Tecnológico de Monterrey

# Actividad Semanal 6 Visualización

## Ciencia y Analítica de datos

## *Maestría en Inteligencia Artificial Aplicada (MNA-V)*

***Victor Hugo Avila Felipe - A01794425***

## Enlace a github para subir las tareas.

[https://github.com/PosgradoMNA/actividades-de-aprendizaje-flynnGodslayer (https://github.com/PosgradoMNA/actividades-de-aprendizaje-flynnGodslayer)](https://github.com/PosgradoMNA/actividades-de-aprendizaje-flynnGodslayer)

## 1. Descarga los datos y carga el dataset en tu libreta.

### Información del dataset:

**Nombre dataset:** case of customers' default payments in Taiwan.

**Descripción de datos:** Pagos por default de clientes en Taiwan. Compara la precisión de la predicción entre seis metodos de minería de datos. Se desea obtener los clientes creíbles y no creíbles para gestionar el riego de asignación de créditos. La probabilidad real de incumplimiento es dada por Y, y la variable independiente es X. El resultado de la regresión lineal simple (Y = A + BX) muestra que el modelo de pronóstico producido por la red neuronal artificial tiene el coeficiente de determinación más alto; su intersección de regresión (A) es cercana a cero y el coeficiente de regresión (B) a uno. Por lo tanto, entre las seis técnicas de minería de datos, la red neuronal artificial es la única que puede estimar con precisión la probabilidad real de incumplimiento.

**Obtenido de:** Yeh, I. C., & Lien, C. H. (2009). The comparisons of data mining techniques for the predictive accuracy of probability of default of credit card clients. Expert Systems with Applications, 36(2), 2473-2480.

A continuación se van a importar las librerías a utilizar y el dataset:

In [4]:
```python
import pandas as pd
import numpy as np

# Cargamos el dataframe desde la liga proporcionada
mypath = "https://raw.githubusercontent.com/PosgradoMNA/Actividades_Aprendizaj

df = pd.read_csv(mypath, index_col=0)
df.index.name = None
```

# 2. Obten la información del DataFrame con los métodos y propiedades: shape, columns, head(), dtypes, info(), isna()

**head()**

In [5]:
```python
# Revisamos que se cargó correctamenta el dataframe y visualizamos los primero

df.head()
```

Out[5]:

| | X1 | X2 | X3 | X4 | X5 | X6 | X7 | X8 | X9 | X10 | ... | X15 | X16 | X17 | X18 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **1** | 20000 | 2.0 | 2.0 | 1.0 | 24.0 | 2.0 | 2.0 | -1.0 | -1.0 | -2.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | |
| **2** | 120000 | 2.0 | 2.0 | 2.0 | 26.0 | -1.0 | 2.0 | 0.0 | 0.0 | 0.0 | ... | 3272.0 | 3455.0 | 3261.0 | 0.0 | |
| **3** | 90000 | 2.0 | 2.0 | 2.0 | 34.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 14331.0 | 14948.0 | 15549.0 | 1518.0 | |
| **4** | 50000 | 2.0 | 2.0 | 1.0 | 37.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 28314.0 | 28959.0 | 29547.0 | 2000.0 | |
| **5** | 50000 | 1.0 | 2.0 | 1.0 | 57.0 | -1.0 | 0.0 | -1.0 | 0.0 | 0.0 | ... | 20940.0 | 19146.0 | 19131.0 | 2000.0 | 3 |

5 rows × 24 columns

**shape()**

In [6]:
```python
# Revisamos las dimensiones del dataframe (Registros x Columnas)
df.shape
```

Out[6]:
```
(30000, 24)
```

**columns()**

In [7]:
```python
# Revisamos un listado del nombre de las columnas asignada al datadframe

df.columns
```

Out[7]:
```
Index(['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')
```

**dtypes()**

In [8]:
```
# Se aplica un listado de los tipos de datos que se asignaron de manera automá

df.dtypes
```

Out[8]:
```
X1        int64
X2      float64
X3      float64
X4      float64
X5      float64
X6      float64
X7      float64
X8      float64
X9      float64
X10     float64
X11     float64
X12     float64
X13     float64
X14     float64
X15     float64
X16     float64
X17     float64
X18     float64
X19     float64
X20     float64
X21     float64
X22     float64
X23     float64
Y       float64
dtype: object
```

### info()

In [9]:
```python
# Se imprime un sumario de la información general del dataframe

df.info
```

Out[9]:
```
<bound method DataFrame.info of              X1    X2    X3    X4    X5    X6    X7
X8    X9  X10  ...      X15  \
1       20000  2.0  2.0  1.0  24.0  2.0  2.0 -1.0 -1.0 -2.0  ...      0.0
2      120000  2.0  2.0  2.0  26.0 -1.0  2.0  0.0  0.0  0.0  ...   3272.0
3       90000  2.0  2.0  2.0  34.0  0.0  0.0  0.0  0.0  0.0  ...  14331.0
4       50000  2.0  2.0  1.0  37.0  0.0  0.0  0.0  0.0  0.0  ...  28314.0
5       50000  1.0  2.0  1.0  57.0 -1.0  0.0 -1.0  0.0  0.0  ...  20940.0
...       ...  ...  ...  ...   ...  ...  ...  ...  ...  ...  ...      ...
29996  220000  1.0  3.0  1.0  39.0  0.0  0.0  0.0  0.0  0.0  ...  88004.0
29997  150000  1.0  3.0  2.0  43.0 -1.0 -1.0 -1.0 -1.0  0.0  ...   8979.0
29998   30000  1.0  2.0  2.0  37.0  4.0  3.0  2.0 -1.0  0.0  ...  20878.0
29999   80000  1.0  3.0  1.0  41.0  1.0 -1.0  0.0  0.0  0.0  ...  52774.0
30000   50000  1.0  2.0  1.0  46.0  0.0  0.0  0.0  0.0  0.0  ...  36535.0

           X16      X17      X18      X19      X20     X21      X22     X23
\
1          0.0      0.0      0.0    689.0      0.0     0.0      0.0     0.0
2       3455.0   3261.0      0.0   1000.0   1000.0  1000.0      0.0  2000.0
3      14948.0  15549.0   1518.0   1500.0   1000.0  1000.0   1000.0  5000.0
4      28959.0  29547.0   2000.0   2019.0   1200.0  1100.0   1069.0  1000.0
5      19146.0  19131.0   2000.0  36681.0  10000.0  9000.0    689.0   679.0
...        ...      ...      ...      ...      ...     ...      ...     ...
29996  31237.0  15980.0   8500.0  20000.0   5003.0  3047.0   5000.0  1000.0
29997   5190.0      0.0   1837.0   3526.0   8998.0   129.0      0.0     0.0
29998  20582.0  19357.0      0.0      0.0  22000.0  4200.0   2000.0  3100.0
29999  11855.0  48944.0  85900.0   3409.0   1178.0  1926.0  52964.0  1804.0
30000  32428.0  15313.0   2078.0   1800.0   1430.0  1000.0   1000.0  1000.0

         Y
1      1.0
2      1.0
3      0.0
4      0.0
5      0.0
...    ...
29996  0.0
29997  0.0
29998  1.0
29999  1.0
30000  1.0

[30000 rows x 24 columns]>
```

### isna()

In [10]:
```
# Función que define si existe un valor nulo (NA)
# dentro de los registros del dataframe.

df.isna
```

Out[10]:
```
<bound method DataFrame.isna of            X1   X2   X3   X4    X5   X6   X7
X8   X9  X10  ...      X15  \
1       20000  2.0  2.0  1.0  24.0  2.0  2.0 -1.0 -1.0 -2.0  ...      0.0
2      120000  2.0  2.0  2.0  26.0 -1.0  2.0  0.0  0.0  0.0  ...   3272.0
3       90000  2.0  2.0  2.0  34.0  0.0  0.0  0.0  0.0  0.0  ...  14331.0
4       50000  2.0  2.0  1.0  37.0  0.0  0.0  0.0  0.0  0.0  ...  28314.0
5       50000  1.0  2.0  1.0  57.0 -1.0  0.0 -1.0  0.0  0.0  ...  20940.0
...       ...  ...  ...  ...   ...  ...  ...  ...  ...  ...  ...      ...
29996  220000  1.0  3.0  1.0  39.0  0.0  0.0  0.0  0.0  0.0  ...  88004.0
29997  150000  1.0  3.0  2.0  43.0 -1.0 -1.0 -1.0 -1.0  0.0  ...   8979.0
29998   30000  1.0  2.0  2.0  37.0  4.0  3.0  2.0 -1.0  0.0  ...  20878.0
29999   80000  1.0  3.0  1.0  41.0  1.0 -1.0  0.0  0.0  0.0  ...  52774.0
30000   50000  1.0  2.0  1.0  46.0  0.0  0.0  0.0  0.0  0.0  ...  36535.0

           X16      X17      X18      X19      X20     X21      X22     X23
\
1          0.0      0.0      0.0    689.0      0.0     0.0      0.0     0.0
2       3455.0   3261.0      0.0   1000.0   1000.0  1000.0      0.0  2000.0
3      14948.0  15549.0   1518.0   1500.0   1000.0  1000.0   1000.0  5000.0
4      28959.0  29547.0   2000.0   2019.0   1200.0  1100.0   1069.0  1000.0
5      19146.0  19131.0   2000.0  36681.0  10000.0  9000.0    689.0   679.0
...        ...      ...      ...      ...      ...     ...      ...     ...
29996  31237.0  15980.0   8500.0  20000.0   5003.0  3047.0   5000.0  1000.0
29997   5190.0      0.0   1837.0   3526.0   8998.0   129.0      0.0     0.0
29998  20582.0  19357.0      0.0      0.0  22000.0  4200.0   2000.0  3100.0
29999  11855.0  48944.0  85900.0   3409.0   1178.0  1926.0  52964.0  1804.0
30000  32428.0  15313.0   2078.0   1800.0   1430.0  1000.0   1000.0  1000.0

         Y
1      1.0
2      1.0
3      0.0
4      0.0
5      0.0
...    ...
29996  0.0
29997  0.0
29998  1.0
29999  1.0
30000  1.0

[30000 rows x 24 columns]>
```

In [11]:
```
# Para tener una mejor visualización de los datos NA, obtenemos
# una flag de si existen en el dataframe.

df.isna().values.any()
```

Out[11]:
```
True
```

In [12]:
```
# Tambien revisamos que columnas contienen valores NA

df.isna().any()
```

Out[12]:
```
X1     False
X2      True
X3      True
X4      True
X5      True
X6      True
X7      True
X8      True
X9      True
X10     True
X11     True
X12     True
X13     True
X14     True
X15     True
X16     True
X17     True
X18     True
X19     True
X20     True
X21     True
X22     True
X23     True
Y       True
dtype: bool
```

In [13]:
```
# Verificamos la totalidad de registros que contienen en por lo menos uno de
# sus registros con NA
df[df.isna().any(axis=1)]
```

Out[13]:

| | X1 | X2 | X3 | X4 | X5 | X6 | X7 | X8 | X9 | X10 | ... | X15 | X16 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **19** | 360000 | 2.0 | 1.0 | 1.0 | 49.0 | 1.0 | -2.0 | -2.0 | -2.0 | -2.0 | ... | NaN | NaN | |
| **39** | 50000 | 1.0 | 1.0 | 2.0 | 25.0 | 1.0 | -1.0 | -1.0 | -2.0 | -2.0 | ... | 0.0 | 0.0 | |
| **50** | 20000 | 1.0 | 1.0 | 2.0 | 24.0 | 0.0 | 0.0 | 0.0 | 0.0 | NaN | ... | 19865.0 | 20480.0 | 200 |
| **65** | 130000 | 2.0 | 2.0 | 1.0 | 51.0 | -1.0 | -1.0 | -2.0 | -2.0 | -1.0 | ... | 0.0 | 2353.0 | |
| **161** | 30000 | 1.0 | 1.0 | 2.0 | 41.0 | 2.0 | 2.0 | 2.0 | NaN | 2.0 | ... | 28168.0 | 27579.0 | 283 |
| **174** | 50000 | 2.0 | 1.0 | 2.0 | 24.0 | 1.0 | -2.0 | -2.0 | -2.0 | -2.0 | ... | -2898.0 | -3272.0 | -32 |
| **176** | 130000 | 1.0 | 3.0 | 1.0 | 56.0 | 1.0 | 2.0 | 2.0 | 2.0 | 2.0 | ... | 68557.0 | NaN | 713 |
| **183** | 500000 | 2.0 | 1.0 | 1.0 | NaN | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 122967.0 | 108834.0 | 700 |
| **220** | 310000 | 2.0 | 1.0 | 2.0 | NaN | -1.0 | -1.0 | -1.0 | -1.0 | -2.0 | ... | 0.0 | 0.0 | |
| **234** | 190000 | 1.0 | 2.0 | 2.0 | 34.0 | 2.0 | 0.0 | 0.0 | 0.0 | NaN | ... | 142323.0 | 140120.0 | 1500 |
| **240** | 140000 | 2.0 | 2.0 | 3.0 | NaN | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 19068.0 | 16409.0 | 163 |
| **241** | 60000 | 2.0 | 1.0 | 2.0 | 28.0 | 1.0 | 2.0 | 2.0 | -2.0 | -2.0 | ... | 0.0 | NaN | 22 |
| **246** | 20000 | 2.0 | 2.0 | 2.0 | 40.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 19575.0 | 0.0 | |
| **247** | 250000 | 2.0 | 2.0 | 1.0 | 75.0 | 0.0 | -1.0 | -1.0 | -1.0 | -1.0 | ... | 1010.0 | 5572.0 | 7 |
| **248** | 100000 | 2.0 | 2.0 | 2.0 | 27.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 30337.0 | 30997.0 | 329 |
| **6228** | 30000 | 1.0 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | ... | NaN | NaN | 211 |
| **6233** | 60000 | 2.0 | 2.0 | 2.0 | 29.0 | 2.0 | 2.0 | 2.0 | 0.0 | 0.0 | ... | NaN | NaN | |
| **6269** | 120000 | 1.0 | 2.0 | 2.0 | 29.0 | 1.0 | 2.0 | 0.0 | 0.0 | NaN | ... | 91380.0 | 93263.0 | 950 |
| **6270** | 400000 | 2.0 | 1.0 | 2.0 | 27.0 | 1.0 | -2.0 | -1.0 | 0.0 | NaN | ... | 12640.0 | 14805.0 | 2 |
| **6271** | 50000 | 2.0 | 1.0 | 1.0 | 55.0 | 0.0 | 0.0 | 0.0 | 0.0 | NaN | ... | 19431.0 | 19837.0 | 202 |
| **6277** | 30000 | 1.0 | 3.0 | 1.0 | 32.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | NaN | NaN | 14 |
| **6278** | 110000 | 2.0 | 1.0 | 2.0 | 32.0 | 1.0 | 2.0 | 0.0 | 0.0 | 0.0 | ... | NaN | NaN | 254 |
| **6279** | 20000 | 2.0 | 3.0 | 2.0 | 54.0 | 0.0 | 0.0 | 2.0 | 2.0 | 2.0 | ... | NaN | NaN | 91 |
| **6383** | 80000 | 2.0 | 2.0 | 1.0 | 30.0 | 1.0 | 2.0 | 0.0 | 0.0 | NaN | ... | 75374.0 | 77158.0 | 787 |
| **6384** | 90000 | 1.0 | 2.0 | 2.0 | 29.0 | 0.0 | 0.0 | 0.0 | 0.0 | NaN | ... | 69362.0 | 70537.0 | 567 |
| **6385** | 20000 | 2.0 | 2.0 | 2.0 | 22.0 | 0.0 | 0.0 | 0.0 | 0.0 | NaN | ... | 20055.0 | 19606.0 | 199 |
| **17990** | 80000 | 1.0 | 2.0 | 1.0 | 46.0 | 0.0 | 0.0 | NaN | 0.0 | 0.0 | ... | 64237.0 | 47890.0 | 480 |
| **17991** | 50000 | 1.0 | 3.0 | 2.0 | 41.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 50269.0 | 29182.0 | 290 |
| **18004** | 70000 | 1.0 | 2.0 | 2.0 | 33.0 | 1.0 | 2.0 | NaN | 7.0 | 7.0 | ... | 38922.0 | 38318.0 | 381 |
| **23944** | 20000 | 1.0 | 2.0 | 2.0 | 24.0 | 0.0 | 0.0 | 2.0 | 3.0 | 2.0 | ... | 17755.0 | 17967.0 | 175 |
| **23991** | 60000 | 1.0 | 2.0 | 2.0 | 26.0 | 0.0 | 0.0 | 0.0 | NaN | 0.0 | ... | 58938.0 | 30101.0 | 297 |
| **23992** | 50000 | 1.0 | 3.0 | 2.0 | 25.0 | 0.0 | 0.0 | 0.0 | NaN | 2.0 | ... | 49485.0 | 49083.0 | 451 |
| **23993** | 50000 | 1.0 | 2.0 | 2.0 | 24.0 | 0.0 | 0.0 | 0.0 | NaN | 0.0 | ... | 41286.0 | 16764.0 | 169 |

| | X1 | X2 | X3 | X4 | X5 | X6 | X7 | X8 | X9 | X10 | ... | X15 | X16 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **24124** | 50000 | 1.0 | 1.0 | 2.0 | 29.0 | NaN | NaN | NaN | NaN | NaN | ... | NaN | NaN | |
| **24366** | 130000 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | ... | NaN | NaN | |
| **29734** | 300000 | 1.0 | 2.0 | 1.0 | 37.0 | -2.0 | -2.0 | -1.0 | -1.0 | NaN | ... | NaN | NaN | |
| **29735** | 500000 | 1.0 | 2.0 | 1.0 | 49.0 | -1.0 | -1.0 | 2.0 | -1.0 | NaN | ... | NaN | NaN | |
| **29736** | 50000 | 1.0 | 3.0 | 1.0 | 40.0 | 2.0 | 0.0 | 0.0 | 0.0 | NaN | ... | NaN | NaN | |
| **29825** | 40000 | 1.0 | 1.0 | 1.0 | 47.0 | 2.0 | 2.0 | 2.0 | 2.0 | 2.0 | ... | NaN | NaN | |
| **29826** | 50000 | 1.0 | 2.0 | 1.0 | 41.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | NaN | NaN | |
| **29833** | 20000 | 1.0 | 2.0 | 2.0 | 30.0 | 0.0 | NaN | NaN | NaN | NaN | ... | NaN | NaN | 200 |
| **29834** | 20000 | 1.0 | 2.0 | 2.0 | 31.0 | -1.0 | NaN | NaN | NaN | NaN | ... | NaN | NaN | 191 |

# 3. Limpia los datos eliminando los registros nulos o rellena con la media de la columna

#### Eliminación registros NA

In [14]:
```
# Eliminamos los valores NA del dataframe, creando una copia primero para
# aplicar la solución 2
df_not_na = df.copy()
df_not_na.dropna(inplace = True)

# Verificamos que no existan valores NA en el dataframe copia.
# Se eliminaron 42 filas de 30,000; lo cual equivale a un 0.14% del dataframe
# original
df_not_na.isnull().values.any()
```

Out[14]:
```
False
```

#### Sustitución por Media

In [15]:
```
# Se aplica una función para sustituir la Media
df_sust_med = df.copy()
for column in df_sust_med.columns:
    avg = df_sust_med[column].mean()
    df_sust_med[column].fillna(value = avg, inplace = True)

# Y verificamos que la operación fue exitosa.

df_sust_med.isna().values.any()
```

Out[15]:
```
False
```

# 4. Calcula la estadística descriptiva con describe() y explica las medidas de tendencia central y dispersión

In [16]:
```
# Nos quedamos con la primera solución con pérdida de pocos datos.
df_not_na.describe
```

Out[16]:
```
<bound method NDFrame.describe of            X1   X2   X3   X4    X5   X6   X
7   X8   X9  X10  ...      X15  \
1       20000  2.0  2.0  1.0  24.0  2.0  2.0 -1.0 -1.0 -2.0  ...      0.0
2      120000  2.0  2.0  2.0  26.0 -1.0  2.0  0.0  0.0  0.0  ...   3272.0
3       90000  2.0  2.0  2.0  34.0  0.0  0.0  0.0  0.0  0.0  ...  14331.0
4       50000  2.0  2.0  1.0  37.0  0.0  0.0  0.0  0.0  0.0  ...  28314.0
5       50000  1.0  2.0  1.0  57.0 -1.0  0.0 -1.0  0.0  0.0  ...  20940.0
...       ...  ...  ...  ...   ...  ...  ...  ...  ...  ...  ...      ...
29996  220000  1.0  3.0  1.0  39.0  0.0  0.0  0.0  0.0  0.0  ...  88004.0
29997  150000  1.0  3.0  2.0  43.0 -1.0 -1.0 -1.0 -1.0  0.0  ...   8979.0
29998   30000  1.0  2.0  2.0  37.0  4.0  3.0  2.0 -1.0  0.0  ...  20878.0
29999   80000  1.0  3.0  1.0  41.0  1.0 -1.0  0.0  0.0  0.0  ...  52774.0
30000   50000  1.0  2.0  1.0  46.0  0.0  0.0  0.0  0.0  0.0  ...  36535.0

           X16      X17      X18      X19      X20     X21      X22     X23
\
1          0.0      0.0      0.0    689.0      0.0     0.0      0.0     0.0
2       3455.0   3261.0      0.0   1000.0   1000.0  1000.0      0.0  2000.0
3      14948.0  15549.0   1518.0   1500.0   1000.0  1000.0   1000.0  5000.0
4      28959.0  29547.0   2000.0   2019.0   1200.0  1100.0   1069.0  1000.0
5      19146.0  19131.0   2000.0  36681.0  10000.0  9000.0    689.0   679.0
...        ...      ...      ...      ...      ...     ...      ...     ...
29996  31237.0  15980.0   8500.0  20000.0   5003.0  3047.0   5000.0  1000.0
29997   5190.0      0.0   1837.0   3526.0   8998.0   129.0      0.0     0.0
29998  20582.0  19357.0      0.0      0.0  22000.0  4200.0   2000.0  3100.0
29999  11855.0  48944.0  85900.0   3409.0   1178.0  1926.0  52964.0  1804.0
30000  32428.0  15313.0   2078.0   1800.0   1430.0  1000.0   1000.0  1000.0

         Y
1      1.0
2      1.0
3      0.0
4      0.0
5      0.0
...    ...
29996  0.0
29997  0.0
29998  1.0
29999  1.0
30000  1.0

[29958 rows x 24 columns]>
```

In [17]:
```
# Ahora, comenzamos a definir los usos del describe(). Inicialmente
# aplicaremos la descripción de todos los datos.
df_not_na.describe(include='all')
```

Out[17]:

| | X1 | X2 | X3 | X4 | X5 | X6 |
|---|---|---|---|---|---|---|
| **count** | 29958.000000 | 29958.000000 | 29958.000000 | 29958.000000 | 29958.000000 | 29958.000000 |
| **mean** | 167555.900928 | 1.604012 | 1.853094 | 1.551739 | 35.483443 | -0.017124 |
| **std** | 129737.299088 | 0.489070 | 0.790471 | 0.521952 | 9.214319 | 1.123989 |
| **min** | 10000.000000 | 1.000000 | 0.000000 | 0.000000 | 21.000000 | -2.000000 |
| **25%** | 50000.000000 | 1.000000 | 1.000000 | 1.000000 | 28.000000 | -1.000000 |
| **50%** | 140000.000000 | 2.000000 | 2.000000 | 2.000000 | 34.000000 | 0.000000 |
| **75%** | 240000.000000 | 2.000000 | 2.000000 | 2.000000 | 41.000000 | 0.000000 |
| **max** | 1000000.000000 | 2.000000 | 6.000000 | 3.000000 | 79.000000 | 8.000000 |

8 rows × 24 columns

Las medidas de tendencia central son:

**Conteo:** Muestra el número total de registros contados.

**Media:** Es el promedio de los valores.

**STD (Desviación estandar:** Es la dispersión de los datos.

**Mínimo:** Muestra el valor mínimo de la serie de datos.

**25%, 50%, 75%:** Los valores que wobrepasan el % definido de la distribución.

**Máximo:** Valor máximo de la distribución.

# 5. Realiza el conteo de las variables categóricas

Para realizar esta distinción, debemos revisar la descripción de las variables. A continuación se describen los datos:

- **X1**: credit given.
- **X2**: Gender (1 = male; 2 = female).
- **X3**: Education (1 = graduate school; 2 = university; 3 = high school; 4 = others).
- **X4**: Marital status (1 = married; 2 = single; 3 = others).
- **X5**: Age (year).
- **X6 - X11**: History of past payment. We tracked the past monthly payment records (from April to September, 2005) as follows: X6 = the repayment status in September, 2005; X7 = the repayment status in August, 2005; . . .; X11 = the repayment status in April, 2005. The measurement scale for the repayment status is:

```
  -1 = pay duly;
  1 = payment delay for one month;
  2 = payment delay for two months; . . .;
  8 = payment delay for eight months; 9 = payment delay for nine m
onths and above.
```

- **X12-X17**: Amount of bill statement (NT dollar). X12 = amount of bill statement in September, 2005; X13 = amount of bill statement in August, 2005; . . .; X17 = amount of bill statement in April, 2005.
- **X18-X23**: Amount of previous payment (NT dollar). X18 = amount paid in September, 2005; X19 = amount paid in August, 2005; . . .; X23 = amount paid in April, 2005.

Se identifican los siguientes nombres de columnas que serán sustituidos en el dataframe para

In [18]:
```
# Definimos los nombres de las columnas de acuerdo a la documentación
# proprocionada junto con el dataframe.
df_not_na.columns
df_not_na.rename(columns = {'X1' : 'credit_giv',
                           'X2' : 'gender',
                           'X3' : 'education',
                           'X4' : 'marital',
                           'X5' : 'age',
                           'X6' : 'repay_stat_sep',
                           'X7' : 'repay_stat_aug',
                           'X8' : 'repay_stat_jul',
                           'X9' : 'repay_stat_jun',
                           'X10' : 'repay_stat_may',
                           'X11' : 'repay_stat_apr',
                           'X12' : 'bill_stmnt_dll_sep',
                           'X13' : 'bill_stmnt_dll_aug',
                           'X14' : 'bill_stmnt_dll_jul',
                           'X15' : 'bill_stmnt_dll_jun',
                           'X16' : 'bill_stmnt_dll_may',
                           'X17' : 'bill_stmnt_dll_apr',
                           'X18' : 'prev_pay_sep',
                           'X19' : 'prev_pay_aug',
                           'X20' : 'prev_pay_jul',
                           'X21' : 'prev_pay_jun',
                           'X22' : 'prev_pay_may',
                           'X23' : 'prev_pay_apr',
                            'Y' : 'default_pay'}, inplace = True)

# Renombramos el dataframe en una nueva copia y ordenamos los datos para una
# lectura mas sencilla

df_names = df_not_na[['credit_giv',
                           'gender',
                           'education',
                           'marital',
                           'age',
                           'repay_stat_apr',
                           'repay_stat_may',
                           'repay_stat_jun',
                           'repay_stat_jul',
                           'repay_stat_aug',
                           'repay_stat_sep',
                           'bill_stmnt_dll_apr',
                           'bill_stmnt_dll_may',
                           'bill_stmnt_dll_jun',
                           'bill_stmnt_dll_jul',
                           'bill_stmnt_dll_aug',
                           'bill_stmnt_dll_sep',
                           'prev_pay_apr',
                           'prev_pay_may',
                           'prev_pay_jun',
                           'prev_pay_jul',
                           'prev_pay_aug',
                           'prev_pay_sep',
                           'default_pay'
```

```
]]
```

Out[18]: df_pagos

| | credit_giv | gender | education | marital | age | repay_stat_apr | repay_stat_may | repay_stat_ju |
|---|---|---|---|---|---|---|---|---|
| **1** | 20000 | 2.0 | 2.0 | 1.0 | 24.0 | -2.0 | -2.0 | -1 |
| **2** | 120000 | 2.0 | 2.0 | 2.0 | 26.0 | 2.0 | 0.0 | C |
| **3** | 90000 | 2.0 | 2.0 | 2.0 | 34.0 | 0.0 | 0.0 | C |
| **4** | 50000 | 2.0 | 2.0 | 1.0 | 37.0 | 0.0 | 0.0 | C |
| **5** | 50000 | 1.0 | 2.0 | 1.0 | 57.0 | 0.0 | 0.0 | C |
| **...** | ... | ... | ... | ... | ... | ... | ... | |
| **29996** | 220000 | 1.0 | 3.0 | 1.0 | 39.0 | 0.0 | 0.0 | C |
| **29997** | 150000 | 1.0 | 3.0 | 2.0 | 43.0 | 0.0 | 0.0 | -1 |
| **29998** | 30000 | 1.0 | 2.0 | 2.0 | 37.0 | 0.0 | 0.0 | -1 |
| **29999** | 80000 | 1.0 | 3.0 | 1.0 | 41.0 | -1.0 | 0.0 | C |
| **30000** | 50000 | 1.0 | 2.0 | 1.0 | 46.0 | 0.0 | 0.0 | C |

Como variables categóricas, se identifican las siguientes:

- gender
- education
- marital
- age
- repay_stat_apr
- repay_stat_may
- repay_stat_jun
- repay_stat_jul
- repay_stat_aug
- repay_stat_sep

Ya que para su correcta interpretación, se requiere de sustituír los valores numéricos con sus mapeos en sus catálogos respectivos. por ejemplo, el 1 y 2 para hombre y mujer respectivamente en gender. Estos valores indican categoría, mas no monto, como lo es en el caso de las variables eliminadas de este listado.

In [19]:
```
# Se aplica la visualización de los valores categóricos
df_names[['gender', 'education', 'marital', 'age', 'repay_stat_apr',
         'repay_stat_may', 'repay_stat_jun', 'repay_stat_jul',
         'repay_stat_aug', 'repay_stat_sep']]
```

Out[19]:

| | gender | education | marital | age | repay_stat_apr | repay_stat_may | repay_stat_jun | repay_s |
|---|---|---|---|---|---|---|---|---|
| **1** | 2.0 | 2.0 | 1.0 | 24.0 | -2.0 | -2.0 | -1.0 | |
| **2** | 2.0 | 2.0 | 2.0 | 26.0 | 2.0 | 0.0 | 0.0 | |
| **3** | 2.0 | 2.0 | 2.0 | 34.0 | 0.0 | 0.0 | 0.0 | |
| **4** | 2.0 | 2.0 | 1.0 | 37.0 | 0.0 | 0.0 | 0.0 | |
| **5** | 1.0 | 2.0 | 1.0 | 57.0 | 0.0 | 0.0 | 0.0 | |
| **...** | ... | ... | ... | ... | ... | ... | ... | |
| **29996** | 1.0 | 3.0 | 1.0 | 39.0 | 0.0 | 0.0 | 0.0 | |
| **29997** | 1.0 | 3.0 | 2.0 | 43.0 | 0.0 | 0.0 | -1.0 | |
| **29998** | 1.0 | 2.0 | 2.0 | 37.0 | 0.0 | 0.0 | -1.0 | |
| **29999** | 1.0 | 3.0 | 1.0 | 41.0 | -1.0 | 0.0 | 0.0 | |
| **30000** | 1.0 | 2.0 | 1.0 | 46.0 | 0.0 | 0.0 | 0.0 | |

29958 rows × 10 columns

# 6. Escala los datos, si consideras necesario

La escala de datos se usa cuando los datos de entrada se van a aplicar a un modelo. Para esto, solo vamos a aplicar el escalado a las variables no categóricas.

In [21]:
```
# Tomamos las librerías que usaremos
from sklearn.preprocessing import StandardScaler

# Estandarizamos los valores con un escalador
df_scaled = StandardScaler().fit_transform(df_names)

# Regresamos a un dataframe
df_scaled = pd.DataFrame(df_scaled)

df_scaled
```

Out[21]:

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | |
|---|---|---|---|---|---|---|---|---|---|
| **0** | -1.137363 | 0.809689 | 0.185849 | -1.057086 | -1.246282 | -1.486513 | -1.530700 | -0.666630 | -0.69 |
| **1** | -0.366561 | 0.809689 | 0.185849 | 0.858831 | -1.029224 | 1.993916 | 0.235635 | 0.189241 | 0.13 |
| **2** | -0.597802 | 0.809689 | 0.185849 | 0.858831 | -0.160996 | 0.253701 | 0.235635 | 0.189241 | 0.13 |
| **3** | -0.906122 | 0.809689 | 0.185849 | -1.057086 | 0.164590 | 0.253701 | 0.235635 | 0.189241 | 0.13 |
| **4** | -0.906122 | -1.235043 | 0.185849 | -1.057086 | 2.335161 | 0.253701 | 0.235635 | 0.189241 | -0.69 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | |
| **29953** | 0.404240 | -1.235043 | 1.450938 | -1.057086 | 0.381647 | 0.253701 | 0.235635 | 0.189241 | 0.13 |
| **29954** | -0.135321 | -1.235043 | 1.450938 | 0.858831 | 0.815761 | 0.253701 | 0.235635 | -0.666630 | -0.69 |
| **29955** | -1.060283 | -1.235043 | 0.185849 | 0.858831 | 0.164590 | 0.253701 | 0.235635 | -0.666630 | 1.8 |
| **29956** | -0.674882 | -1.235043 | 1.450938 | -1.057086 | 0.598704 | -0.616406 | 0.235635 | 0.189241 | 0.13 |
| **29957** | -0.906122 | -1.235043 | 0.185849 | -1.057086 | 1.141347 | 0.253701 | 0.235635 | 0.189241 | 0.13 |

29958 rows × 24 columns

# 7. Reduce las dimensiones con PCA, si consideras necesario

In [38]:
```
# Tomamos las librerías que usemos
from sklearn.decomposition import PCA

# Se aplican 10 componentes a la función de PCA
pca_eval = PCA(n_components = 10)
componentes_principales = pca_eval.fit_transform(df_scaled)

print('PCA: ', componentes_principales.shape)
print('PCA explained variance ratio: ', pca_eval.explained_variance_ratio_)
print('PCA singular values : ', pca_eval.singular_values_)
```

```
PCA:  (29958, 10)
PCA explained variance ratio:  [0.27301007 0.17504995 0.06470639 0.06144887
0.04336966 0.04058541
 0.0381546  0.03780229 0.03693108 0.03630422]
PCA singular values :  [443.04859217 354.7668489  215.69278946 210.19335175 1
76.58550786
 170.82326144 165.62866101 164.862194   162.95136692 161.56250512]
```

Se busca encontrtar la variabilidad de los datos atraves de los componentes principales. Se busca reducir dimensiones y hacer una transformación a los componentes principales (PC).

### 7.1. Indica la varianza de los datos explicada por cada componente seleccionado. Para actividades de exploración de los datos la varianza > 70%

### 7.2 Indica la importancia de las variables en cada componente

### 8. Elabora los histogramas de los atributos para visualizar su distribución

### 9. Realiza la visualización de los datos usando por lo menos 3 gráficos que consideres adecuados: plot, scatter, jointplot, boxplot, areaplot, pie chart, pairplot, bar chart, etc.

### 10. Interpreta y explica cada uno de los gráficos indicando cuál es la información más relevante que podría ayudar en el proceso de toma de decisiones.

**Bibliografía:**

Shanthababu, P. (2021). Effective Data Visualization Techniques in Data Science Using Python. Analytics Vidhya. Recuperado de: [https://www.analyticsvidhya.com/blog/2021/08/effective-data-visualization-techniques-in-data-science-using-python/Enlaces (https://www.analyticsvidhya.com/blog/2021/08/effective-data-visualization-techniques-in-data-science-using-python/Enlaces)] a un sitio externo.

Rajbangshi, A. (2020). Importance of Data Storytelling in Data Science. Artificial Intelligence in Plain English. Recuperado de: [https://ai.plainenglish.io/importance-of-data-storytelling-in-data-science-494f49273027Enlaces (https://ai.plainenglish.io/importance-of-data-storytelling-in-data-science-494f49273027Enlaces)] a un sitio externo.

Das, A. (2020). Data Visualization in Data Science. Towards Data Science. Recuperado de: [https://towardsdatascience.com/data-visualization-in-data-science-5681cbdde5bf (https://towardsdatascience.com/data-visualization-in-data-science-5681cbdde5bf)]