Actividad-1

October 2, 2022

1 Actividad 1

1.1 TC4029 Ciencia y analítica de datos (Gpo 10)

Profesores : Jobish Vallikavungal Devassia, Victoria Guerrero Orozco

Alumno: Armando Bringas Corpus (A01200230)

1.2 1. Fundamentos de Bases de Datos

1.2.1 1.1. Fundamentos de Bases de Datos y para Ciencia de Datos.

Una Base de Datos en Ciencia de Datos es una colección de datos, podemos pensar en ella como una hoja de cálculo (*spreadsheet*), con registros representados por las filas y con sus atributos representados en cada columna, esta tabla representa una entidad que se refiere al objeto o concepto que representa y los datos que captura sobre el mismo; una base de datos puede almacenar diferentes subconjuntos de datos, es decir, colecciones de tablas relacionadas, información sobre estas tablas y su relación entre sí, las cuales definen la estructura de la base de datos (Teate, 2021).

Podemos encontrar que los datos se pueden almacenar en muchos tipos de formas y estructuras, sin embargo, los datos estructurados, por lo general se organizan en un formato tabular en forma de una Base de Datos utilizando diferentes aplicaciones de software (Teate, 2021), empleando un sistema DBMS (*Database Management System*) como puede ser SQL (*Structured Query Language*).

Utilizar Bases de Datos para hacer Ciencia de Datos ofrece múltiples ventajas como tener consistencia en los datos, reducir la reduncia de los datos, así como mejorar la productividad y accesibilidad para trabajarlas.

1.2.2 2.1. Fundamentos de almacenes de datos (Data Warehouse) para Ciencia de Datos

Los Almacenes de Datos (Data Warehouse) se refieren a bases de datos muy grandes que pueden contener información de diferentes fuentes, las cuales pueden representar un desafío para transformar su información en una estructura que nos permita hacer consultas de forma simultánea y hacer inferencias a través del análisis de datos (Kane, 2017).

Estos almacenes de datos pueden contener información ya sea de los datos crudos (raw data) que se extrae de forma directa de otras bases de datos o bien puede contener tablas que combinan o transforman estos datos en crudo y presentan la información en forma de tablas de sumarios (Teate, 2021). Por lo tanto, el principal reto de trabajar con almacenes de datos es que si la información viene de los datos en crudo es que se tienen normalizar los datos.

Este proceso de normalizar los datos implica tener los datos estructurados para remover los datos redundantes, lo cual implica remover los datos recurrentes, trabajar con las anomalías que pueden venir de datos con valores atípicos, datos faltantes, datos corruptos, verificar que la relación entre las diferentes bases de datos use la misma escala, etc.

1.3 2. Selección y Limpieza de los Datos en Python

[1]:
```
# Importación de Librerías a utilizar

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

[2]:
```
# Mostrar todas las filas y columnas

#pd.set_option('display.rows', None)
pd.set_option('display.max_columns', None)
```

1.3.1 Importación de los Datos

[3]:
```
# Obtener datos 'raw' desde URL

input_data = 'https://raw.githubusercontent.com/PosgradoMNA/
 ↪Actividades_Aprendizaje-/main/default%20of%20credit%20card%20clients.csv'
```

[4]:
```
# Convertir los datos importados en un DataFrame

df = pd.read_csv(input_data)
```

Datos desde un archivo .'csv' (comma separated values) Guardado de los datos en un archivo '.csv'

[5]:
```
df.to_csv('default_of_credit_card_clients.csv')
```

Lectura de los datos desde el archivo '.csv' creado

[6]:
```
df = pd.read_csv('default_of_credit_card_clients.csv', index_col=0)
df
```

[6]:

	ID	X1	X2	X3	X4	X5	X6	X7	X8	X9	X10	X11	\
0	1	20000	2.0	2.0	1.0	24.0	2.0	2.0	-1.0	-1.0	-2.0	-2.0	

```
1          2  120000  2.0  2.0  2.0  26.0 -1.0  2.0  0.0  0.0  0.0  2.0
2          3   90000  2.0  2.0  2.0  34.0  0.0  0.0  0.0  0.0  0.0  0.0
3          4   50000  2.0  2.0  1.0  37.0  0.0  0.0  0.0  0.0  0.0  0.0
4          5   50000  1.0  2.0  1.0  57.0 -1.0  0.0 -1.0  0.0  0.0  0.0
…        …      …  …  …  …  …   …  …  …  …  …  …  …
29995  29996  220000  1.0  3.0  1.0  39.0  0.0  0.0  0.0  0.0  0.0  0.0
29996  29997  150000  1.0  3.0  2.0  43.0 -1.0 -1.0 -1.0 -1.0  0.0  0.0
29997  29998   30000  1.0  2.0  2.0  37.0  4.0  3.0  2.0 -1.0  0.0  0.0
29998  29999   80000  1.0  3.0  1.0  41.0  1.0 -1.0  0.0  0.0  0.0 -1.0
29999  30000   50000  1.0  2.0  1.0  46.0  0.0  0.0  0.0  0.0  0.0  0.0

            X12       X13       X14      X15      X16      X17      X18  \
0        3913.0    3102.0     689.0      0.0      0.0      0.0      0.0
1        2682.0    1725.0    2682.0   3272.0   3455.0   3261.0      0.0
2       29239.0   14027.0   13559.0  14331.0  14948.0  15549.0   1518.0
3       46990.0   48233.0   49291.0  28314.0  28959.0  29547.0   2000.0
4        8617.0    5670.0   35835.0  20940.0  19146.0  19131.0   2000.0
…          …         …         …        …        …        …
29995  188948.0  192815.0  208365.0  88004.0  31237.0  15980.0   8500.0
29996    1683.0    1828.0    3502.0   8979.0   5190.0      0.0   1837.0
29997    3565.0    3356.0    2758.0  20878.0  20582.0  19357.0      0.0
29998   -1645.0   78379.0   76304.0  52774.0  11855.0  48944.0  85900.0
29999   47929.0   48905.0   49764.0  36535.0  32428.0  15313.0   2078.0

            X19      X20     X21       X22     X23    Y
0         689.0      0.0     0.0       0.0     0.0  1.0
1        1000.0   1000.0  1000.0       0.0  2000.0  1.0
2        1500.0   1000.0  1000.0    1000.0  5000.0  0.0
3        2019.0   1200.0  1100.0    1069.0  1000.0  0.0
4       36681.0  10000.0  9000.0     689.0   679.0  0.0
…         …       …       …         …       …  …
29995   20000.0   5003.0  3047.0    5000.0  1000.0  0.0
29996    3526.0   8998.0   129.0       0.0     0.0  0.0
29997       0.0  22000.0  4200.0    2000.0  3100.0  1.0
29998    3409.0   1178.0  1926.0   52964.0  1804.0  1.0
29999    1800.0   1430.0  1000.0    1000.0  1000.0  1.0

[30000 rows x 25 columns]
```

1.3.2 Descripción de los Datos

Descripción de atributos del data frame (columnas)

[7]:
```
# listado del nombre de las columnas (encabezado)

df.columns
```

[7]:
```
Index(['ID', 'X1', 'X2', 'X3', 'X4', 'X5', 'X6', 'X7', 'X8', 'X9', 'X10',
       'X11', 'X12', 'X13', 'X14', 'X15', 'X16', 'X17', 'X18', 'X19', 'X20',
       'X21', 'X22', 'X23', 'Y'],
      dtype='object')
```

Este conjunto de datos contiene información sobre factores demográficos y datos crediticiosde clientes de tarjetas de crédito en Taiwán desde abril de 2005 hasta septiembre de 2005. (Descripción del dataset)

- **ID**: ID del cliente
- **X1 [CREDITO]**: Monto del crédito otorgado (NT dólares): incluye tanto el crédito de consumo individual como su crédito familiar (complementario)
- **X2 [SEXO]**: (1 = masculino; 2 = femenino)
- **X3 [ESCOLARIDAD]**: (1 = posgrado; 2 = universidad; 3 = secundaria; 4 = otros)
- **X4 [ESTADO_CIVIL]**: (1 = casado; 2 = soltero; 3 = otros)
- **X5 [EDAD]**: Edad (años)
- **X6 - X11**: Meses de retraso de abril a septiembre (-1 = pagado; 1 = retraso en el pago de un mes; 2 = retraso en el pago de dos meses; . . .; 8 = retraso en el pago de ocho meses; 9 = retraso en el pago de nueve meses o más..)
  - **X6 [RETRASO_SEP]** = cantidad de meses de retraso de pago en septiembre de 2005;
  - **X7 [RETRASO_AUG]** = cantidad de meses de retraso de pago en agosto de 2005;
  - **X8 [RETRASO_JUL]** = cantidad de meses de retraso de pago en julio de 2005;
  - **X9 [RETRASO_JUN]** = cantidad de meses de retraso de pago en junio de 2005;
  - **X10 [RETRASO_MAY]** = cantidad de meses de retraso de pago en mayo de 2005;
  - **X11 [RETRASO_APR]** = cantidad de meses de retraso de pago en abril de 2005;
- **X12-X17**: Monto del estado de cuenta (NT dólares).
  - **X12 [SALDO_SEP]** = monto del estado de cuenta en septiembre de 2005;
  - **X13 [SALDO_AUG]** = monto del estado de cuenta en agosto de 2005;
  - **X14 [SALDO_JUL]** = monto del estado de cuenta en julio de 2005;
  - **X15 [SALDO_JUN]** = monto del estado de cuenta en junio de 2005;
  - **X16 [SALDO_MAY]** = monto del estado de cuenta en mayo de 2005;
  - **X17 [SALDO_APR]** = monto del estado de cuenta en abril de 2005;
- **X18-X23**: Monto del pago anterior (NT dólares).
  - **X18 [PAGO_SEP]** = monto pagado en septiembre de 2005;
  - **X19 [PAGO_AUG]** = monto pagado en agosto de 2005;
  - **X20 [PAGO_JUL]** = monto pagado en julio de 2005;
  - **X21 [PAGO_JUN]** = monto pagado en junio de 2005;
  - **X22 [PAGO_MAY]** = monto pagado en mayo de 2005;
  - **X23 [PAGO_APR]** = monto pagado en abril de 2005;
- **Y [Y_PREPAGO]** = pago predeterminado: (Sí = 1, No = 0)

Renombrado de los datos Consideramos importante renombrar los atributos del conjunto de datos para que sea más sencillo poder identificarlos y almacenarlos en nuevo dataframe.

[8]:
```
df = df.rename(columns={'X1': 'CREDITO',
                        'X2': 'SEXO',
                        'X3': 'ESCOLARIDAD',
                        'X4': 'ESTADO_CIVIL',
```

```
                        'X5': 'EDAD',
                        'X6': 'RETRASO_SEP',
                        'X7': 'RETRASO_AUG',
                        'X8': 'RETRASO_JUL',
                        'X9': 'RETRASO_JUN',
                        'X10': 'RETRASO_MAY',
                        'X11': 'RETRASO_APR',
                        'X12': 'SALDO_SEP',
                        'X13': 'SALDO_AUG',
                        'X14': 'SALDO_JUL',
                        'X15': 'SALDO_JUN',
                        'X16': 'SALDO_MAY',
                        'X17': 'SALDO_APR',
                        'X18': 'PAGO_SEP',
                        'X19': 'PAGO_AUG',
                        'X20': 'PAGO_JUL',
                        'X21': 'PAGO_JUN',
                        'X22': 'PAGO_MAY',
                        'X23': 'PAGO_APR',
                        'Y': 'Y_PREPAGO'})

df.head()
```

[8]:

	ID	CREDITO	SEXO	ESCOLARIDAD	ESTADO_CIVIL	EDAD	RETRASO_SEP	\
0	1	20000	2.0	2.0	1.0	24.0	2.0	
1	2	120000	2.0	2.0	2.0	26.0	-1.0	
2	3	90000	2.0	2.0	2.0	34.0	0.0	
3	4	50000	2.0	2.0	1.0	37.0	0.0	
4	5	50000	1.0	2.0	1.0	57.0	-1.0	

	RETRASO_AUG	RETRASO_JUL	RETRASO_JUN	RETRASO_MAY	RETRASO_APR	SALDO_SEP	\
0	2.0	-1.0	-1.0	-2.0	-2.0	3913.0	
1	2.0	0.0	0.0	0.0	2.0	2682.0	
2	0.0	0.0	0.0	0.0	0.0	29239.0	
3	0.0	0.0	0.0	0.0	0.0	46990.0	
4	0.0	-1.0	0.0	0.0	0.0	8617.0	

	SALDO_AUG	SALDO_JUL	SALDO_JUN	SALDO_MAY	SALDO_APR	PAGO_SEP	PAGO_AUG	\
0	3102.0	689.0	0.0	0.0	0.0	0.0	689.0	
1	1725.0	2682.0	3272.0	3455.0	3261.0	0.0	1000.0	
2	14027.0	13559.0	14331.0	14948.0	15549.0	1518.0	1500.0	
3	48233.0	49291.0	28314.0	28959.0	29547.0	2000.0	2019.0	
4	5670.0	35835.0	20940.0	19146.0	19131.0	2000.0	36681.0	

	PAGO_JUL	PAGO_JUN	PAGO_MAY	PAGO_APR	Y_PREPAGO
0	0.0	0.0	0.0	0.0	1.0
1	1000.0	1000.0	0.0	2000.0	1.0

```
2    1000.0    1000.0    1000.0    5000.0        0.0
3    1200.0    1100.0    1069.0    1000.0        0.0
4   10000.0    9000.0     689.0     679.0        0.0
```

Recodificación de los datos Procedemos a transformar aquellos datos que son cuantitativos a recodificarlos como tipos de datos categóricos. De igual manera procedemos a cambiar el tipo de dato de *CREDITO* de 'int' a 'float' y el de *EDAD* de 'float' a 'int' para tener consistencia con los otros datos

[9]:
```
df['ID'] = df['ID'].astype('category')
df['CREDITO'] = df['CREDITO'].astype('float64')
df['SEXO'] = df['SEXO'].astype('category')
df['ESCOLARIDAD'] = df['ESCOLARIDAD'].astype('category')
df['ESTADO_CIVIL'] = df['ESTADO_CIVIL'].astype('category')
df['Y_PREPAGO'] = df['Y_PREPAGO'].astype('category')
df.dtypes
```

[9]:
```
ID              category
CREDITO          float64
SEXO            category
ESCOLARIDAD     category
ESTADO_CIVIL    category
EDAD             float64
RETRASO_SEP      float64
RETRASO_AUG      float64
RETRASO_JUL      float64
RETRASO_JUN      float64
RETRASO_MAY      float64
RETRASO_APR      float64
SALDO_SEP        float64
SALDO_AUG        float64
SALDO_JUL        float64
SALDO_JUN        float64
SALDO_MAY        float64
SALDO_APR        float64
PAGO_SEP         float64
PAGO_AUG         float64
PAGO_JUL         float64
PAGO_JUN         float64
PAGO_MAY         float64
PAGO_APR         float64
Y_PREPAGO       category
dtype: object
```

Descripción de los datos

[10]:
```
df.describe()
```

[10]:

	CREDITO	EDAD	RETRASO_SEP	RETRASO_AUG	RETRASO_JUL
count	30000.000000	29995.000000	29997.000000	29995.000000	29993.000000
mean	167484.322667	35.484214	-0.016635	-0.133689	-0.166405
std	129747.661567	9.218024	1.123829	1.197254	1.196048
min	10000.000000	21.000000	-2.000000	-2.000000	-2.000000
25%	50000.000000	28.000000	-1.000000	-1.000000	-1.000000
50%	140000.000000	34.000000	0.000000	0.000000	0.000000
75%	240000.000000	41.000000	0.000000	0.000000	0.000000
max	1000000.000000	79.000000	8.000000	8.000000	8.000000

	RETRASO_JUN	RETRASO_MAY	RETRASO_APR	SALDO_SEP	SALDO_AUG
count	29991.000000	29984.000000	29986.000000	29989.000000	29989.000000
mean	-0.220800	-0.266342	-0.291136	51236.862750	49190.734669
std	1.169153	1.133296	1.150134	73645.219278	71183.385123
min	-2.000000	-2.000000	-2.000000	-165580.000000	-69777.000000
25%	-1.000000	-1.000000	-1.000000	3565.000000	2986.000000
50%	0.000000	0.000000	0.000000	22387.000000	21207.000000
75%	0.000000	0.000000	0.000000	67139.000000	64027.000000
max	8.000000	8.000000	8.000000	964511.000000	983931.000000

	SALDO_JUL	SALDO_JUN	SALDO_MAY	SALDO_APR
count	2.998700e+04	29985.000000	29983.000000	29990.000000
mean	4.702535e+04	43275.652326	40324.493980	38881.135745
std	6.936086e+04	64345.500073	60809.984983	59561.312967
min	-1.572640e+05	-170000.000000	-81334.000000	-339603.000000
25%	2.667500e+03	2329.000000	1763.500000	1256.250000
50%	2.008900e+04	19052.000000	18107.000000	17081.000000
75%	6.018200e+04	54560.000000	50213.000000	49208.250000
max	1.664089e+06	891586.000000	927171.000000	961664.000000

	PAGO_SEP	PAGO_AUG	PAGO_JUL	PAGO_JUN
count	29992.000000	2.999100e+04	29992.000000	29989.000000
mean	5662.945886	5.922489e+03	5225.623400	4827.252526
std	16564.165089	2.304418e+04	17608.422625	15668.751975
min	0.000000	0.000000e+00	0.000000	0.000000
25%	1000.000000	8.355000e+02	390.000000	296.000000
50%	2100.000000	2.009000e+03	1800.000000	1500.000000
75%	5006.000000	5.000000e+03	4505.500000	4014.000000
max	873552.000000	1.684259e+06	896040.000000	621000.000000

	PAGO_MAY	PAGO_APR
count	29989.000000	29995.000000
mean	4800.297209	5216.259977
std	15280.842069	17778.848359
min	0.000000	0.000000
25%	251.000000	118.000000
50%	1500.000000	1500.000000

```
75%       4033.000000    4000.000000
max     426529.000000  528666.000000
```

1.3.3 Verificación de que no falten datos

Obtenemos información de a qué atributos les faltan datos y los contabilizamos

[11]:
```
# Datos faltantes por atributo

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

[11]:
```
ID              0
CREDITO         0
SEXO            1
ESCOLARIDAD     2
ESTADO_CIVIL    2
EDAD            5
RETRASO_SEP     3
RETRASO_AUG     5
RETRASO_JUL     7
RETRASO_JUN     9
RETRASO_MAY    16
RETRASO_APR    14
SALDO_SEP      11
SALDO_AUG      11
SALDO_JUL      13
SALDO_JUN      15
SALDO_MAY      17
SALDO_APR      10
PAGO_SEP        8
PAGO_AUG        9
PAGO_JUL        8
PAGO_JUN       11
PAGO_MAY       11
PAGO_APR        5
Y_PREPAGO       3
dtype: int64
```

Procedemos a calcular el procentaje de datos faltantes para darnos una idea de la relación del porcentaje de los datos que faltan con respecto a la cantidad total de datos que tenemos

[12]:
```
# Total de datos faltantes

def imprimir_datos_faltantes(n_df):
    total_datos_faltantes = n_df.isnull().sum().sum()
    porcentaje_total_datos_faltantes = total_datos_faltantes * 100 / len(n_df)
    print(f'Total de datos faltantes: {total_datos_faltantes}, %:␣
 ↪{porcentaje_total_datos_faltantes}')
```

```
imprimir_datos_faltantes(df)
```

```
Total de datos faltantes: 196, %: 0.6533333333333333
```

1.3.4 Verificación de que no haya datos duplicados

Vamos a verificar que no tengamos registros duplicados en el dataframe

[13]:
```
df.duplicated().any()
```

[13]:
```
False
```

1.3.5 Limpieza de los datos

Procedemos a hacer una exploración más detallada de los registros (filas) a las que les faltan datos y procedemos con su limpieza.

Eliminación de registros Analizamos los datos faltantes en la columna 'Y_PREPAGO' que representa la salida (output -> 'Y') de nuestros datos

[14]:
```
# Observemos el dataframe filtrando los datos faltantes de la columna␣
↪'Y_PREPAGO'

df_NaN = df[df['Y_PREPAGO'].isna()]
df_NaN
```

[14]:

	ID	CREDITO	SEXO	ESCOLARIDAD	ESTADO_CIVIL	EDAD	RETRASO_SEP	\
6232	6233	60000.0	2.0	2.0	2.0	29.0	2.0	
29824	29825	40000.0	1.0	1.0	1.0	47.0	2.0	
29825	29826	50000.0	1.0	2.0	1.0	41.0	0.0	

	RETRASO_AUG	RETRASO_JUL	RETRASO_JUN	RETRASO_MAY	RETRASO_APR	\
6232	2.0	2.0	0.0	0.0	0.0	
29824	2.0	2.0	2.0	2.0	2.0	
29825	0.0	0.0	0.0	0.0	0.0	

	SALDO_SEP	SALDO_AUG	SALDO_JUL	SALDO_JUN	SALDO_MAY	SALDO_APR	\
6232	41387.0	42117.0	NaN	NaN	NaN	NaN	
29824	NaN	NaN	NaN	NaN	NaN	NaN	
29825	NaN	NaN	NaN	NaN	NaN	NaN	

	PAGO_SEP	PAGO_AUG	PAGO_JUL	PAGO_JUN	PAGO_MAY	PAGO_APR	Y_PREPAGO
6232	NaN	NaN	NaN	NaN	NaN	NaN	NaN
29824	NaN	NaN	NaN	NaN	NaN	NaN	NaN
29825	NaN	NaN	NaN	NaN	NaN	NaN	NaN

Vamos a proceder a eliminar los registros con valores *NaN* en la columna de prepago ya que de igual manera le faltan datos en más de dos columnas, por lo que estamos empleando el método de

eliminación por lista

[15]:
```
# Registros a los que les faltan más de dos valores con respecto al dataset␣
↪anterior

thresh = 2
df_NaN.count(1) >= thresh
```

[15]:
```
6232     True
29824    True
29825    True
dtype: bool
```

[16]:
```
# Remover datos faltantes en columna 'Y_PREPAGO'

df = df[df['Y_PREPAGO'].notna()]
imprimir_datos_faltantes(df)
```

```
Total de datos faltantes: 159, %: 0.53005300530053
```

Vamos a analizar qué columna es la que tiene más datos faltantes para poderlos observar y determinar si hay columnas que podemos eliminar

[17]:
```
# Empleamos el método 'idxmin()' para obtener la columna con mayor datos␣
↪faltantes

df.count().idxmin()
```

[17]:
```
'RETRASO_MAY'
```

[18]:
```
# Observemos el dataframe filtrando los datos faltantes de la columna␣
↪'RETRASO_MAY'

df_NaN = df[df['RETRASO_MAY'].isna()]
df_NaN
```

[18]:

	ID	CREDITO	SEXO	ESCOLARIDAD	ESTADO_CIVIL	EDAD	RETRASO_SEP	\
49	50	20000.0	1.0	1.0	2.0	24.0	0.0	
233	234	190000.0	1.0	2.0	2.0	34.0	2.0	
6227	6228	30000.0	1.0	NaN	NaN	NaN	NaN	
6268	6269	120000.0	1.0	2.0	2.0	29.0	1.0	
6269	6270	400000.0	2.0	1.0	2.0	27.0	1.0	
6270	6271	50000.0	2.0	1.0	1.0	55.0	0.0	
6382	6383	80000.0	2.0	2.0	1.0	30.0	1.0	
6383	6384	90000.0	1.0	2.0	2.0	29.0	0.0	
6384	6385	20000.0	2.0	2.0	2.0	22.0	0.0	
24123	24124	50000.0	1.0	1.0	2.0	29.0	NaN	
24365	24366	130000.0	NaN	NaN	NaN	NaN	NaN	
29733	29734	300000.0	1.0	2.0	1.0	37.0	-2.0	

29734	29735	500000.0	1.0	2.0	1.0	49.0	-1.0
29735	29736	50000.0	1.0	3.0	1.0	40.0	2.0
29832	29833	20000.0	1.0	2.0	2.0	30.0	0.0
29833	29834	20000.0	1.0	2.0	2.0	31.0	-1.0

	RETRASO_AUG	RETRASO_JUL	RETRASO_JUN	RETRASO_MAY	RETRASO_APR
49	0.0	0.0	0.0	NaN	0.0
233	0.0	0.0	0.0	NaN	2.0
6227	NaN	NaN	NaN	NaN	NaN
6268	2.0	0.0	0.0	NaN	NaN
6269	-2.0	-1.0	0.0	NaN	NaN
6270	0.0	0.0	0.0	NaN	NaN
6382	2.0	0.0	0.0	NaN	NaN
6383	0.0	0.0	0.0	NaN	NaN
6384	0.0	0.0	0.0	NaN	NaN
24123	NaN	NaN	NaN	NaN	NaN
24365	NaN	NaN	NaN	NaN	NaN
29733	-2.0	-1.0	-1.0	NaN	NaN
29734	-1.0	2.0	-1.0	NaN	NaN
29735	0.0	0.0	0.0	NaN	NaN
29832	NaN	NaN	NaN	NaN	NaN
29833	NaN	NaN	NaN	NaN	NaN

	SALDO_SEP	SALDO_AUG	SALDO_JUL	SALDO_JUN	SALDO_MAY	SALDO_APR
49	17447.0	18479.0	19476.0	19865.0	20480.0	20063.0
233	129801.0	131383.0	134379.0	142323.0	140120.0	150052.0
6227	NaN	NaN	NaN	NaN	NaN	21120.0
6268	87376.0	85135.0	86024.0	91380.0	93263.0	95079.0
6269	140.0	140.0	12527.0	12640.0	14805.0	238.0
6270	25043.0	26411.0	27451.0	19431.0	19837.0	20232.0
6382	82891.0	80997.0	81380.0	75374.0	77158.0	78710.0
6383	86294.0	88412.0	89853.0	69362.0	70537.0	56775.0
6384	17807.0	18520.0	39053.0	20055.0	19606.0	19925.0
24123	NaN	NaN	NaN	NaN	NaN	NaN
24365	NaN	NaN	NaN	NaN	NaN	NaN
29733	NaN	NaN	NaN	NaN	NaN	NaN
29734	NaN	NaN	NaN	NaN	NaN	NaN
29735	NaN	NaN	NaN	NaN	NaN	NaN
29832	NaN	NaN	NaN	NaN	NaN	20090.0
29833	NaN	NaN	NaN	NaN	NaN	19103.0

	PAGO_SEP	PAGO_AUG	PAGO_JUL	PAGO_JUN	PAGO_MAY	PAGO_APR	Y_PREPAGO
49	1318.0	1315.0	704.0	928.0	912.0	1069.0	0.0
233	5000.0	5000.0	10000.0	0.0	12118.0	2769.0	1.0
6227	1586.0	1365.0	1663.0	5106.0	1050.0	0.0	0.0
6268	0.0	3200.0	6800.0	3500.0	3500.0	0.0	1.0
6269	140.0	12527.0	253.0	2305.0	238.0	238.0	0.0

```
6270     1775.0    1815.0     695.0     719.0     724.0     750.0       0.0
6382        0.0    3299.0    2699.0    3000.0    2987.0    3100.0       0.0
6383     4400.0    4111.0    2100.0    2200.0    2510.0    2000.0       0.0
6384     1298.0    1909.0    1666.0    5000.0     703.0     775.0       0.0
24123       NaN       NaN       NaN       NaN       NaN     360.0       0.0
24365       NaN       NaN       NaN       NaN       NaN     390.0       0.0
29733       NaN       NaN       NaN       NaN       NaN    3081.0       0.0
29734       NaN       NaN       NaN       NaN       NaN     237.0       1.0
29735       NaN       NaN       NaN       NaN       NaN     406.0       1.0
29832    1279.0    3000.0    6668.0     468.0     507.0    5582.0       0.0
29833     390.0     780.0       0.0     806.0   19103.0    1000.0       1.0
```

[19]:
```
# Registros a los que les faltan más de dos valores con respecto al dataset␣
 ↪anterior

thresh = 2
df_NaN.count(1) >= thresh
```

[19]:
```
49       True
233      True
6227     True
6268     True
6269     True
6270     True
6382     True
6383     True
6384     True
24123    True
24365    True
29733    True
29734    True
29735    True
29832    True
29833    True
dtype: bool
```

[20]:
```
# Remover datos faltantes en columna 'RETRASO_MAY'

df = df[df['RETRASO_MAY'].notna()]
imprimir_datos_faltantes(df)
```

```
Total de datos faltantes: 34, %: 0.11340515659917948
```

Volvemos a iterar y obtenemos la siguiente columna a la que le hace falta más valores

[21]:
```
# Empleamos el método 'idxmin()' para obtener la columna con mayor datos␣
 ↪faltantes
```

```
df.count().idxmin()
```

[21]:
```
'SALDO_MAY'
```

[22]:
```
df_NaN = df[df['SALDO_MAY'].isna()]
df_NaN
```

[22]:

	ID	CREDITO	SEXO	ESCOLARIDAD	ESTADO_CIVIL	EDAD	RETRASO_SEP	\
18	19	360000.0	2.0	1.0	1.0	49.0	1.0	
175	176	130000.0	1.0	3.0	1.0	56.0	1.0	
240	241	60000.0	2.0	1.0	2.0	28.0	1.0	
6276	6277	30000.0	1.0	3.0	1.0	32.0	0.0	
6277	6278	110000.0	2.0	1.0	2.0	32.0	1.0	
6278	6279	20000.0	2.0	3.0	2.0	54.0	0.0	

	RETRASO_AUG	RETRASO_JUL	RETRASO_JUN	RETRASO_MAY	RETRASO_APR	\
18	-2.0	-2.0	-2.0	-2.0	-2.0	
175	2.0	2.0	2.0	2.0	3.0	
240	2.0	2.0	-2.0	-2.0	-1.0	
6276	0.0	0.0	0.0	0.0	-1.0	
6277	2.0	0.0	0.0	0.0	2.0	
6278	0.0	2.0	2.0	2.0	0.0	

	SALDO_SEP	SALDO_AUG	SALDO_JUL	SALDO_JUN	SALDO_MAY	SALDO_APR	\
18	NaN	NaN	NaN	NaN	NaN	NaN	
175	64617.0	65978.0	67282.0	68557.0	NaN	71345.0	
240	21501.0	20650.0	0.0	0.0	NaN	2285.0	
6276	28292.0	30494.0	28317.0	NaN	NaN	1400.0	
6277	58679.0	56871.0	56279.0	NaN	NaN	25453.0	
6278	9777.0	10140.0	9360.0	NaN	NaN	9116.0	

	PAGO_SEP	PAGO_AUG	PAGO_JUL	PAGO_JUN	PAGO_MAY	PAGO_APR	Y_PREPAGO
18	0.0	0.0	0.0	0.0	0.0	0.0	0.0
175	3000.0	3000.0	3000.0	5500.0	0.0	0.0	1.0
240	0.0	0.0	0.0	0.0	2285.0	0.0	0.0
6276	3000.0	1431.0	483.0	0.0	700.0	0.0	0.0
6277	1210.0	2056.0	1200.0	2300.0	0.0	1000.0	1.0
6278	2000.0	1000.0	1000.0	0.0	500.0	500.0	0.0

[23]:
```
# Registros a los que les faltan más de dos valores con respecto al dataset␣
 ↪anterior

thresh = 2
df_NaN.count(1) >= thresh
```

[23]:
```
18      True
175     True
```

```
240     True
6276    True
6277    True
6278    True
dtype: bool
```

[24]:
```
# Remover datos faltantes en columna 'RETRASO_MAY'

df = df[df['SALDO_MAY'].notna()]
imprimir_datos_faltantes(df)
```

```
Total de datos faltantes: 20, %: 0.06672226855713094
```

Para la siguiente iteración vamos remover aquellas columnas que les falten dos o más valores en la misma fila

[25]:
```
df = df.dropna(thresh=len(df.columns)-1)
df
```

[25]:

	ID	CREDITO	SEXO	ESCOLARIDAD	ESTADO_CIVIL	EDAD	RETRASO_SEP	\
0	1	20000.0	2.0	2.0	1.0	24.0	2.0	
1	2	120000.0	2.0	2.0	2.0	26.0	-1.0	
2	3	90000.0	2.0	2.0	2.0	34.0	0.0	
3	4	50000.0	2.0	2.0	1.0	37.0	0.0	
4	5	50000.0	1.0	2.0	1.0	57.0	-1.0	
…	…	…	…	…	…	…	…	
29995	29996	220000.0	1.0	3.0	1.0	39.0	0.0	
29996	29997	150000.0	1.0	3.0	2.0	43.0	-1.0	
29997	29998	30000.0	1.0	2.0	2.0	37.0	4.0	
29998	29999	80000.0	1.0	3.0	1.0	41.0	1.0	
29999	30000	50000.0	1.0	2.0	1.0	46.0	0.0	

	RETRASO_AUG	RETRASO_JUL	RETRASO_JUN	RETRASO_MAY	RETRASO_APR	\
0	2.0	-1.0	-1.0	-2.0	-2.0	
1	2.0	0.0	0.0	0.0	2.0	
2	0.0	0.0	0.0	0.0	0.0	
3	0.0	0.0	0.0	0.0	0.0	
4	0.0	-1.0	0.0	0.0	0.0	
…	…	…	…	…	…	
29995	0.0	0.0	0.0	0.0	0.0	
29996	-1.0	-1.0	-1.0	0.0	0.0	
29997	3.0	2.0	-1.0	0.0	0.0	
29998	-1.0	0.0	0.0	0.0	-1.0	
29999	0.0	0.0	0.0	0.0	0.0	

	SALDO_SEP	SALDO_AUG	SALDO_JUL	SALDO_JUN	SALDO_MAY	SALDO_APR	\
0	3913.0	3102.0	689.0	0.0	0.0	0.0	
1	2682.0	1725.0	2682.0	3272.0	3455.0	3261.0	

```
2        29239.0    14027.0    13559.0    14331.0    14948.0    15549.0
3        46990.0    48233.0    49291.0    28314.0    28959.0    29547.0
4         8617.0     5670.0    35835.0    20940.0    19146.0    19131.0
…            …          …          …          …          …          …
29995   188948.0   192815.0   208365.0    88004.0    31237.0    15980.0
29996     1683.0     1828.0     3502.0     8979.0     5190.0        0.0
29997     3565.0     3356.0     2758.0    20878.0    20582.0    19357.0
29998    -1645.0    78379.0    76304.0    52774.0    11855.0    48944.0
29999    47929.0    48905.0    49764.0    36535.0    32428.0    15313.0

       PAGO_SEP  PAGO_AUG  PAGO_JUL  PAGO_JUN  PAGO_MAY  PAGO_APR Y_PREPAGO
0           0.0     689.0       0.0       0.0       0.0       0.0        1.0
1           0.0    1000.0    1000.0    1000.0       0.0    2000.0        1.0
2        1518.0    1500.0    1000.0    1000.0    1000.0    5000.0        0.0
3        2000.0    2019.0    1200.0    1100.0    1069.0    1000.0        0.0
4        2000.0   36681.0   10000.0    9000.0     689.0     679.0        0.0
…           …         …         …         …         …         …          …
29995    8500.0   20000.0    5003.0    3047.0    5000.0    1000.0        0.0
29996    1837.0    3526.0    8998.0     129.0       0.0       0.0        0.0
29997       0.0       0.0   22000.0    4200.0    2000.0    3100.0        1.0
29998   85900.0    3409.0    1178.0    1926.0   52964.0    1804.0        1.0
29999    2078.0    1800.0    1430.0    1000.0    1000.0    1000.0        1.0

[29972 rows x 25 columns]
```

[26]:
```
imprimir_datos_faltantes(df)
```

```
Total de datos faltantes: 14, %: 0.04671026291205125
```

Reemplazo de datos nulos en los registro Para decidir que hacer con los datos resultantes vamos a realizar un análisis de sus distribuciones, posteriormente reemplazar los valores faltantes por la media y ver si se modificaron sus distribuciones para ver si no introducimos un sesgo inconsciente.

[27]:
```
# Imprimir columnas con valores faltantes

columns = df.columns[df.isnull().any()]
columns
```

[27]:
```
Index(['EDAD', 'RETRASO_JUL', 'RETRASO_JUN', 'SALDO_JUL', 'SALDO_APR',
       'PAGO_AUG', 'PAGO_JUN', 'PAGO_MAY', 'PAGO_APR'],
      dtype='object')
```

[28]:
```
# Subconjunto de datos del conjunto original, solo con aquellas columnos donde␣
↪hacen falta valores

df_subset = df[columns]
df_subset.head()
```

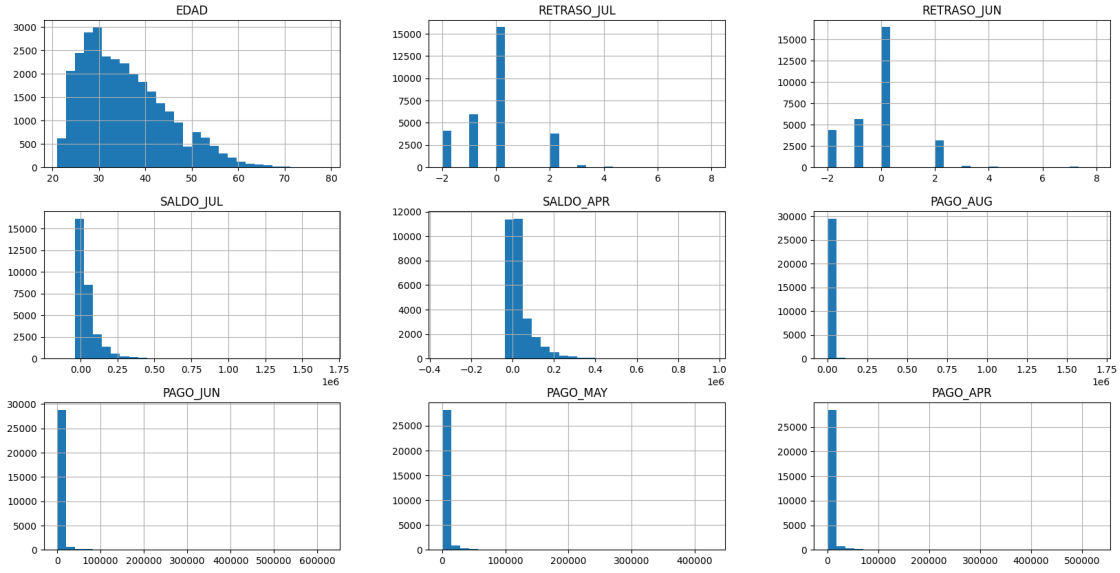
[28]:

	EDAD	RETRASO_JUL	RETRASO_JUN	SALDO_JUL	SALDO_APR	PAGO_AUG	PAGO_JUN	\
0	24.0	-1.0	-1.0	689.0	0.0	689.0	0.0	
1	26.0	0.0	0.0	2682.0	3261.0	1000.0	1000.0	
2	34.0	0.0	0.0	13559.0	15549.0	1500.0	1000.0	
3	37.0	0.0	0.0	49291.0	29547.0	2019.0	1100.0	
4	57.0	-1.0	0.0	35835.0	19131.0	36681.0	9000.0	

	PAGO_MAY	PAGO_APR
0	0.0	0.0
1	0.0	2000.0
2	1000.0	5000.0
3	1069.0	1000.0
4	689.0	679.0

[29]:
```
# Análisis de las distribuciones

df_subset.hist(bins=30, figsize=(20,10))
```

[29]:
```
array([[<AxesSubplot: title={'center': 'EDAD'}>,
        <AxesSubplot: title={'center': 'RETRASO_JUL'}>,
        <AxesSubplot: title={'center': 'RETRASO_JUN'}>],
       [<AxesSubplot: title={'center': 'SALDO_JUL'}>,
        <AxesSubplot: title={'center': 'SALDO_APR'}>,
        <AxesSubplot: title={'center': 'PAGO_AUG'}>],
       [<AxesSubplot: title={'center': 'PAGO_JUN'}>,
        <AxesSubplot: title={'center': 'PAGO_MAY'}>,
        <AxesSubplot: title={'center': 'PAGO_APR'}>]], dtype=object)
```



[30]:
```
stats_subset = df_subset.describe()
stats_subset
```

[30]:

	EDAD	RETRASO_JUL	RETRASO_JUN	SALDO_JUL	SALDO_APR
count	29969.000000	29970.000000	29969.000000	2.997100e+04	29971.000000
mean	35.481965	-0.166834	-0.220928	4.702711e+04	38884.426946
std	9.214738	1.195968	1.169245	6.937403e+04	59572.075536
min	21.000000	-2.000000	-2.000000	-1.572640e+05	-339603.000000
25%	28.000000	-1.000000	-1.000000	2.663500e+03	1256.000000
50%	34.000000	0.000000	0.000000	2.008700e+04	17068.000000
75%	41.000000	0.000000	0.000000	6.017850e+04	49207.500000
max	79.000000	8.000000	8.000000	1.664089e+06	961664.000000

	PAGO_AUG	PAGO_JUN	PAGO_MAY	PAGO_APR
count	2.997100e+04	29971.000000	29971.000000	29971.000000
mean	5.924788e+03	4828.903674	4801.544026	5219.619332
std	2.305159e+04	15673.233370	15284.891095	17785.528195
min	0.000000e+00	0.000000	0.000000	0.000000
25%	8.340000e+02	296.500000	251.000000	118.000000
50%	2.009000e+03	1500.000000	1500.000000	1500.000000
75%	5.000000e+03	4014.000000	4037.000000	4000.000000
max	1.684259e+06	621000.000000	426529.000000	528666.000000

Obtención de las medias para las columnas donde hace falta el valor en alguno de sus registros

[31]:
```
df['EDAD'].fillna(df['EDAD'].mean(), inplace=True)
df['RETRASO_JUL'].fillna(df['RETRASO_JUL'].mean(), inplace=True)
df['RETRASO_JUN'].fillna(df['RETRASO_JUN'].mean(), inplace=True)
df['SALDO_JUL'].fillna(df['SALDO_JUL'].mean(), inplace=True)
df['SALDO_APR'].fillna(df['SALDO_APR'].mean(), inplace=True)
df['PAGO_AUG'].fillna(df['PAGO_AUG'].mean(), inplace=True)
df['PAGO_JUN'].fillna(df['PAGO_JUN'].mean(), inplace=True)
df['PAGO_MAY'].fillna(df['PAGO_MAY'].mean(), inplace=True)
df['PAGO_APR'].fillna(df['PAGO_APR'].mean(), inplace=True)
```

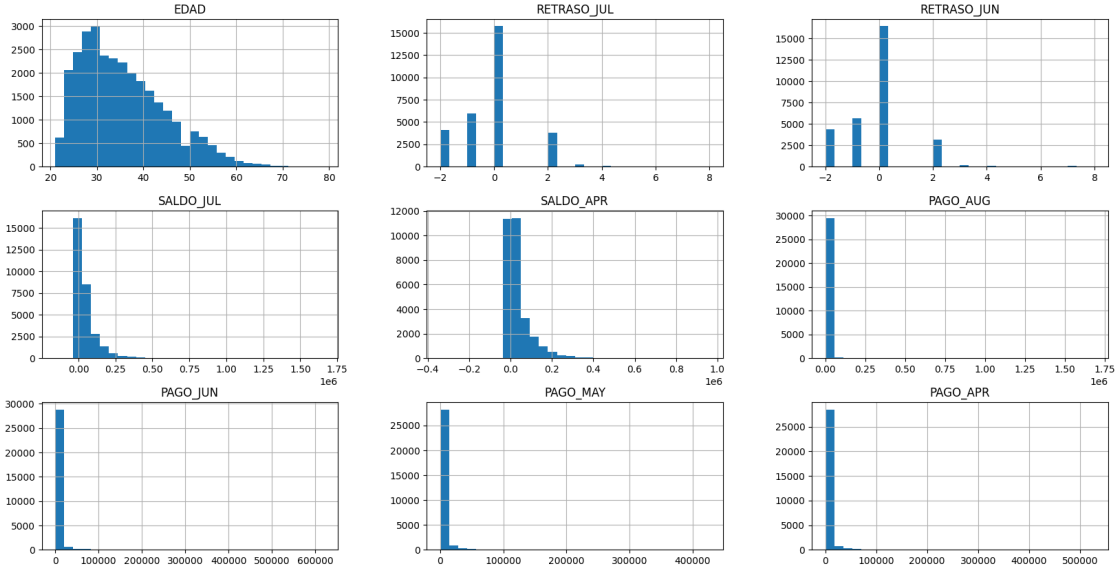
[32]:
```
df.isnull().values.any()
```

[32]:
```
False
```

[33]:
```
df_subset_dropna = df[columns]
df_subset.hist(bins=30, figsize=(20,10))
```

[33]:
```
array([[<AxesSubplot: title={'center': 'EDAD'}>,
        <AxesSubplot: title={'center': 'RETRASO_JUL'}>,
        <AxesSubplot: title={'center': 'RETRASO_JUN'}>],
       [<AxesSubplot: title={'center': 'SALDO_JUL'}>,
        <AxesSubplot: title={'center': 'SALDO_APR'}>,
        <AxesSubplot: title={'center': 'PAGO_AUG'}>],
```

```
[<AxesSubplot: title={'center': 'PAGO_JUN'}>,
 <AxesSubplot: title={'center': 'PAGO_MAY'}>,
 <AxesSubplot: title={'center': 'PAGO_APR'}>]], dtype=object)
```



[34]:
```
stats_subset_dropna = df_subset_dropna.describe()
stats_subset_dropna
```

[34]:

	EDAD	RETRASO_JUL	RETRASO_JUN	SALDO_JUL	SALDO_APR	\
count	29972.000000	29972.000000	29972.000000	2.997200e+04	29972.000000	
mean	35.481965	-0.166834	-0.220928	4.702711e+04	38884.426946	
std	9.214276	1.195928	1.169186	6.937287e+04	59571.081699	
min	21.000000	-2.000000	-2.000000	-1.572640e+05	-339603.000000	
25%	28.000000	-1.000000	-1.000000	2.663750e+03	1256.000000	
50%	34.000000	0.000000	0.000000	2.008750e+04	17071.000000	
75%	41.000000	0.000000	0.000000	6.017775e+04	49206.750000	
max	79.000000	8.000000	8.000000	1.664089e+06	961664.000000	

	PAGO_AUG	PAGO_JUN	PAGO_MAY	PAGO_APR
count	2.997200e+04	29972.000000	29972.000000	29972.000000
mean	5.924788e+03	4828.903674	4801.544026	5219.619332
std	2.305121e+04	15672.971894	15284.636098	17785.231480
min	0.000000e+00	0.000000	0.000000	0.000000
25%	8.345000e+02	296.750000	251.000000	118.000000
50%	2.009000e+03	1500.000000	1500.000000	1500.000000
75%	5.000000e+03	4014.000000	4037.750000	4000.000000
max	1.684259e+06	621000.000000	426529.000000	528666.000000

Comparamos las distribuciones antes y después de hacer el reemplazo de los datos y no vimos cambios significativos, sin embargo para corroborarlo comparamos las estadísticas antes y después de la

sustitución de los valores nulos por la media y de corroboramos que no hay cambios significativos.

En la tabla que se muestra a continuación los valores que se despliegan como *NaN* significa que el valor es el mismo para ambos casos.

[35]:
```
stats_subset.compare(stats_subset_dropna)
```

[35]:

	EDAD		RETRASO_JUL		RETRASO_JUN	\
	self	other	self	other	self	
count	29969.000000	29972.000000	29970.000000	29972.000000	29969.000000	
mean	35.481965	35.481965	NaN	NaN	NaN	
std	9.214738	9.214276	1.195968	1.195928	1.169245	
25%	NaN	NaN	NaN	NaN	NaN	
50%	NaN	NaN	NaN	NaN	NaN	
75%	NaN	NaN	NaN	NaN	NaN	

	SALDO_JUL		SALDO_APR		\
	other	self	other	self	other
count	29972.000000	29971.000000	29972.000000	29971.000000	29972.000000
mean	NaN	NaN	NaN	38884.426946	38884.426946
std	1.169186	69374.026355	69372.868993	59572.075536	59571.081699
25%	NaN	2663.500000	2663.750000	NaN	NaN
50%	NaN	20087.000000	20087.500000	17068.000000	17071.000000
75%	NaN	60178.500000	60177.750000	49207.500000	49206.750000

	PAGO_AUG		PAGO_JUN		PAGO_MAY	\
	self	other	self	other	self	
count	29971.000000	29972.000000	29971.000000	29972.000000	29971.000000	
mean	NaN	NaN	4828.903674	4828.903674	NaN	
std	23051.591035	23051.206467	15673.233370	15672.971894	15284.891095	
25%	834.000000	834.500000	296.500000	296.750000	NaN	
50%	NaN	NaN	NaN	NaN	NaN	
75%	NaN	NaN	NaN	NaN	4037.000000	

	PAGO_APR		
	other	self	other
count	29972.000000	29971.000000	29972.00000
mean	NaN	NaN	NaN
std	15284.636098	17785.528195	17785.23148
25%	NaN	NaN	NaN
50%	NaN	NaN	NaN
75%	4037.750000	NaN	NaN

Análisis de variables categóricas Necesitamos verificar que las variables categóricas tengan valores válidos con base a la codificación que se les asignó, en este caso para *'SEXO','ESCOLARIDAD','ESTADO-CIVIL','Y-PREPAGO'*

[36]:
```
# factorización de los datos

df['SEXO'] = df['SEXO'].astype('int')
df['ESCOLARIDAD'] = df['ESCOLARIDAD'].astype('int')
df['ESTADO_CIVIL'] = df['ESTADO_CIVIL'].astype('int')
df['Y_PREPAGO'] = df['Y_PREPAGO'].astype('int')
```

Variable *'SEXO'* Verificación para variable *'SEXO'* que debe tener valores en el rango [1,2] (1 = masculino; 2 = femenino)

[37]:
```
# Guardar columna en una serie

col_sexo = df['SEXO']
```

[38]:
```
# Checar valor mínimo que no se salgo del rango

col_sexo.min()
```

[38]: 1

[39]:
```
# Checar valor máximo que no se salgo del rango

col_sexo.max()
```

[39]: 2

[40]:
```
# Checar que no haya un valor fuera del rango

for idx, value in enumerate(col_sexo):
    if value <= 0 or value >= 3:
        print(f'index: {idx}, val: {value}')
```

[41]:
```
# refactorización del dato

df['SEXO'] = df['SEXO'].astype('category')
```

Los valores de la variable *'SEXO'* son correctos

Variable '*Y_PREPAGO*' Verificación para variable '*Y-PREPAGO*' que debe tener valores en el rango [0,1] (0 = No; 1 = Sí)

[42]:
```
# Guardar columna en una serie

col_prepago = df['Y_PREPAGO']
```

[43]:
```
# Checar valor mínimo que no se salgo del rango

col_prepago.min()
```

[43]: 0

[44]:
```
# Checar valor máximo que no se salgo del rango

col_prepago.max()
```

[44]: 1

[45]:
```
# refactorización del dato

df['Y_PREPAGO'] = df['Y_PREPAGO'].astype('category')
```

Los valores de la variable *'Y-PREPAGO'* son correctos

Verificación para variable *'ESCOLARIDAD'* que debe tener valores en el rango [1,4] (1 = posgrado; 2 = universidad; 3 = secundaria; 4 = otros)

Variable *'ESCOLARIDAD'* Verificación para variable *'ESCOLARIDAD'* que debe tener valores en el rango [1,4] (1 = posgrado; 2 = universidad; 3 = secundaria; 4 = otros)

[46]:
```
# Guardar columna en una serie

col_escolaridad = df['ESCOLARIDAD']
```

[47]:
```
# Checar valor mínimo que no se salgo del rango

col_escolaridad.min()
```

[47]: 0

[48]:
```
# Checar valor máximo que no se salgo del rango

col_escolaridad.max()
```

[48]: 6

[49]:
```
count_0 = 0
count_5 = 0
count_6 = 0

for idx, value in enumerate(col_escolaridad):
    if value == 0:
        count_0 += 1
    elif value == 5:
        count_5 += 1
    elif value == 6:
        count_6 += 1
```

```
print(f'n_valores = 0: {count_0}, n_valores = 5: {count_5}, n_valores = 6:␣
 ↪{count_6}')
```

```
n_valores = 0: 14, n_valores = 5: 280, n_valores = 6: 51
```

En *'ESCOLARIDAD'* tenemos el problema de que no sabemos qué significa cuando el registro tiene un valor de 0, 5 ó 6. en este caso vamos a hacer la suposición de que el valor fue erróneamente codificado y remplazaremos el valor incorrecto por el más cercano en el rango, es decir los ceros por unos, y los cincos y seis por cuatro.

[50]:
```
df['ESCOLARIDAD'].replace(0, 1, inplace=True)
df['ESCOLARIDAD'].replace(5, 4, inplace=True)
df['ESCOLARIDAD'].replace(6, 4, inplace=True)
```

[51]:
```
# refactorización del dato

df['ESCOLARIDAD'] = df['ESCOLARIDAD'].astype('category')
```

Variable '*ESTADO-CIVIL*' Verificación para variable '*ESTADO-CIVIL*' que debe tener valores en el rango [1,3] (1 = casado; 2 = soltero; 3 = otros)

[52]:
```
# Guardar columna en una serie

col_estado_civil = df['ESTADO_CIVIL']
```

[53]:
```
# Checar valor mínimo que no se salgo del rango

col_estado_civil.min()
```

[53]:
```
0
```

[54]:
```
# Checar valor máximo que no se salgo del rango

col_estado_civil.max()
```

[54]:
```
3
```

[55]:
```
count_0 = 0
for idx, value in enumerate(col_estado_civil):
    if value == 0:
        count_0 += 1

print(f'n_valores = 0: {count_0}')
```

```
n_valores = 0: 54
```

En '*ESTADO-CIVIL*' tenemos el problema de que no sabemos qué significa cuando el registro tiene un valor de 0, en este caso vamos a hacer la suposición de que el valor fue erróneamente codificado y remplazaremos el valor incorrecto por el más cercano en el rango, es decir los ceros por unos.

[56]:
```
# Aplicamos el método replace para sustituir los valores

df['ESTADO_CIVIL'].replace(0, 1, inplace=True)
```

[57]:
```
# refactorización del dato

df['ESTADO_CIVIL'] = df['ESTADO_CIVIL'].astype('category')
```

1.3.6 Exportado de los Datos Limpados

Una vez teniendo ya los datos sin valores nulos o incorrectos procedemos a exportarlos como archivo '.csv' para su manejo posterior

[58]:
```
df.to_csv('default_of_credit_card_clients-clean_data.csv')
```

1.3.7 Análisis de Correlación entre los Datos

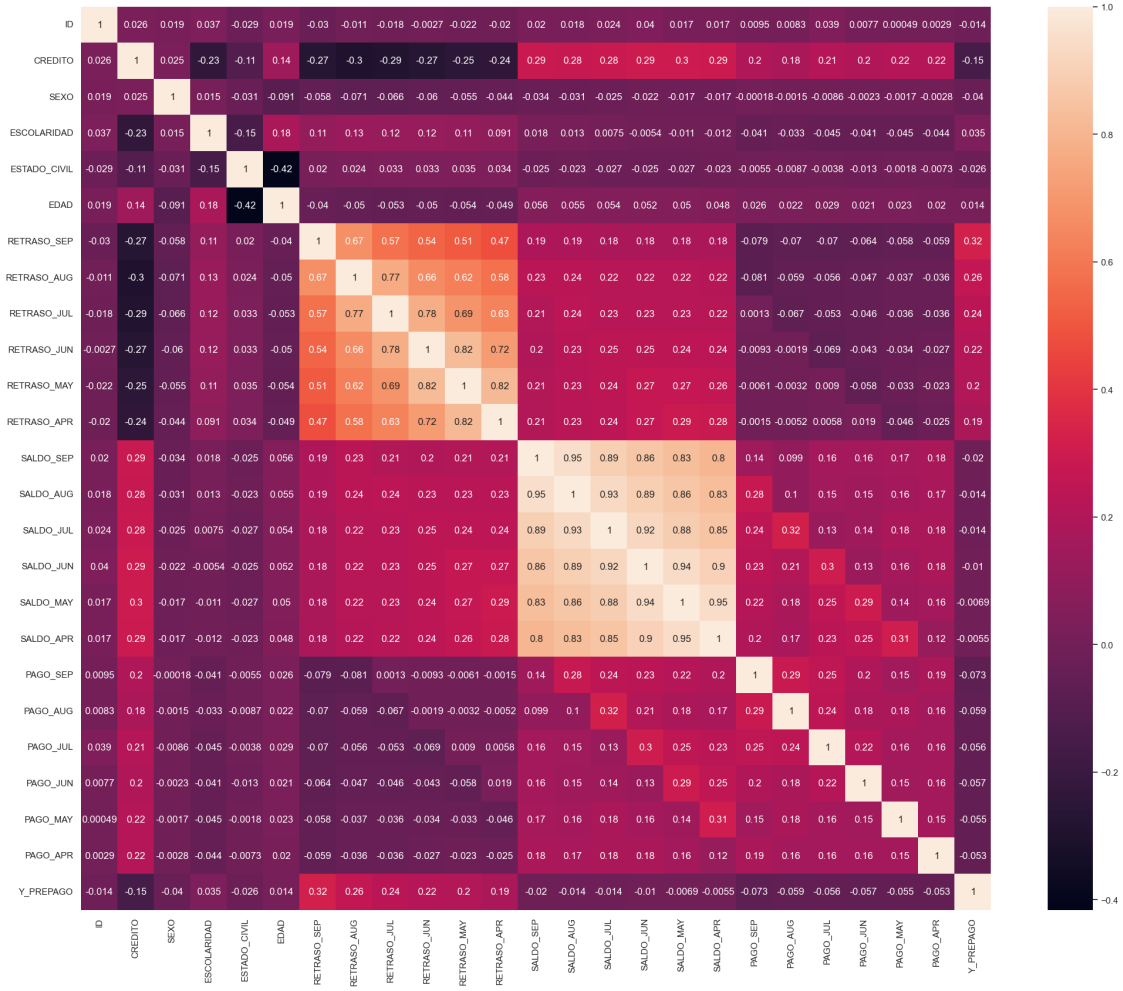
Antes de obtener nuestro análisis de correlación, necesitamos factorizar las variables categóricas para que podamos utilizar los métodos de la librería de pandas para obtener la matriz de correlación y los valores de correlación (utilizando el método de Pearson, Kendall o Spearman.

[59]:
```
# factorización de los datos

df['ID'] = df['ID'].astype('int')
df['SEXO'] = df['SEXO'].astype('int')
df['ESCOLARIDAD'] = df['ESCOLARIDAD'].astype('int')
df['ESTADO_CIVIL'] = df['ESTADO_CIVIL'].astype('int')
df['Y_PREPAGO'] = df['Y_PREPAGO'].astype('int')
```

Matriz de correlación

[60]:
```
# Obtención y gráfica de matriz de correlación
matriz_de_correlacion = df.corr()

# Gráfica de matriz de correlación con seaborn
sns.set(rc={'figure.figsize':(25,20)})
sns.heatmap(matriz_de_correlacion, annot=True)
```

[60]:
```
<AxesSubplot: >
```



Obtención de rango de Spearman

[61]:
```
print(df[df.columns].corrwith(df['Y_PREPAGO'], method='spearman'))
```

```
ID              -0.248558
CREDITO         -0.134090
SEXO            -0.036201
ESCOLARIDAD      0.022687
ESTADO_CIVIL     0.047631
EDAD            -0.009264
RETRASO_SEP      0.239113
RETRASO_AUG      0.154318
RETRASO_JUL      0.137980
RETRASO_JUN      0.117471
RETRASO_MAY      0.106554
RETRASO_APR      0.082581
SALDO_SEP       -0.032513
```

```
SALDO_AUG      -0.025879
SALDO_JUL      -0.026843
SALDO_JUN      -0.029422
SALDO_MAY      -0.024710
SALDO_APR      -0.019671
PAGO_SEP       -0.122130
PAGO_AUG       -0.128784
PAGO_JUL       -0.134488
PAGO_JUN       -0.103708
PAGO_MAY       -0.094538
PAGO_APR       -0.100423
Y_PREPAGO       1.000000
dtype: float64
```

1.4 3. Preparación de los Datos

1.4.1 3.1. ¿Qué datos consideró mas importantes? ¿Por qué?

Realizamos un análisis de correlación empleando el método de Spearman. La correlación de rango de Spearman es una prueba no paramétrica que mide el nivel de asociación entre dos variables, no hace ninguna inferencia sobre la distribución de los datos y permite teneder datos ordinales (Statistics Solutions, s.f.).

$$\rho = 1 - \frac{6 \sum d_i^2}{n(n^2 - 1)}$$

De acuerdo al análisis realizado haciendo la correlación de cada una de las variables con respecto al *output* de los datos, variable *'Y'* (*'Y_PAGO*'), estás fueron las variables más importantes:

ATRIBUTO	RANGO DE SPEARMAN
EDAD	0.004957
RETRASO_SEP	0.292186
RETRASO_AUG	0.216866
RETRASO_JUL	0.194749
RETRASO_JUN	0.173553
RETRASO_MAY	0.158890
RETRASO_APR	0.142169

1.4.2 3.2. ¿Se eliminaron o reemplazaron datos nulos? ¿Qué se hizo y por qué?

Se realizaron ambas, sin embargo, primero se describieron los datos, se analizaron y se procedió con los siguientes pasos:

Para la eliminación de registros, se decidió por emplear el criterio de eliminación por lista que es la opción predeterminada ya que inferimos que faltan datos completamente al azar (MCAR - Missed at random) y eliminamos sólo aquellos en cuyo registro (fila) hubieran dos o más datos faltantes :

1. Primero tomamos la columna '*Y-PREPAGO*' que es la salida ('*y*' - *Output* de los datos_).

2. Analizamos cuál era la columna con más datos nulos, en este caso la de '*RETRASO-MAY*' y empleamos el mismo criterio de eliminación por lista anterior.
3. Volvimos a iterarar el paso dos. En este caso la columna con más datos nulos es la de '*SALDO-MAY*'.
4. De los datos faltantes empleamos el mismo método anterior pero ya con respecto a todas las columnas que nos quedamos y en este caso empleando empleando un *'tresh'* para quedarnos sólo con las filas que no tengan dos o más valores nulos.

Con los datos que quedaron procedimos con hacer una sustitución media ya que partimos de la inferencia de que faltan esos datos de forma aleatoria y verificamos que no estuvieramos introduciendo un sesgo inconsciente:

1. Obtuvimos las estadísticas y distrubuciones antes de sustituir los datos faltantes.
2. Procedimos a obtener las medias de aquellas columnas donde habían datos faltantes.
3. Reemplazamos los datos faltantes por las medias.
4. Comparamos y verificamos que no hubieran cambios que introducieran un sesgo inconsciente.

1.4.3 3.3. ¿Es necesario ordenar los datos para el análisis? Sí / No / ¿Por qué?

No es necesario ya que estamos trabajando con datos estructurados que tienen un esquema con campos (atributos) definidos. También si nos fijamos en el atributo de *ID* podemos ver que ya hay un ordenamiento preestablecido con base a la identificación del cliente en la base de datos. Sin embargo, eso no significa que no tengamos que generar subconjuntos de datos y agruparlos para poderlos analizar de mejor forma.

1.4.4 3.4. ¿Existen problemas de formato que deban solucionar antes del proceso de modelado? Sí / No / Por qué.

Consideramos que sí:

- Es mejor renombrar los atributos originales de los datos para que sean más descriptivos y entendibles con base a la Descripción del dataset. y se facilite su análisis posterior.
- La codificación de los datos, atributos como *'ID', 'SEXO', 'ESCOLARIDAD', 'ESTADO-CIVIL', 'Y-PREPAGO'* deben estar como tipo de datos categóricos.

1.4.5 3.5. ¿Qué ajustes se realizaron en el proceso de limpieza de datos (agregar, integrar, eliminar, modificar registros (filas), cambiar atributos (columnas)?

Empleamos diferentes métodos:

- Eliminar registros: ver respuesta 3.2
  - Modificar registros:
    - Sustituación Media: ver respuesta 3.2
    - Análisis de las variables categóricas, que los valores que tienen sean correctas y esten dentro del rango indicado en la Descripción del dataset.

ATRIBUTO	VALORES VÁLIDOS
SEXO	SÍ
ESCOLARIDAD	NO, n_valores = 0: 14, n_valores = 5: 280, n_valores = 6: 51
ESTADO_CIVIL	NO, n_valores = 0: 54
Y_PREPAGO	SÍ

En este caso hicimos la suposición de que el dato fue incorrectamente codificados y procedimos con reemplazarlos por el valor más cercano en el rango permitido.

1.5 Referencias

Kane, F. (2017). Hands-On Data Science and Python Machine Learning. Packt Publishing. Obtenido de https://newoutlook.it/download/python/hands-on-data-science.pdf

Teate, R. M. (2021). SQL for Data Scientists. A Beginner's Guide for Building Datasets for Analysis. Wiley. Obtenido de https://learning.oreilly.com/library/view/sql-for-data/9781119669364/

Statistics Solutions. (s.f.). Correlation (Pearson, Kendall, Spearman). Recuperado el 2 de october de 2022, de Complete Dissertation. Expert Guidance Every Step of the Way: https://www.statisticssolutions.com/free-resources/directory-of-statistical-analyses/correlation-pearson-kendall-spearman/