

Análisis de estado crediticio de clientes aplicando Inteligencia Artificial en Institución Financiera

Por

LIBIA JOHANA SÁNCHEZ ESPINOZA

Resumen

Este proyecto tiene como objetivo analizar el estado crediticio de los clientes mediante la implementación de técnicas de Inteligencia Artificial, clasificando los créditos en función de la existencia de mora. Inicialmente, se realiza una revisión de la literatura sobre tecnologías disruptivas en el contexto de la Industria 4.0, y se analizan trabajos similares previos que sustentan la selección de los modelos de Regresión Logística, Árbol de Decisión, Random Forest y XGBoost. Para desarrollar la solución, se aplica la metodología CRISP-DM, comenzando con la comprensión del giro de negocio de la Institución Financiera, destacando la importancia de mantener una gestión de créditos sostenible y rentable, dado que estos representan el mayor índice de ingresos económicos. A continuación, se analiza la estructura de la base de datos SQL Server, de donde se extrae información histórica de los clientes y sus créditos. Se desarrolla un script con Python, que permite adecuar los datos para ser usados en modelos de Aprendizaje Automático. Posteriormente se balancean los datos con la técnica de sobre muestreo aleatorio ROS y SMOTE para que en la fase de modelado se pueda entrenar y probar los modelos previamente seleccionados. Con los índices de precisión obtenidos, se elabora una matriz comparativa que permite concluir que el modelo XGBoost, combinado con SMOTE, alcanza el mayor índice de precisión, con un 79%. Finalmente se recomienda su implementación.

Palabras clave: predecir, mora, Inteligencia Artificial, Machine Learning, XGBoost.

Índice

1. Objetivos.....	8
1.1. General	8
1.2. Específicos.....	8
2. Metodología.....	8
2.1. Metodología <i>CRISP-DM</i>	8
3. Desarrollo de la solución	9
3.1. Fase de comprensión del negocio o problema.....	9
3.2. Fase de comprensión de los datos	9
3.2.1. Fuente de datos original	9
3.2.2. Extracción de datos	9
3.2.3. Archivo obtenido después de la extracción de datos.....	9
3.3. Fase de preparación de los datos	9
3.3.1. Lectura de datos	10
3.3.2. Análisis exploratorio de datos (EDA).....	10
3.3.3. Análisis y limpieza de datos.....	15
3.3.4. Preparación y transformación de variables	28
3.4. Fase de modelado	42
3.4.1. Balanceo de datos con sobre muestreo aleatorio.....	43
3.4.2. Modelo Regresión Logística con sobre muestreo aleatorio.	44
3.4.3. Modelo Árbol de decisión con sobre muestreo aleatorio.....	45
3.4.4. Modelo <i>Random Forest</i> con sobre muestreo aleatorio.....	46
3.4.5. Modelo <i>XGBoost</i> con sobre muestreo aleatorio.....	47
3.4.6. Balanceo de datos con la técnica <i>SMOTE</i>	49

3.4.7.	Modelo Regresión Logística con balanceo <i>SMOTE</i>	49
3.4.8.	Modelo Árbol de decisión con balanceo <i>SMOTE</i>	50
3.4.9.	Modelo <i>Random Forest</i> con balanceo <i>SMOTE</i>	51
3.4.10.	Modelo <i>XGBoost</i> con sobre muestreo <i>SMOTE</i>	52
3.5.	Fase de evaluación.....	54
3.6.	Fase de implementación	55
4.	Conclusiones y líneas futuras	55
	Referencias bibliográficas.....	57

Índice de figuras

Figura 1. Estructura del <i>DataFrame</i> de créditos. Fuente: Elaboración propia.	11
Figura 2. Matriz de información del <i>DataFrame</i> original. Fuente: Elaboración propia.....	11
Figura 3. Número de filas y columnas. Fuente: Elaboración propia.	11
Figura 4. Matriz de valores nulos <i>DataFrame</i> . Fuente: Elaboración propia	12
Figura 5. Matriz con la frecuencia de valores de “TipoActividad”. Fuente: Elaboración propia.....	12
Figura 6. Valores de los 4 registros con datos ausentes en la variable “TipoActividad”. Fuente: Elaboración propia.	13
Figura 7. Verificación de que no existen valores ausentes en “TipoActividad”. Fuente: Elaboración propia	13
Figura 8. Matriz con la frecuencia de valores de “vivienda”. Fuente: Elaboración propia.	14
Figura 9. Matriz con la frecuencia de valores de “Vivienda” sin valores nulos. Fuente: Elaboración propia.	14
Figura 10. Estadística descriptiva de las variables numéricas. Fuente: Elaboración propia.....	15
Figura 11. Resumen estadístico de las variables categóricas. Fuente: Elaboración propia.....	15
Figura 12. Información resumen del <i>DataFrame</i> con variables numéricas tipo decimal transformadas. Fuente: Elaboración propia.....	17
Figura 13. Distribución de frecuencias de las variables categóricas. Fuente: Elaboración propia.....	19
Figura 14. Tipo de Actividad frecuente por Instrucción y valores corregidos de “TipoActividad”. Fuente: Elaboración propia.	20
Figura 15. Distribución de frecuencias de “TipoActividad” sin valores redundantes. Fuente: Elaboración propia.	21
Figura 16. Gráfico de barras categóricas para visualizar la distribución de las variables categóricas con respecto a la variable “MoraCredito”. Fuente: Elaboración propia.....	22
Figura 17. Test Chi Cuadrado de “Sexo” y “MoraCredito”. Fuente: Elaboración propia.	23

Figura 18. Test Chi Cuadrado entre “EstadoCivil” y “MoraCredito”. Fuente: Elaboración propia.	24
Figura 19. Test Chi Cuadrado entre “Instrucción” y “MoraCredito”. Fuente: Elaboración propia.	25
Figura 20. Test Chi Cuadrado entre Calificacion y MoraCredito.	26
Figura 21. Test Chi Cuadrado entre “Tipo” y “MoraCredito”. Fuente: Elaboración propia.....	27
Figura 22. Test Chi Cuadrado entre “vivienda” y “MoraCredito”. Fuente: Elaboración propia.....	28
Figura 23. Información de <i>DataFrame</i> con las nuevas variables creadas. Fuente: Elaboración propia.	29
Figura 24. Distribución de los años de nacimiento. Fuente: Elaboración propia.	29
Figura 25. Distribución de “Instruccion” con valores numéricos. Fuente: Elaboración propia.....	30
Figura 26. Distribución de frecuencias de “CodigoBarrio”. Fuente: Elaboración propia.	30
Figura 27. Distribución de frecuencias de “NumeroCargas”. Fuente: Elaboración propia.....	31
Figura 28. Distribución de frecuencias de la variable “TipoActividad” de tipo numérico. Fuente: Elaboración propia.	31
Figura 29. Distribución de frecuencias de la variable “NumeroCredito”. Fuente: Elaboración propia.	32
Figura 30. Distribución de frecuencias de la variable “NumeroCuotas”. Fuente: Elaboración propia.	32
Figura 31. Distribución de frecuencias de la variable “FechaEntrega”. Fuente: Elaboración propia.	33
Figura 32. Distribución de frecuencias de la variable “EdadAlCredito”. Fuente: Elaboración propia.	34
Figura 33. Gráfico de caja de la variable “EdadAlCredito”. Fuente: Elaboración propia.	34
Figura 34. Distribución de frecuencias de la nueva variable “GrupoEdad”. Fuente: Elaboración propia.	35
Figura 35. Distribución de frecuencias de la variable “Tasa”.....	35

Figura 36. Distribución de frecuencias de la variable “Calificación” de tipo numérico. Fuente: Elaboración propia.	36
Figura 37. Distribución de frecuencias de la variable “Tipo” como numérica. Fuente: Elaboración propia.	37
Figura 38. Información del <i>DataFrame</i> con las nuevas variables de “vivienda”. Fuente: Elaboración propia.	37
Figura 39. Distribución de frecuencias de la variable “Patrimonio”. Fuente: Elaboración propia.....	38
Figura 40. Distribución de frecuencias de la nueva variable “Disponible”. Fuente: Elaboración propia.	38
Figura 41. Distribución de la variable a predecir “MoraCredito”. Fuente: Elaboración propia.....	38
Figura 42. Información de <i>DataFrame</i> con las variables pre definitivas. Fuente: Elaboración propia.	39
Figura 43. Mapa de calor, con el método de Pearson. Fuente: Elaboración propia.....	40
Figura 44. Matriz de correlación con el método Pearson. Fuente: Elaboración propia.....	41
Figura 45. Información del <i>DataFrame</i> con las variables definitivas. Fuente: Elaboración propia.	42
Figura 46. Número de registros para entrenamiento. Fuente: Elaboración propia.	42
Figura 47. Número de registros para pruebas. Fuente: Elaboración propia.....	43
Figura 48. Distribución de frecuencias de la variable” MoraCredito” con datos de entrenamiento. Fuente: Elaboración propia.	43
Figura 49. Distribución de frecuencias de la variable “MoraCredito” con datos de prueba. Fuente: Elaboración propia.	43
Figura 50. Distribución de frecuencia de “MoraCredito” después del sobre muestreo en datos de entrenamiento. Fuente: Elaboración propia.	44
Figura 51. Matriz de confusión del modelo Regresión Logística. Fuente: Elaboración propia.....	45
Figura 52. Valor de precisión del modelo con Regresión Logística. Fuente: Elaboración propia.....	45
Figura 53. Matriz de confusión del modelo con Árbol de decisión. Fuente: Elaboración propia.....	46
Figura 54. Valor de precisión del modelo con Árbol de Decisión. Fuente: Elaboración propia.....	46

Figura 55. Matriz de confusión con el modelo <i>Random Forest</i> . Fuente: Elaboración propia.....	47
Figura 56. Valor de precisión del modelo con Random Forest.....	47
Figura 57. Precisión y matriz de confusión del modelo XGBoost. Fuente: Elaboración propia.....	48
Figura 58. Datos después del sobre muestreo con la técnica <i>SMOTE</i> . Fuente: Elaboración propia	49
Figura 59. Matriz de confusión del modelo Regresión Logística con técnica SMOTE. Fuente: Elaboración propia	50
Figura 60. Valor de precisión del modelo con <i>Regresión Logística</i> con balanceo <i>SMOTE</i>	50
Figura 61. Matriz de confusión del modelo Árbol de decisión con sobre muestreo SMOTE. Fuente: Elaboración propia.....	51
Figura 62. Valor de precisión del modelo Árbol de decisión con balanceo SMOTE.	51
Figura 63. Matriz de confusión del modelo Random Forest con sobre muestreo SMOTE. Fuente: Elaboración propia	52
Figura 64. Precisión del modelo <i>Random Forest</i> después del sobre muestreo SMOTE. Fuente: Elaboración propia	52
Figura 65. Precisión y matriz de confusión del modelo XGBoost después del sobre muestreo con SMOTE. Fuente: Elaboración propia.	53

1. Objetivos

1.1.General

El **objetivo general** es implementar modelos basados en técnicas de Inteligencia Artificial para predecir la existencia de mora en el pago de créditos de clientes de la Institución Financiera.

1.2.Específicos

Para lograr el objetivo general propuesto se proponen los siguientes objetivos específicos:

1. **Revisar la implementación de la Inteligencia Artificial en el sector financiero:** Estudiar la teoría y el estado del arte de los modelos de predicción.
2. **Preprocesar los datos históricos de los clientes:** Extraer, transformar y cargar los datos, asegurando su calidad y coherencia para el análisis.
3. **Analizar y limpiar los datos:** Realizar un análisis exploratorio de datos (EDA) y aplicar técnicas de limpieza para asegurar la calidad y consistencia de los datos.
4. **Desarrollar modelos de Aprendizaje Automático:** Implementar y entrenar los modelos predictivos utilizando técnicas de Aprendizaje Automático supervisado.
5. **Evaluar el índice de precisión del modelo:** Medir y analizar la precisión y efectividad del modelo.

2. Metodología

2.1.Metodología *CRISP-DM*

La metodología *CRISP-DM* (*CRoss-Industry Standard Process for Data mining*) es la guía de referencia más altamente utilizada en proyectos de *Data mining* y propone 6 fases:

- Fase de Comprensión del negocio o problema
- Fase de comprensión de los datos
- Fase de preparación de los datos
- Fase de modelado
- Fase de evaluación
- Fase de implementación

3. Desarrollo de la solución

3.1. Fase de comprensión del negocio o problema

Institución Financiera con 5000 socios, posee un historial de los clientes con sus datos demográficos, datos de cuentas y créditos con su comportamiento de pago, valor de crédito, frecuencia, interés, destino, saldo, estado, mora entre otros. La información actualmente se encuentra centralizada en el motor de base de datos *SQL Server* con tablas debidamente relacionadas.

3.2. Fase de comprensión de los datos

3.2.1. Fuente de datos original

Los datos se encuentran almacenados en una base de datos *Microsoft SQL Server*, que se encuentra operativa en modo *On-premise* en las instalaciones del *Data Center* propio de la Institución Financiera.

3.2.2. Extracción de datos

Por medio del Lenguaje *SQL (Structured Query Language)* se crea un *script* que extrae la información de las tablas de la base de datos *SQL Server*. El *script* se ejecuta realizando las siguientes operaciones:

- Selecciona Variables
- Calcula Activos y Pasivos
- Calcula Ingresos y Gastos
- Determina Mora de Crédito
- Combina Datos

3.2.3. Archivo obtenido después de la extracción de datos

Los datos se encuentran listos para ser cargados y leídos desde un archivo en formato *CSV*, separado por “;”.

3.3. Fase de preparación de los datos

La variable “MoraCredito” es la variable objetivo o variable a predecir, ya que contiene un dato binario donde 0 indica “Puntualidad sin mora” y 1 indica “Impuntualidad con mora”.

3.3.1. Lectura de datos

El análisis de los datos del archivo *CSV* es ejecutado por medio de un *script* codificado en Python con la ayuda de librerías propias para este objetivo y con el servicio de *Google Colab* para la edición de código.

Se importan las librerías necesarias y se procede con el acceso al archivo *CSV*.

```
# Importamos las librerías que necesitamos

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker
import seaborn as sns
import missingno as msno
from dateutil.relativedelta import relativedelta
from imblearn.over_sampling import RandomOverSampler
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, precision_score
from sklearn.metrics import accuracy_score, recall_score, f1_score
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import LabelEncoder
from scipy.stats import chi2_contingency
```

Se conecta al contenedor de Google Drive para acceder al archivo *CSV* almacenado.

```
from google.colab import drive
drive.mount('/content/drive')
```

Se obtiene un *DataFrame* a analizar por medio de la función *read_csv* de la librería Pandas.

```
# Cargamos el archivo CSV con los datos

df =
pd.read_csv("/content/drive/MyDrive/Colab_Notebooks/TFM_final/data_creditos_completo9.csv",
encoding = 'unicode_escape', engine = 'python', sep = ";")
df
```

3.3.2. Análisis exploratorio de datos (EDA)

Se visualiza el *DataFrame* y su estructura. En la Figura 1 se visualiza los datos retornados.

	Sexo	EstadoCivil	FechaNacimiento	Instruccion	CodigoBarrio	NumeroCargas	TipoActividad	NumeroCredito	Monto	NumeroCuotas	...	Tasa	Calificacion
0	Femenino	Casado	10/6/1970	P	18096707	0	Independiente	2891	600	10	...	23.00	Microcrédito
1	Masculino	Soltero	15/9/1992	G	18095004	0	Empleado Privado	3821	600	6	...	23.50	Microcrédito
2	Masculino	Soltero	3/4/1986	U	18010101	0	Empleado Privado	6223	8000	38	...	21.50	Microcrédito
3	Masculino	Soltero	3/6/1995	S	18016508	0	Independiente	6932	3000	24	...	18.00	Microcrédito
4	Masculino	Casado	4/2/1990	P	18096504	1	Independiente	3089	1500	18	...	21.00	Microcrédito

Figura 1. Estructura del *DataFrame* de créditos. Fuente: Elaboración propia.

Se ejecuta instrucción *info()* para observar las variables del *DataFrame*. En la Figura 2 se visualiza los datos retornados.

```
df.info()
```

```
Data columns (total 22 columns):
#   Column              Non-Null Count  Dtype
---  -
0   Sexo                 7680 non-null  object
1   EstadoCivil          7680 non-null  object
2   FechaNacimiento      7680 non-null  object
3   Instruccion          7680 non-null  object
4   CodigoBarrio         7680 non-null  int64
5   NumeroCargas         7680 non-null  int64
6   TipoActividad        7676 non-null  object
7   NumeroCredito        7680 non-null  int64
8   Monto                7680 non-null  object
9   NumeroCuotas         7680 non-null  int64
10  ValorCuota           7680 non-null  object
11  FechaEntrega         7680 non-null  object
12  Tasa                 7680 non-null  float64
13  Calificacion         7680 non-null  object
14  encaje               7680 non-null  object
15  tipo                 7680 non-null  object
16  vivienda             7668 non-null  object
17  totalActivo          7680 non-null  object
18  totalPasivo          7680 non-null  object
19  Ingresos             7680 non-null  object
20  Gastos               7680 non-null  object
21  MoraCredito          7680 non-null  int64
dtypes: float64(1), int64(5), object(16)
memory usage: 1.3+ MB
```

Figura 2. Matriz de información del *DataFrame* original. Fuente: Elaboración propia.

Se ejecuta instrucción *shape()* para observar el número de filas y columnas del *DataFrame*. En la Figura 3 se visualiza los datos retornados.

```
# Revisamos el tamaño del DataFrame
# El primer número son las filas y el segundo las columnas
```

```
df.shape
```

```
(7680, 22)
```

Figura 3. Número de filas y columnas. Fuente: Elaboración propia.

Se verifica la existencia de valores nulos en la *DataFrame*. En la Figura 4 se visualiza los datos retornados.

```
#Verificar si existen NULL o vacios en el DataFrame
# Se observa la distribución de los valores nulos en el DataFrame.

df.isnull().sum()
```

Instruccion	0
CodigoBarrio	0
NumeroCargas	0
TipoActividad	4
NumeroCredito	0
Monto	0
NumeroCuotas	0
ValorCuota	0
FechaEntrega	0
Tasa	0
Calificacion	0
encaje	0
tipo	0
vivienda	12

Figura 4. Matriz de valores nulos *DataFrame*. Fuente: Elaboración propia

Se observa en la matriz de la Figura 5, que la variable “TipoActividad” tiene 4 valores ausentes. Se observa la variable “TipoActividad”.

```
# Se observa la variable “TipoActividad”

df[“TipoActividad”].value_counts()
```

TipoActividad	
Independiente	4292
–Selecione Tipo Empleo–	2235
Empleado Privado	1026
Empleado Publico	123

Figura 5. Matriz con la frecuencia de valores de “TipoActividad”. Fuente: Elaboración propia.

Se visualiza los registros con valores ausentes en “TipoActividad”. En la Figura 6 se visualiza los datos retornados.

```
# Se filtrar los registros donde la columna 'TipoActividad' es nula

nulos = df[df['TipoActividad'].isnull()]
```

```
# Mostrar los registros nulos
print(nulos)
```

	Sexo	EstadoCivil	FechaNacimiento	Instruccion	CodigoBarrio	\
1476	Femenino	Soltero	20/11/1956	P	18015503	
4168	Femenino	Soltero	20/11/1956	P	18015503	
4324	Femenino	Soltero	20/11/1956	P	18015503	
4986	Femenino	Soltero	20/11/1956	P	18015503	

	NumeroCargas	TipoActividad	NumeroCredito	Monto	NumeroCuotas	...	\
1476	0	NaN	5838	4000	31	...	
4168	0	NaN	3193	2000	24	...	
4324	0	NaN	4753	2000	24	...	
4986	0	NaN	1651	2000	24	...	

	Tasa	Calificacion	encaje	tipo	vivienda	\
1476	21.5	Microcrédito	0	Ordinario Cuota Fija	VIVE CON FAMILIARES	
4168	23.0	Microcrédito	0	Ordinario Cuota Fija	VIVE CON FAMILIARES	
4324	21.5	Microcrédito	0	Ordinario Cuota Fija	VIVE CON FAMILIARES	
4986	22.5	Microcrédito	0	Ordinario Cuota Fija	VIVE CON FAMILIARES	

	totalActivo	totalPasivo	Ingresos	Gastos	MoraCredito
1476	2000	0	300	220	0
4168	2000	0	300	220	0
4324	2000	0	300	220	0
4986	2000	0	300	220	1

[4 rows x 22 columns]

Figura 6. Valores de los 4 registros con datos ausentes en la variable “TipoActividad”. Fuente: Elaboración propia.

Se elimina los registros nulos de la variable “TipoActividad” y se verifica que no tenga valores ausentes. En la Figura 7 se visualiza los datos retornados.

```
# Se elimina registros donde TipoActividad es NULL

df = df.dropna(subset=['TipoActividad'])
# Verificar si la columna 'TipoActividad' tiene valores nulos
has_null_TipoActividad = df['TipoActividad'].isna().any()
print(has_null_TipoActividad)
```

False

Figura 7. Verificación de que no existen valores ausentes en “TipoActividad”. Fuente: Elaboración propia

Se observa la variable “vivienda” que contiene valores nulos. En la Figura 8 se visualiza los datos retornados.

```
# Se observa la variable “vivienda”

df[“vivienda”].value_counts()
```

VIVE CON FAMILIARES	3879
PROPIA NO HIPOTECADA	3463
ARRENDADA	147
PROPIA HIPOTECADA	120
PRESTADA	55
dtype: int64	

Figura 8. Matriz con la frecuencia de valores de “vivienda”. Fuente: Elaboración propia.

Se extrae el valor más frecuente de “vivienda” y se reemplaza en los valores nulos. Se decide esta solución por ser 12 registros modificados. En la Figura 9 se visualiza los datos retornados.

```
# Se calcula la moda de la columna 'vivienda'
moda_vivienda = df['vivienda'].mode()[0]

# Se reemplaza los valores NaN en 'vivienda' con la moda utilizando .loc
df.loc[:, 'vivienda'] = df['vivienda'].fillna(modas_vivienda)
```

vivienda	
VIVE CON FAMILIARES	3891
PROPIA NO HIPOTECADA	3463
ARRENDADA	147
PROPIA HIPOTECADA	120
PRESTADA	55
dtype: int64	

Figura 9. Matriz con la frecuencia de valores de “Vivienda” sin valores nulos. Fuente: Elaboración propia.

Se ejecuta la función *df.describe()* para obtener la estadística descriptiva de las columnas con datos numéricos. Los resultados obtenidos revelan varias tendencias interesantes que se observan en la Figura 10 como:

El número de cargas familiares tiene un promedio bajo (0.57), lo que indica que la mayoría de los clientes tienen pocos o ningún dependiente. De hecho, el 50% de los clientes no tiene cargas familiares (percentil 50 en 0), mientras que algunos pocos alcanzan hasta 6 dependientes.

El número de cuotas tiene una media de 19.7, con un rango amplio que llega hasta 96 cuotas, lo que refleja una variedad significativa en los plazos de los créditos.

En cuanto a la tasa de interés, su media es del 21%, lo que es coherente con tasas de mercado típicas. Sin embargo, se observan tasas más bajas (mínimo 0%) y un máximo de 36%, lo que podría estar relacionado con distintas políticas de crédito.

La variable “MoraCredito”, que mide la morosidad, muestra que, en promedio, solo el 20% de los clientes presenta algún tipo de atraso en los pagos. Se observa que la mayoría de los clientes no tiene mora con el percentil 75 en 0.

Las variables “CodigoBarrio”, “NumeroCredito” son códigos que identifican el barrio y el crédito respectivamente y no tiene sentido valorar su estadística descriptiva.

```
# Se analiza un resumen estadístico de las columnas numéricas
```

```
df.describe()
```

	CodigoBarrio	NumeroCargas	NumeroCredito	NumeroCuotas	Tasa	MoraCredito
count	7.676000e+03	7676.000000	7676.000000	7676.000000	7676.000000	7676.000000
mean	1.805791e+07	0.568395	4555.821391	19.708051	21.013138	0.204534
std	5.784717e+05	0.943422	2681.144726	12.558225	2.851126	0.403387
min	2.010301e+06	0.000000	1.000000	1.000000	0.000000	0.000000
25%	1.807020e+07	0.000000	2273.750000	12.000000	20.500000	0.000000
50%	1.809630e+07	0.000000	4527.500000	15.000000	21.500000	0.000000
75%	1.809661e+07	1.000000	6703.250000	24.000000	22.440000	0.000000
max	2.301020e+07	6.000000	10817.000000	96.000000	36.000000	1.000000

Figura 10. Estadística descriptiva de las variables numéricas. Fuente: Elaboración propia.

Se obtiene un resumen estadístico de los variables categóricas. En la Figura 11 se visualiza los datos retornados.

```
# Resumen estadístico de variables categóricas (tipo object)
```

```
df.describe(include=["O"])
```

	Sexo	EstadoCivil	FechaNacimiento	Instruccion	TipoActividad	Monto	ValorCuota	FechaEntrega	Calificacion	encaje	tipo	vivienda	totalActivo	totalPasivo
count	7676	7676	7676	7676	7676	7676	7676	7676	7676	7676	7676	7664	7676	7676
unique	2	5	2337	6	4	502	2646	2941	5	222	2	5	595	823
top	Femenino	Casado	6/5/1990	P	Independiente	1000	83.33	16/12/2016	Microcrédito	0	Ordinario Cuota Fija	VIVE CON FAMILIARES	0	0
freq	4117	3984	30	4816	4292	1043	1117	25	7397	6790	6440	3879	2431	3282

Figura 11. Resumen estadístico de las variables categóricas. Fuente: Elaboración propia.

3.3.3. Análisis y limpieza de datos

Se observa el resumen estadístico de las variables categóricas de tipo Object, pero su contenido es de tipo decimal: “Monto”, “ValorCuota”, “encaje”, “totalActivo”, “totalPasivo”, “Ingresos” y “Gastos” es necesario convertirlas a tipo *float*. En la Figura 12 se visualiza los datos retornados.

```
# Se observa la categoría "Monto"
# Se reemplaza la "," por "."
# Se transforma a decimal

df["Monto"].value_counts()
df["Monto"] = df["Monto"].str.replace(",", ".")
df['Monto'] = df['Monto'].astype(float)
df["Monto"].value_counts()

# Se observa la variable "ValorCuota"
# Se reemplaza la "," por "."
# Se transforma a decimal
df["ValorCuota"].value_counts()
df["ValorCuota"] = df["ValorCuota"].str.replace(",", ".")
df['ValorCuota'] = df['ValorCuota'].astype(float)
df["ValorCuota"].value_counts()

# Se observa la categoría "encaje"
# Se reemplaza la "," por "."
# Se transforma a decimal
df["encaje"].value_counts()
df["encaje"] = df["encaje"].str.replace(",", ".")
df['encaje'] = df['encaje'].astype(float)
df["encaje"].value_counts()

# Se observa la categoría "totalActivo"
# Se reemplaza la "," por "."
# Se transforma a decimal
df["totalActivo"].value_counts()
df["totalActivo"] = df["totalActivo"].str.replace(",", ".")
df['totalActivo'] = df['totalActivo'].astype(float)
df["totalActivo"].value_counts()

# Se observa la categoría "totalPasivo"
# Se reemplaza la "," por "."
# Se transforma a decimal
df["totalPasivo"].value_counts()
df["totalPasivo"] = df["totalPasivo"].str.replace(",", ".")
df['totalPasivo'] = df['totalPasivo'].astype(float)
df["totalPasivo"].value_counts()
```



```
# Se observa la categoría "Ingresos"
# Se reemplaza la "," por "."
# Se transforma a decimal
df["Ingresos"].value_counts()
df["Ingresos"] = df["Ingresos"].str.replace(",", ".")
df['Ingresos'] = df['Ingresos'].astype(float)
df["Ingresos"].value_counts()

# Se observa la categoría "Gastos"
# Se reemplaza la "," por "."
# Se transforma a decimal
df["Gastos"].value_counts()
df["Gastos"] = df["Gastos"].str.replace(",", ".")
df['Gastos'] = df['Gastos'].astype(float)
df["Gastos"].value_counts()

#Observamos un resumen del DataFrame
df.info()
```

```
# Column Non-Null Count Dtype
---
0 Sexo 7676 non-null object
1 EstadoCivil 7676 non-null object
2 FechaNacimiento 7676 non-null object
3 Instruccion 7676 non-null object
4 CodigoBarrio 7676 non-null int64
5 NumeroCargas 7676 non-null int64
6 TipoActividad 7676 non-null object
7 NumeroCredito 7676 non-null int64
8 Monto 7676 non-null float64
9 NumeroCuotas 7676 non-null int64
10 ValorCuota 7676 non-null float64
11 FechaEntrega 7676 non-null object
12 Tasa 7676 non-null float64
13 Calificacion 7676 non-null object
14 encaje 7676 non-null float64
15 tipo 7676 non-null object
16 vivienda 7676 non-null object
17 totalActivo 7676 non-null float64
18 totalPasivo 7676 non-null float64
19 Ingresos 7676 non-null float64
20 Gastos 7676 non-null float64
21 MoraCredito 7676 non-null int64
dtypes: float64(8), int64(5), object(9)
memory usage: 1.3+ MB
```

Figura 12. Información resumen del *DataFrame* con variables numéricas tipo decimal transformadas. Fuente: Elaboración propia.

Se observan las variables categóricas “Sexo”, “EstadoCivil”, “Instruccion”, “TipoActividad”, “Calificacion”, “tipo” y “vivienda”. En la Figura 13 se visualiza los datos retornados.

```
sexo_counts = df["Sexo"].value_counts()
estado_civil_counts = df["EstadoCivil"].value_counts()
```

```
instruccion_counts = df["Instruccion"].value_counts()
tipo_actividad_counts = df["TipoActividad"].value_counts()
calificacion_counts = df["Calificacion"].value_counts()
tipo_counts = df["tipo"].value_counts()
vivienda_counts = df["vivienda"].value_counts()

# Mostrar resultados
print("Conteos de valores en la columna 'Sexo':")
print(sexo_counts)
print()

print("Conteos de valores en la columna 'EstadoCivil':")
print(estado_civil_counts)
print()

print("Conteos de valores en la columna 'Instruccion':")
print(instruccion_counts)
print()

print("Conteos de valores en la columna 'TipoActividad':")
print(tipo_actividad_counts)
print()

print("Conteos de valores en la columna 'Calificacion':")
print(calificacion_counts)
print()

print("Conteos de valores en la columna 'Tipo':")
print(tipo_counts)
print()

print("Conteos de valores en la columna 'vivienda':")
print(vivienda_counts)
```

```

Conteos de valores en la columna 'Sexo':
Sexo
Femenino      4117
Masculino     3559
Name: count, dtype: int64

Conteos de valores en la columna 'EstadoCivil':
EstadoCivil
Casado        3984
Soltero       2548
Divorciado     553
Unión Libre    382
Viudo         289
Name: count, dtype: int64

Conteos de valores en la columna 'Instruccion':
Instruccion
P      4816
S      2818
U       632
N       133
G        54
T         31
Name: count, dtype: int64

Conteos de valores en la columna 'TipoActividad':
TipoActividad
Independiente      4292
--Seleccione Tipo Empleo--  2235
Empleado Privado   1826
Empleado Publico    123
Name: count, dtype: int64

Conteos de valores en la columna 'Calificacion':
Calificacion
Microcrédito    7397
Consumo         242
Vivienda         23
Inmobiliario     12
Comercial         2
Name: count, dtype: int64

Conteos de valores en la columna 'Tipo':
tipo
Ordinario Cuota Fija      6448
Ordinario Cuota Variable  1236
Name: count, dtype: int64

```

Figura 13. Distribución de frecuencias de las variables categóricas. Fuente: Elaboración propia.

Se observa un valor erróneo “--Seleccione Tipo Empleo--” en la variable “TipoActividad”. En la Figura 14 se visualiza los datos retornados.

```

#Se reemplaza el valor erróneo “--Seleccione Tipo Empleo--” de “TipoActividad”
# y los valores vacíos por el valor que más se repite según la variable “Instruccion”
# Paso 1: Calcular la moda de “TipoActividad” para cada “Instrucción”
def calcular_moda(grupo):
    try:
        return grupo.mode().iloc[0]
    except IndexError:
        return None # Si no hay moda, devolver None

# Filtrar las filas que no tienen el valor erróneo y agrupar por “Instrucción” para calcular
la moda

```

```

moda_por_instruccion = df[df['TipoActividad'] != '--Seleccione Tipo Empleo--'].groupby('Instruccion')['TipoActividad'].apply(calcular_moda)

# Imprimir la moda para verificar
print("Moda por Instrucción:\n", moda_por_instruccion)

# Paso 2: Reemplazar los valores erróneos y nulos según la moda por "Instrucción"
def reemplazar_tipo_actividad(row):
    instruccion = row['Instruccion']
    tipo_actividad = row['TipoActividad']

    # Se verifica si el TipoActividad es el valor incorrecto o si es nulo
    if pd.isnull(tipo_actividad) or tipo_actividad == '--Seleccione Tipo Empleo--':
        # Se reemplaza con la moda correspondiente si está disponible
        if instruccion in moda_por_instruccion.index:
            return moda_por_instruccion[instruccion]
        else:
            # Si la instrucción no tiene moda calculada, se mantiene el valor original (nulo o erróneo)
            return tipo_actividad
    else:
        return tipo_actividad

# Se aplica la función al DataFrame
df['TipoActividad'] = df.apply(reemplazar_tipo_actividad, axis=1)

# Verificar los cambios en la columna TipoActividad
df["TipoActividad"].value_counts()
  
```

Moda por Instrucción:	
Instruccion	
G	Independiente
N	Independiente
P	Independiente
S	Independiente
T	Independiente
U	Empleado Privado

TipoActividad	
Independiente	6339
Empleado Privado	1214
Empleado Publico	123
dtype: int64	

Figura 14. Tipo de Actividad frecuente por Instrucción y valores corregidos de "TipoActividad". Fuente: Elaboración propia.

Se corrige valores redundantes en la variable "Calificación". En la Figura 15 se visualiza los datos retornados.

```

# Se observa en la categoría "Calificacion" valores redundantes

# Unificar valores "Comercial" y "Microcredito" de la variable
# "Calificacion" (Microcredito se considera para comercial)
  
```

```
df["Calificacion"] = df["Calificacion"].str.replace("Comercial", "Microcrédito")
df["Calificacion"].value_counts()
```

Calificacion	
Microcrédito	7399
Consumo	242
Vivienda	23
Inmobiliario	12
dtype: int64	

Figura 15. Distribución de frecuencias de “TipoActividad” sin valores redundantes. Fuente: Elaboración propia.

Se crean los gráficos de barras categóricas para visualizar la distribución de las variables con respecto a la variable “MoraCredito”. En la Figura 16 se visualiza los gráficos obtenidos.

```
# Se crea lista de variables categóricas
categorical_columns = ['Sexo', 'EstadoCivil', 'Instruccion', 'TipoActividad',
'Calificacion', 'vivienda']

# Se crea gráficos de barra para cada variable categórica con respecto a “MoraCredito”
for column in categorical_columns:
    plt.figure(figsize=(10, 6))
    ax = sns.countplot(x=column, hue='MoraCredito', data=df)

    # Se añade la cuadrícula
    plt.grid(True, linestyle='--', alpha=0.7)

    # Se muestra los valores en cada barra
    for p in ax.patches:
        ax.annotate(f'{int(p.get_height())}',
                    (p.get_x() + p.get_width() / 2., p.get_height()),
                    ha='center', va='center', xytext=(0, 10),
                    textcoords='offset points',
                    fontsize=10, color='black', weight='bold')

    plt.title(f'{column} vs MoraCredito')
    plt.xlabel(column)
    plt.ylabel('Count')
    plt.legend(title='MoraCredito')

# Se muestra el gráfico
plt.show()
```

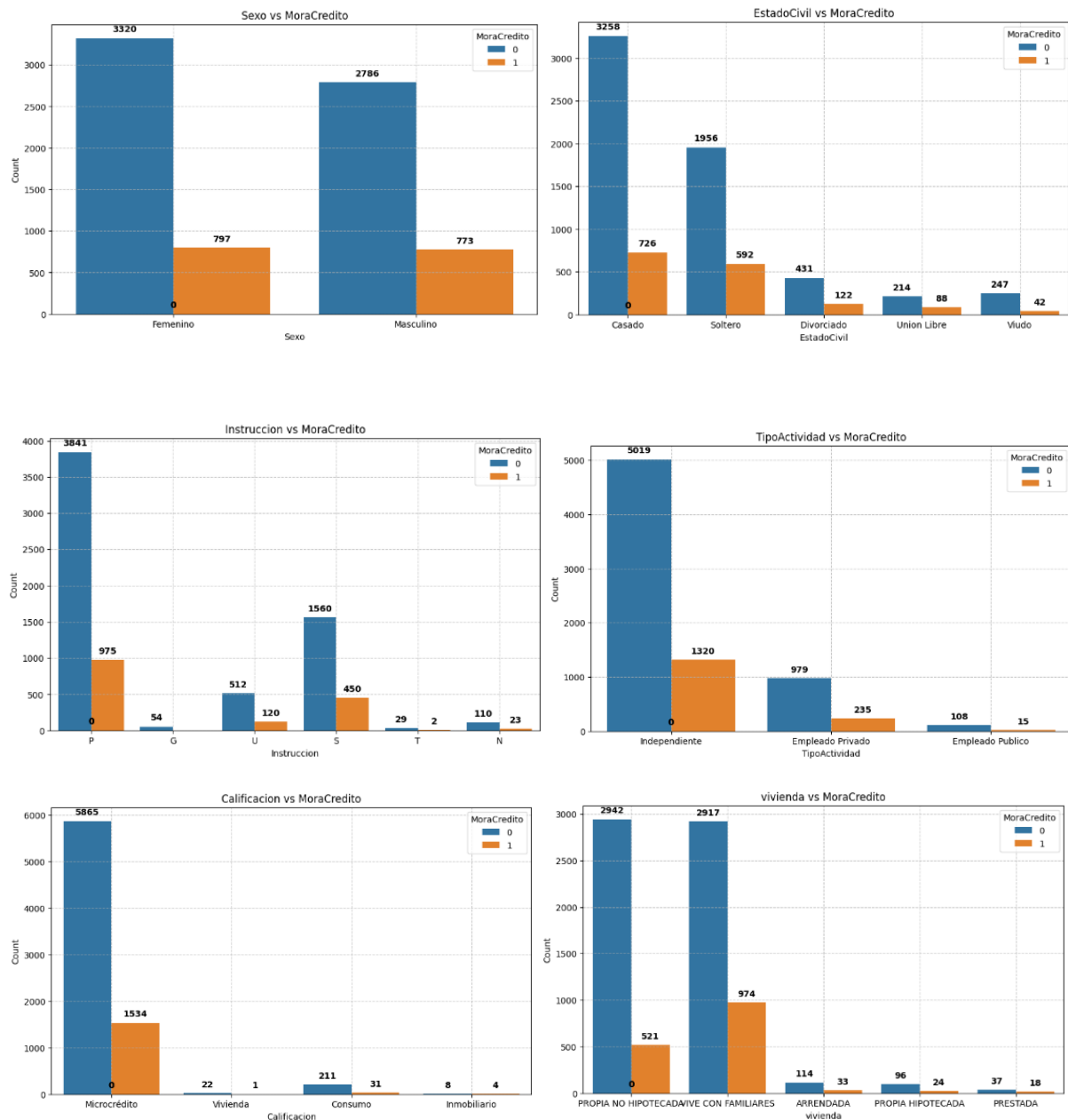


Figura 16. Gráfico de barras categóricas para visualizar la distribución de las variables categóricas con respecto a la variable “MoraCredito”. Fuente: Elaboración propia.

Se interpreta los gráficos y se visualiza que hay más cantidad de créditos para el sexo femenino que el masculino pero la cantidad de morosidad con respecto al sexo masculino no varía mucho.

Se visualiza que hay mayor cantidad de créditos para clientes de estado civil casados pero el valor de morosidad no varía mucho con respecto a clientes de estado civil soltero.

Se observa que cuando la instrucción de estudios es secundaria (S) la morosidad es mayor comparado con los de instrucción primaria o con clientes que no tienen instrucción.

Se visualiza que cuando el cliente es empleado independiente la morosidad es mayor y si es empleado público la morosidad es menor.

En los datos de calificación se observa que existen muy pocos créditos de inmobiliario pero que la morosidad es bastante alta.

En los datos de vivienda se observa que cuando el cliente vive con familiares o la vivienda es prestada la morosidad es mayor.

Se aplica el test Chi Cuadrado para evaluar la asociación con un nivel de significancia de 0.05.

Entre la variable “Sexo” y “MoraCredito”.

En la Figura 17 se visualiza los datos retornados.

```
#Se aplica el test Chi Cuadrado para evaluar la asociación
# Entre la variable Sexo y la variable objetivo "MoraCredito"

# Crear la tabla de contingencia
contingenciaTablaSexo = pd.crosstab(df['Sexo'], df['MoraCredito'])
print("Tabla de Contingencia:")
print(contingenciaTablaSexo)

# Aplicar la prueba de chi-cuadrado
chi2, p, dof, expected = chi2_contingency(contingenciaTablaSexo)

print(f"\nChi2: {chi2}")
print(f"P-value: [1]")
print(f"Grados de libertad: {dof}")
print(f"Tabla de frecuencias esperadas:\n{expected}")
```

```
Tabla de Contingencia:
MoraCredito  0    1
Sexo
Femenino    3320  797
Masculino    2786  773

Chi2: 6.394771426430642
P-value: 0.011445696651363621
Degrees of Freedom: 1
Expected Frequencies Table:
[[3274.93512246  842.06487754]
 [2831.06487754  727.93512246]]
```

Figura 17. Test Chi Cuadrado de “Sexo” y “MoraCredito”. Fuente: Elaboración propia.

El test chi-cuadrado muestra una relación significativa entre el sexo y la mora, con un valor de 6.39 y un p-valor de 0.0114. Dado que el p-valor es menor que 0.05, se rechaza la hipótesis nula, indicando que el género influye en la probabilidad de caer en mora.

Entre la variable “EstadoCivil” y “MoraCredito”.

En la Figura 18 se visualiza los datos retornados.

```
# Se aplica el test Chi Cuadrado para evaluar la asociación
# Entre la variable "EstadoCivil" y variable objetivo "MoraCredito"

# Crear la tabla de contingencia
contingenciaTablaCivil = pd.crosstab(df['EstadoCivil'], df['MoraCredito'])
print("Tabla de Contingencia:")
print(contingenciaTablaCivil)

# Aplicar la prueba de chi-cuadrado
chi2, p, dof, expected = chi2_contingency(contingenciaTablaCivil)

print(f"\nChi2: {chi2}")
print(f"P-value: [1]")
print(f"Grados de libertad: {dof}")
print(f"Tabla de frecuencias esperadas:\n{expected}")
```

```
Tabla de Contingencia:
MoraCredito      0      1
EstadoCivil
Casado           3258    726
Divorciado       431    122
Soltero          1956    592
Union Libre       214     88
Viudo            247     42

Chi2: 45.39877696609361
P-value: 3.2848684297392316e-09
Degrees of Freedom: 4
Expected Frequencies Table:
[[3169.13809276  814.86190724]
 [ 439.89291298  113.10708702]
 [2026.84835852  521.15164148]
 [ 240.2308494   61.7691506 ]
 [ 229.88978635  59.11021365]]
```

Figura 18. Test Chi Cuadrado entre “EstadoCivil” y “MoraCredito”. Fuente: Elaboración propia.

El test chi-cuadrado muestra una relación significativa entre el estado civil y la mora, con un valor de 45.40 y un p-valor de 3.28e-09. Dado que el p-valor es mucho menor que 0.05, se rechaza la hipótesis nula, indicando que el estado civil influye considerablemente en la probabilidad de caer en mora.

Entre la variable “Instrucción” y “MoraCredito”.

En la Figura 19 se visualiza los datos retornados.

```
# Se aplica el test Chi Cuadrado para evaluar la asociación
# Entre la variable "Instruccion" y variable objetivo "MoraCredito"
# SIN ESTUDIOS ='N'
# PRIMARIA ='P'
```



```

# SECUNDARIA ='S'
# FORMACIÓN INTERMEDIA (TÉCNICA - TECNOLOGÍA) ='T'
# UNIVERSITARIA ='U'
# SUPERIOR ='G'

# Crear la tabla de contingencia
contingenciaTablaInt = pd.crosstab(df['Instruccion'], df['MoraCredito'])
print("Tabla de Contingencia:")
print(contingenciaTablaInt)

# Aplicar la prueba de chi-cuadrado
chi2, p, dof, expected = chi2_contingency(contingenciaTablaInt)

print(f"\nChi2: {chi2}")
print(f"P-value: [1]")
print(f"Grados de libertad: {dof}")
print(f"Tabla de frecuencias esperadas:\n{expected}")

```

```

Tabla de Contingencia:
MoraCredito    0    1
Instruccion
G              54    0
N             110   23
P            3841  975
S            1560  450
T              29    2
U              512  120

Chi2: 24.024008808624195
P-value: 0.00021481857318553097
Degrees of Freedom: 5
Expected Frequencies Table:
[[ 42.95518499  11.04481501]
 [ 105.7970297  27.2029703 ]
 [3830.96612819 985.03387181]
 [1598.88744138 411.11255862]
 [ 24.65945805   6.34054195]
 [ 502.73475769 129.26524231]]

```

Figura 19. Test Chi Cuadrado entre “Instrucción” y “MoraCredito”. Fuente: Elaboración propia.

El test chi-cuadrado muestra una relación significativa entre el nivel de instrucción y la mora, con un valor de 24.02 y un p-valor de 0.0002. Como el p-valor es mucho menor que 0.05, se rechaza la hipótesis nula, indicando que el nivel de instrucción influye significativamente en la probabilidad de caer en mora.

Entre la variable “Calificacion” y “MoraCredito”.

En la Figura 20 se visualiza los datos retornados.

```

# Se aplica el test Chi Cuadrado para evaluar la asociación
# Entre la variable "Calificacion" y variable objetivo "MoraCredito"

# Crear la tabla de contingencia

```

```
contingenciaTablaCali = pd.crosstab(df['Calificacion'], df['MoraCredito'])
print("Tabla de Contingencia:")
print(contingenciaTablaCali)

# Aplicar la prueba de chi-cuadrado
chi2, p, dof, expected = chi2_contingency(contingenciaTablaCali)

print(f"\nChi2: {chi2}")
print(f"P-value: [1]")
print(f"Grados de libertad: {dof}")
print(f"Tabla de frecuencias esperadas:\n{expected}")
```

```
Tabla de Contingencia:
MoraCredito    0    1
Calificacion
Consumo         211   31
Inmobiliario     8    4
Microcrédito  5865 1534
Vivienda         22    1

Chi2: 13.934564996862072
P-value: 0.002995587617678681
Degrees of Freedom: 3
Expected Frequencies Table:
[[1.92502866e+02  4.94971339e+01]
 [9.54559666e+00  2.45440334e+00]
 [5.88565581e+03  1.51334419e+03]
 [1.82957269e+01  4.70427306e+00]]
```

Figura 20. Test Chi Cuadrado entre Calificacion y MoraCredito.

El test chi-cuadrado revela una relación significativa entre la calificación del crédito y la mora, con un valor de 13.93 y un p-valor de 0.003. Como el p-valor es menor que 0.05, se rechaza la hipótesis nula, indicando que la calificación del crédito influye notablemente en la probabilidad de mora.

Entre la variable “Tipo” y “MoraCredito”.

En la Figura 21 se visualiza los datos retornados.

```
# Se aplica el test Chi Cuadrado para evaluar la asociación
# Entre la variable "Tipo" y variable objetivo "MoraCredito"

# Crear la tabla de contingencia
contingenciaTablaTipo = pd.crosstab(df['tipo'], df['MoraCredito'])
print("Tabla de Contingencia:")
print(contingenciaTablaTipo)

# Aplicar la prueba de chi-cuadrado
chi2, p, dof, expected = chi2_contingency(contingenciaTablaTipo)
```

```
print(f"\nChi2: {chi2}")
print(f"P-value: [1]")
print(f"Grados de libertad: {dof}")
print(f"Tabla de frecuencias esperadas:\n{expected}")
```

```
Tabla de Contingencia:
MoraCredito      0      1
tipo
Ordinario Cuota Fija      5141  1299
Ordinario Cuota Variable  965   271

Chi2: 1.8561656562765485
P-value: 0.17306799154532593
Degrees of Freedom: 1
Expected Frequencies Table:
[[5122.80354351  1317.19645649]
 [ 983.19645649   252.80354351]]
```

Figura 21. Test Chi Cuadrado entre “Tipo” y “MoraCredito”. Fuente: Elaboración propia.

El test chi-cuadrado muestra que no hay una relación significativa entre el tipo de crédito (Ordinario Cuota Fija o Variable) y la mora, con un valor de 1.86 y un p-valor de 0.173. Dado que el p-valor es mayor que 0.05, no se rechaza la hipótesis nula, indicando que el tipo de crédito no afecta significativamente la probabilidad de mora.

Entre la variable “vivienda” y “MoraCredito”.

En la Figura 22 se visualiza los datos retornados.

```
# Se aplica el test Chi Cuadrado para evaluar la asociación
# Entre la variable "vivienda" y variable objetivo "MoraCredito"

# Crear la tabla de contingencia
contingenciaTablavivienda = pd.crosstab(df['vivienda'], df['MoraCredito'])
print("Tabla de Contingencia:")
print(contingenciaTablavivienda)

# Aplicar la prueba de chi-cuadrado
chi2, p, dof, expected = chi2_contingency(contingenciaTablavivienda)

print(f"\nChi2: {chi2}")
print(f"P-value: [1]")
print(f"Grados de libertad: {dof}")
print(f"Tabla de frecuencias esperadas:\n{expected}")
```

Tabla de Contingencia:

MoraCredito	0	1
vivienda		
ARRENDADA	114	33
PRESTADA	37	18
PROPIA HIPOTECADA	96	24
PROPIA NO HIPOTECADA	2942	521
VIVE CON FAMILIARES	2917	974

Chi2: 117.86996690905632
P-value: 1.5224473564062367e-24
Degrees of Freedom: 4
Expected Frequencies Table:

[116.93355915	30.06644085]
[43.75065138	11.24934862]
[95.45596665	24.54403335]
[2754.70010422	708.29989578]
[3095.1597186	795.8402814]

Figura 22. Test Chi Cuadrado entre “vivienda” y “MoraCredito”. Fuente: Elaboración propia.

El test chi-cuadrado muestra una relación significativa entre el tipo de vivienda y la mora en el crédito, con un valor de 117.87 y un p-valor de 1.52e-24. Este p-valor, mucho menor que 0.05, indica que el tipo de vivienda influye significativamente en la probabilidad de mora.

3.3.4. Preparación y transformación de variables

Se convierte la variable “Sexo” y “EstadoCivil” en variables *dummies*. Se añaden nuevas variables de acuerdo a las categorías existentes en cada variable. En la Figura 23 se visualiza los datos retornados.

```
#Se convierte la categoría “Sexo”, en variable dummies

df = pd.get_dummies(df, columns=['Sexo'], dtype=int)

#Se convertir la categoría “EstadoCivil” en variable dummies
df = pd.get_dummies(df, columns=['EstadoCivil'], dtype=int)
df.info()
```

```

Data columns (total 27 columns):
#   Column                                Non-Null Count  Dtype
---  -
0   FechaNacimiento                       7676 non-null   object
1   Instruccion                           7676 non-null   object
2   CodigoBarrio                          7676 non-null   int64
3   NumeroCargas                          7676 non-null   int64
4   TipoActividad                         7676 non-null   object
5   NumeroCredito                         7676 non-null   int64
6   Monto                                 7676 non-null   float64
7   NumeroCuotas                          7676 non-null   int64
8   ValorCuota                            7676 non-null   float64
9   FechaEntrega                          7676 non-null   object
10  Tasa                                  7676 non-null   float64
11  Calificacion                           7676 non-null   object
12  encaje                                7676 non-null   float64
13  tipo                                  7676 non-null   object
14  vivienda                              7676 non-null   object
15  totalActivo                           7676 non-null   float64
16  totalPasivo                           7676 non-null   float64
17  Ingresos                              7676 non-null   float64
18  Gastos                                7676 non-null   float64
19  MoraCredito                           7676 non-null   int64
20  Sexo_Femenino                         7676 non-null   int64
21  Sexo_Masculino                        7676 non-null   int64
22  EstadoCivil_Casado                    7676 non-null   int64
23  EstadoCivil_Divorciado                7676 non-null   int64
24  EstadoCivil_Soltero                   7676 non-null   int64
25  EstadoCivil_Union Libre                7676 non-null   int64
26  EstadoCivil_Viudo                     7676 non-null   int64
dtypes: float64(8), int64(12), object(7)
  
```

Figura 23. Información de *DataFrame* con las nuevas variables creadas. Fuente: Elaboración propia.

Se separa los datos de “FechaNacimiento” para calcular posteriormente la columna “EdadAlCredito”. En la Figura 24 se visualiza los datos retornados.

```

# Separamos los datos de “FechaNacimiento” para calcular la columna “EdadAlCredito”

df[["DiaNacimiento",          "MesNacimiento",          "AnioNacimiento"]] =
df["FechaNacimiento"].str.split("/", n = 2, expand = True)
  
```

AnioNacimiento	
1991	286
1981	259
1990	257
1984	253
1992	243
...	...
1942	3
1937	3
1936	2
1935	2
1940	2
70 rows x 1 columns	

Figura 24. Distribución de los años de nacimiento. Fuente: Elaboración propia.

Se define un mapeo para convertir la variable “Instrucción” de categórica a numérica. El orden es ascendente desde estudios básicos a superiores. En la Figura 25 se visualiza los datos retornados.

```

# Definir un mapeo para convertir de Instruccion categorica a numerica
#El orden es desde estudios básicos a superiores
mapeo_instruccion = {
    'N': 0, # SIN ESTUDIOS
    'P': 1, # PRIMARIA
    'S': 2, # SECUNDARIA
    'T': 3, # FORMACIÓN INTERMEDIA (TÉCNICA - TECNOLOGÍA)
    'U': 4, # UNIVERSITARIA
    'G': 5 # SUPERIOR
}

df['Instruccion'] = df['Instruccion'].map(mapeo_instruccion)
df["Instruccion"].value_counts()

```

Instruccion	
1	4816
2	2010
4	632
0	133
5	54
3	31
dtype: int64	

Figura 25. Distribución de “Instruccion” con valores numéricos. Fuente: Elaboración propia.

Se observa la categoría “CodigoBarrio”. En la Figura 26 se visualiza los datos retornados.

```

# Se observa la categoría “CodigoBarrio”
df["CodigoBarrio"].value_counts()

```

CodigoBarrio	
18095004	319
18096801	298
18096804	273
18096504	234
18096606	211
...	...
17011901	1
17013001	1
18045024	1
18016306	1
18085201	1
318 rows × 1 columns	

Figura 26. Distribución de frecuencias de “CodigoBarrio”. Fuente: Elaboración propia.

Se observa la variable “NumeroCargas”. En la Figura 27 se visualiza los datos retornados.

```

# Se observa la variable “NumeroCargas”
df["NumeroCargas"].value_counts()

```

NumeroCargas	
0	5142
1	1216
2	917
3	311
4	74
5	12
6	4

dtype: int64

Figura 27. Distribución de frecuencias de “NumeroCargas”. Fuente: Elaboración propia.

Se convierte la variable “TipoActividad” en numérica con el método *Label Encoding*. En la Figura 28 se visualiza los datos retornados.

```
#Se convierte la categoría “TipoActividad” en numérica
# Con el método Label Encoding

# Instanciamos el LabelEncoder
LabelEncoderAct = LabelEncoder()

df['TipoActividad'] = LabelEncoderAct.fit_transform(df['TipoActividad'])

# Ver el mapeo
mapping = dict(zip(LabelEncoderAct.classes_,
LabelEncoderAct.transform(LabelEncoderAct.classes_)))
print(mapping)

df[“TipoActividad”].value_counts()
```

{ 'Empleado Privado': 0, 'Empleado Publico': 1, 'Independiente': 2 }	
count	
TipoActividad	
2	6339
0	1214
1	123

dtype: int64

Figura 28. Distribución de frecuencias de la variable “TipoActividad” de tipo numérico. Fuente: Elaboración propia.

Se observa la variable “NumeroCredito”.

```
#Se observa “NumeroCredito”
df[“NumeroCredito”].value_counts()
```

NumeroCredito	
4729	8
9389	8
2219	6
3003	6
1253	6
...	...
5900	1
6782	1
4698	1
370	1
547	1
7600 rows × 1 columns	

Figura 29. Distribución de frecuencias de la variable “NumeroCredito”. Fuente: Elaboración propia.

Se decide eliminar la variable “NumeroCredito” por ser un valor único que no aporta valor a la predicción.

```
# Se decide eliminar la variable “NumeroCredito”
# por ser un valor único que no aporta valor a la predicción
# de MoraCredito

df.drop(“NumeroCredito”, axis = 1, inplace = True)
```

Se observa la variable “NumeroCuotas”. En la Figura 30 se visualiza los datos retornados.

```
# Se observa la variable “NumeroCuotas”
df[“NumeroCuotas”].value_counts()
```

NumeroCuotas	
12	1985
24	1319
36	509
18	475
8	340
...	...
80	1
58	1
96	1
82	1
73	1
68 rows × 1 columns	

Figura 30. Distribución de frecuencias de la variable “NumeroCuotas”. Fuente: Elaboración propia.

Se observa la categoría “FechaEntrega” y se obtiene el año para calcular posteriormente la columna “EdadAlCredito”. En la Figura 31 se visualiza los datos retornados.

```
# Se observa la categoría “FechaEntrega”
# Separamos los datos de “FechaEntrega” para calcular posteriormente la columna
“EdadAlCredito”
```



```

df["FechaEntrega"].value_counts()
df[["DiaEntrega", "MesEntrega", "AnioEntrega"]] = df["FechaEntrega"].str.split("/", n = 2,
expand = True)
df["AnioEntrega"].value_counts()

```

2019	943
2020	837
2018	766
2021	743
2017	688
2016	672
2022	637
2014	604
2015	549
2013	400
2011	325
2012	318
2023	193
2024	1

dtype: int64

Figura 31. Distribución de frecuencias de la variable "FechaEntrega". Fuente: Elaboración propia.

Se calcula una nueva variable "EdadAlCredito". En la Figura 32 se visualiza los datos retornados.

```

# Se cambia los datos de las columnas "AñoNacimiento" y "AñoEntrega" a números, para calcular
la columna "EdadAlCredito"
cambio_anios = {"AnioNacimiento": int, "AnioEntrega": int}
df = df.astype(cambio_anios)

# Creamos la columna "EdadAlCredito"
df["EdadAlCredito"] = df["AnioEntrega"] - df["AnioNacimiento"]
# Se elimina la variable "FechaNacimiento"
df.drop("FechaNacimiento", axis = 1, inplace = True)
# Se elimina la variable "FechaEntrega"
df.drop("FechaEntrega", axis = 1, inplace = True)
# Se elimina la variable "DiaNacimiento"
df.drop("DiaNacimiento", axis = 1, inplace = True)
# Se elimina la variable "MesNacimiento"
df.drop("MesNacimiento", axis = 1, inplace = True)
# Se elimina la variable "AnioNacimiento"
df.drop("AnioNacimiento", axis = 1, inplace = True)
# Se elimina la variable "DiaEntrega"
df.drop("DiaEntrega", axis = 1, inplace = True)
# Se elimina la variable "MesEntrega"
df.drop("MesEntrega", axis = 1, inplace = True)

```

```
# Se elimina la variable "AnioEntrega"
df.drop("AnioEntrega", axis = 1, inplace = True)
df["EdadAlCredito"].value_counts()
```

EdadAlCredito	
25	271
26	263
29	262
23	255
24	252
...	...
73	6
84	1
79	1
80	1
78	1
64 rows x 1 columns	

Figura 32. Distribución de frecuencias de la variable "EdadAlCredito". Fuente: Elaboración propia.

Se observa en la Figura 33 la nueva variable creada con un gráfico de caja.

La mediana de edad al solicitar crédito es de 35 años, lo que indica que el 50% de los solicitantes tienen 35 años o menos. El rango intercuartílico abarca de 30 a 45 años. Los bigotes muestran un rango de edad de 20 a 60 años, y los círculos por encima de 80 años representan valores atípicos. La mayoría de los clientes tienen entre 30 y 45 años al solicitar crédito.

```
# Se observa la nueva variable creada con un gráfico de caja
df.boxplot(column=["EdadAlCredito"])
plt.title("EdadAlCredito")
plt.show()
```

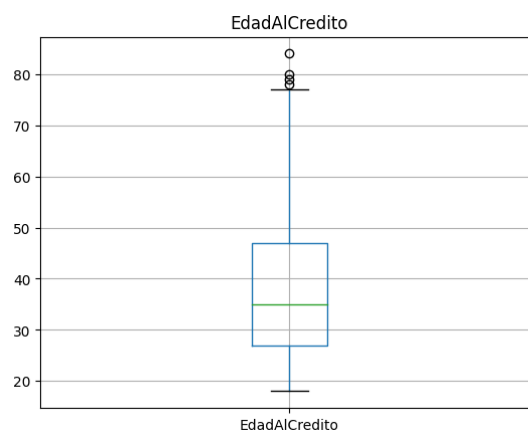


Figura 33. Gráfico de caja de la variable "EdadAlCredito". Fuente: Elaboración propia.

Se crean grupos con las edades para facilitar su distribución en Juvenil, Adulto y Senior. En la Figura 34 se visualiza los datos retornados.

```
# Crear una función para clasificar las edades en grupos personalizados
```

```

def clasificar_edad_personalizada(edad):
    if edad < 30:
        return 'Juvenil'
    elif 30 <= edad < 50:
        return 'Adulto'
    else:
        return 'Senior'

# Aplicar la función a la columna de edad
df['GrupoEdad'] = df['EdadAlCredito'].apply(clasificar_edad_personalizada)
# Eliminar la variable "EdadAlCredito"
df.drop("EdadAlCredito", axis = 1, inplace = True)

# Mapeo de categorías a valores numéricos
mapeo_edad = {'Juvenil': 0, 'Adulto': 1, 'Senior': 2}

# Aplicar el mapeo a la columna 'GrupoEdad'
df['GrupoEdad'] = df['GrupoEdad'].map(mapeo_edad)

# Verificar los valores numéricos en 'GrupoEdad'
print(df["GrupoEdad"].value_counts())
  
```

```

GrupoEdad
1    3423
0    2586
2    1667
Name: count, dtype: int64
  
```

Figura 34. Distribución de frecuencias de la nueva variable “GrupoEdad”. Fuente: Elaboración propia.

Se observa la categoría “Tasa”. En la Figura 35 se visualiza los datos retornados.

```

# Se observa la categoría “Tasa”
df["Tasa"].value_counts()
  
```

```

Tasa
21.50    2601
22.50     776
21.00     569
23.00     536
19.75     438
22.44     398
20.50     377
19.97     350
19.50     271
23.50     234
18.00     175
24.50     151
22.00      98
  
```

Figura 35. Distribución de frecuencias de la variable “Tasa”.

Se convierte la categoría “Calificacion” en numérica con el método *Label Encoding*. En la Figura 36 se visualiza los datos retornados.

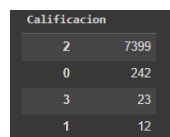
```
#Se convierte la categoría “Calificacion” en numérica
# Con el método Label Encoding

# Instanciamos el LabelEncoder
LabelEncoderCali = LabelEncoder()

df['Calificacion'] = LabelEncoderCali.fit_transform(df['Calificacion'])

# Ver el mapeo
mapping = dict(zip(LabelEncoderCali.classes_,
LabelEncoderCali.transform(LabelEncoderCali.classes_)))
print(mapping)

df[“Calificacion”].value_counts()
```



Calificacion	
2	7399
0	242
3	23
1	12

Figura 36. Distribución de frecuencias de la variable “Calificacion” de tipo numérico. Fuente: Elaboración propia.

Se convierte la categoría “Tipo” en numérica con el método *Label Encoding*.

```
#Se convierte la categoría “Tipo” en numérica
# Con el método Label Encoding

# Instanciamos el LabelEncoder
LabelEncoderTipo = LabelEncoder()

df['tipo'] = LabelEncoderTipo.fit_transform(df['tipo'])

# Ver el mapeo
mapping = dict(zip(LabelEncoderTipo.classes_,
LabelEncoderTipo.transform(LabelEncoderTipo.classes_)))
print(mapping)

df[“tipo”].value_counts()
```

tipo	
0	6440
1	1236

Figura 37. Distribución de frecuencias de la variable “Tipo” como numérica. Fuente: Elaboración propia.

Se convierte la variable “vivienda” en variables *dummies*. En la Figura 38 se visualiza los datos retornados.

#Se convierte la categoría “vivienda” en variable dummies

```
df = pd.get_dummies(df, columns=['vivienda'], dtype=int)
df.info()
```

```
# Column Non-Null Count Dtype
---
0 Instrucion 7676 non-null int64
1 Codigobarrio 7676 non-null int64
2 NumeroCargas 7676 non-null int64
3 TipoActividad 7676 non-null int64
4 Monto 7676 non-null float64
5 NumeroCuotas 7676 non-null int64
6 ValorCuota 7676 non-null float64
7 Tasa 7676 non-null float64
8 Calificacion 7676 non-null int64
9 encaje 7676 non-null float64
10 tipo 7676 non-null int64
11 totalActivo 7676 non-null float64
12 totalPasivo 7676 non-null float64
13 Ingresos 7676 non-null float64
14 Gastos 7676 non-null float64
15 MoraCredito 7676 non-null int64
16 Sexo_Femenino 7676 non-null int64
17 Sexo_Masculino 7676 non-null int64
18 EstadoCivil_Casado 7676 non-null int64
19 EstadoCivil_Divorciado 7676 non-null int64
20 EstadoCivil_Soltero 7676 non-null int64
21 EstadoCivil_Union Libre 7676 non-null int64
22 EstadoCivil_Viudo 7676 non-null int64
23 GrupoEdad 7676 non-null int64
24 vivienda_ARRENDADA 7676 non-null int64
25 vivienda_PRESTADA 7676 non-null int64
26 vivienda_PROPIA HIPOTECADA 7676 non-null int64
27 vivienda_PROPIA NO HIPOTECADA 7676 non-null int64
28 vivienda_VIVE CON FAMILIARES 7676 non-null int64
dtypes: float64(8), int64(21)
memory usage: 1.8 MB
```

Figura 38. Información del *DataFrame* con las nuevas variables de “vivienda”. Fuente: Elaboración propia.

Se calcula la diferencia entre “Activos – Pasivos” y se identifica como “Patrimonio”. En la Figura 39 se visualiza los datos retornados.

Se calcula la diferencia entre Activos - Pasivos

```
df['Patrimonio'] = df['totalActivo'] - df['totalPasivo']
df["Patrimonio"].value_counts()
```

Patrimonio	
0.00	2444
1500.00	44
1000.00	39
5000.00	35
4000.00	32
...	...
12999.99	1
1920.00	1
3199.99	1
19263.81	1
34900.00	1
1234 rows x 1 columns	

Figura 39. Distribución de frecuencias de la variable “Patrimonio”. Fuente: Elaboración propia.

Se calcula la diferencia entre “Ingresos – Gastos” y se identifica como “Disponible”. En la Figura 40 se visualiza los datos retornados.

```
# Se calcula la diferencia entre Ingresos - Gastos

df['Disponible'] = df['totalActivo'] - df['totalPasivo']
df["Disponible"].value_counts()
```

Disponible	
0.00	2444
1500.00	44
1000.00	39
5000.00	35
4000.00	32
...	...
12999.99	1
1920.00	1
3199.99	1
19263.81	1
34900.00	1
1234 rows x 1 columns	

Figura 40. Distribución de frecuencias de la nueva variable “Disponible”. Fuente: Elaboración propia.

Se observa la categoría a predecir “MoraCredito” y se visualiza un desbalance en los valores como se observa en la Figura 41. Luego se aplicará un método de balanceo.

```
# Se observa la categoría a predecir “MoraCredito”
df["MoraCredito"].value_counts()
```

MoraCredito	
0	6106
1	1570

Figura 41. Distribución de la variable a predecir “MoraCredito”. Fuente: Elaboración propia.

Se verifica que las variables sean numéricas para implementarlas en modelos de Aprendizaje Automático como se observa en la Figura 42.

```
# Revisamos que las variables estén en el formato numérico
df.info()
```

```
0 Instruccion          7676 non-null   int64
1 CodigoBarrio         7676 non-null   int64
2 NumeroCargas         7676 non-null   int64
3 TipoActividad        7676 non-null   int64
4 Monto                7676 non-null   float64
5 NumeroCuotas         7676 non-null   int64
6 ValorCuota           7676 non-null   float64
7 Tasa                 7676 non-null   float64
8 Calificacion         7676 non-null   int64
9 encaje              7676 non-null   float64
10 tipo               7676 non-null   int64
11 totalActivo         7676 non-null   float64
12 totalPasivo         7676 non-null   float64
13 Ingresos            7676 non-null   float64
14 Gastos              7676 non-null   float64
15 MoraCredito         7676 non-null   int64
16 Sexo_Femenino       7676 non-null   int64
17 Sexo_Masculino      7676 non-null   int64
18 EstadoCivil_Casado  7676 non-null   int64
19 EstadoCivil_Divorciado 7676 non-null   int64
20 EstadoCivil_Soltero 7676 non-null   int64
21 EstadoCivil_Union Libre 7676 non-null   int64
22 EstadoCivil_Viudo   7676 non-null   int64
23 GrupoEdad          7676 non-null   int64
24 vivienda_ARRENDADA  7676 non-null   int64
25 vivienda_PRESTADA  7676 non-null   int64
26 vivienda_PROPIA HIPOTECADA 7676 non-null   int64
27 vivienda_PROPIA NO HIPOTECADA 7676 non-null   int64
28 vivienda_VIVE CON FAMILIARES 7676 non-null   int64
29 Patrimonio         7676 non-null   float64
30 Disponible         7676 non-null   float64
dtypes: float64(10), int64(21)
memory usage: 1.9 MB
```

Figura 42. Información de *DataFrame* con las variables pre definitivas. Fuente: Elaboración propia.

Se calcula la matriz de correlación y se muestra el mapa de calor con el método de *Pearson* de todas las variables con respecto a “MoraCredito”. Se elige este método considerando que se requiere observar la correlación de las variables explicativas cuantitativas con las variables “MoraCredito” que es dicotómica con valores de 0 (no mora) o 1(mora). En la Figura 43 se visualiza los datos retornados.

```
# Calcular la matriz de correlación de Pearson
MatrizCorrelacionPearson = df.corr(method='pearson')

# Crear un mapa de calor utilizando seaborn
plt.figure(figsize=(10, 8)) # Tamaño del gráfico
sns.heatmap(MatrizCorrelacionPearson, annot=True, cmap='coolwarm', fmt='.2f',
linewidths=0.5)
plt.title('Mapa de Calor de la Matriz de Correlación de Pearson')
plt.show()
```

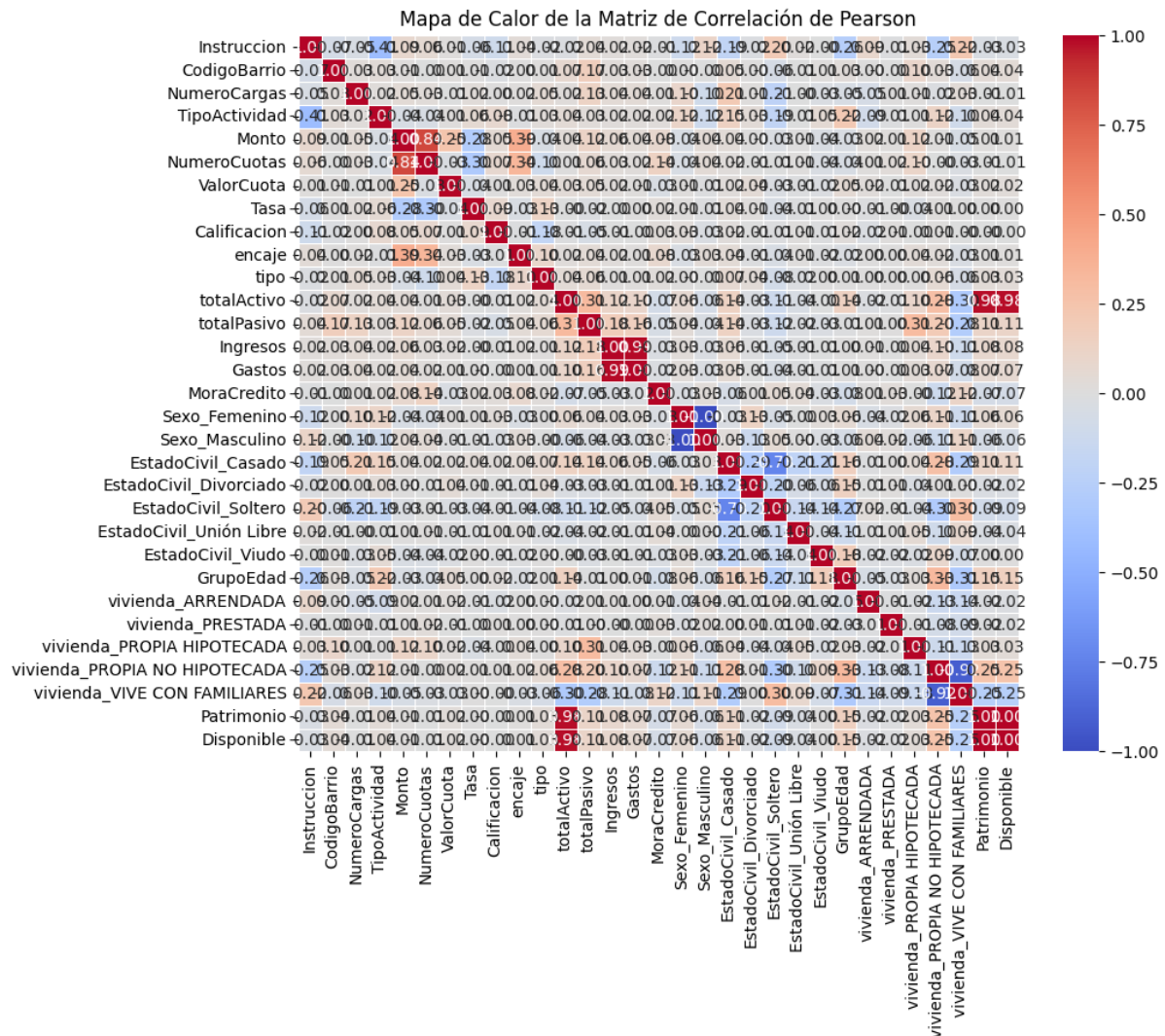


Figura 43. Mapa de calor, con el método de Pearson. Fuente: Elaboración propia.

Para una mejor observación se obtiene la matriz de correlación con el método de *Pearson*. En la Figura 44 se visualiza los datos retornados.

```
# Calcula la correlación de Pearson de todas las variables con respecto a 'MoraCredito'
correlacion_pearson = df.corr(method='pearson')['MoraCredito'].sort_values(ascending=False)

# Imprime el resultado
print(correlacion_pearson)
```


MoraCredito	1.000000
NumeroCuotas	0.139309
vivienda_VIVE CON FAMILIARES	0.115094
encaje	0.084312
Monto	0.080868
EstadoCivil_Soltero	0.048592
EstadoCivil_Unión Libre	0.043578
Sexo_Masculino	0.029187
Calificacion	0.028785
vivienda_PRESTADA	0.025850
Tasa	0.024919
TipoActividad	0.016186
tipo	0.015990
NumeroCargas	0.013907
EstadoCivil_Divorciado	0.011109
vivienda_ARRENDADA	0.006913
CodigoBarrio	-0.000592
vivienda_PROPIA HIPOTECADA	-0.001416
Instruccion	-0.013005
Gastos	-0.023136
EstadoCivil_Viudo	-0.029032
Sexo_Femenino	-0.029187
Ingresos	-0.030324
ValorCuota	-0.032557
totalPasivo	-0.052567
EstadoCivil_Casado	-0.057442
Patrimonio	-0.065605
Disponible	-0.065605
totalActivo	-0.073512
GrupoEdad	-0.083544
vivienda_PROPIA NO HIPOTECADA	-0.121569
Name: MoraCredito, dtype: float64	

Figura 44. Matriz de correlación con el método Pearson. Fuente: Elaboración propia.

Los resultados de la correlación de Pearson muestran que la variable “NumeroCuotas” tiene la mayor correlación positiva con “MoraCredito” (0.139), sugiriendo que más cuotas están ligeramente asociadas con un mayor riesgo de mora. La variable “vivienda_PROPIA NO HIPOTECADA” muestra la correlación negativa más significativa (-0.122), indicando que tener una vivienda propia sin hipotecar está asociado con un menor riesgo de mora. Las correlaciones para variables como “Monto” y “encaje” son moderadas, mientras que las correlaciones negativas con “GrupoEdad”, “Patrimonio” y “totalActivo” sugieren que personas mayores y aquellos con más activos tienen menor probabilidad de incurrir en mora. Las variables como “Ingresos”, “Gastos” y “EstadoCivil” tienen correlaciones débiles con la mora..

De acuerdo a la matriz observada se decide eliminar la variable “CodigoBarrio” por no tener asociación con la variable a predecir.

Se muestra el *DataFrame* después de preprocesar su información y acondicionar las variables para implementarlas en los modelos de ML. En la Figura 45 se visualiza los datos retornados.

```
df.drop("CodigoBarrio", axis = 1, inplace = True)
df.info()
```

#	Column	Non-Null Count	Dtype
0	Instruccion	7676 non-null	int64
1	NumeroCargas	7676 non-null	int64
2	TipoActividad	7676 non-null	int64
3	Monto	7676 non-null	float64
4	NumeroCuotas	7676 non-null	int64
5	ValorCuota	7676 non-null	float64
6	Tasa	7676 non-null	float64
7	Calificacion	7676 non-null	int64
8	encaje	7676 non-null	float64
9	tipo	7676 non-null	int64
10	totalActivo	7676 non-null	float64
11	totalPasivo	7676 non-null	float64
12	Ingresos	7676 non-null	float64
13	Gastos	7676 non-null	float64
14	MoraCredito	7676 non-null	int64
15	Sexo_Femenino	7676 non-null	int64
16	Sexo_Masculino	7676 non-null	int64
17	EstadoCivil_Casado	7676 non-null	int64
18	EstadoCivil_Divorciado	7676 non-null	int64
19	EstadoCivil_Soltero	7676 non-null	int64
20	EstadoCivil_Union Libre	7676 non-null	int64
21	EstadoCivil_Viudo	7676 non-null	int64
22	GrupoEdad	7676 non-null	int64
23	vivienda_ARRENDADA	7676 non-null	int64
24	vivienda_PRESTADA	7676 non-null	int64
25	vivienda_PROPIA HIPOTECADA	7676 non-null	int64
26	vivienda_PROPIA NO HIPOTECADA	7676 non-null	int64
27	vivienda_VIVE CON FAMILIARES	7676 non-null	int64
28	Patrimonio	7676 non-null	float64
29	Disponible	7676 non-null	float64

dtypes: float64(10), int64(20)
memory usage: 1.8 MB

Figura 45. Información del *DataFrame* con las variables definitivas. Fuente: Elaboración propia.

3.4.Fase de modelado

Para la fase de modelado se escoge trabajar con modelos de Aprendizaje Automático supervisado, considerando que el objetivo es la predicción de la variable “MoraCredito”.

Se cuenta con la variable explicada en formato binario “MoraCredito” y las variables explicativas en formato numérico.

Se separa las variables independientes (x) y la variable dependiente (y).

```
# Se separa las variables independientes (x) y la variable dependiente (y)
x_data = df.drop(["MoraCredito"], axis = 1)
y_data = df["MoraCredito"]
```

Se separa los datos para pruebas y entrenamiento, definiendo un 30% para pruebas y el 70% de datos para entrenamiento.

```
# separar los datos en entrenamiento y prueba
x_val, x_test_, y_val, y_test_ = train_test_split(x_data, y_data, test_size = 0.3, stratify
= y_data, random_state = 42)
```

Se observa cuantos datos se posee para entrenamiento. En la Figura 46 se visualiza los datos retornados.

```
# cuantos datos tenemos de entrenamiento
len(x_val)
```

5373

Figura 46. Número de registros para entrenamiento. Fuente: Elaboración propia.

Se observa cuantos datos se posee para pruebas. En la Figura 47 se visualiza los datos retornados.

```
# cuantos datos tenemos de prueba
len(x_test_)
```

2303

Figura 47. Número de registros para pruebas. Fuente: Elaboración propia.

Se observa la distribución de la variable a predecir con datos de entrenamiento. En la Figura 48 se visualiza los datos retornados.

```
# Se observa como es la partición de la variable a predecir
# de los datos de entrenamiento
y_val.value_counts()
```

MoraCredito	
0	4274
1	1099

Figura 48. Distribución de frecuencias de la variable "MoraCredito" con datos de entrenamiento. Fuente: Elaboración propia.

Se observa la distribución de la variable a predecir con datos de prueba. En la Figura 49 se visualiza los datos retornados.

```
# Se observa como es la particion de la variable a predecir
# de los datos de prueba
y_test_.value_counts()
```

MoraCredito	
0	1832
1	471

Figura 49. Distribución de frecuencias de la variable "MoraCredito" con datos de prueba. Fuente: Elaboración propia.

Como es evidente se observa un desbalance en los valores de "MoraCredito", existiendo mayor cantidad de valores 0.

3.4.1. Balanceo de datos con sobre muestreo aleatorio

Para balancear los datos se aplica el sobre muestreo aleatorio sobre los datos de entrenamiento. La estrategia de esta técnica es aumentar la cantidad de la clase minoritaria hasta alcanzar el 80% de la cantidad de la clase mayoritaria. Se decide esta técnica considerando que hay más valores en “MoraCredito” = 0 y requerimos balancear los valores de “MoraCredito” = 1. En la Figura 50 se visualiza los datos retornados.

```
# Se aplica sobremuestreo aleatorio sobre los datos de entrenamiento

ROS = RandomOverSampler(sampling_strategy = 0.8, random_state = 1)
balance_training_x, balance_training_y = ROS.fit_resample(x_val, y_val)
```

Se observa los datos de “MoraCredito” después del sobre muestreo. En la Figura 56 se visualiza los datos retornados.

```
# Se observan los datos de “MoraCredito” después del sobremuestreo
balance_training_y[“MoraCredito”].value_counts()
```

MoraCredito	
0	4274
1	3419

Figura 50. Distribución de frecuencia de “MoraCredito” después del sobre muestreo en datos de entrenamiento. Fuente: Elaboración propia.

3.4.2. Modelo Regresión Logística con sobre muestreo aleatorio.

```
##### Regresion Logística #####
# Instanciamos el modelo
modelo_regresion_log = LogisticRegression(random_state=42, max_iter=1000,
solver='liblinear')
# Entrenamos el modelo con los datos de entrenamiento (train)
modelo_regresion_log = modelo_regresion_log.fit(balance_training_x, balance_training_y)
# Hacemos la predicción del modelo entrenado sobre los datos de prueba (test)
y_pred_rl = modelo_regresion_log.predict(x_test_)
```

Se obtiene la matriz de confusión para evaluar el rendimiento del modelo. En la Figura 51 se visualiza los datos retornados.

```
# Matriz de confusión para evaluar el rendimiento del modelo
# Con Regresión Logística

confusion_rl = confusion_matrix(y_test_, y_pred_rl)
confusion_rl
```

```
array([[1435, 397],
       [ 269, 202]])
```

Figura 51. Matriz de confusión del modelo Regresión Logística. Fuente: Elaboración propia

Es preciso recordar que está tratando de predecir la variable MoraCredito donde:

0 = No tiene mora (sin mora, clase negativa).

1 = Tiene mora (con mora, clase positiva).

- **1435 (Verdaderos negativos):** Clientes sin mora correctamente clasificados.
- **397 (Falsos positivos):** Clientes sin mora clasificados incorrectamente como con mora.
- **269 (Falsos negativos):** Clientes con mora clasificados incorrectamente como sin mora.
- **202 (Verdaderos positivos):** Clientes con mora correctamente clasificados.

El modelo clasifica correctamente a 1435 clientes sin mora y 202 con mora, pero comete 397 falsos positivos, afectando la precisión en clientes sin mora, y 269 falsos negativos, lo que representa un riesgo al no identificar muchos clientes con mora.

Se obtiene el índice de precisión del modelo con Regresión Logística. En la Figura 52 se visualiza los datos retornados.

```
# Evaluamos el accuracy del modelo entrenado
# Con Regresión Logistica
accuracy_score(y_test_, y_pred_rl)
```

```
0.7108119843682154
```

Figura 52. Valor de precisión del modelo con Regresión Logística. Fuente: Elaboración propia.

3.4.3. Modelo Árbol de decisión con sobre muestreo aleatorio

```
##### Arbol de decision #####
```

```
# Instanciamos el modelo
modelo_arbol = DecisionTreeClassifier(max_depth = 5, random_state = 42)

# Entrenamos el modelo con los datos de entrenamiento (train)
modelo_arbol = modelo_arbol.fit(balance_training_x, balance_training_y)

# Hacemos la predicción del modelo entrenado sobre los datos de prueba (test)
y_pred_ad = modelo_arbol.predict(x_test_)
```

Se obtiene la matriz de confusión para evaluar el rendimiento del modelo. En la Figura 53 se visualiza los datos retornados.

```
# Matriz de confusión para evaluar el modelo
confusion_ad = confusion_matrix(y_test_, y_pred_ad)
confusion_ad
```

```
array([[1246,  586],
       [ 190,  281]])
```

Figura 53. Matriz de confusión del modelo con Árbol de decisión. Fuente: Elaboración propia

- **1246 (Verdaderos negativos):** Clientes sin mora correctamente clasificados.
- **586 (Falsos positivos):** Clientes sin mora clasificados incorrectamente como con mora.
- **190 (Falsos negativos):** Clientes con mora clasificados incorrectamente como sin mora.
- **281 (Verdaderos positivos):** Clientes con mora correctamente clasificados.

El modelo clasifica correctamente a 1246 clientes sin mora y 281 con mora, pero comete 586 falsos positivos, afectando la confianza en clientes sin mora, y 190 falsos negativos, lo que implica riesgos al no identificar algunos clientes con mora.

Se obtiene la precisión del modelo con Árbol de Decisión. En la Figura 54 se visualiza los datos retornados.

```
# Evaluamos el accuracy del modelo entrenado
accuracy_score(y_test_, y_pred_ad)
```

```
0.6630481980026053
```

Figura 54. Valor de precisión del modelo con Árbol de Decisión. Fuente: Elaboración propia.

3.4.4. Modelo *Random Forest* con sobre muestreo aleatorio

```
##### Random Forest #####

# Instanciamos el modelo
modelo_bosque = RandomForestClassifier(n_estimators = 500, random_state = 42, max_depth = 5)

# Entrenamos el modelo con los datos de entrenamiento (train)
modelo_bosque.fit(balance_training_x, balance_training_y)

# Hacemos la predicción del modelo entrenado sobre los datos de prueba (test)
y_pred_ba = modelo_bosque.predict(x_test_)
```

Se obtiene la matriz de confusión para evaluar el rendimiento del modelo. En la Figura 55 se visualiza los datos retornados.

```
# Matriz de confusión para evaluar el modelo

confusion_ba = confusion_matrix(y_test, y_pred_ba)
confusion_ba
```

```
array([[1428, 404],
       [ 254, 217]])
```

Figura 55. Matriz de confusión con el modelo *Random Forest*. Fuente: Elaboración propia.

- **1428 (Verdaderos negativos):** Clientes sin mora correctamente clasificados.
- **404 (Falsos positivos):** Clientes sin mora clasificados incorrectamente como con mora.
- **254 (Falsos negativos):** Clientes con mora clasificados incorrectamente como sin mora.
- **217 (Verdaderos positivos):** Clientes con mora correctamente clasificados.

El análisis refleja que el modelo clasifica bien a los clientes sin mora, con 1428 verdaderos negativos, pero genera 404 falsos positivos, lo que puede afectar la confianza de clientes clasificados erróneamente. Los 254 falsos negativos que representan clientes con mora que no son identificados, puede ser un riesgo financiero importante. Aunque detecta 217 verdaderos positivos.

Se obtiene la precisión del modelo con *Random Forest*. En la Figura 56 se visualiza los datos retornados.

```
# Evaluamos el accuracy del modelo entrenado
accuracy_score(y_test_, y_pred_ba)
```

```
0.7142857142857143
```

Figura 56. Valor de precisión del modelo con *Random Forest*.

3.4.5. Modelo *XGBoost* con sobre muestreo aleatorio

```
# Crear una instancia del clasificador XGBoost
import xgboost as xgb
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix
xgb_model = xgb.XGBClassifier(
    n_estimators=100,      # Número de árboles
    max_depth=5,          # Profundidad máxima del árbol
```

```

learning_rate=0.1,      # Tasa de aprendizaje
random_state=42,        # Asegura la reproducibilidad
use_label_encoder=False # Desactivar la codificación de etiquetas para evitar
advertencias)

# Entrenar el modelo
xgb_model.fit(balance_training_x, balance_training_y)

# Hacer predicciones en el conjunto de prueba
y_pred_xgboost1 = xgb_model.predict(x_test_)

# Calcular la precisión del modelo
accuracy = accuracy_score(y_test_, y_pred_xgboost1)
print(f"Accuracy: {accuracy:.2f}")

# Mostrar el reporte de clasificación
print("Classification Report:\n", classification_report(y_test_, y_pred_xgboost1))

# Mostrar la matriz de confusión
print("Confusion Matrix:\n", confusion_matrix(y_test_, y_pred_xgboost1))

```

Se visualiza la precisión y la matriz de confusión del modelo con XGBoost. En la Figura 57 se visualiza los datos retornados.

```

Accuracy: 0.72
Classification Report:
              precision    recall  f1-score   support

     0       0.87        0.76        0.81       1832
     1       0.38        0.56        0.45        471

   accuracy          0.72
  macro avg          0.62
 weighted avg          0.72

Confusion Matrix:
[[1393  439]
 [ 207  264]]

```

Figura 57. Precisión y matriz de confusión del modelo XGBoost. Fuente: Elaboración propia.

- **1393 (Verdaderos negativos):** Clientes sin mora correctamente clasificados.
- **439 (Falsos positivos):** Clientes sin mora clasificados incorrectamente como con mora.
- **207 (Falsos negativos):** Clientes con mora clasificados incorrectamente como sin mora.
- **264 (Verdaderos positivos):** Clientes con mora correctamente clasificados.

El modelo XGBoost presenta un desempeño moderado en la predicción de la variable *moraCredito*, con un índice de precisión del 72% que refleja un buen ajuste general. Se observa que el modelo funciona mucho mejor para identificar a los clientes sin mora (clase 0), con una precisión del 87% y un recall del 76%, que en la identificación de los clientes con mora (clase 1), donde la precisión desciende al 38% y el recall alcanza solo el 56%.

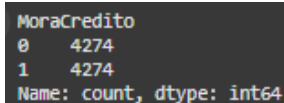
3.4.6. Balanceo de datos con la técnica *SMOTE*

Con la intención de mejorar la precisión del modelo se aplica la técnica de sobre muestreo *SMOTE* (*Synthetic Minority Over-sampling Technique*) [2] para balancear los datos. Esta técnica aumenta la cantidad de muestras en la clase minoritaria de forma conjunta tanto en variables explicativas como explicadas. En la Figura 58 se visualiza los datos retornados.

```
# Se implementa la técnica de sobremuestreo SMOTE
# Aumenta la cantidad de muestras en la clase minoritaria
from imblearn.over_sampling import SMOTE
smote = SMOTE() # Puedes ajustar random_state para reproducibilidad

# Ajustar el predictor y la variable objetivo
balance_smote_training_x, balance_smote_training_y = smote.fit_resample(x_val, y_val)

# Ver el resultado de la variable MoraCredito balanceada con SMOTE
print(balance_smote_training_y.value_counts())
```



```
MoraCredito
0    4274
1    4274
Name: count, dtype: int64
```

Figura 58. Datos después del sobre muestreo con la técnica *SMOTE*. Fuente: Elaboración propia

3.4.7. Modelo Regresión Logística con balanceo *SMOTE*

```
##### Regresión Logística #####

# Instanciamos el modelo
modelo_regresion_log = LogisticRegression(random_state=42, max_iter=1000,
solver='liblinear')

# Entrenamos el modelo con los datos de entrenamiento (train)
modelo_regresion_log = modelo_regresion_log.fit(balance_smote_training_x,
balance_smote_training_y)

# Hacemos la predicción del modelo entrenado sobre los datos de prueba (test)
```

```
y_pred_rl = modelo_regresion_log.predict(x_test_)
```

Se observa la matriz de confusión con el modelo *Regresión Logística*, con el balanceo *SMOTE*. En la Figura 59 se visualiza los datos retornados.

```
# Matriz de confusión para evaluar el rendimiento del modelo
# Con Regresión Logística

confusion_rl = confusion_matrix(y_test_, y_pred_rl)
confusion_rl
```

```
array([[1342, 490],
       [ 250, 221]])
```

Figura 59. Matriz de confusión del modelo Regresión Logística con técnica SMOTE. Fuente: Elaboración propia

- **1342 (Verdaderos negativos):** Clientes sin mora correctamente clasificados.
- **490 (Falsos positivos):** Clientes sin mora clasificados incorrectamente como con mora.
- **250 (Falsos negativos):** Clientes con mora clasificados incorrectamente como sin mora.
- **221 (Verdaderos positivos):** Clientes con mora correctamente clasificados.

Se obtiene la precisión del modelo con *Regresión logística*. En la Figura 60 se visualiza los datos retornados.

```
# Evaluamos el accuracy del modelo entrenado

accuracy_score(y_test_, y_pred_ba)
```

```
0.6786799826313504
```

Figura 60. Valor de precisión del modelo con *Regresión Logística* con balanceo *SMOTE*.

3.4.8. Modelo Árbol de decisión con balanceo *SMOTE*

```
##### Arbol de decision #####

# Instanciamos el modelo
modelo_arbol = DecisionTreeClassifier(max_depth = 5, random_state = 42)

# Entrenamos el modelo con los datos de entrenamiento (train)
modelo_arbol = modelo_arbol.fit(balance_smote_training_x, balance_smote_training_y)

# Hacemos la predicción del modelo entrenado sobre los datos de prueba (test)
y_pred_ad = modelo_arbol.predict(x_test_)
```

Se observa la matriz de confusión con el modelo *Árbol de decisión*, con el balanceo *SMOTE*. En la Figura 61 se visualiza los datos retornados.

```
# Matriz de confusión para evaluar el modelo

confusion_ad = confusion_matrix(y_test_, y_pred_ad)

confusion_ad
```

```
array([[1423, 409],
       [ 289, 182]])
```

Figura 61. Matriz de confusión del modelo *Árbol de decisión* con sobre muestreo *SMOTE*. Fuente: Elaboración propia

- **1423 (Verdaderos negativos):** Clientes sin mora correctamente clasificados.
- **409 (Falsos positivos):** Clientes sin mora clasificados incorrectamente como con mora.
- **289 (Falsos negativos):** Clientes con mora clasificados incorrectamente como sin mora.
- **182 (Verdaderos positivos):** Clientes con mora correctamente clasificados.

El modelo clasifica bien a 1423 clientes sin mora, pero comete 409 falsos positivos y 289 falsos negativos, lo que refleja dificultades para identificar clientes con mora dando 182 verdaderos positivos.

Se obtiene la precisión del modelo con *Árbol de decisión*. En la Figura 62 se visualiza los datos retornados.

```
# Evaluamos el accuracy del modelo entrenado

accuracy_score(y_test_, y_pred_ad)
```

```
0.6969170646982197
```

Figura 62. Valor de precisión del modelo *Árbol de decisión* con balanceo *SMOTE*.

3.4.9. Modelo *Random Forest* con balanceo *SMOTE*

```
##### Random Forest #####
```

```
# Instanciamos el modelo

modelo_bosque = RandomForestClassifier(n_estimators = 500, random_state = 42, max_depth = 5)

# Entrenamos el modelo con los datos de entrenamiento (train)

modelo_bosque.fit(balance_smote_training_x, balance_smote_training_y)
```

```
# Hacemos la predicción del modelo entrenado sobre los datos de prueba (test)
y_pred_ba_smote = modelo_bosque.predict(x_test_)
```

Se observa la matriz de confusión con el modelo *Random Forest*, con el sobre muestreo *SMOTE*. En la Figura 63 se visualiza los datos retornados.

```
# Matriz de confusión para evaluar el modelo

confusion_ad = confusion_matrix(y_test_, y_pred_ba_smote)
confusion_ad
```

```
array([[1360,  472],
       [ 231,  240]])
```

Figura 63. Matriz de confusión del modelo *Random Forest* con sobre muestreo *SMOTE*. Fuente: Elaboración propia

- **1360 (Verdaderos negativos):** Clientes sin mora correctamente clasificados.
- **472 (Falsos positivos):** Clientes sin mora clasificados incorrectamente como con mora.
- **231 (Falsos negativos):** Clientes con mora clasificados incorrectamente como sin mora.
- **240 (Verdaderos positivos):** Clientes con mora correctamente clasificados.

El modelo clasifica correctamente a 1360 clientes sin mora y 240 con mora, pero comete 472 falsos positivos y 231 falsos negativos, lo que indica que tiene margen de mejora, especialmente en la detección de clientes con mora.

Se obtiene la precisión del modelo con *Random Forest*. En la Figura 64 se visualiza los datos retornados

```
# Evaluamos el accuracy del modelo entrenado
accuracy_score(y_test, y_pred_ba_smote)
```

```
0.6947459834997829
```

Figura 64. Precisión del modelo *Random Forest* después del sobre muestreo *SMOTE*. Fuente: Elaboración propia

3.4.10. Modelo *XGBoost* con sobre muestreo *SMOTE*

```
##### XGBoost #####
# Entrenar el modelo

xgb_model.fit(balance_smote_training_x, balance_smote_training_y)

# Hacer predicciones en el conjunto de prueba
y_pred_xgboost2 = xgb_model.predict(x_test_)

# Calcular la precisión del modelo
```

```

accuracy = accuracy_score(y_test_, y_pred_xgboost2)
print(f"Accuracy: {accuracy:.2f}")

# Mostrar el reporte de clasificación
print("Classification Report:\n", classification_report(y_test_, y_pred_xgboost2))

# Mostrar la matriz de confusión
print("Confusion Matrix:\n", confusion_matrix(y_test_, y_pred_xgboost2))
    
```

Se visualiza la precisión y la matriz de confusión del modelo con XGBoost con el balanceo *SMOTE*. En la Figura 65 se visualiza los datos retornados.

```

warnings.warn(smsg, UserWarning)
Accuracy: 0.79
Classification Report:
      precision    recall  f1-score   support

     0       0.83      0.92      0.87      1832
     1       0.46      0.27      0.34       471

 accuracy          0.79      2303
 macro avg       0.64      0.59      0.60      2303
 weighted avg    0.75      0.79      0.76      2303

Confusion Matrix:
[[1683  149]
 [ 346  125]]
    
```

Figura 65. Precisión y matriz de confusión del modelo XGBoost después del sobre muestreo con SMOTE. Fuente: Elaboración propia.

- **1683 (Verdaderos negativos):** Clientes sin mora correctamente clasificados.
- **149 (Falsos positivos):** Clientes sin mora clasificados incorrectamente como con mora.
- **346 (Falsos negativos):** Clientes con mora clasificados incorrectamente como sin mora.
- **125 (Verdaderos positivos):** Clientes con mora correctamente clasificados.

El modelo XGBoost con balanceo SMOTE muestra un buen rendimiento al clasificar clientes sin mora (precisión del 83% y recall del 92%), pero tiene dificultades al identificar clientes con mora (precisión del 46% y recall del 27%). La precisión general es del 79% y se presenta como la más alta de los modelos entrenados.

3.5.Fase de evaluación

Modelo	Técnica de balanceo de datos	Precisión del modelo	Efectividad	Observación
Regresión Logística	Muestreo Aleatorio ROS	0.7108	Media	No confiable
Árbol de Decisión	Muestreo Aleatorio ROS	0.6630	Baja	No confiable
<i>Random Forest</i>	Muestreo Aleatorio ROS	0.7142	Media	No confiable
<i>XGBoost</i>	Muestreo Aleatorio ROS	0.72	Media	Mejor precisión de los modelos con muestreo aleatorio
Regresión Logística	<i>SMOTE</i>	0.6786	Baja	No confiable
Árbol de decisión	<i>SMOTE</i>	0.6969	Baja	No confiable
<i>Random Forest</i>	<i>SMOTE</i>	0.6947	Baja	No confiable
<i>XGBoost</i>	<i>SMOTE</i>	0.79	Alta	Muestra un rendimiento superior en comparación con otros modelos, especialmente en un contexto de clases desbalanceadas.

Tabla 1. Comparación de modelo con técnicas de balanceo.

Los resultados obtenidos en la predicción de morosidad en créditos según la Tabla, revelan un desempeño variable entre los modelos, dependiendo de la técnica de balanceo de datos utilizada.

En primer lugar, el modelo Árbol de Decisión con muestreo aleatorio presenta un rendimiento bajo, con precisión de 0.6630, los demás modelos con el mismo balanceo presentan un índice confiable medianamente.

Por otro lado, el modelo *XGBoost* con balanceo SMOTE alcanza una precisión Alta de 0.79, siendo el mejor entre los modelos que utilizan esta técnica, aunque aún muestra limitaciones en su capacidad predictiva, los demás modelos tienen una confiabilidad Baja. Cuando se aplica la técnica *SMOTE*, que equilibra mejor las clases desbalanceadas, se observa una mejora significativa en los resultados.

Finalmente, el modelo *XGBoost* con *SMOTE* se destaca con una precisión Alta de 0.79, lo que lo convierte en el modelo más confiable y preciso para predecir la morosidad en este contexto, demostrando un rendimiento superior frente a los demás modelos, especialmente al manejar datos desbalanceados.

3.6. Fase de implementación

Esta fase queda fuera de lo planificado, pero se propone y se deja en consideración de las autoridades de la Institución Financiera su posible implementación. Se detalla la propuesta como parte de Conclusiones y líneas futuras.

4. Conclusiones y líneas futuras

En este trabajo se han implementado modelos basados en técnicas de Inteligencia Artificial para predecir el estado crediticio de clientes de una Institución Financiera.

Para lograr este objetivo principal fue importante revisar varios trabajos similares donde implementaron concretamente modelos de ML supervisados para predecir mora o cuotas impagas en créditos. Según la literatura revisada se concluye que en estos escenarios es muy común el uso de *Random Forest*, Regresión Logística, *XGBoost* y el árbol de decisión (no muy aplicado).

Se ejecutó la extracción, transformación y carga de los datos desde la base de datos *SQL Server* a un archivo *CSV*. Estos se preprocesaron mediante un *script* codificado con Python y sus fases fueron guiadas por la metodología *CRISP-DM*.

Se realizó un Análisis Exploratorio de Datos (EDA), logrando entender la estructura del *DataFrame*, eliminar valores ausentes, datos redundantes, completar datos faltantes, calcular

nuevas variables, formatear valores decimales, entender la distribución de las frecuencias, verificar la asociación de variables, eliminar variables no significativas, obtener la matriz de correlación de variables, graficar un mapa de calor de correlación de Pearson y acondicionar las variables para los modelos de ML.

Se implementaron y entrenaron los modelos Regresión Logística, Árbol de decisión, *Random Forest* y *XGBoost*. Durante el preprocesamiento de los datos, se observó un desequilibrio en la variable objetivo “MoraCredito” con más casos igual a 0 (puntualidad) que igual a 1 (impuntualidad). Para abordar este desequilibrio se aplicó la técnica de sobre muestreo aleatorio alcanzando una precisión baja, posteriormente se aplicó la técnica *SMOTE*, se reentrenaron los modelos *Random Forest* y *XGBoost* y se observa una mejora significativa en la precisión.

Se evaluó el índice de precisión creando una tabla y categorizando su efectividad como Baja, Media, y Alta según su valor, también se evaluó la matriz de confusión de cada modelo.

Se concluye que el modelo *XGBoost* logró el mayor índice de precisión con una efectividad denominada Alta según la Tabla 1, demostrando un rendimiento superior comparado con los demás modelos, convirtiéndose en el más confiable y preciso para predecir la morosidad en créditos. Además, la técnica de muestreo *SMOTE* fue la que mejor balanceó los datos.

Aunque el modelo no será implementado directamente, se recomienda su adopción debido a los resultados obtenidos, que demuestran un alto potencial para mejorar la precisión en la predicción de mora crediticia. Esto refuerza la idea de que mejores predicciones conducen a decisiones más informadas y destaca la necesidad de incorporar procesos basados en Inteligencia Artificial para garantizar la salud Financiera y la sostenibilidad de la Cooperativa a largo plazo.

Finalmente, si se llegara a implementar, se podría diseñar y ejecutar estrategias más efectivas para otorgar préstamos a clientes con alta probabilidad de cumplimiento, optimizando el servicio de crédito y reduciendo el riesgo de incumplimiento.

Referencias bibliográficas

- [1] P. Haya. "La metodología CRISP-DM en ciencia de datos." myendnoteweb.com. <https://www.iic.uam.es/innovacion/metodologia-crisp-dm-ciencia-de-datos/> (accessed Jun. 03, 2024).
- [2] K. D. Grzebień. (2023) Predicción del riesgo de impago en los préstamos P2P. *Revista de Economía y Finanzas*. Available: <https://www.reveyf.es/index.php/REyF/article/view/352/159>