entrenado"""
        try:
            value_var = self.df[nombre_columna].astype("category")
            codificador_oneHot = OneHotEncoder(handle_unknown='ignore', drop='first')
            codificacion = codificador_oneHot.fit(pd.DataFrame(value_var, columns=[value_var]))
            return codificacion
        except Exception as error:
            print("Ocurrió un error:", error)

    def Transf_OneHot(self, nombre_columna):
        """crea objeto para Transformar los valores de la columna indicada
        usando OneHot encoder de sklearn, devuelve el transformador entrenado """
        try:
            value_var = self.df[nombre_columna].astype("category")
            codificador_oneHot = OneHotEncoder(handle_unknown='ignore')
            codificacion = codificador_oneHot.fit(pd.DataFrame(value_var, columns=[value_var]))
            return codificacion
        except Exception as error:
            print("Ocurrió un error:", error)

    def Transf_woe(self,nombre_columna,nombre_target):
        """crea el objeto para Transformar los valores de la columna indicada
        usando woe de OptimalBinning,
        Distingue de variables numericas y categoricas
        Devuelve el objeto transoformador"""
        try:
            data_type = self.df[nombre_columna].dtypes
            target = self.df[nombre_target]
            x = self.df.loc[:,nombre_columna]
            if (data_type == 'object' or data_type.name == 'category'):
                optb = OptimalBinning(name = nombre_columna,dtype ='categorical',sol
                optb.fit(x,target)
            elif np.issubdtype(data_type, np.number):
                optb = OptimalBinning(name = nombre_columna,dtype = 'numerical',sol
                optb.fit(x,target)
            else:
                print("La variable se de convertir a tipo numerica o categorica")
                optb = None

            return optb
        except Exception as error:
            print("Ocurrió un error:", error)

    def calc_matriz_correlacion(self, columnas=None,filas=None):
        """metodo para la creacion de la matriz de correlacion, se pude ajustar el
        a calcular para la correlacion"""
        try:
            if self.matriz_correlacion is None or self.matriz_correlacion.empty:
                if columnas:

```

```

# Se confirma que las columnas existan y sean numericas
columnas = [ col for col in columnas if col in self.variables_numericas()]
else:
    columnas = self.variables_numericas()
if filas:
    data_set = self.df[columnas].sample(filas)
    data_set.reset_index(inplace=True, drop=True)
else:
    data_set = self.df[columnas]
self.matriz_correlacion = data_set.corr(method='pearson', numeric_only=True)
except Exception as error:
    print("Ocurrió un error:", error)

def clasificacion_correlacion(self, target_name, nivel_correlacion_target = 0.3):
    """Metodo para clasificar las variables en función del nivel de correlación
    1. Entre las variables y el objetivo (target_name), escogiendo las que tengan una Moderada correlación |corr| > 0.3 pero se puede ajustar si se desea.
    2. Entre las mismas variables permitiendo hasta una Moderada correlación |corr| <= 0.3 con el objetivo de reducir la multicolinalidad"""
try:
    # Calculo de la correlacion para todas las variables
    self.calcular_matriz_correlacion()
    matriz = self.matriz_correlacion

    # Verificación de la correlación entre todas las variables y el target, se guardan los resultados en signals
    signals = []
    for i in range(len(matriz)):
        for j in range(i+1, len(matriz)):
            signal_1 = matriz.columns[i]
            signal_2 = matriz.columns[j]
            correlation = abs(matriz.iloc[i, j])
            signals.append([signal_1, signal_2, correlation])
    df_correlacion = pd.DataFrame(signals, columns=['sig_1', 'sig_2', 'nivel_correlacion'])
    df_correlacion_mod = df_correlacion[df_correlacion['sig_2'] == target_name]
    variables_x = list(df_correlacion_mod[df_correlacion_mod['nivel_correlacion'] >= nivel_correlacion_target].index)

    # Verificación de la correlación entre variables
    for i, variable1 in enumerate(variables_x):
        for variable2 in variables_x[i+1:]:
            mask = (df_correlacion['sig_1'] == variable1) & (df_correlacion['sig_2'] == variable2)
            val_corr = list(df_correlacion[mask]['nivel_correlacion'])
            if not val_corr:
                mask = (df_correlacion['sig_1'] == variable2) & (df_correlacion['sig_2'] == variable1)
                val_corr = list(df_correlacion[mask]['nivel_correlacion'])
            val_corr = val_corr[0] if val_corr else 0
            if val_corr > nivel_correlacion_inter_variables:
                if variable2 in variables_x:
                    variables_x.remove(variable2)
    return variables_x
except Exception as error:
    print("Ocurrió un error:", error)

def crea_EDAD(self, nombre_columna = 'EDAD'):
    """Para crear una variable EDAD se usa como referencia la fecha actual, independiente del momento en que se corra el programa, se calcula la diferencia de edad entre la fecha actual y la fecha de nacimiento"""

```

```

        la fecha de nacimiento provista y la fecha actual
"""

try:
    fecha_actual = datetime.datetime.now()
    self.df[nombre_columna] = (fecha_actual - self.df['FECHANACIMIENTO']).days
    self.df.drop(columns=['ANIO', 'MES', 'FECHANACIMIENTO'], inplace=True)
    # Se actualiza la instancia de la clase revisor_data_csv
    #self.revisor_data.df = self.df
except Exception as error:
    print("Ocurrió un error:", error)

def crea_EDAD_NORM(self,nombre_columna = 'EDAD'):
    """Para crear una variable EDAD normalizada se usa como referencia la fecha
    independiente del momento en que se corra el programa, se calcula la diferencia
    entre la fecha de nacimiento provista y la fecha actual, se aplica una eliminación
    y una transformación estandar normalizando entre el valor mínimo y máximo.
    """
    try:
        fecha_actual = datetime.datetime.now()
        self.df[nombre_columna] = (fecha_actual - self.df['FECHANACIMIENTO']).days
        self.eliminacion_outlier(nombre_columna)
        escalador = self.Transf_MinMaxScaler(nombre_columna)
        self.df[nombre_columna] = escalador.transform(self.df[nombre_columna].values)
        self.guardar_transformador(nombre_columna,escalador)
        self.df.drop(columns=['ANIO', 'MES', 'FECHANACIMIENTO'], inplace=True)

        # Se actualiza la instancia de la clase revisor_data_csv
        #self.revisor_data.df = self.df
    except Exception as error:
        print("Ocurrió un error:", error)

def mod_columna_Transf_Quantile(self,nombre_columna):
    """Modifica la columna indicada utilizando el Transf_Quantile,
    almacena el nombre de la columna modificada y el objeto utilizado para la transformación"""
    try:
        escalador = self.Transf_Quantile(nombre_columna)
        self.df[nombre_columna] = escalador.transform(self.df[nombre_columna].values)
        self.guardar_transformador(nombre_columna,escalador)

        # Se actualiza la instancia de la clase revisor_data_csv
        #self.revisor_data.df = self.df
    except Exception as error:
        print("Ocurrió un error:", error)

def mod_columna_OneHot_binario(self,nombre_columna):
    """Modifica la columna indicada utilizando OneHot_binario,
    almacena el nombre de la columna modificada y
    el objeto utilizado para la transformación"""
    try:
        escalador = self.Transf_OneHot_binario(nombre_columna)
        arreglo = escalador.transform(preproceso.df[[nombre_columna]]).toarray()
        self.df[nombre_columna] = arreglo.astype(int)
        self.guardar_transformador(nombre_columna,escalador)

        # Se actualiza la instancia de la clase revisor_data_csv
        #self.revisor_data.df = self.df

```

```

        except Exception as error:
            print("Ocurrió un error:", error)

    def mod_columna_OneHot(self,nombre_columna):
        try:
            escalador = self.Transf_OneHot(nombre_columna)
            arreglo = escalador.transform(preproceso.df[[nombre_columna]]).toarray()
            columnas_codificadas = escalador.get_feature_names_out([nombre_columna])
            df_codificado = pd.DataFrame(arreglo, columns=columnas_codificadas)

            #Se introducen en df los valores codificados
            pos = self.df.columns.get_loc(nombre_columna)
            for col in df_codificado.columns:
                self.df.insert(pos, col, df_codificado[col])
                pos += 1
            self.df.drop(columns=[nombre_columna], inplace=True)
            self.guardar_transformador(nombre_columna,escalador)

            # Se actualiza la instancia de la clase revisor_data_csv
            #self.revisor_data.df = self.df
        except Exception as error:
            print("Ocurrió un error:", error)

    def mod_woe(self,nombre_columna,nombre_target,metrica = "woe"):
        """Metodo para transformar los elementos de una columna
        usando woe de OptimalBinning, se pueden utilizar otras metricas
        como: "event_rate", "woe", "indices" and "bins" ."""
        try:
            escalador = self.Transf_woe(nombre_columna,nombre_target)
            x = self.df.loc[:,nombre_columna]
            self.df[nombre_columna] = escalador.transform(x, metric=metrica)
            self.guardar_transformador(nombre_columna,escalador)

            # Se actualiza la instancia de la clase revisor_data_csv
            #self.revisor_data.df = self.df
        except Exception as error:
            print("Ocurrió un error:", error)
    preproceso = preprocesamiento(revision)

    def grafica_confusion_matrix(y_test, y_test_predictions):
        conf_matrix = confusion_matrix(y_test, y_test_predictions)
        fig, ax = plt.subplots(figsize=(8,6), dpi=100)
        display = ConfusionMatrixDisplay(conf_matrix)
        ax.set(title='Confusion Matrix')
        display.plot(ax=ax)

```

In [8]: `preproceso = preprocesamiento(revision)`

Cada columna o variable tiene características que debe ser analizada para hacer la transformación más adecuada, en este caso se pueden distinguir las siguientes:

- La variable 'EDAD' registra la edad del cliente al momento del registro, clasificándola como una variable cuantitativa de intervalo. Esta variable está relacionada con 'ANIO', 'MES' y 'FECHANACIMIENTO'. Con el objetivo de simplificar y capturar la información de

las edades de los clientes, proponemos convertir la variable 'EDAD' en una variable cuantitativa de razón. Para lograr esto, utilizaremos el tiempo actual como referencia para calcular la edad y normalizaremos los valores, preservando únicamente la información relativa a las edades de los clientes.

- La variable 'INGRESO' presenta una significativa cantidad de outliers, sin embargo no se deben eliminar porque en realidad representan información valida,

por eso se procede a normalizar utilizando intercuartiles, ya que esta normalización disminuye el impacto de los outliers.

- Las variables 'SEXO' y 'ESTADOCIVIL' se pueden codificar usando OneHotEncoder de sklearn.
- El resto de las variables se puede normalizar usando diferentes métodos, sin embargo por simplicidad se utilizará el weight of evidence apoyandonos en la librería OptimalBinning

```
In [9]: preprocesso.crea_EDAD_NORM('EDAD')
preprocesso.mod_columna_Transf_Quantile('INGRESO')
preprocesso.mod_columna_OneHot_binario('SEXO')
preprocesso.mod_columna_OneHot('ESTADOCIVIL')
preprocesso.mod_woe('ESTADO','TARGET')

columnas_faltantes = ['MORAS', 'MARCACIONES', 'CONTACTOS', 'M1', 'C1', 'M2', 'C2', 'C3']
for columna in columnas_faltantes:
    preprocesso.mod_woe(columna,'TARGET')
```

```
In [4]: #Para utilizar las graficas se debe instanciar la clase con el nuevo df
graficas = revisor_grafico(revision.df)
```

Una vez que se tienen las variables normalizadas y ajustadas para su implementación en los modelos se pueden escoger varios métodos para hacer la selección de las variables más importantes.

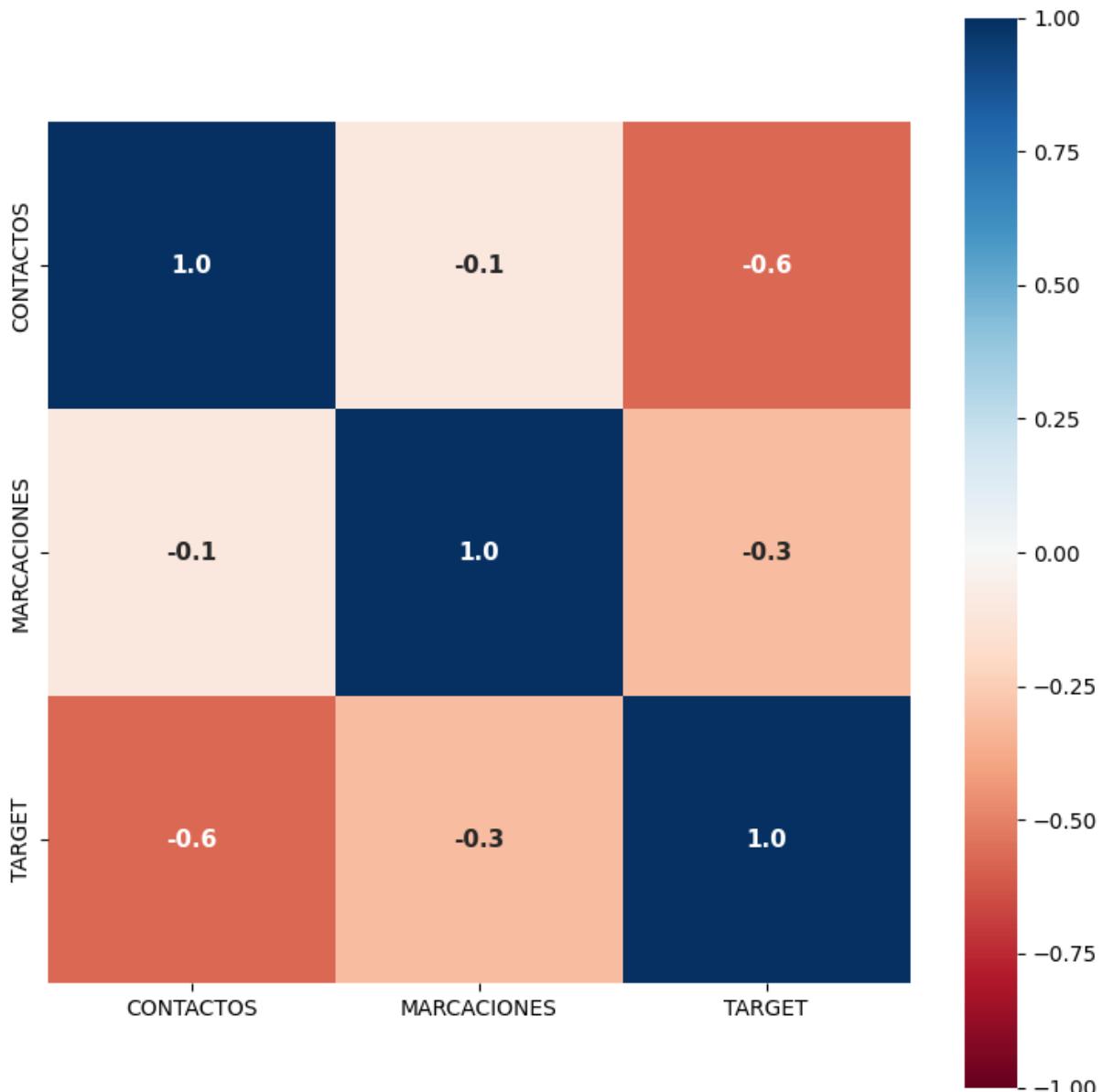
9.1 Filtro: Correlación para Selección de factores influyentes

```
In [134...]: # utilizando la correlación como indicador
variables_a_utilizar = preprocesso.clasificacion_correlacion('TARGET')
variables_a_utilizar = variables_a_utilizar+[ 'TARGET']
print("Permitiendo una correlación mínima de 0.3 variables/target e intervariables")
```

Permitiendo una correlación mínima de 0.3 variables/target e intervariables las columnas a usar son :

['CONTACTOS', 'MARCACIONES', 'TARGET']

```
In [6]: graficas.correlacion(variables_a_utilizar)
```

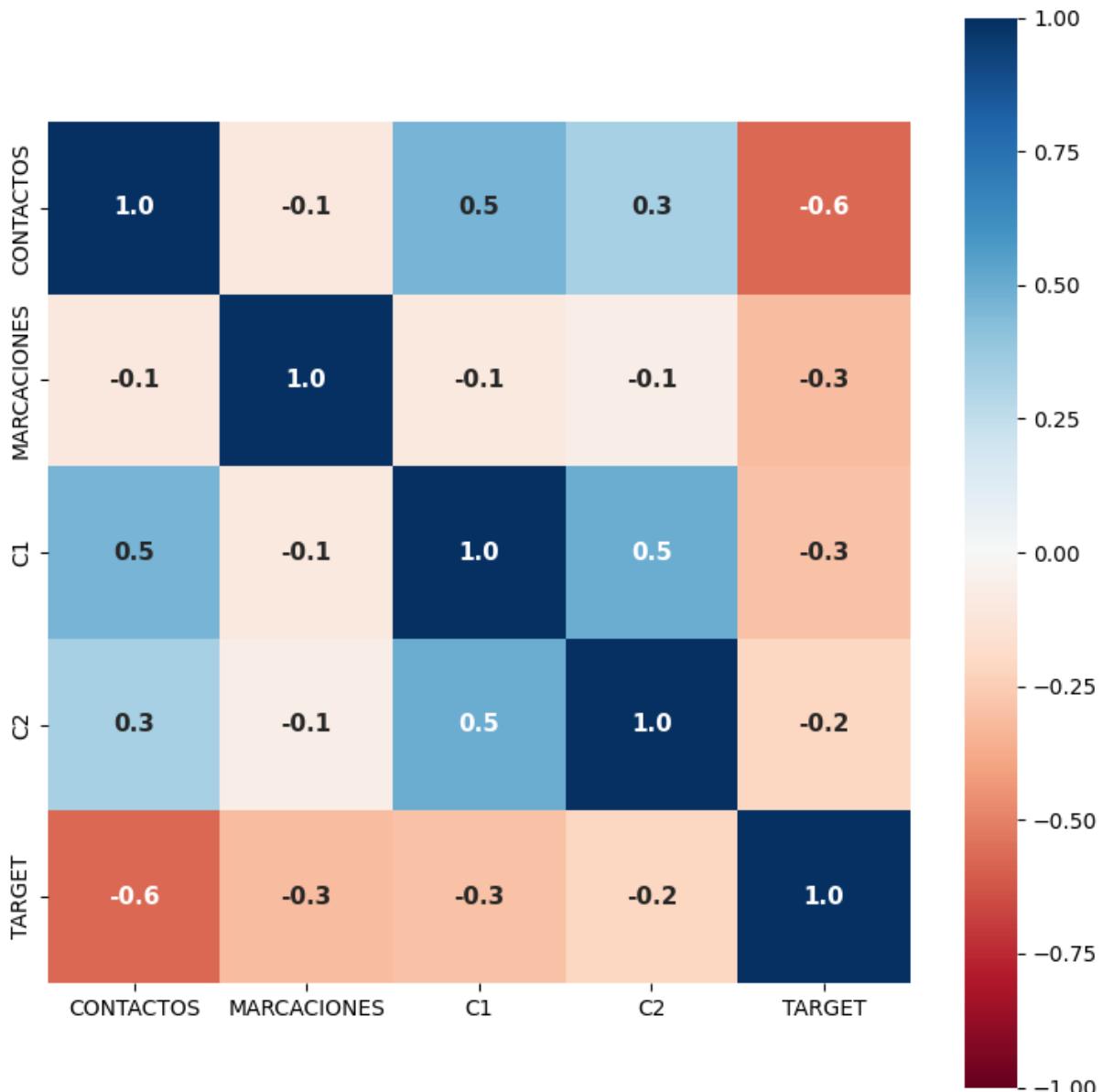


```
In [136]: # Correlacion dejando mas Libertad
min_corr_target = 0.2
max_corr_inter = 0.5
variables_a_utilizar = preproceso.clasificacion_correlacion('TARGET',min_corr_target,max_corr_inter)
variables_a_utilizar = variables_a_utilizar+[ 'TARGET']
print(f"Permitiendo una correlacion minima de {min_corr_target} variables/target y max {max_corr_inter} intervariables")
```

Permitiendo una correlacion minima de 0.2 variables/target y max 0.5 intervariables
las columnas a usar son :

['CONTACTOS', 'MARCACIONES', 'C1', 'C2', 'TARGET']

```
In [8]: graficas.correlacion(variables_a_utilizar)
```



9.2 Filtro: Information Value (IV) para Selección de factores influyentes

El Information Value (IV) es una medida para evaluar la capacidad predictiva de una variable en relación con la variable objetivo. Se calcula analizando la distribución de la variable en diferentes categorías y su relación con la variable objetivo. El IV se utiliza como criterio para seleccionar características relevantes. Se busca un IV alto (por encima de un umbral) para considerar una variable como informativa. Esto ayuda a simplificar y mejorar la precisión del modelo al reducir el ruido y la redundancia en los datos.

El IV cuantifica la predictividad de una variable de la siguiente manera:

- $IV < 0.02$: Variable sin poder predictivo.
- $0.02 \leq IV < 0.1$: Variable con un poder predictivo débil.
- $0.1 \leq IV < 0.3$: Variable con un poder predictivo medio.

- $0.3 \leq IV < 0.5$: Variable con un poder predictivo fuerte.
- $IV \geq 0.5$: Variable con un poder predictivo muy fuerte.

In [100...]

```
#En esta caso se pueden convertir todas las variables usando woe
preproceso_woe = preprocesamiento(revision)

#Se modifica primero edad usando MinMaxScaler
preproceso_woe.crea_EDAD_NORM('EDAD')

#Se aplica woe a todas las variables (menos a target)
for columna in preproceso_woe.df.drop(columns = ['TARGET']):
    preproceso_woe.mod_woe(columna,'TARGET')

# Se puede visualizar la informacion en el proceso de transformacion
tablas_transformacion = {}
for nombre_columna , transformador in zip(preproceso_woe.transformaciones.keys(), p
    tablas_transformacion[nombre_columna] = transformador[-1].binning_table.build()

# O tambien obtener la informacion necesaria directamente
def encontrarIV(preproceso_obj):
    nombre_Columnas = []
    columnas_IV = []
    columnas_gini = []
    columnas_JS = []
    for nombre_columna , transformador in zip(preproceso_obj.transformaciones.keys(
        nombre_Columnas.append(nombre_columna)
        columnas_IV.append(transformador[-1].binning_table.iv)
        columnas_gini.append(transformador[-1].binning_table.gini)
        columnas_JS.append(transformador[-1].binning_table.js)

    df_Information_values = pd.DataFrame({'Feature': nombre_Columnas,'IV' :columnas
    df_Information_values.sort_values(by='IV', ascending=False,inplace=True)
    df_Information_values.reset_index(inplace=True,drop=True)
    return df_Information_values

df_Information_values = encontrarIV(preproceso_woe)

print(f"filtrando para variables con valor predictivo mayor a moderado (IV>0.3) las
print(df_Information_values[df_Information_values['IV']>0.3])
```

filtrando para variables con valor predictivo mayor a moderado ($IV>0.3$) las columnas a usar son :

	Feature	IV	gini	JS
0	CONTACTOS	1.694075	0.582097	0.182586
1	MARCACIONES	0.462279	0.335012	0.053102
2	C1	0.391705	0.286734	0.046200

9.3 Filtro: SelectKBest Chi-cuadrado para Selección de factores influyentes

SelectKBest es una clase de la biblioteca scikit-learn que se utiliza para seleccionar las mejores características según una función de puntuación dada. En este caso, se utiliza la función de puntuación chi2, que calcula el estadístico chi-cuadrado para evaluar la

dependencia entre dos variables categóricas. Cuanto mayor sea la puntuación chi-cuadrado, mayor será la dependencia entre la característica y la variable objetivo. SelectKBest selecciona las k características con las puntuaciones chi-cuadrado más altas.

Este código realiza transformaciones en los datos utilizando la métrica event_rate, aplica la transformación a las variables, utiliza SelectKBest con chi2 como función de puntuación para seleccionar las mejores características y muestra los resultados en un DataFrame.

```
In [3]: # esto porque chi2 no permite valores negativos
preproceso_chi2 = preprocesamiento(revision)

#Se modifica primero edad usando MinMaxScaler
preproceso_chi2.crea_EDAD_NORM('EDAD')

#Se aplica woe a todas las variables (menos a target)
for columna in preproceso_chi2.df.columns[:-1]:
    preproceso_chi2.mod_woe(columna,'TARGET','event_rate')

x = preproceso_chi2.df.iloc[:, :-1]
y = preproceso_chi2.df[['TARGET']]

prueba = SelectKBest(score_func=chi2,k = 5)
entrenamiento = prueba.fit(x,y)

x_final = prueba.transform(x)

df_SelectKBest = pd.DataFrame({'Feature': x.columns , 'kBest_chi2' :entrenamiento.scores_})
df_SelectKBest.sort_values(by='kBest_chi2', ascending=False,inplace=True)
df_SelectKBest.reset_index(inplace=True,drop=True)
df_SelectKBest.head()

#caracteristicas = entrenamiento.transform(x)
```

	Feature	kBest_chi2
0	CONTACTOS	7945.805251
1	MARCACIONES	699.555893
2	C1	523.399643
3	C2	155.959107
4	M1	90.524811

9.4 Filtro: Análisis de Componentes Principales (PCA)

El Análisis de Componentes Principales (PCA, por sus siglas en inglés) es una técnica de reducción de dimensionalidad que se utiliza para identificar patrones y estructuras subyacentes en conjuntos de datos de alta dimensionalidad. El objetivo del PCA es transformar un conjunto de variables correlacionadas en un nuevo conjunto de variables no correlacionadas llamadas componentes principales usando la Descomposición de Valor

Singular (SVD) de los datos para proyectarlos a un espacio dimensional más bajo. Los datos de entrada se centran pero no se escalan para cada característica antes de aplicar el SVD.

La implementación del PCA utilizando la biblioteca scikit-learn en Python se realiza a través del módulo sklearn.decomposition y la clase .Po de los componentes principales.

El PCA permite reducir la dimensionalidad de los datos al seleccionar un número adecuado de componentes principales que capturan la mayor parte de la varianza. Estos componentes principales se pueden utilizar posteriormente en tareas de análisis exploratorio, visualización de datos o como entrada para alg

La desventaja de este metodo es que se pierde la relacion entre la data y los resultados de cualquier modelo.oritmos de aprendizaje automático.A.

In [53]:

```
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

preproceso = preprocesamiento(revision)
preproceso.df = preproceso.df.sample(10000)

# Se convierten todas las variables
preproceso.crea_EDAD_NORM('EDAD')

# para variar un poco se ustilizara la conversion mas sencilla
# en las variables categoricas
preproceso.mod_columna_OneHot('ESTADO')
preproceso.mod_columna_OneHot('ESTADOCIVIL')
preproceso.mod_columna_OneHot_binario('SEXO')
preproceso.mod_columna_Transf_Quantile('INGRESO')

#Tambien se pueden utilizar otras formas de normalizar
def trans_StandardScaler(df_org,columnas):
    df = df_org[columnas]
    escalador = StandardScaler()
    escalador.fit(df)
    data_transf = escalador.transform(df)
    df_trans = pd.DataFrame (revisar,columns = otras_columnas)
    for columna in df_trans.columns:
        df_org[columna] = df_trans[columna]
    return df_org
otras_columnas = ['MORAS','MARCACIONES', 'CONTACTOS', 'M1', 'C1', 'M2', 'C2','M3',
preproceso.df = trans_StandardScaler(preproceso.df,otras_columnas)

#Se define la cantidad componentes a utilizar
k = 2
# Creamos instancia de la clase `PCA`:
pca = PCA(n_components=k)

X = preproceso.df.drop(columns = ['TARGET'])
y = preproceso.df['TARGET'].to_numpy()
```

```
pca.fit(X)

componentes_principales = pca.components_
varianza_explicada = pca.explained_variance_ratio_

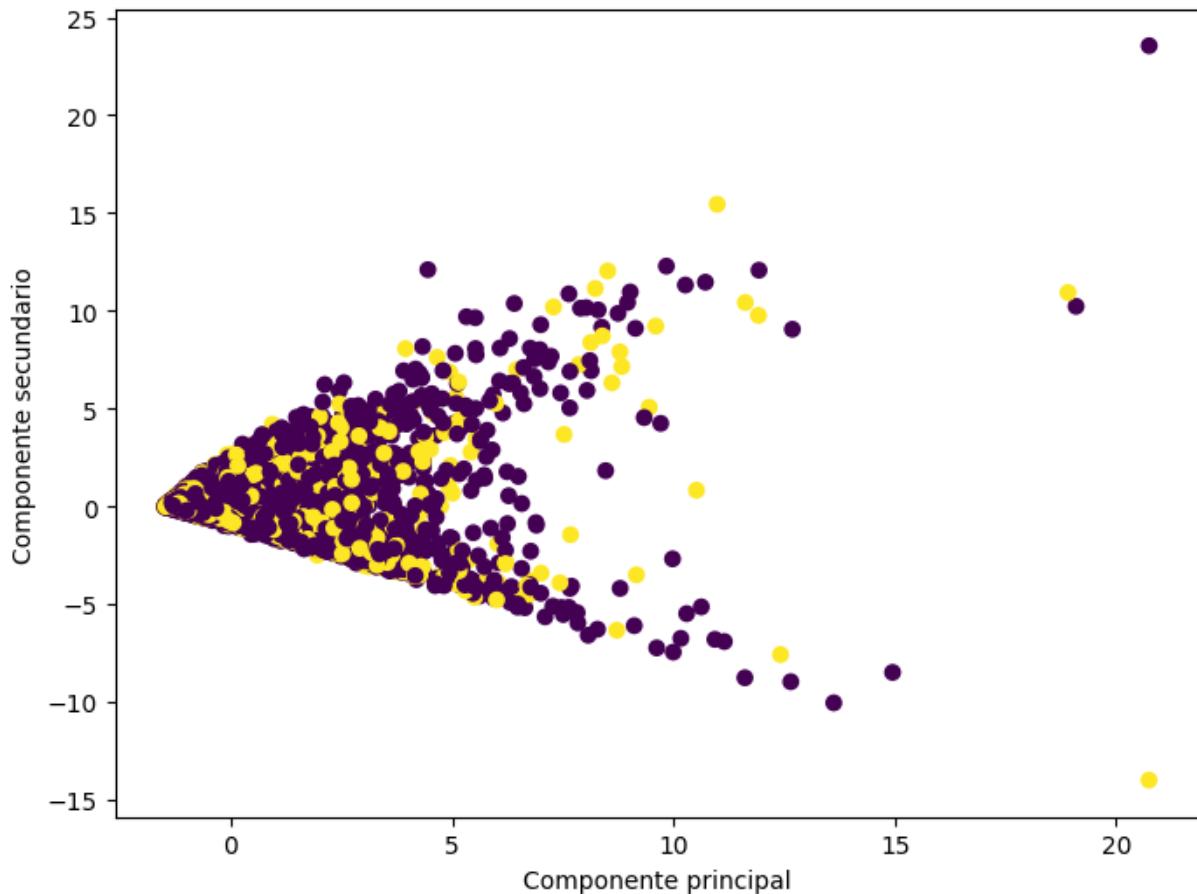
X_transformado = pca.transform(X)

print('varianza_explicada:', varianza_explicada)

plt.figure(figsize= (8,6))
plt.scatter(X_transformado[:,0],X_transformado[:,1],c = y)
plt.xlabel('Componente principal')
plt.ylabel('Componente secundario')
```

varianza_explicada: [0.23349505 0.16387174]

Out[53]: Text(0, 0.5, 'Componente secundario')



9.5: Envoltura: Feature Importance Tree Classifier

Se puede obtener la importancia de la característica de cada variable/columna del conjunto de datos utilizando la propiedad de importancia de la característica del modelo. La importancia de las características es una clase incorporada que viene con los clasificadores basados en árboles, usaremos el `ExtraTreesClassifier` para extraer las 5 características principales para el conjunto de datos. En el caso del clasificador `ExtraTreesClassifier` de scikit-learn, la importancia de las características se calcula por defecto mediante la

medida "Gini importance" pero se puede modificar para que utilice `{"gini", "entropy", "log_loss"}`. Esta métrica mide la reducción media de la impureza de Gini que se logra al dividir un nodo utilizando una característica en particular en el árbol de decisión. prueba de dimensiones `(n_samples, n_features)`.

La implementación interna de `ExtraTreesClassifier` se basa en la construcción y entrenamiento de múltiples árboles de decisión utilizando una combinación de Bootstrap Aggregating (Bagging) y selección aleatoria de características. Cada árbol se entrena utilizando una versión modificada del algoritmo de árboles de decisión CART (Classification and Regression Trees).

El proceso de selección de características aleatorias implica seleccionar un subconjunto aleatorio de características en cada nodo de división durante la construcción de los árboles. La importancia de cada característica se calcula utilizando la métrica de importancia Gini o Entropía de cada árbol, y luego se promedia para obtener una importancia global de las características.

La predicción se realiza tomando las decisiones de cada árbol individual en el conjunto y realizando llevar a un mejor rendimiento en la clasificación de datos.

In [130...]

```
from sklearn.ensemble import ExtraTreesClassifier

preproceso = preprocesamiento(revision)

# Para este caso solo se convertira la edad
preproceso.crea_EDAD_NORM('EDAD')

#Se aplica woe a todas las variables (menos a target)
for columna in preproceso.df.drop(columns = ['TARGET']):
    preproceso.mod_woe(columna,'TARGET','event_rate')

X = preproceso.df.drop(columns = ['TARGET'])
y = preproceso.df['TARGET'].to_numpy()

modelo = ExtraTreesClassifier()
modelo.fit(X,y)

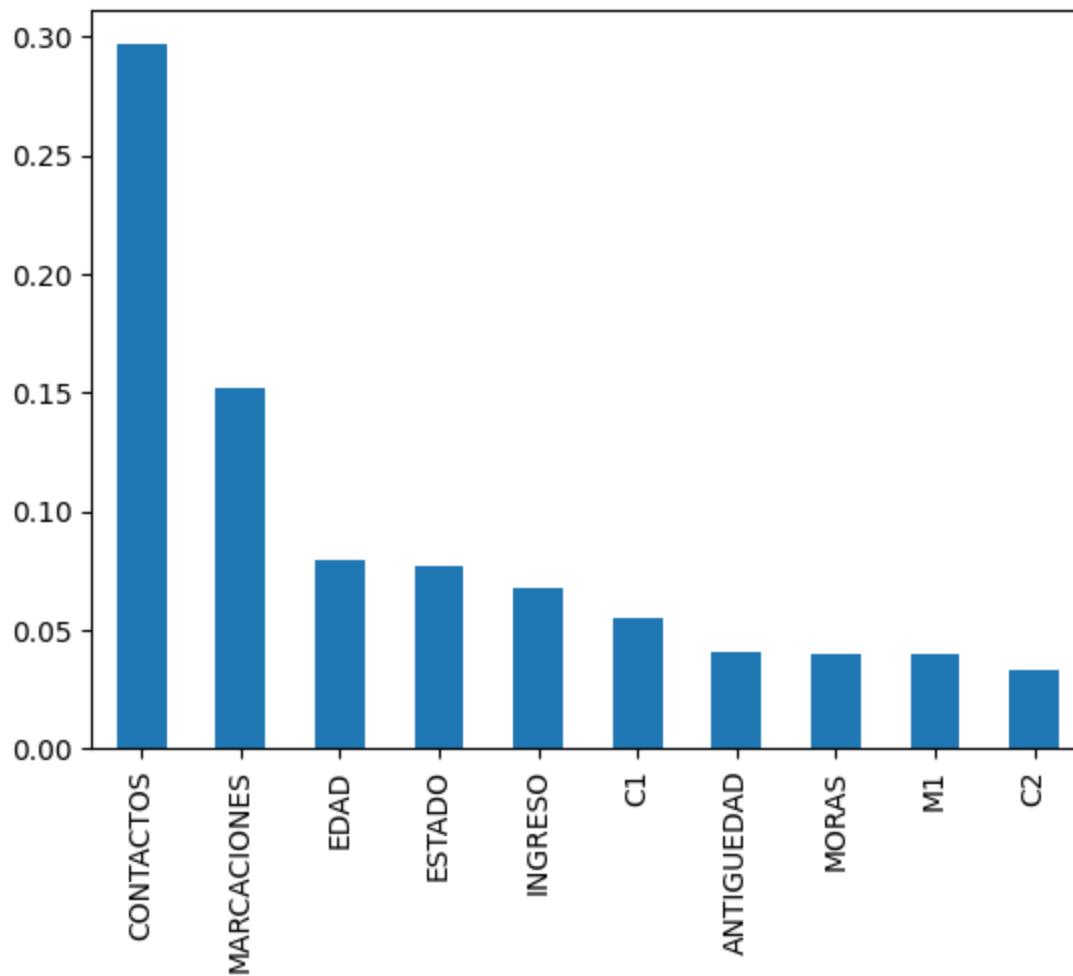
feature_imp = pd.Series(modelo.feature_importances_,index=X.columns)
feature_imp = feature_imp.sort_values(ascending=False)

print('Las 5 columnas mas importantes y su nivel de importancia es:')
for columna,importancia in zip(feature_imp.head().index,feature_imp.head().values):
    print(f" {columna} :\t {round(importancia,2)}")

feature_imp.nlargest(10).plot(kind = 'bar')
plt.show()
```

Las 5 columnas mas importantes y su nivel de importancia es:

CONTACTOS :	0.3
MARCACIONES :	0.15
EDAD :	0.08
ESTADO :	0.08
INGRESO :	0.07



9.6 Envoltura: Recursive Feature Elimination

La eliminación de características recursivas (Recursive Feature Elimination o RFE) es un algoritmo de optimización que busca encontrar el subconjunto de características con mejor rendimiento. Se basa en la eliminación iterativa de características menos importantes según una medida de importancia calculada por un modelo de aprendizaje automático. El proceso se repite hasta alcanzar un número deseado de características o algún criterio de parada. Al eliminar características menos informativas.

In [155]:

```
from sklearn.feature_selection import RFE

# Ejemplo 1 de modelo de clasificación
from sklearn.svm import SVC

# Ejemplo 2 de modelo de clasificación
from sklearn.linear_model import LogisticRegression
```

```

preproceso = preprocesamiento(revision)
preproceso.df = preproceso.df.sample(1000)

preproceso.crea_EDAD_NORM('EDAD')

for columna in preproceso.df.drop(columns=['TARGET']):
    preproceso.mod_woe(columna, 'TARGET', 'woe')

X = preproceso.df.drop(columns=['TARGET'])
y = preproceso.df['TARGET'].to_numpy()
n_features = 5

# Modelo de clasificación de Support Vector Machine (SVM)

estimator = SVC(kernel='linear', coef0=1, probability=True)
selector_SVC = RFE(estimator, n_features_to_select=n_features)
selector_SVC.fit(X, y)
#selected_features = selector.transform(X)
print("Utilizando Support Vector Machine las variables a utilizar serian: \n", X.co

# Modelo de clasificación Regresion Logistica
modelo = LogisticRegression()
selector_Lr = RFE(modelo, n_features_to_select=n_features)
selector_Lr.fit(X,y)
print("Utilizando Regresion Logistica las variables a utilizar serian: \n", X.colu

```

Utilizando Support Vector Machine las variables a utilizar serian:
Index(['ESTADO', 'ESTADOCIVIL', 'MARCACIONES', 'CONTACTOS', 'M2'], dtype='object')
Utilizando Regresion Logistica las variables a utilizar serian:
Index(['ESTADOCIVIL', 'MARCACIONES', 'CONTACTOS', 'M2', 'C2'], dtype='object')

10. Nuevas variables.

Todos los procesos hasta ahora vistos modifican las variables de entrada, el punto es probar de acuerdo a los resultados estadísticos cuales poseen buen poder predictivo con respecto a la variable objetivo.

Sin embargo se pueden crear variables que aporten informacion mas concentrada, por ejemplo, la variable ingreso se puede hacer una operacion del tipo Rel_ingreso_Edad = ((ingreso)-(ingreso_promedio))/(edad**2) esta nueva variable tiene informacion relativa a la edad y el ingreso realtivo, posiblemente de mas informacion que los que la conforman con realacion a la variable target.

Adicionalmene la libreria de Sklear posee el metodo PolynomialFeatures que tambien es muy util para vericar el posible aporte de variables combinadas.

notas: <https://arxiv.org/pdf/1701.07852.pdf>

In [117...]: preproceso = preprocesamiento(revision)
preproceso.df = preproceso.df.sample(1000)

```
fecha_actual = datetime.datetime.now()
preproceso.df['EDAD'] = (fecha_actual - preproceso.df['FECHANACIMIENTO']).dt.days
preproceso.df = preproceso.df.drop(columns=['ANIO', 'FECHANACIMIENTO', 'MES'])
preproceso.df['RELACION_EDAD_INGRESO'] = (preproceso.df['INGRESO'] - preproceso.df['INGRESO'].mean()) / preproceso.df['INGRESO'].std()

for columna in preproceso.df.drop(columns = ['TARGET']):
    preproceso.mod_woe(columna,'TARGET')

def encontrarIV(preproceso_obj):
    nombre_Columnas = []
    columnas_IV = []
    columnas_gini = []
    columnas_JS = []
    for nombre_columna , transformador in zip(preproceso_obj.transformaciones.keys(),
                                                preproceso_obj.transformaciones.values()):
        nombre_Columnas.append(nombre_columna)
        transformador[-1].binning_table.build()
        columnas_IV.append(transformador[-1].binning_table.iv)
        columnas_gini.append(transformador[-1].binning_table.gini)
        columnas_JS.append(transformador[-1].binning_table.js)

    df_Information_values = pd.DataFrame({'Feature': nombre_Columnas,'IV' :columnas_IV})
    df_Information_values.sort_values(by='IV', ascending=False,inplace=True)
    df_Information_values.reset_index(inplace=True,drop=True)
    return df_Information_values

df_Information_values = encontrarIV(preproceso)
df_Information_values
```

Out[117]:

	Feature	IV	gini	JS
0	CONTACTOS	1.689238	0.598108	0.188914
1	MARCACIONES	0.478234	0.354694	0.055551
2	C1	0.373286	0.238910	0.042423
3	M1	0.272019	0.261438	0.032790
4	M2	0.205382	0.214917	0.025098
5	MORAS	0.142330	0.198449	0.017558
6	ESTADO	0.132297	0.184313	0.015994
7	C2	0.122927	0.157750	0.015166
8	RELACION_EDAD_INGRESO	0.115808	0.180283	0.014213
9	C3	0.100111	0.121556	0.012192
10	INGRESO	0.083592	0.154553	0.010397
11	M3	0.065796	0.096129	0.008031
12	EDAD	0.033109	0.101547	0.004127
13	ANTIGUEDAD	0.015189	0.050602	0.001891
14	ESTADOCIVIL	0.007830	0.046547	0.000978
15	SEXO	0.000905	0.014989	0.000113

11. Preparar un conjunto de entrenamiento y validación

La elección del porcentaje utilizado para el conjunto de entrenamiento y validación en modelos de machine learning es una decisión importante y puede depender del tamaño del conjunto de datos, la cantidad de muestras disponibles y la naturaleza del problema. En este proyecto se utilizan los siguientes:

1. División estándar: Una práctica común es utilizar una división estándar, como el 70% para entrenamiento y el 30% para validación. Esta división proporciona una proporción equilibrada entre los conjuntos de datos, donde el conjunto de entrenamiento es más grande y se utiliza para entrenar el modelo, mientras que el conjunto de validación se utiliza para evaluar el rendimiento del modelo.
2. Validación cruzada: En lugar de realizar una sola división entre entrenamiento y validación, la validación cruzada (cross-validation) divide el conjunto de datos en múltiples partes llamadas "folds". Por ejemplo, en la validación cruzada de 5 folds, se divide el conjunto de datos en 5 partes iguales. Luego, se entrena y evalúa el modelo 5 veces, utilizando cada parte como conjunto de validación una vez y las restantes como

conjunto de entrenamiento. Esto proporciona una evaluación más robusta del modelo al promediar los resultados de las iteraciones.

3. Validación estratificada: En algunos casos, es importante mantener una distribución de clases equilibrada en los conjuntos de entrenamiento y validación, especialmente cuando hay desequilibrio de clases en los datos. La validación estratificada garantiza que la proporción de cada clase se mantenga en ambos conjuntos. Esto puede ser útil cuando se busca un rendimiento más preciso en clases minoritarias.
4. Muestreo balanceado: En situaciones en las que hay un desequilibrio significativo entre las clases, se puede utilizar el muestreo balanceado para asegurar una representación adecuada de todas las clases en el conjunto de entrenamiento y validación. Por ejemplo, se puede aumentar o disminuir el número de muestras de ciertas clases para lograr un equilibrio en los con proyecto.

Modelos de clasificación

12. Elaboracion de modelos de clasificación

Se evaluara el comportamiento de los diferentes modelos y se almacenara los resultados de los mismos en un df para su posterios evaluacion.

```
In [ ]: df_modelos = pd.DataFrame([],columns = ['Nombre_modelo','accuracy_score','precision
```

12.1 Regresión Logística básica

Al utilizar la Regresión Logística como primer modelo, se obtiene una visión inicial del rendimiento y comportamiento de los datos, lo que ayuda a guiar la exploración de otros modelos más complejos y avanzados. Este modelo permite:

- Interpretación y comprensión: Proporciona una interpretación fácil de entender de los coeficientes asociados a cada característica, lo que permite comprender cómo influyen en la predicción del resultado.
- Eficiencia computacional: Es un modelo computacionalmente eficiente, lo que significa que es rápido de entrenar y realizar predicciones en comparación con otros algoritmos más complejos.
- Menos propenso a sobreajuste: Ayuda a evitar problemas de generalización y mejora la capacidad de aplicar el modelo a nuevos conjuntos de datos.
- Facilidad de implementación: La Regresión Logística es fácil de implementar y entender, lo que la hace adecuada como punto de partida para introducir conceptos de machine

learning.

El código siguiente realiza un proceso de preprocessamiento de datos, incluyendo muestreo, transformaciones y reducción de características. Luego, se entrena un modelo de clasificación de Regresión Logística con ajuste de desbalanceo de clases. Finalmente, se crea un pipeline que combina las transformaciones y el modelo, y se evalúa el rendimiento del modelo en un conjunto de prueba mediante la visualización de una matriz de confusión.

```
In [2]: preproceso = preprocessamiento(revision)
preproceso.df = preproceso.df.sample(10000)
preproceso.crea_EDAD_NORM('EDAD')

X = preproceso.df.drop(columns=['TARGET'])
y = preproceso.df['TARGET'].to_numpy()

#Una práctica común es utilizar la división estándar, 70% para entrenamiento y el 30% adelante se pueden utilizar Validación cruzada, Validación estratificada o Muestrado aleatorio
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

#Se utiliza ColumnTransformer para ajustar el modelo incluyendo las transformaciones
ct = ColumnTransformer([('Quantile', QuantileTransformer(n_quantiles=10, random_state=42), ['M1', 'C1']),
                        ('MinMax', MinMaxScaler(), ['MORAS']),
                        ('Ohe_bi', OneHotEncoder(handle_unknown='ignore', drop='first'), ['ESTADO', 'EDAD']),
                        ('Ohe', OneHotEncoder(handle_unknown='ignore'), ['MARACIONES', 'CONTACTOS']),
                        ('Normal', Normalizer(), ['M1', 'C1'])],
                        remainder='passthrough')#.set_output(transform='pandas')
ct_out = ct.fit(X_train)
X_train_trans = ct_out.transform(X_train)

# Modelo de clasificación de Support Vector Machine (SVM)
n = 5
estimator = SVC(kernel='linear', coef0=1, probability=True)
selector_SVC = RFE(estimator, n_features_to_select=n)
selector_SVC.fit(X_train_trans, y_train)
X_train_trans_red = selector_SVC.transform(X_train_trans)

# Modelo de clasificación Regresión Logística con ajuste de desbalanceo
def ajuste_desbalance_LogReg(target):
    # Implementación de ajuste en caso de desbalanceo de clases para la implementación
    unique, counts = np.unique(target, return_counts=True)
    numero_clases = len(unique)
    val_TARGET_0 = unique[0]
    val_TARGET_1 = unique[1]
    cant_TARGET_0 = counts[0]
    cant_TARGET_1 = counts[1]
    total = cant_TARGET_0 + cant_TARGET_1
    peso_TARGET_0 = total / (numero_clases * cant_TARGET_0)
    peso_TARGET_1 = total / (numero_clases * cant_TARGET_1)
    return {val_TARGET_0: peso_TARGET_0, val_TARGET_1: peso_TARGET_1}
LR_balance = LogisticRegression(class_weight = ajuste_desbalance_LogReg(y_train)).fit(X_train_trans_red, y_train)

# implementación del modelo completo
```

```

pipeline = Pipeline([
    ('Transformacion_var', ct_out),
    ('Reduccion_var', selector_SVC),
    ('clasificacion', LR_balance)
])

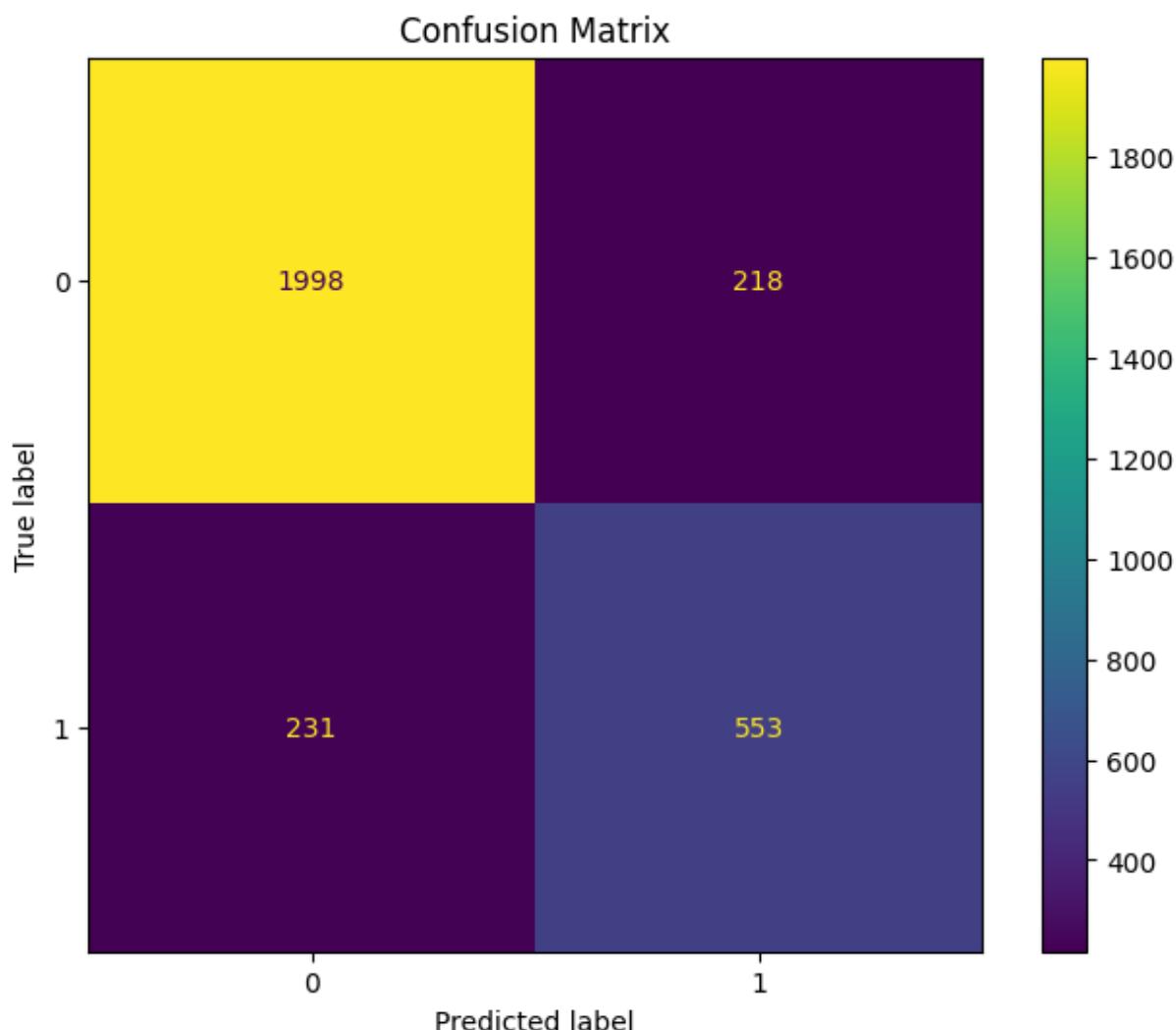
pipe = pipeline.fit(X_train, y_train)
y_pred = pipe.predict(X_test)
grafica_confusion_matrix(y_test, y_pred)

# Calcular métricas de evaluación del modelo
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

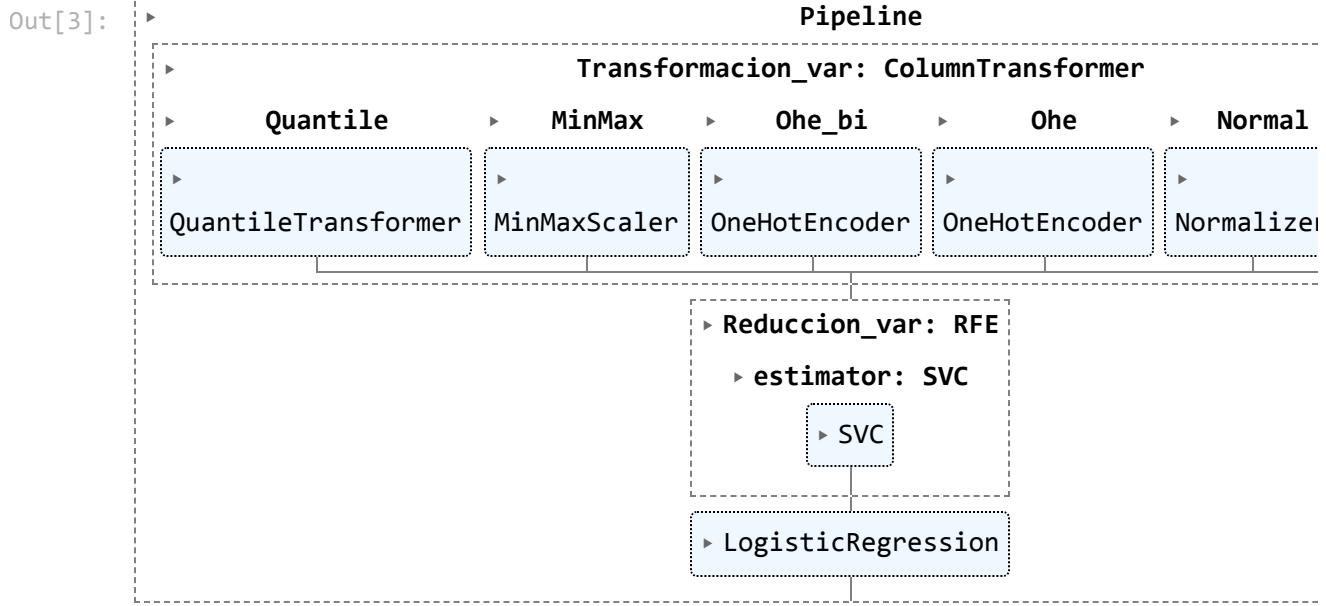
# Probabilidad de clase positiva
y_pred_prob = pipe.predict_proba(X_test)[:, 1]
auc_roc = roc_auc_score(y_test, y_pred_prob)

df_modelos_row = ['LogisticRegression', accuracy, precision, recall, f1, auc_roc, pipe]
df_modelos = pd.concat([df_modelos, pd.DataFrame([df_modelos_row]), columns=df_modelos.columns])

```



```
In [3]: #Descripcion del modelo completo
pipe
```



```
In [4]: # Otra forma de obtener las métricas de precisión, recall y F1-score
report = classification_report(y_test, y_pred)
print("Classification Report:")
print(report)
```

Classification Report:				
	precision	recall	f1-score	support
0	0.90	0.90	0.90	2216
1	0.72	0.71	0.71	784
accuracy			0.85	3000
macro avg	0.81	0.80	0.81	3000
weighted avg	0.85	0.85	0.85	3000

```
In [8]: # Calcular el valor AUC-ROC
y_pred_prob = pipe.predict_proba(X_test)[:, 1] # Probabilidad de clase positiva
auc_roc = roc_auc_score(y_test, y_pred_prob)
print("AUC-ROC Score:", auc_roc)
```

AUC-ROC Score: 0.9067381013777353

```
In [6]: df_modelos
```

```
Out[6]: Nombre_modelo  accuracy_score  precision_score  recall_score  f1_score  roc_auc_score
0  LogisticRegression  0.850333      0.71725       0.705357   0.711254   0.906738
```

12.2 Clasificador K Vecinos

Una segunda opción a considerar es el clasificador K Vecinos (KNN) debido a las siguientes características presentes en los datos:

- Tamaño de muestra suficiente: KNN funciona mejor con un tamaño de muestra grande, ya que se basa en la cercanía de los vecinos. Un tamaño de muestra pequeño puede no representar adecuadamente los patrones en los datos.
- Estructura de datos no lineal: KNN es especialmente útil para datos con estructura no lineal o que no siguen una distribución específica. Es capaz de capturar patrones complejos sin hacer suposiciones sobre la forma de los datos.
- Datos sin sesgo: KNN tiene un mejor desempeño cuando los datos no presentan un sesgo significativo hacia una clase en particular. En caso de existir sesgo, podrían requerirse técnicas adicionales para equilibrar los datos, aunque en este caso no hay un sesgo pronunciado debido a que la data no está balanceada.

Las ventajas de utilizar KNN son: simplicidad, ya que no requiere suposiciones sobre la distribución de los datos ni ajustes de parámetros complejos; versatilidad, ya que puede ser aplicado tanto a problemas de clasificación como de regresión, y puede manejar diferentes tipos de datos; e interpretabilidad, dado que las predicciones de KNN se basan en los vecinos más cercanos, lo que permite una explicación lógica e intuitiva de las decisiones del modelo.

Por otro lado, las desventajas de KNN son: sensibilidad a la escala de las características, lo cual puede afectar su desempeño si las características tienen escalas diferentes; coste computacional, ya que el tiempo de ejecución aumenta con el tamaño del conjunto de datos, especialmente al buscar los vecinos más cercanos en grandes conjuntos de datos, por lo que se recomienda trabajar con muestras; y sensibilidad al ruido y a los valores atípicos, por lo que se debe realizar un buen proceso de ETL e ingeniería de características.

Además, la elección adecuada del parámetro K es crucial para el rendimiento de KNN, ya que un valor incorrecto puede generar sesgos o varianzas subóptimas en el modelo. Para abordar esta cuestión, se puede emplear la técnica de GridSearchCV.

En el código siguiente se realiza un preprocesamiento de los datos y se ajusta un pipeline que incluye el proceso de binning, análisis de componentes principales y el clasificador KNN. Se utilizan los datos de prueba para hacer predicciones y se visualiza una matriz de confusión. Además, se imprime un informe de clasificación que muestra las métricas de precisión, recall y F1-score, y se calcula el área bajo la curva ROC (AUC-ROC) para evaluar el rendimiento del modelo.

```
In [9]: preprocesso = preprocesamiento(revision)
preprocesso.df = preprocesso.df.sample(10000)
```

```
preproceso.crea_EDAD_NORM('EDAD')

X = preproceso.df.drop(columns=['TARGET'])
y = preproceso.df['TARGET'].to_numpy()

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_stan

pipes_bining = []
for columna in X_train.columns:
    pipes_bining.append((columna + '_Tr', Pipeline([('columna', BinningProcess(vari
preprocessor = ColumnTransformer(pipes_bining)
#preprocessor.fit(X_train, y_train)
# X_train_transf = preprocessor.transform(X_train)

pipe = Pipeline((
    ("BinningProcess", preprocessor),
    ("Principal_component_analysis", PCA(n_components=5)),
    ("KNeighborsClassifier", KNeighborsClassifier(n_neighbors=5)))))

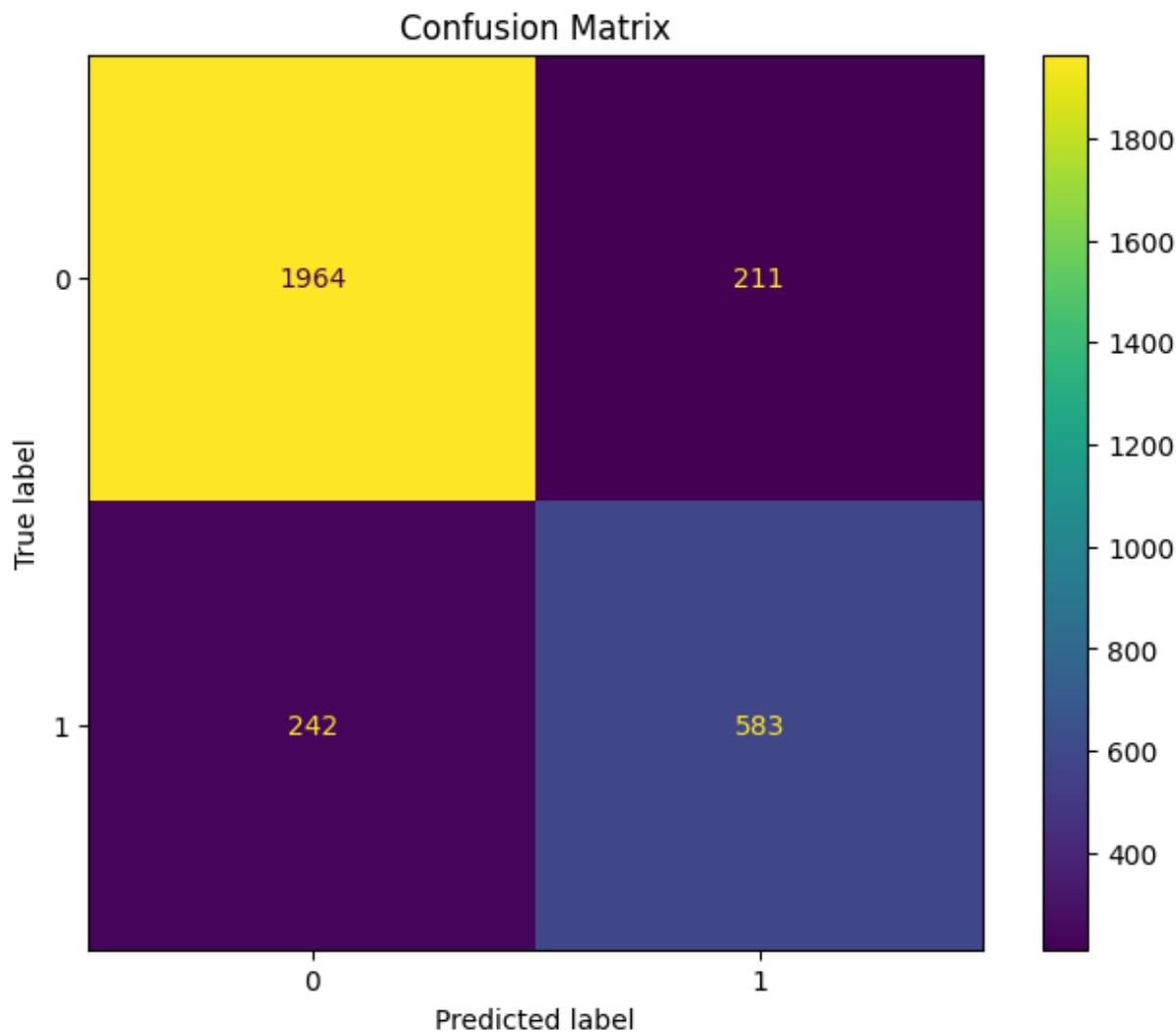
pipe.fit(X_train, y_train)
y_pred = pipe.predict(X_test)

grafica_confusion_matrix(y_test, y_pred)

# Calcular métricas de evaluación del modelo
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

# Probabilidad de clase positiva
y_pred_prob = pipe.predict_proba(X_test)[:, 1]
auc_roc = roc_auc_score(y_test, y_pred_prob)

df_modelos_row = ['KNeighborsClassifier', accuracy, precision, recall, f1, auc_roc, pipe]
df_modelos = pd.concat([df_modelos, pd.DataFrame([df_modelos_row], columns=df_modelos.columns)], ignore_index=True)
```



```
In [12]: df_modelos.to_csv("Modelos.csv", index=False)  
df_modelos
```

```
Out[12]:
```

	Nombre_modelo	accuracy_score	precision_score	recall_score	f1_score	roc_auc_score
0	LogisticRegression	0.850333	0.717250	0.705357	0.711254	0.906738
1	KNeighborsClassifier	0.849000	0.734257	0.706667	0.720198	0.878683

12.3 RandomForestClassifier

RandomForestClassifier es un algoritmo de aprendizaje supervisado utilizado para problemas de clasificación. Se basa en el concepto de un conjunto de árboles de decisión llamados "bosque" (forest). Cada árbol individual en el bosque se construye utilizando un subconjunto aleatorio de datos de entrenamiento y características seleccionadas aleatoriamente. La predicción final se realiza mediante la combinación de las predicciones de

cada árbol individual, utilizando votación para problemas de clasificación (mayoría) y promediando para problemas de regresión.

El funcionamiento de RandomForestClassifier implica los siguientes pasos:

1. Muestreo de datos: Se selecciona aleatoriamente un subconjunto de los datos de entrenamiento, conocido como bootstrap sample. Esto permite que cada árbol tenga datos de entrenamiento diferentes y evita el sobreajuste.
2. Selección de características: En cada división de un árbol, solo se considera un subconjunto aleatorio de características. Esto asegura diversidad y reduce la correlación entre los árboles.
3. Construcción del árbol: Se construye un árbol de decisión utilizando el subconjunto de datos y características seleccionados. Cada árbol se construye dividiendo los datos en función de las características y los umbrales que maximizan la pureza de la división.
4. Predicción: Para realizar una predicción, se propagan los datos a través de todos los árboles del bosque y se obtiene una votación o promedio de las predicciones individuales de cada árbol.

Ventajas de RandomForestClassifier:

- Precisión: RandomForestClassifier tiene una alta precisión en la mayoría de los conjuntos de datos. Al combinar las predicciones de múltiples árboles, se reduce el sesgo y la varianza, lo que lleva a una mayor precisión general.
- Resistencia al sobreajuste: El muestreo aleatorio y la selección de características aleatorias ayudan a reducir el sobreajuste al evitar la dependencia en un solo árbol.
- Robustez: RandomForestClassifier es robusto frente a datos faltantes y valores atípicos, ya que las predicciones se basan en la votación o promedio de varios árboles en lugar de depender de un solo árbol.
- Importancia de características: RandomForestClassifier proporciona una medida de la importancia de las características, lo que permite identificar las características más relevantes para la clasificación.

Desventajas de RandomForestClassifier:

- Interpretación: La interpretación de un modelo de Random Forest puede ser más compleja que un solo árbol de decisión debido a la combinación de múltiples árboles.
- Tiempo de entrenamiento: El entrenamiento de un Random Forest puede llevar más tiempo en comparación con otros algoritmos de clasificación, especialmente si el número de árboles y características es grande.

- Memoria requerida: RandomForestClassifier puede requerir una cantidad significativa de memoria, especialmente si se utiliza un gran número de recursos computacionales. Implementa un proceso de ajuste de un modelo de clasificación utilizando el algoritmo SGDClassifier de scikit-learn. Utiliza técnicas como muestreo aleatorio, preprocessamiento, validación cruzada y búsqueda de hiperparámetros para encontrar el mejor modelo. Al final, se selecciona el modelo con el mejor rendimiento en términos de f1-score para su uso posterior.a precisión, recall y F1-score. También genera una matriz de confusión y calcula el árs al utilizar este algoritmo.

```
In [45]: from sklearn.ensemble import RandomForestClassifier
preproceso = preprocessamiento(revision)
preproceso.df = preproceso.df.sample(10000)
preproceso.crea_EDAD_NORM('EDAD')

X = preproceso.df.drop(columns=['TARGET'])
y = preproceso.df['TARGET'].to_numpy()

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Se utiliza ColumnTransformer para ajustar el modelo incluyendo las transformaciones
pipes_bining = []
for columna in X_train.columns:
    pipes_bining.append((columna + '_Tr', Pipeline([('columna', BinningProcess(variant=ColumnTransformer(pipes_bining)))])))

# implementación del modelo completo
pipeline = Pipeline([
    ('Transformacion_var', ct),
    ("Principal_component_analysis", PCA(n_components=5)),
    ('RandomForestClassifier', RandomForestClassifier())
])
#pipe = pipeline.fit(X_train, y_train)

##### parámetros para Grid Random Forest:
# numero de arboles
n_estimators = [int(x) for x in np.linspace(5,15,5)]

# minimo numero de muestras para dividir un nodo
min_samples_split = [2,5]

# minimo numero de muestras para cada hoja del nodo
min_samples_leaf = [1,2]

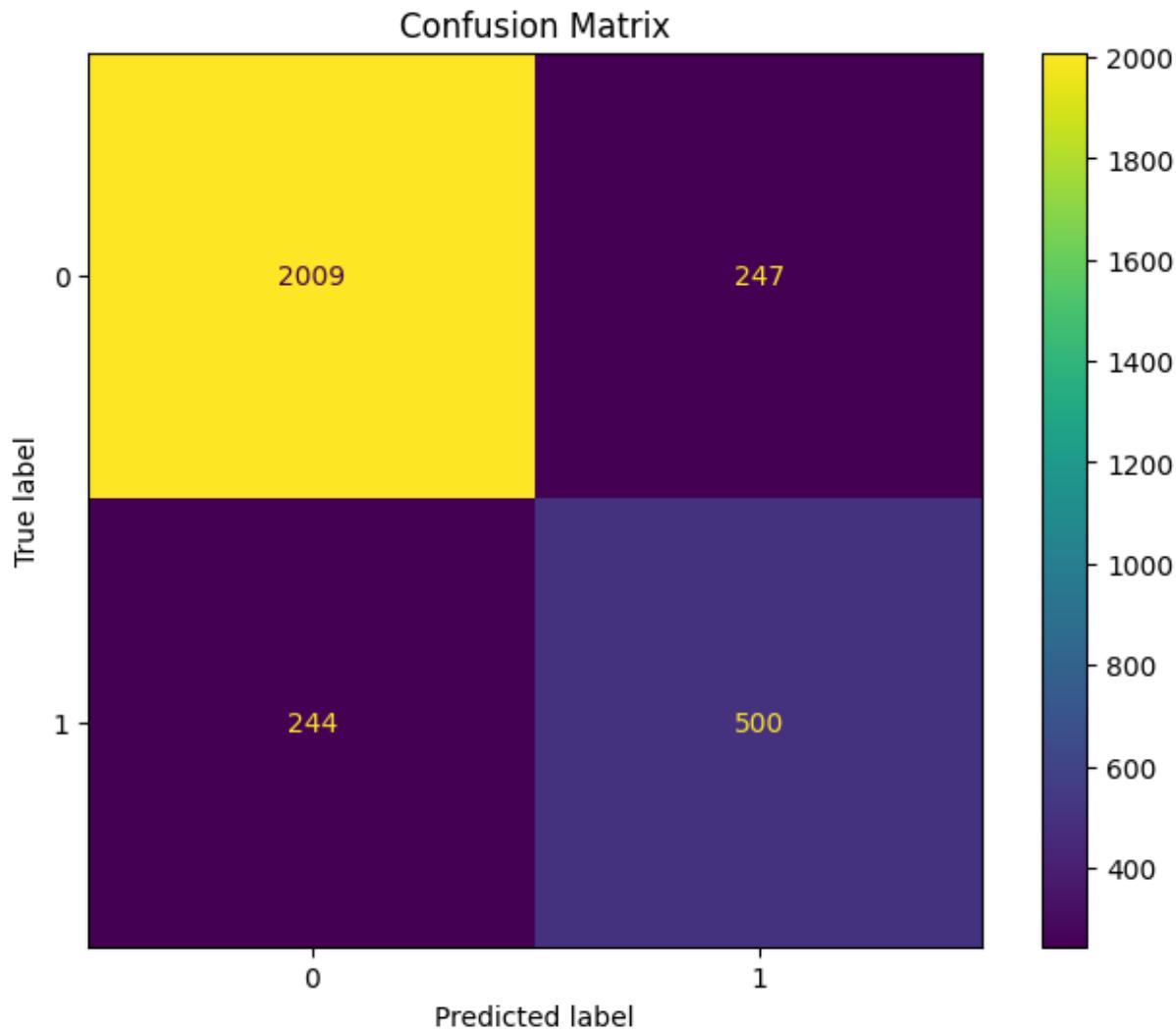
param_grid = {'RandomForestClassifier__n_estimators':n_estimators,
              'RandomForestClassifier__min_samples_split': min_samples_split,
              'RandomForestClassifier__min_samples_leaf':min_samples_leaf }

"""

El F1-score es una métrica comúnmente utilizada en problemas de clasificación, especialmente cuando los datos están desbalanceados. El F1-score combina las métricas de precisión (precision) y recall (sensibilidad) en un solo valor, lo que permite evaluar el equilibrio entre la precisión y la capacidad de

```

```
recuperación del modelo. Al utilizar el F1-score como scoring en GridSearchCV,  
se busca encontrar el modelo que logre el mejor equilibrio entre la precisión y el  
"""  
final_grid = GridSearchCV(estimator = pipeline,  
                         param_grid = param_grid,  
                         scoring = {  
                             'f1_score': make_scorer(f1_score)  
                         },  
                         refit = 'f1_score',  
                         return_train_score = True,  
                         cv = 5)  
  
final_grid.fit(X_train,y_train)  
  
# Obtener Los mejores hiperparámetros y el mejor modelo  
best_params = final_grid.best_params_  
  
best_model = final_grid.best_estimator_  
y_pred= best_model.predict(X_test)  
  
grafica_confusion_matrix(y_test, y_pred)  
  
# Calcular métricas de evaluación del modelo  
accuracy = accuracy_score(y_test, y_pred)  
precision = precision_score(y_test, y_pred)  
recall = recall_score(y_test, y_pred)  
f1 = f1_score(y_test, y_pred)  
  
# Probabilidad de clase positiva  
#y_pred_prob = best_model.predict_proba(X_test)[:, 1]  
#auc_roc = roc_auc_score(y_test, y_pred_prob)  
auc_roc = roc_auc_score(y_test, y_pred)  
  
df_modelos_row = [' RandomForestClassifier',accuracy,precision,recall,f1,auc_roc,be  
df_modelos = pd.concat([df_modelos, pd.DataFrame([df_modelos_row]), columns=df_model
```



```
In [49]: resultados = pd.DataFrame(final_grid.cv_results_)
print(best_params)
resultados[resultados['rank_test_f1_score'] < 5]
```

```
{'RandomForestClassifier__min_samples_leaf': 2, 'RandomForestClassifier__min_samples_
_split': 5, 'RandomForestClassifier__n_estimators': 12}
```

```
Out[49]:      mean_fit_time  std_fit_time  mean_score_time  std_score_time  param_RandomForestClass
```

9	1.267929	0.027782	0.032422	0.000491
17	1.167402	0.040042	0.029820	0.000400
18	1.210245	0.030726	0.031422	0.001498
19	1.257419	0.029147	0.032222	0.000979

4 rows × 23 columns

In [50]: df_modelos

	Nombre.modelo	accuracy_score	precision_score	recall_score	f1_score	roc_auc_soc
0	LogisticRegression	0.850333	0.717250	0.705357	0.711254	0.9067
1	KNeighborsClassifier	0.849000	0.734257	0.706667	0.720198	0.8786
2	RandomForestClassifier	0.836333	0.669344	0.672043	0.670691	0.8832

12.4 SGDClassifier

`SGDClassifier` es una implementación del algoritmo de descenso de gradiente estocástico (SGD) para problemas de clasificación en scikit-learn. Es un clasificador lineal que utiliza el enfoque de optimización iterativa basado en el gradiente para ajustar los parámetros del modelo.

El funcionamiento básico de `SGDClassifier` se puede resumir en los siguientes pasos:

1. Preparación de los datos: Antes de entrenar el modelo, los datos de entrenamiento deben estar en un formato adecuado. Esto generalmente implica realizar la codificación de variables categóricas, escalar las características numéricas y dividir los datos en conjuntos de entrenamiento y prueba.
2. Inicialización de parámetros: Se inicializan los pesos y sesgos del modelo. Estos parámetros se actualizan durante el proceso de entrenamiento para minimizar la función de pérdida.
3. Selección de la función de pérdida: Se elige una función de pérdida apropiada para el problema de clasificación. Algunas opciones comunes incluyen la regresión logística (`log`), la pérdida de entropía cruzada (`log` con el parámetro `loss='log'`), la pérdida de bisagra (`hinge`) para máquinas de vectores de soporte lineales, entre otras.
4. Ciclo de entrenamiento: El modelo itera sobre los datos de entrenamiento en mini lotes (muestras seleccionadas aleatoriamente) y actualiza los parámetros en función del gradiente de la función de pérdida. Esta es la característica principal del SGD, ya que realiza actualizaciones de parámetros más frecuentes en lugar de utilizar todo el conjunto de datos en cada iteración.
5. Actualización de parámetros: Durante cada iteración, se calcula el gradiente de la función de pérdida con respecto a los parámetros del modelo y se actualizan los pesos y sesgos utilizando un tamaño de paso (también llamado tasa de aprendizaje). Esto se

repite hasta que se alcanza un criterio de convergencia predefinido, como el número máximo de iteraciones o una mejora mínima en la función de pérdida.

6. Predicción: Una vez entrenado el modelo, se pueden realizar predicciones en nuevos datos utilizando la función `predict`. El clasificador asigna una etiqueta a cada muestra en función de la función de decisión o probabilidad.

Es importante destacar que `SGDClassifier` es eficiente y escalable, ya que se puede utilizar en conjuntos de datos grandes debido a su enfoque de optimización estocástica y su capacidad para procesar datos en mini lotes. Además, permite el ajuste de hiperparámetros como la tasa de aprendizaje, la regularización L1 o L2 y la penalización elástica, lo que brinda flexibilidad en la configuración del modelo.

En resumen, `SGDClassifier` es un clasificador lineal eficiente y escalable que utiliza el algoritmo de descenso de gradiente estocástico para ajustar los parámetros del modelo. Es adecuado para problemas de clasificación binaria y multiclase y ofrece opciones de regularización y diferentes funciones de pérdida para adaptarse a diferentes escenarios de clasificación.

```
In [73]: # Definición de la función para construir el modelo SGDClassifier
def modelo_SGDClassifier():
    # Paso 1: Creación de transformaciones personalizadas para las variables
    pipes_bining = []
    for columna in X_train.columns:
        pipes_bining.append((columna + '_Tr', Pipeline([('columna', BinningProcess(
            ct_woe = ColumnTransformer(pipes_bining)

    # Paso 2: Selección de características utilizando el test chi2
    filtro = SelectKBest(chi2, k=5)

    # Paso 3: Implementación del modelo completo utilizando un pipeline
    pipeline = Pipeline([
        ('Transformacion_woe', ct_woe),
        ('Transformacion_0_1', MinMaxScaler()),
        ('Reduccion_var', filtro),
        ('classifier', SGDClassifier())
    ])

    # Paso 4: Definición de los posibles valores de hiperparámetros para GridSearch
    param_grid = {
        'classifier_loss': ['hinge', 'log_loss'], # Función de pérdida: Logística
        'classifier_alpha': [0.0001, 0.001, 0.01], # Tasa de regularización
        'classifier_class_weight': ['balanced', {0: 0.2, 1: 0.8}] # Ponderación a
    }

    # Paso 5: Creación del objeto GridSearchCV para ajustar los hiperparámetros
    grid_search = GridSearchCV(pipeline,
                               param_grid,
                               cv=5,
                               scoring='f1') # Utilización de F1-score como métric

    return grid_search
```

```

# Creación del dataframe para almacenar los resultados de evaluación del modelo
df_reporte = pd.DataFrame([], columns=['accuracy_score', 'precision_score', 'recall_score'])

# Creación de una lista para almacenar los resultados de cada repetición del modelo
resultado = []
repeticiones = 5 # Número de repeticiones del modelo
for _ in range(repeticiones):
    print('repetición:', _)
    # Paso 6: Preprocesamiento y división de los datos en conjuntos de entrenamiento y prueba
    preproceso = preprocesamiento(revision)
    preproceso.df = preproceso.df.sample(10000)
    preproceso.crea_EDAD()

    X = preproceso.df.drop(columns=['TARGET'])
    y = preproceso.df['TARGET'].to_numpy()
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3) #, random_state=42)

    # Paso 7: Construcción y ajuste del modelo final
    modelo_final = modelo_SGDClassifier()
    modelo_final.fit(X_train, y_train)

    # Paso 8: Obtención de los mejores hiperparámetros y el mejor modelo
    best_params = modelo_final.best_params_
    best_model = modelo_final.best_estimator_
    resultado.append((best_params, best_model))

    # Paso 9: Predicción en el conjunto de prueba y cálculo de métricas de evaluación
    y_pred = modelo_final.predict(X_test)

    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)
    auc_roc = roc_auc_score(y_test, y_pred)

    # Paso 10: Almacenamiento de las métricas en el dataframe de reporte
    df_reporte_row = [accuracy, precision, recall, f1, auc_roc]
    df_reporte = pd.concat([df_reporte, pd.DataFrame([df_reporte_row], columns=df_reporte.columns)], ignore_index=True)

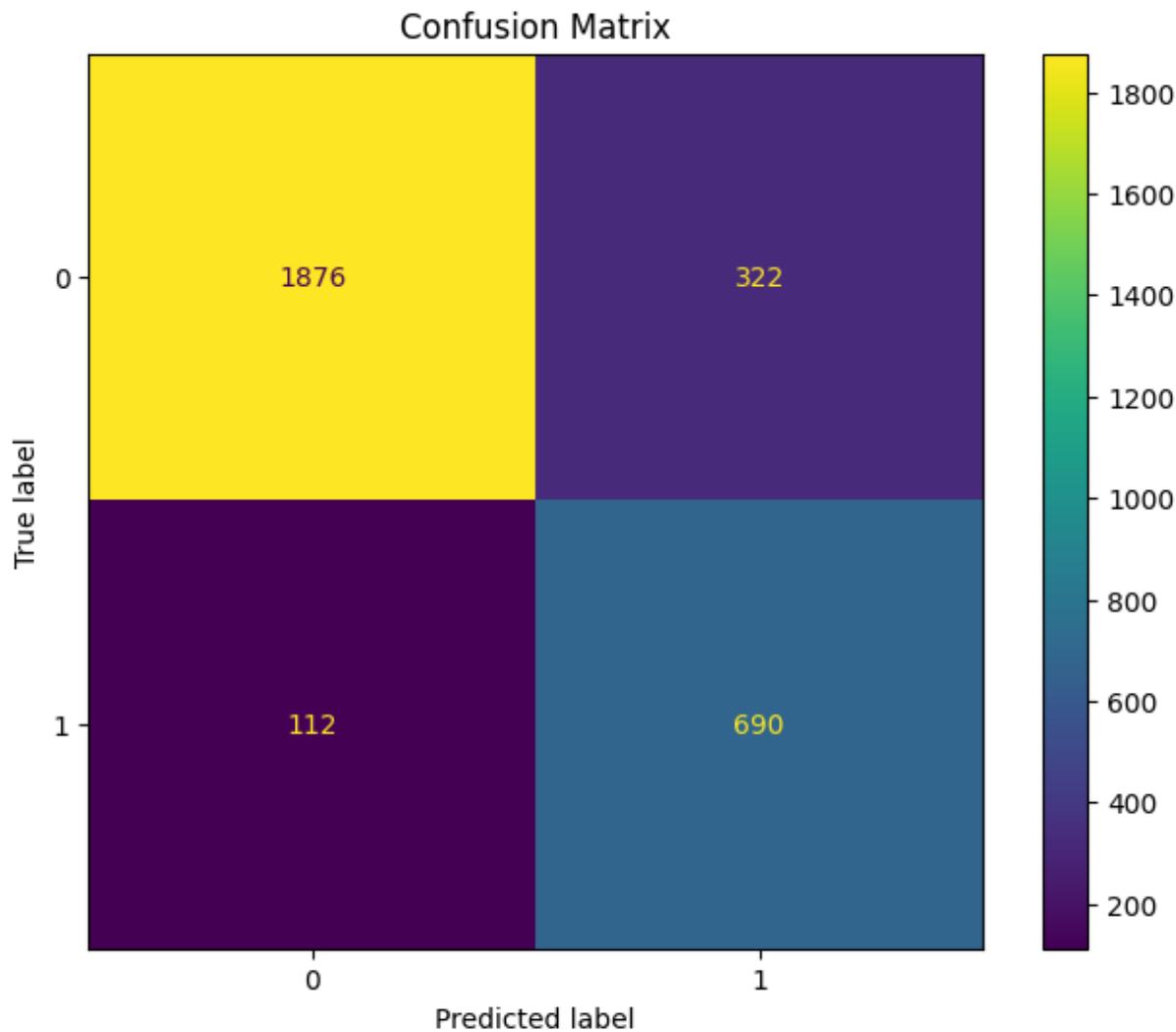
# Selección del modelo con el mayor valor de F1-score
modelo_SGDClassifier = resultado[df_reporte['f1_score'].idxmax()][1]
indice_selecc = df_reporte['f1_score'].idxmax()

y_pred = modelo_SGDClassifier.predict(X_test)
grafica_confusion_matrix(y_test, y_pred)

# Paso 11: Almacenamiento de las métricas para comparar con otros modelos
df_modelos_row = ['SGDClassifier',
                   df_reporte.loc[indice_selecc, 'accuracy_score'],
                   df_reporte.loc[indice_selecc, 'precision_score'],
                   df_reporte.loc[indice_selecc, 'recall_score'],
                   df_reporte.loc[indice_selecc, 'f1_score'],
                   df_reporte.loc[indice_selecc, 'roc_auc_score'],
                   modelo_SGDClassifier]
df_modelos = pd.concat([df_modelos, pd.DataFrame([df_modelos_row], columns=df_modelos.columns)], ignore_index=True)

```

```
repeticion: 0
repeticion: 1
repeticion: 2
repeticion: 3
repeticion: 4
```



In [121]: df_modelos

Out[121]:

	Nombre_modelo	accuracy_score	precision_score	recall_score	f1_score	roc_auc_soc
0	LogisticRegression	0.850333	0.717250	0.705357	0.711254	0.9067
1	KNeighborsClassifier	0.849000	0.734257	0.706667	0.720198	0.8786
2	RandomForestClassifier	0.836333	0.669344	0.672043	0.670691	0.8832
3	SGDClassifier	0.843000	0.660494	0.819923	0.731624	0.8355
4	modelo_compuesto	0.871667	0.774069	0.742611	0.758014	0.8310

13 Métricas para evaluar algoritmos de clasificación

Las métricas de precisión, recall y F1-score se utilizan ampliamente para evaluar algoritmos de clasificación, como regresión logística, KNN, SVM, Naive Bayes, árboles de decisión y bosques aleatorios. Estas métricas proporcionan una evaluación general del rendimiento del modelo en términos de su precisión, capacidad para capturar instancias positivas y una combinación de precisión y recall.

- **Precisión (precision_score):** Es útil para medir la proporción de verdaderos positivos entre todas las predicciones positivas. Esta métrica es especialmente importante en casos en los que los falsos positivos tienen un alto costo.
- **Recall (recall_score):** Mide la proporción de verdaderos positivos que se capturan correctamente entre todas las instancias positivas reales. Esta métrica es importante cuando el objetivo es minimizar los falsos negativos.
- **F1-score (f1_score):** Es una métrica que combina la precisión y el recall en un solo valor, lo que la convierte en una medida general del rendimiento del modelo.
- **AUC-ROC (Area Under the Receiver Operating Characteristic Curve):** Es una métrica que evalúa la capacidad del modelo para distinguir entre las clases positiva y negativa. Representa la probabilidad de que el modelo clasifique correctamente una instancia aleatoria de la clase positiva como más probable que una instancia aleatoria de la clase negativa. Un valor de AUC-ROC cercano a 1 indica un mejor rendimiento del modelo en la clasificación.

Es importante destacar que los modelos se pueden ajustar para mejorar una métrica específica, pero esto puede ser a expensas de otra métrica. Esto no necesariamente es algo malo. Por ejemplo, si queremos predecir la existencia de una enfermedad, es más importante tener un alto valor en recall_score, ya que un falso negativo implicaría una persona con la enfermedad a la que se podría no darle seguimiento. Sin embargo, un falso positivo indicaría una persona sin la enfermedad que se le diagnostica erróneamente. En el peor de los casos, esa persona pasaría por una situación incómoda, pero lo más probable es que una repetición de las pruebas o un estudio más profundo revele que no tiene la enfermedad.

En nuestro caso, se plantea un equilibrio entre los falsos positivos y los falsos negativos, por lo que los modelos se ajustaron utilizando la métrica F1-score.

In [78]: df_modelos

Out[78]:

	Nombre_modelo	accuracy_score	precision_score	recall_score	f1_score	roc_auc_soc
0	LogisticRegression	0.850333	0.717250	0.705357	0.711254	0.9067
1	KNeighborsClassifier	0.849000	0.734257	0.706667	0.720198	0.8786
2	RandomForestClassifier	0.836333	0.669344	0.672043	0.670691	0.8832
3	SGDClassifier	0.843000	0.660494	0.819923	0.731624	0.8355

14. Visualización de la implementación del modelo ganador

Dado que los resultados de los modelos son muy parecidos se implementará una solución alternativa utilizando `VotingClassifier` de Scikit-learn, que es un clasificador de votación que combina múltiples clasificadores individuales para tomar una decisión conjunta. Cada clasificador individual puede tener su propio conjunto de parámetros y configuraciones.

El funcionamiento del `VotingClassifier` se basa en el principio de votación mayoritaria, donde cada clasificador individual emite su propia predicción y la clase final se determina por mayoría de votos. Hay dos tipos de votación compatibles:

1. Votación dura (hard voting): El `VotingClassifier` selecciona la clase con la mayoría de votos de los clasificadores individuales.
2. Votación suave (soft voting): El `VotingClassifier` asigna las probabilidades de clase promediadas de los clasificadores individuales y selecciona la clase con la probabilidad más alta.

Las ventajas de utilizar el `VotingClassifier` son:

1. Mejora de la precisión: Al combinar varios clasificadores, el `VotingClassifier` puede aprovechar la diversidad de los clasificadores individuales y mejorar la precisión de las predicciones.
2. Reducción de la variabilidad: La votación de múltiples clasificadores reduce la variabilidad inherente a un solo clasificador y puede generar predicciones más estables y confiables.
3. Flexibilidad: El `VotingClassifier` permite combinar diferentes tipos de clasificadores y aprovechar sus fortalezas individuales.

Sin embargo, también hay algunas limitaciones a considerar:

1. Clasificadores correlacionados: Si los clasificadores individuales tienen altas correlaciones entre sí, la diversidad y el rendimiento del `VotingClassifier` pueden verse comprometidos.
2. Clasificadores inadecuados: Si los clasificadores individuales tienen un rendimiento deficiente o están mal calibrados, el `VotingClassifier` puede verse afectado negativamente.

Algunos de los parámetros con los que se puede interactuar en el `VotingClassifier` son:

- `estimators` : Una lista de clasificadores individuales que se utilizarán en la votación.
- `voting` : El tipo de votación a utilizar, que puede ser "hard" (votación dura) o "soft" (votación suave).
- `weights` : Pesos opcionales asignados a los clasificadores individuales en la votación. Los clasificadores con pesos más altos tienen más influencia en la decisión final.
- `n_jobs` : El número de trabajos en paralelo para realizar durante el a

El siguiente código utiliza un par de modelos ya desarrollados para crear el modelo final. Problema de clasificación.PDF.

```
In [ ]: #Se pueden seleccionar los modelos que se deseen
modelos = [(fila['Nombre_modelo'], fila['modelo']) for _, fila in df_modelos.iterrows()]

#Pudiendo asignar un peso en función de la métrica que convenga o nuevamente haciendo
metricas = df_modelos[['accuracy_score', 'precision_score', 'recall_score', 'f1_score']]
pesos = list(metricas.sum(axis=1)/metricas.sum(axis=1).mean())

modelo_completo = VotingClassifier(estimators=modelos, voting='hard', weights=pesos)
```

```
In [118...]: #Aunque lo ideal sería trabajar con un mínimo de modelos
df = df_modelos[df_modelos['Nombre_modelo'].isin(['LogisticRegression', 'KNeighborsClassifier'])]
modelos = [(fila['Nombre_modelo'], fila['modelo']) for _, fila in df.iterrows()]

metricas = df[['accuracy_score', 'precision_score', 'recall_score', 'f1_score', 'roc_auc']]
pesos = list(metricas.sum(axis=1)/metricas.sum(axis=1).mean())

modelo_completo = VotingClassifier(estimators=modelos, voting='soft', weights=pesos)
```

```
In [119...]: # Paso 1: Preprocesamiento de datos
# Creación de una instancia de preprocesamiento
preproceso = preprocesamiento(revision)

# Muestreo aleatorio de 10,000 filas del conjunto de datos
preproceso.df = preproceso.df.sample(10000)

# Creación de una variable EDAD normalizada utilizando la función crea_EDAD_NORM
preproceso.crea_EDAD_NORM('EDAD')

# Paso 2: División de datos en conjuntos de entrenamiento y prueba
# Seleccionar las columnas predictoras (X) y el target (y)
X = preproceso.df.drop(columns=['TARGET'])
y = preproceso.df['TARGET'].to_numpy()
```

```
# Dividir los datos en conjuntos de entrenamiento y prueba utilizando train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Paso 3: Ajuste del modelo compuesto
# Entrenar el modelo compuesto utilizando los datos de entrenamiento
modelo_compuesto.fit(X_train, y_train)

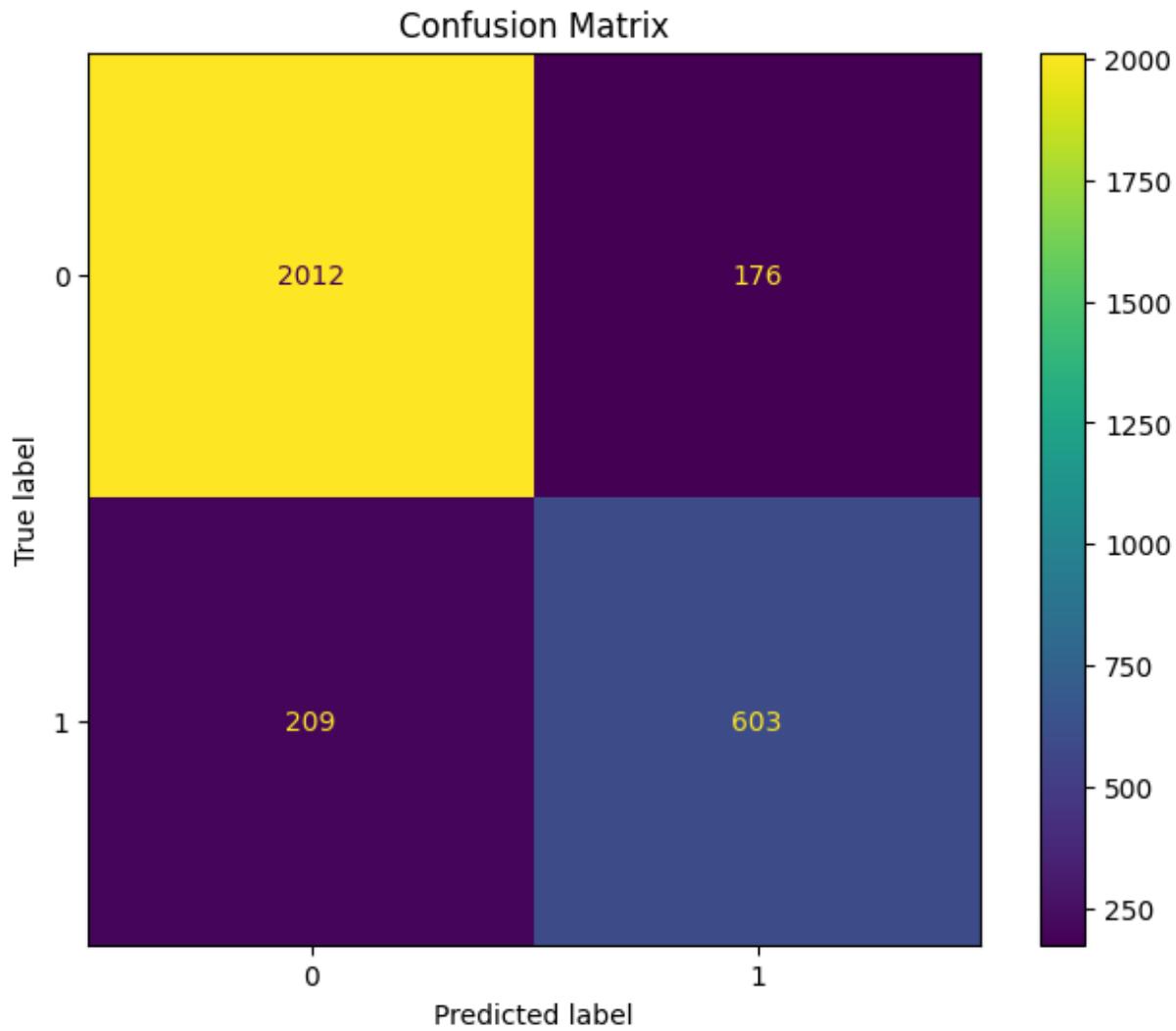
# Paso 4: Predicción y evaluación del modelo
# Realizar predicciones en el conjunto de prueba
y_pred = modelo_compuesto.predict(X_test)

# Graficar la matriz de confusión
grafica_confusion_matrix(y_test, y_pred)

# Calcular métricas de evaluación del modelo: exactitud, precisión, recall, F1-score
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

# Calcular el AUC-ROC Score
auc_roc = roc_auc_score(y_test, y_pred)

# Guardar los resultados del modelo en un DataFrame
df_modelos_row = ['modelo_compuesto', accuracy, precision, recall, f1, auc_roc, modelo_compuesto]
df_modelos = pd.concat([df_modelos, pd.DataFrame([df_modelos_row]), columns=df_modelos.columns], ignore_index=True)
```



```
In [122]: df_modelos.to_csv("Modelos.csv", index=False)
df_modelos
```

```
Out[122]:
```

	Nombre_modelo	accuracy_score	precision_score	recall_score	f1_score	roc_auc_soc
0	LogisticRegression	0.850333	0.717250	0.705357	0.711254	0.9067
1	KNeighborsClassifier	0.849000	0.734257	0.706667	0.720198	0.8786
2	RandomForestClassifier	0.836333	0.669344	0.672043	0.670691	0.8832
3	SGDClassifier	0.843000	0.660494	0.819923	0.731624	0.8355
4	modelo_compuesto	0.871667	0.774069	0.742611	0.758014	0.8310

```
In [129]: # X.analisis_nulos(self)
revision_nueva.analisis_nulos()
```

Out[129]:

	cant_nulos	porcentaje_nulos
INGRESO	179892	1.94
MORAS	2815	0.03
SEXO	2815	0.03
ESTADOCIVIL	2815	0.03
FECHANACIMIENTO	2815	0.03
ANTIGUEDAD	2815	0.03
EDAD	2815	0.03

In [138...]

```
#La implementacion final para cualquier set de nueva data seria:

#carga y procesamiento del archivo .csv con la nueva data...
revision = revisor_data_csv("../\\datos_internos/20210513_Challenge_AA_MANANA.csv")

# Eliminacion de valores nulos
revision.nulos_EliminacionSimple(['MORAS', 'SEXO', 'ESTADOCIVIL', 'FECHANACIMIENTO'])
revision.nulos_MediaCondicional('INGRESO',[ 'ESTADO','EDAD','ESTADOCIVIL','SEXO'])
revision.nulos_EliminacionSimple(['INGRESO'])
revision.df['MORAS'] = revision.df['MORAS'].astype(int)
revision.df['ANTIGUEDAD'] = revision.df['ANTIGUEDAD'].astype(int)
revision.df['EDAD'] = revision.df['EDAD'].astype(int)
revision.df['FECHANACIMIENTO'] = pd.to_datetime(revision.df['FECHANACIMIENTO']).dt.
revision.df['FECHANACIMIENTO'] = pd.to_datetime(revision.df['FECHANACIMIENTO'])
revision.eliminar_por_condicion('ESTADOCIVIL', ' ')
revision.df = revision.df.drop(['CLIENTE'], axis=1)

preproceso = preprocesamiento(revision)

# Suponiendo que se carga 100.000 datos nuevos:
preproceso.df = preproceso.df.sample(100000)

# Creación de una variable EDAD normalizada utilizando la función crea_EDAD_NORM
preproceso.crea_EDAD_NORM('EDAD')

# Paso 2: División de datos en conjuntos de entrenamiento y prueba
# Seleccionar las columnas predictoras (X) y el target (y)
X = preproceso.df.drop(columns=['TARGET'])
y = preproceso.df['TARGET'].to_numpy()

modelo_final = df_modelos.loc[4,'modelo']

# Realizar predicciones
y_pred = modelo_compuesto.predict(X)

#Como tenemos la data y podemos evaluar
#gaficar la matriz de confusión
grafica_confusion_matrix(y,y_pred)
```

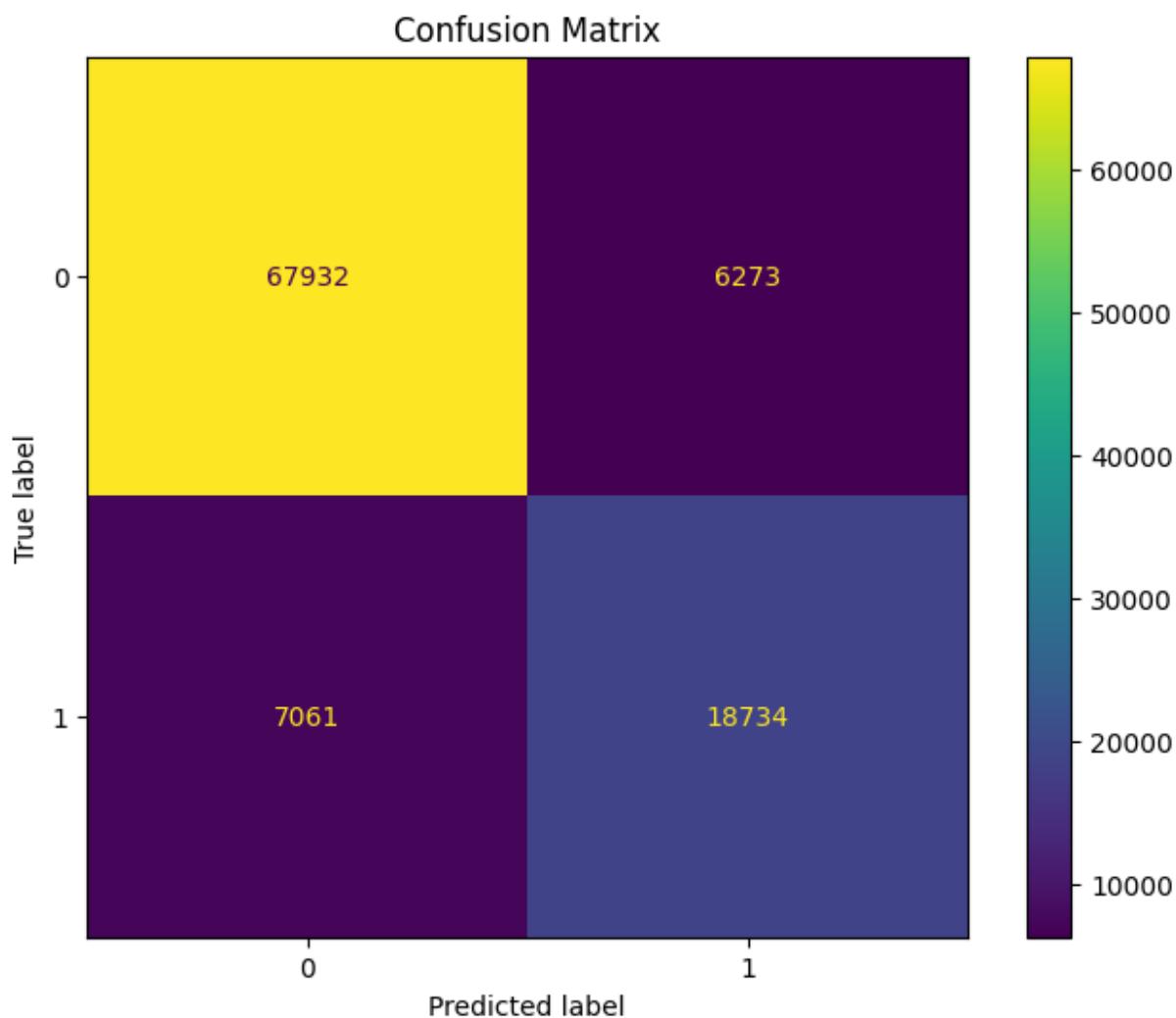
```
# Calcular métricas de evaluación del modelo: exactitud, precisión, recall, F1-score
accuracy = accuracy_score(y, y_pred)
precision = precision_score(y, y_pred)
recall = recall_score(y, y_pred)
f1 = f1_score(y, y_pred)

# Calcular el AUC-ROC Score
auc_roc = roc_auc_score(y, y_pred)

df_metricas_prediccion = pd.DataFrame([[accuracy, precision, recall, f1, auc_roc]]),
df_metricas_prediccion
```

Out[138]:

	accuracy_score	precision_score	recall_score	f1_score	roc_auc_score
0	0.86666	0.74915	0.726265	0.73753	0.820864



In [1]: #Resumen acciones

```
import os # Librería para interactuar con el sistema operativo
import datetime # Librería para trabajar con fechas y horas
from IPython.display import clear_output # Librería para limpiar la salida en el notebook
import pandas as pd # Librería para manipulación y análisis de datos en formato tabular
import numpy as np # Librería para operaciones numéricas eficientes
```

```

import seaborn as sns # Librería para visualización de datos estadísticos
import matplotlib.pyplot as plt # Librería para visualización de gráficos
from scipy.stats import skew # Librería para cálculo de sesgo en distribuciones
from scipy.stats import kurtosis # Librería para cálculo de curtosis en distribuciones
import statsmodels.api as sm # Librería para modelado estadístico y análisis de regresión
from optbinning import OptimalBinning # Librería para el binning óptimo de variables
from optbinning import BinningProcess # Librería para el proceso de binning
import sklearn # Librería para aprendizaje automático (machine learning)
from sklearn import preprocessing # Submódulo de sklearn para preprocesamiento de datos
from sklearn import linear_model # Submódulo de sklearn para modelos Lineales
from sklearn import model_selection # Submódulo de sklearn para selección de modelos
from sklearn.metrics import classification_report # Función para generar informe de clasificación
from sklearn.metrics import confusion_matrix # Función para generar matriz de confusión
from sklearn.metrics import accuracy_score # Función para calcular la precisión de los modelos
from sklearn.model_selection import train_test_split # Función para dividir datos
from sklearn.preprocessing import OneHotEncoder # Codificación one-hot para variables categóricas
from sklearn.feature_selection import SelectKBest # Selección de características basada en la variancia
from sklearn.preprocessing import StandardScaler # Estandarización de variables numéricas
from sklearn.feature_selection import SelectFromModel # Selección de características basada en el modelo
from sklearn.svm import SVC # Máquina de vectores de soporte (SVM)
from sklearn.compose import ColumnTransformer # Transformador de columnas para el pipeline
from sklearn.preprocessing import QuantileTransformer # Transformación cuantílica
from sklearn.preprocessing import MinMaxScaler # Escalado de variables al rango [0, 1]
from sklearn.preprocessing import Normalizer # Normalización de variables
from sklearn.pipeline import Pipeline # Construcción de pipelines de procesamiento
from sklearn.pipeline import make_pipeline # Construcción de pipelines de procesamiento
from sklearn.feature_selection import RFE # Eliminación recursiva de características
from sklearn.neighbors import KNeighborsClassifier # Clasificador k-NN
from sklearn.linear_model import LogisticRegression # Regresión logística
from sklearn.metrics import ConfusionMatrixDisplay # Visualización de matriz de confusión
from sklearn.decomposition import PCA # Análisis de componentes principales (PCA)
from sklearn.model_selection import GridSearchCV # Búsqueda de hiperparámetros con validación cruzada
from sklearn.metrics import precision_score, recall_score, f1_score, make_scorer # Funciones para medir el rendimiento
from sklearn.ensemble import RandomForestClassifier # Clasificador de bosques aleatorios
from sklearn.ensemble import VotingClassifier # Implementación de clasificador por votación
from sklearn.linear_model import SGDClassifier # Implementación del clasificador SGD
from sklearn.preprocessing import FunctionTransformer # aplicar transformaciones personalizadas
# from sklearn.datasets import make_classification

```

```
print("librerias cargadas correctamente")
```

```

class revisor_data_csv:
    """Clase para carga y revisión de archivos con formato .csv
    Con esto podemos visualizar y manipular la data de forma controlada minimizando errores
    pudiendo posteriormente crear un pipeline del proceso ETL"""
    def __init__(self, file_path):
        """
        Constructor de la clase que recibe la ruta del archivo y carga el archivo
        Se asume archivos .csv estándar, sep = ","
        """
        self.file_path = file_path
    try:
        # Se carga el df

```

```

        self.mensaje("cargando", self.file_path)
        self.df = pd.read_csv(self.file_path)
        self.mensaje("cargado")
    except Exception as error:
        print("Ocurrió un error:", error)

    def mensaje(self, tipo, arg=None):
        tipos_mensajes = {"cargando": f"Cargando archivo {arg}",
                           "cargado": "Carga completa"}
        if tipo in tipos_mensajes:
            clear_output(wait=True)
            print(tipos_mensajes[tipo])
        else:
            clear_output(wait=True)
            print("Tipo de mensaje no válido")
            print(f"{list(tipos_mensajes.keys())}")

    def analisis_nulos(self):
        cuenta_nulos = self.df.isnull().sum()
        cuenta_nulos = cuenta_nulos[cuenta_nulos != 0]
        porcentaje_nulos = round((cuenta_nulos / self.df.shape[0]) * 100, 2)
        return pd.DataFrame({"cant_nulos": cuenta_nulos, "porcentaje_nulos": porcen

    def nulos_EliminacionSimple(self, lista):
        """Se eliminarán las filas que contengan valores nulos en las columnas indicadas"""
        try:
            self.df = self.df.dropna(subset=lista)
        except Exception as error:
            print("Ocurrió un error:", error)

    def nulos_MediaCondicional(self, variable, lista):
        """Se asignará a los valores nulos de la columna variable, el valor promedio agrupando con las columnas indicadas por la lista"""
        try:
            self.df.loc[self.df[variable].isnull(), variable] = self.df.groupby(list
        except Exception as error:
            print("Ocurrió un error:", error)

    def eliminar_por_condicion(self, columna, condicion):
        """Se eliminarán las filas que contengan valores 'condicion' en la columna indicada"""
        try:
            self.df = self.df.drop(self.df[self.df[columna] == condicion].index)
        except Exception as error:
            print("Ocurrió un error:", error)

#revision = revisor_data_csv("../datos_internos/20210513_Challenge_AA_MANANA.csv")
revision = revisor_data_csv("df_prueba_MANANA.csv")

# Eliminación de valores nulos
revision.nulos_EliminacionSimple(['MORAS', 'SEXO', 'ESTADOCIVIL', 'FECHANACIMIENTO'])
revision.nulos_MediaCondicional('INGRESO', ['ESTADO', 'EDAD', 'ESTADOCIVIL', 'SEXO'])
revision.nulos_EliminacionSimple(['INGRESO'])
revision.df['MORAS'] = revision.df['MORAS'].astype(int)
revision.df['ANTIGUEDAD'] = revision.df['ANTIGUEDAD'].astype(int)
revision.df['EDAD'] = revision.df['EDAD'].astype(int)
revision.df['FECHANACIMIENTO'] = pd.to_datetime(revision.df['FECHANACIMIENTO']).dt.

```

```

revision.df['FECHANACIMIENTO'] = pd.to_datetime(revision.df['FECHANACIMIENTO'])
revision.eliminar_por_condicion('ESTADOCIVIL',' ')

revision.df = revision.df.drop(['CLIENTE'], axis=1)

class preprocesamiento:
    def __init__(self,revisor_data):
        """
        Constructor de la clase recibe el objeto instanciado de la clase revisor_data
        seria mejor usar un patron de diseño DataFrameSingleton pero esta division
        para fines demostrativos,en codigo final todos los metodos deben
        pertenecer a una sola clase.
        """
        try:
            # Se carga el df
            self.revisor_data = revisor_data
            self.df = self.revisor_data.df
            self.matriz_correlacion = None
            self.transformaciones = {}
        except Exception as error:
            print("Ocurrió un error:", error)

#Este metodo esta repetido en la clase graficos, al final se debe mejorar el código
def variables_numericas(self):
    """Devuelve una lista con las variables numericas de un df"""
    return [column_name for column_name, data_type in zip(self.df.columns, self.df.dtypes) if data_type in [np.int64, np.float64]]


def detencion_outlier(self,nombre_columna,q=.1):
    """
    Funcion para detectar valores atípicos (utiliers) de una columna especifica
    Por defecto se usa Deciles / dividiendo la distribucion en 10 partes, pero
    q = 0.25 se trabajaria con cuartiles.
    La funcion devuelve una "lista/pandas.core.indexes.numeric.Int64Index" con
    valores atípicos del df
    """
    # try:
    #calculo de cuantiles
    Q1 = self.df[nombre_columna].quantile(q)
    Q3 = self.df[nombre_columna].quantile(1-q)
    IQR = Q3-Q1
    limite_inferior = Q1 - 1.5 * IQR
    limite_superior = Q3 + 1.5 * IQR
    indice_filas_eliminar = self.df.index[(self.df[nombre_columna] < limite_inferior) | (self.df[nombre_columna] > limite_superior)]
    return indice_filas_eliminar


def eliminacion_outlier(self,nombre_columna,q=0.1):
    """
    Funcion para eliminar valores atípicos (outliers) de una columna especifica
    La funcion no devuelve nada porque los cambios se hacen en el df que se pasa
    """
    try:
        if nombre_columna in self.variables_numericas():
            indices = self.detencion_outlier(nombre_columna,q)
            self.df = self.df.drop(indices)
            self.df.reset_index(inplace=True,drop=True)
        else:
    
```

```

        print("La variable no es numerica")
    except Exception as error:
        print("Ocurrió un error:", error)

    def guardar_transformador(self,nombre_columna,transformador):
        """Metodo para guardar transformador aplicado, se almacenan en lista, si se
        aplican dos o mas se almacenaran en la secuencia aplicada"""
        if nombre_columna in list(self.transformaciones.keys()):
            self.transformaciones[nombre_columna] = self.transformaciones[nombre_co
        else:
            self.transformaciones[nombre_columna] = [transformador]

    def Transf_MinMaxScaler(self,nombre_columna):
        """crea objeto que Transforma los valores de la columna indicada
        usando MinMaxScaler de sklear, devuelve el objeto para transformar futuros
        try:
            if nombre_columna in self.variables_numericas():
                scaler = preprocessing.MinMaxScaler()
                escalador = scaler.fit(self.df[nombre_columna].values.reshape(-1, 1
                return escalador
            else:
                print("La variable no es numerica")
        except Exception as error:
            print("Ocurrió un error:", error)

    def Transf_Quantile(self,nombre_columna):
        """crea objeto para Transformar los valores de la columna indicada
        usando QuantileTransformer de sklearn"""
        try:
            if nombre_columna in self.variables_numericas():
                scaler = preprocessing.QuantileTransformer()
                escalador = scaler.fit(self.df[nombre_columna].values.reshape(-1, 1
                return escalador
            else:
                print("La variable no es numerica")
        except Exception as error:
            print("Ocurrió un error:", error)

    def Transf_OneHot_binario(self, nombre_columna):
        """Crea objeto para transformacion OneHot cuando la variable es binaria, de
        el transformador ya entrenado"""
        try:
            value_var = self.df[nombre_columna].astype("category")
            codificador_oneHot = OneHotEncoder(handle_unknown='ignore', drop='first'
            codificacion = codificador_oneHot.fit(pd.DataFrame(value_var, columns=[

            return codificacion
        except Exception as error:
            print("Ocurrió un error:", error)

    def Transf_OneHot(self, nombre_columna):
        """crea objeto para Transformar los valores de la columna indicada
        usando OneHot encoder de sklearn, devuelve el transformador entrenado """
        try:
            value_var = self.df[nombre_columna].astype("category")
            codificador_oneHot = OneHotEncoder(handle_unknown='ignore')
            codificacion = codificador_oneHot.fit(pd.DataFrame(value_var, columns=[


```

```

        return codificacion
    except Exception as error:
        print("Ocurrió un error:", error)

def Transf_woe(self,nombre_columna,nombre_target):
    """crea el objeto para Transformar los valores de la columna indicada
    usando woe de OptimalBinning,
    Distingue de variables numericas y categoricas
    Devuelve el objeto transformador"""
    try:
        data_type = self.df[nombre_columna].dtypes
        target = self.df[nombre_target]
        x = self.df.loc[:,nombre_columna]
        if (data_type == 'object' or data_type.name == 'category'):
            optb = OptimalBinning(name = nombre_columna,dtype ='categorical',so
            optb.fit(x,target)
        elif np.issubdtype(data_type, np.number):
            optb = OptimalBinning(name = nombre_columna,dtype = 'numerical',sol
            optb.fit(x,target)
        else:
            print("La variable se de convertir a tipo numerica o categorica")
            optb = None

        return optb
    except Exception as error:
        print("Ocurrió un error:", error)

def calc_matriz_correlacion(self, columnas=None,filas=None):
    """metodo para la creacion de la matriz de correlacion, se pude ajustar el
    a calcular para la correlacion"""
    try:
        if self.matriz_correlacion is None or self.matriz_correlacion.empty:
            if columnas:
                # Se confirma que las columnas existan y sean numericas
                columnas = [ col for col in columnas if col in self.variables_nu
            else:
                columnas = self.variables_numericas()
        if filas:
            data_set = self.df[columnas].sample(filas)
            data_set.reset_index(inplace=True,drop=True)
        else:
            data_set = self.df[columnas]
        self.matriz_correlacion = data_set.corr(method='pearson', numeric_o
    except Exception as error:
        print("Ocurrió un error:", error)

def clasificacion_correlacion(self, target_name ,nivel_correlacion_target = 0.3
    """Metodo para clasificar las variables en funcion del nivel de correlacion
    1. Entre las variables y el objetivo (target_name), escogiendo las que teng
    Moderada correlación |corr| > 0.3 pero se puede ajustar si se desea.
    2. Entre las mismas variables permitiendo hasta una Moderada correlación |c
    con el objetivo de reducir la multicolinalidad"""
    try:
        # Calculo de la correlacion para todas las variables
        self.calc_matriz_correlacion()

```

```

        matriz = self.matriz_correlacion

        #Verificacion de la correlacion entre todas las variables y el target,
        signals = []
        for i in range(len(matriz)):
            for j in range(i+1, len(matriz)):
                signal_1 = matriz.columns[i]
                signal_2 = matriz.columns[j]
                correlation = abs(matriz.iloc[i, j])
                signals.append([signal_1, signal_2, correlation])
        df_correlacion = pd.DataFrame(signals, columns=['sig_1', 'sig_2', 'nive
df_correlacion_mod = df_correlacion[df_correlacion['sig_2'] == target_n
variables_x = list(df_correlacion_mod [df_correlacion_mod['nivel_correl

        # Verificacion de la correlacion entre variables
        for i, variable1 in enumerate(variables_x):
            for variable2 in variables_x[i+1:]:
                mask = (df_correlacion['sig_1'] == variable1) & (df_correlacion
                val_corr = list(df_correlacion[mask]['nivel_correlacion'])
                if not val_corr:
                    mask = (df_correlacion['sig_1'] == variable2) & (df_correla
                    val_corr = list(df_correlacion[mask]['nivel_correlacion'])
                val_corr = val_corr[0] if val_corr else 0
                if val_corr > nivel_correlacion_inter_variables:
                    if variable2 in variables_x:
                        variables_x.remove(variable2)
        return variables_x
    except Exception as error:
        print("Ocurrió un error:", error)

def crea_EDAD(self,nombre_columna = 'EDAD'):
    """Para crear una variable EDAD se usa como referencia la fecha actual,
    independiente del momento en que se corra el programa, se calcula la dife
    la fecha de nacimiento provista y la fecha actual
    """
    try:
        fecha_actual = datetime.datetime.now()
        self.df[nombre_columna] = (fecha_actual - self.df['FECHANACIMIENTO']).d
        self.df.drop(columns=['ANIO', 'MES','FECHANACIMIENTO'], inplace=True)
        # Se actualiza la instancia de la clase revisor_data_csv
        #self.revisor_data.df = self.df
    except Exception as error:
        print("Ocurrió un error:", error)

def crea_EDAD_NORM(self,nombre_columna = 'EDAD'):
    """Para crear una variable EDAD normalizada se usa como referencia la fecha
    independiente del momento en que se corra el programa, se calcula la dife
    la fecha de nacimiento provista y la fecha actual, se aplica una eliminacio
    y una transformacion estandar normalizando entre el valor minimo y maximo.
    """
    try:
        fecha_actual = datetime.datetime.now()
        self.df[nombre_columna] = (fecha_actual - self.df['FECHANACIMIENTO']).d
        self.eliminacion_outlier(nombre_columna)
        escalador = self.Transf_MinMaxScaler(nombre_columna)
        self.df[nombre_columna] = escalador.transform(self.df[nombre_columna].v

```

```

        self.guardar_transformador(nombre_columna,escalador)
        self.df.drop(columns=['ANIO', 'MES','FECHANACIMIENTO'], inplace=True)

        # Se actualiza la instancia de la clase revisor_data_csv
        #self.revisor_data.df = self.df
    except Exception as error:
        print("Ocurrió un error:", error)

    def mod_columna_Transf_Quantile(self,nombre_columna):
        """Modifica la columna indicada utilizando el Transf_Quantile,
        almacena el nombre de la columna modificada y el objeto utilizado para la t
    try:
        escalador = self.Transf_Quantile(nombre_columna)
        self.df[nombre_columna] = escalador.transform(self.df[nombre_columna]).v
        self.guardar_transformador(nombre_columna,escalador)

        # Se actualiza la instancia de la clase revisor_data_csv
        #self.revisor_data.df = self.df
    except Exception as error:
        print("Ocurrió un error:", error)

    def mod_columna_OneHot_binario(self,nombre_columna):
        """Modifica la columna indicada utilizando OneHot_binario,
        almacena el nombre de la columna modificada y
        el objeto utilizado para la transformacion"""
    try:
        escalador = self.Transf_OneHot_binario(nombre_columna)
        arreglo = escalador.transform(preproceso.df[[nombre_columna]]).toarray()
        self.df[nombre_columna] = arreglo.astype(int)
        self.guardar_transformador(nombre_columna,escalador)

        # Se actualiza la instancia de la clase revisor_data_csv
        #self.revisor_data.df = self.df
    except Exception as error:
        print("Ocurrió un error:", error)

    def mod_columna_OneHot(self,nombre_columna):
        try:
            escalador = self.Transf_OneHot(nombre_columna)
            arreglo = escalador.transform(preproceso.df[[nombre_columna]]).toarray()
            columnas_codificadas = escalador.get_feature_names_out([nombre_columna])
            df_codificado = pd.DataFrame(arreglo, columns=columnas_codificadas)

            #Se introducen en df los valores codificados
            pos = self.df.columns.get_loc(nombre_columna)
            for col in df_codificado.columns:
                self.df.insert(pos, col, df_codificado[col])
                pos += 1
            self.df.drop(columns=[nombre_columna], inplace=True)
            self.guardar_transformador(nombre_columna,escalador)

            # Se actualiza la instancia de la clase revisor_data_csv
            #self.revisor_data.df = self.df
        except Exception as error:
            print("Ocurrió un error:", error)

```

```
def mod_woe(self,nombre_columna,nombre_target,metrica = "woe"):
    """Metodo para transformar los elementos de una columna
    usando woe de OptimalBinning, se pueden utilizar otras metricas
    como: "event_rate", "woe", "indices" and "bins" ."""
    try:
        escalador = self.Transf_woe(nombre_columna,nombre_target)
        x = self.df.loc[:,nombre_columna]
        self.df[nombre_columna] = escalador.transform(x, metric=metrica)
        self.guardar_transformador(nombre_columna,escalador)

        # Se actualiza la instancia de la clase revisor_data_csv
        #self.revisor_data.df = self.df
    except Exception as error:
        print("Ocurrió un error:", error)
preproceso = preprocesamiento(revision)

def grafica_confusion_matrix(y_test, y_test_predictions):
    conf_matrix = confusion_matrix(y_test, y_test_predictions)
    fig, ax = plt.subplots(figsize=(8,6), dpi=100)
    display = ConfusionMatrixDisplay(conf_matrix)
    ax.set(title='Confusion Matrix')
    display.plot(ax=ax)
```

Carga completa

15. Informe resumen.

Se anexa en carpeta de trabajo el informe 'Informe resumen.pdf'