Lichman, M. (2013). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science.

IMPORTAR LIBRERÍAS Y DATASETS

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

```
In [ ]: # Montar el disco usando los siguientes comandos:

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

```
Mounted at /content/drive
```

```
In [ ]: # Enlace al archivo csv que contiene el conjunto de datos
        creditcard_df = pd.read_csv('/content/drive/MyDrive/01_Oriol/01_Machine Learning/07_Masterclass-IA-Moderna/04_
```

```
In [ ]: creditcard_df
```

Out[ ]:

	ID	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_0	PAY_2	PAY_3	PAY_4	PAY_5	PAY_6	BILL_AMT1	BILL_AMT2	BILL_AMT
0	1	20000.0	2	2	1	24	2	2	-1	-1	-2	-2	3913.0	3102.0	689
1	2	120000.0	2	2	2	26	-1	2	0	0	0	2	2682.0	1725.0	2682
2	3	90000.0	2	2	2	34	0	0	0	0	0	0	29239.0	14027.0	13559
3	4	50000.0	2	2	1	37	0	0	0	0	0	0	46990.0	48233.0	49291
4	5	50000.0	1	2	1	57	-1	0	-1	0	0	0	8617.0	5670.0	35835
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
29995	29996	220000.0	1	3	1	39	0	0	0	0	0	0	188948.0	192815.0	208365
29996	29997	150000.0	1	3	2	43	-1	-1	-1	-1	0	0	1683.0	1828.0	3502
29997	29998	30000.0	1	2	2	37	4	3	2	-1	0	0	3565.0	3356.0	2758
29998	29999	80000.0	1	3	1	41	1	-1	0	0	0	-1	-1645.0	78379.0	76304
29999	30000	50000.0	1	2	1	46	0	0	0	0	0	0	47929.0	48905.0	49764

30000 rows × 25 columns

```
In [ ]: creditcard_df.info()
        # 24 características con un total de 30000 puntos de datos
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30000 entries, 0 to 29999
Data columns (total 25 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   ID                          30000 non-null  int64
 1   LIMIT_BAL                   30000 non-null  float64
 2   SEX                         30000 non-null  int64
 3   EDUCATION                   30000 non-null  int64
 4   MARRIAGE                    30000 non-null  int64
 5   AGE                         30000 non-null  int64
 6   PAY_0                       30000 non-null  int64
 7   PAY_2                       30000 non-null  int64
 8   PAY_3                       30000 non-null  int64
 9   PAY_4                       30000 non-null  int64
 10  PAY_5                       30000 non-null  int64
 11  PAY_6                       30000 non-null  int64
 12  BILL_AMT1                   30000 non-null  float64
 13  BILL_AMT2                   30000 non-null  float64
 14  BILL_AMT3                   30000 non-null  float64
 15  BILL_AMT4                   30000 non-null  float64
 16  BILL_AMT5                   30000 non-null  float64
 17  BILL_AMT6                   30000 non-null  float64
 18  PAY_AMT1                    30000 non-null  float64
 19  PAY_AMT2                    30000 non-null  float64
 20  PAY_AMT3                    30000 non-null  float64
 21  PAY_AMT4                    30000 non-null  float64
 22  PAY_AMT5                    30000 non-null  float64
 23  PAY_AMT6                    30000 non-null  float64
 24  default.payment.next.month  30000 non-null  int64
dtypes: float64(13), int64(12)
memory usage: 5.7 MB
```

```
In [ ]: creditcard_df.describe()
        # la media de LIMIT_BAL = 1500, min = 1 y max = 30000
        # la media de EDAD = 25 años, mínimo = 21 y máximo = 79
        # El promedio de PAY_AMT es de alrededor de 5000
```

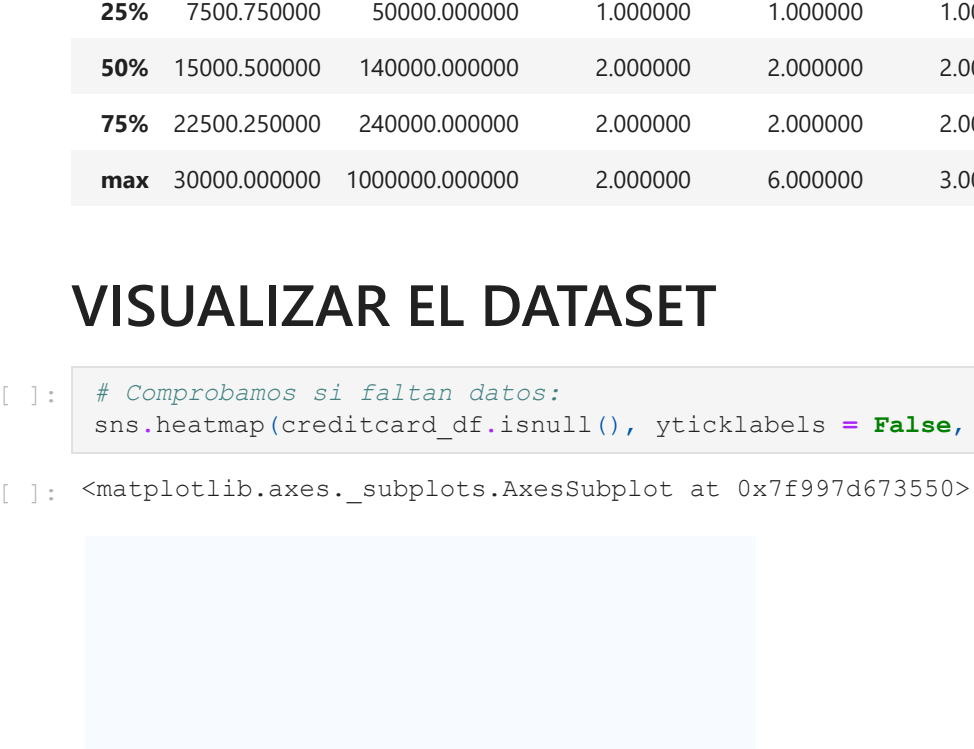
Out[ ]:

	ID	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_0	PAY_2	PAY_3
count	30000.000000	30000.000000	30000.000000	30000.000000	30000.000000	30000.000000	30000.000000	30000.000000	30000.000000
mean	15000.500000	167484.322667	1.603733	1.853133	1.551867	35.485500	-0.016700	-0.133767	-0.166200
std	8660.398374	129747.661567	0.489129	0.790349	0.521970	9.217904	1.123802	1.197186	1.196868
min	1.000000	10000.000000	1.000000	0.000000	0.000000	21.000000	-2.000000	-2.000000	-2.000000
25%	7500.750000	50000.000000	1.000000	1.000000	1.000000	28.000000	-1.000000	-1.000000	-1.000000
50%	15000.500000	140000.000000	2.000000	2.000000	2.000000	34.000000	0.000000	0.000000	0.000000
75%	22500.250000	240000.000000	2.000000	2.000000	2.000000	41.000000	0.000000	0.000000	0.000000
max	30000.000000	1000000.000000	2.000000	6.000000	3.000000	79.000000	8.000000	8.000000	8.000000

VISUALIZAR EL DATASET

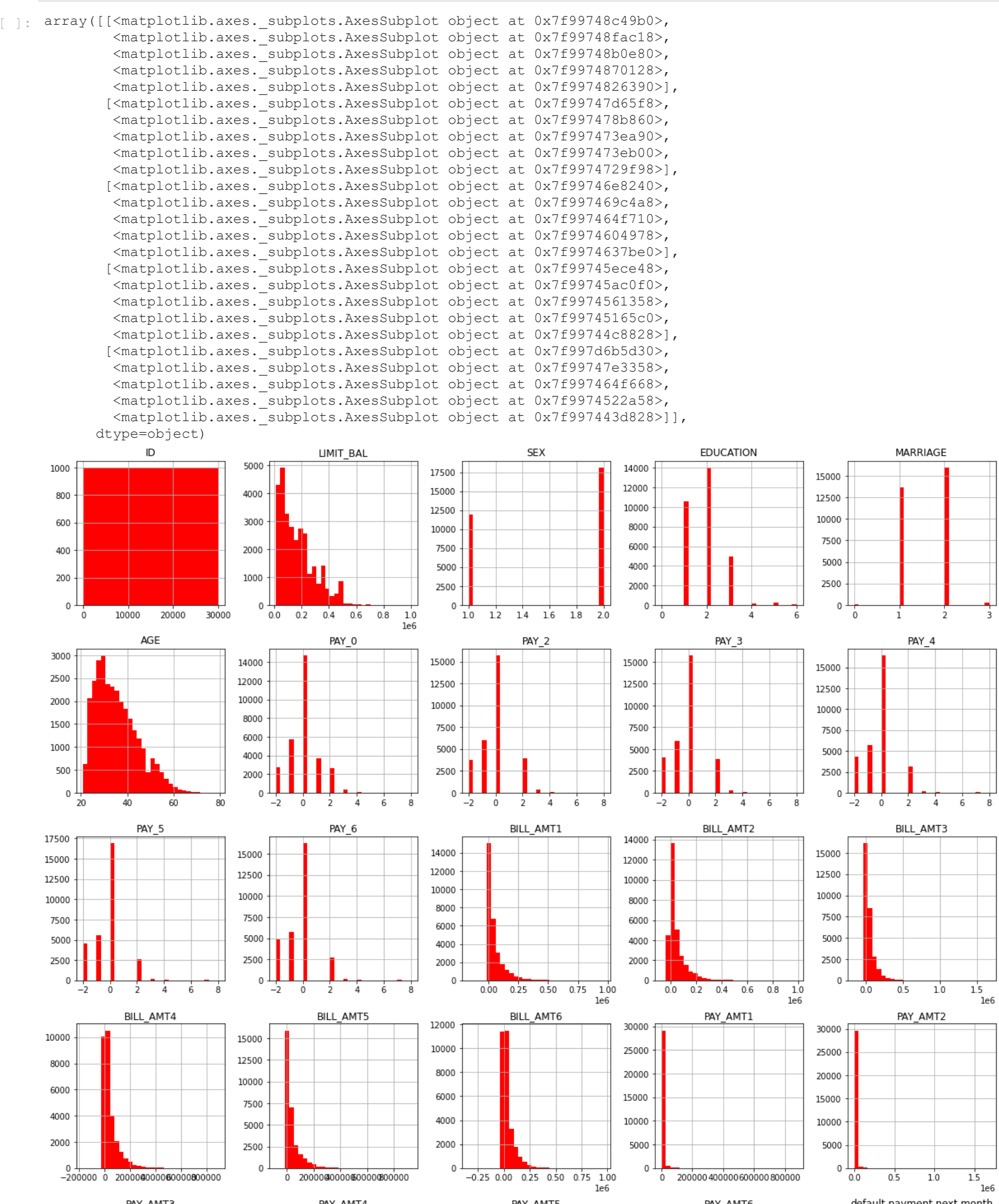
```
In [ ]: # Comprobamos si faltan datos:
        sns.heatmap(creditcard_df.isnull(), yticklabels = False, cbar = False, cmap="Blues")
```

```
Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7f997d673550>
```



```
In [ ]: creditcard_df.hist(bins = 30, figsize = (20,20), color = 'r')
```

```
Out[ ]: array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7f99748c49b0>,
                <matplotlib.axes._subplots.AxesSubplot object at 0x7f99748fac18>,
                <matplotlib.axes._subplots.AxesSubplot object at 0x7f99748b0e80>,
                <matplotlib.axes._subplots.AxesSubplot object at 0x7f9974870128>,
                <matplotlib.axes._subplots.AxesSubplot object at 0x7f9974826390>],
               [<matplotlib.axes._subplots.AxesSubplot object at 0x7f99747d65f8>,
                <matplotlib.axes._subplots.AxesSubplot object at 0x7f997478b860>,
                <matplotlib.axes._subplots.AxesSubplot object at 0x7f997473ea90>,
                <matplotlib.axes._subplots.AxesSubplot object at 0x7f997473eb00>,
                <matplotlib.axes._subplots.AxesSubplot object at 0x7f9974729f98>],
               [<matplotlib.axes._subplots.AxesSubplot object at 0x7f99746e8240>,
                <matplotlib.axes._subplots.AxesSubplot object at 0x7f997469c4a8>,
                <matplotlib.axes._subplots.AxesSubplot object at 0x7f997464f710>,
                <matplotlib.axes._subplots.AxesSubplot object at 0x7f9974604978>,
                <matplotlib.axes._subplots.AxesSubplot object at 0x7f9974637be0>],
               [<matplotlib.axes._subplots.AxesSubplot object at 0x7f99745ece48>,
                <matplotlib.axes._subplots.AxesSubplot object at 0x7f99745ac0f0>,
                <matplotlib.axes._subplots.AxesSubplot object at 0x7f9974561358>,
                <matplotlib.axes._subplots.AxesSubplot object at 0x7f99745165c0>,
                <matplotlib.axes._subplots.AxesSubplot object at 0x7f99744c8828>],
               [<matplotlib.axes._subplots.AxesSubplot object at 0x7f997d6b5d30>,
                <matplotlib.axes._subplots.AxesSubplot object at 0x7f99747e3358>,
                <matplotlib.axes._subplots.AxesSubplot object at 0x7f997464f668>,
                <matplotlib.axes._subplots.AxesSubplot object at 0x7f9974522a58>,
                <matplotlib.axes._subplots.AxesSubplot object at 0x7f997443d828>]],
              dtype=object)
```



```
In [ ]: # Eliminación de la columna ID
        creditcard_df.drop(['ID'], axis=1, inplace=True)
```

```
In [ ]: creditcard_df
```

Out[ ]:

	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_0	PAY_2	PAY_3	PAY_4	PAY_5	PAY_6	BILL_AMT1	BILL_AMT2	BILL_AMT3
0	20000.0	2	2	1	24	2	2	-1	-1	-2	-2	3913.0	3102.0	689.0
1	120000.0	2	2	2	26	-1	2	0	0	0	2	2682.0	1725.0	2682.0
2	90000.0	2	2	2	34	0	0	0	0	0	0	29239.0	14027.0	13559.0
3	50000.0	2	2	1	37	0	0	0	0	0	0	46990.0	48233.0	49291.0
4	50000.0	1	2	1	57	-1	0	-1	0	0	0	8617.0	5670.0	35835.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
29995	220000.0	1	3	1	39	0	0	0	0	0	0	188948.0	192815.0	208365.0
29996	150000.0	1	3	2	43	-1	-1	-1	-1	0	0	1683.0	1828.0	3502.0
29997	30000.0	1	2	2	37	4	3	2	-1	0	0	3565.0	3356.0	2758.0
29998	80000.0	1	3	1	41	1	-1	0	0	0	-1	-1645.0	78379.0	76304.0
29999	50000.0	1	2	1	46	0	0	0	0	0	0	47929.0	48905.0	49764.0

30000 rows × 24 columns

```
In [ ]: # Cuántos clientes podrían incumplir con el pago de la tarjeta de crédito!
        cc_default_df   = creditcard_df[creditcard_df['default.payment.next.month'] == 1]
        cc_nodefault_df = creditcard_df[creditcard_df['default.payment.next.month'] == 0]
```

```
In [ ]: # Número de clientes que se quedaron y se fueron
        # Estamos ante un conjunto de datos desbalanceado

        print("Total =", len(creditcard_df))

        print("Número de clientes que incumplieron sus pagos con tarjeta de crédito =", len(cc_default_df))
        print("Porcentaje de clientes que incumplieron sus pagos con tarjeta de crédito =", 1.*len(cc_default_df)/len(c

        print("Número de clientes que cumplieron sus pagos con tarjeta de crédito (pagan la deuda) =", len(cc_nodefault
        print("Porcentaje de clientes que cumplieron sus pagos con tarjeta de crédito (pagan la deuda) =", 1.*len(cc_no
```

```
Total = 30000
Número de clientes que incumplieron sus pagos con tarjeta de crédito = 6636
Porcentaje de clientes que incumplieron sus pagos con tarjeta de crédito = 22.12 %
Número de clientes que cumplieron sus pagos con tarjeta de crédito (pagan la deuda) = 23364
Porcentaje de clientes que cumplieron sus pagos con tarjeta de crédito (pagan la deuda) = 77.88000000000001 %
```

```
In [ ]: # Se compara la media y el error estándar de los clientes que cumplen y los que no
        cc_default_df.describe()
```

Out[ ]:

	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_0	PAY_2	PAY_3	PAY_4	PAY_5
count	6636.000000	6636.000000	6636.000000	6636.000000	6636.000000	6636.000000	6636.000000	6636.000000	6636.000000	6636.000000
mean	130109.656420	1.567058	1.894665	1.528029	35.725738	0.668174	0.458258	0.362116	0.254521	0.167872
std	115378.540571	0.495520	0.728096	0.525433	9.693438	1.383252	1.502243	1.499401	1.508535	1.482833
min	10000.000000	1.000000	1.000000	0.000000	21.000000	-2.000000	-2.000000	-2.000000	-2.000000	-2.000000
25%	50000.000000	1.000000	1.000000	1.000000	28.000000	0.000000	0.000000	-1.000000	-1.000000	-1.000000
50%	90000.000000	2.000000	2.000000	2.000000	34.000000	1.000000	0.000000	0.000000	0.000000	0.000000
75%	200000.000000	2.000000	2.000000	2.000000	42.000000	2.000000	2.000000	2.000000	2.000000	0.000000
max	740000.000000	2.000000	6.000000	3.000000	75.000000	8.000000	7.000000	8.000000	8.000000	8.000000

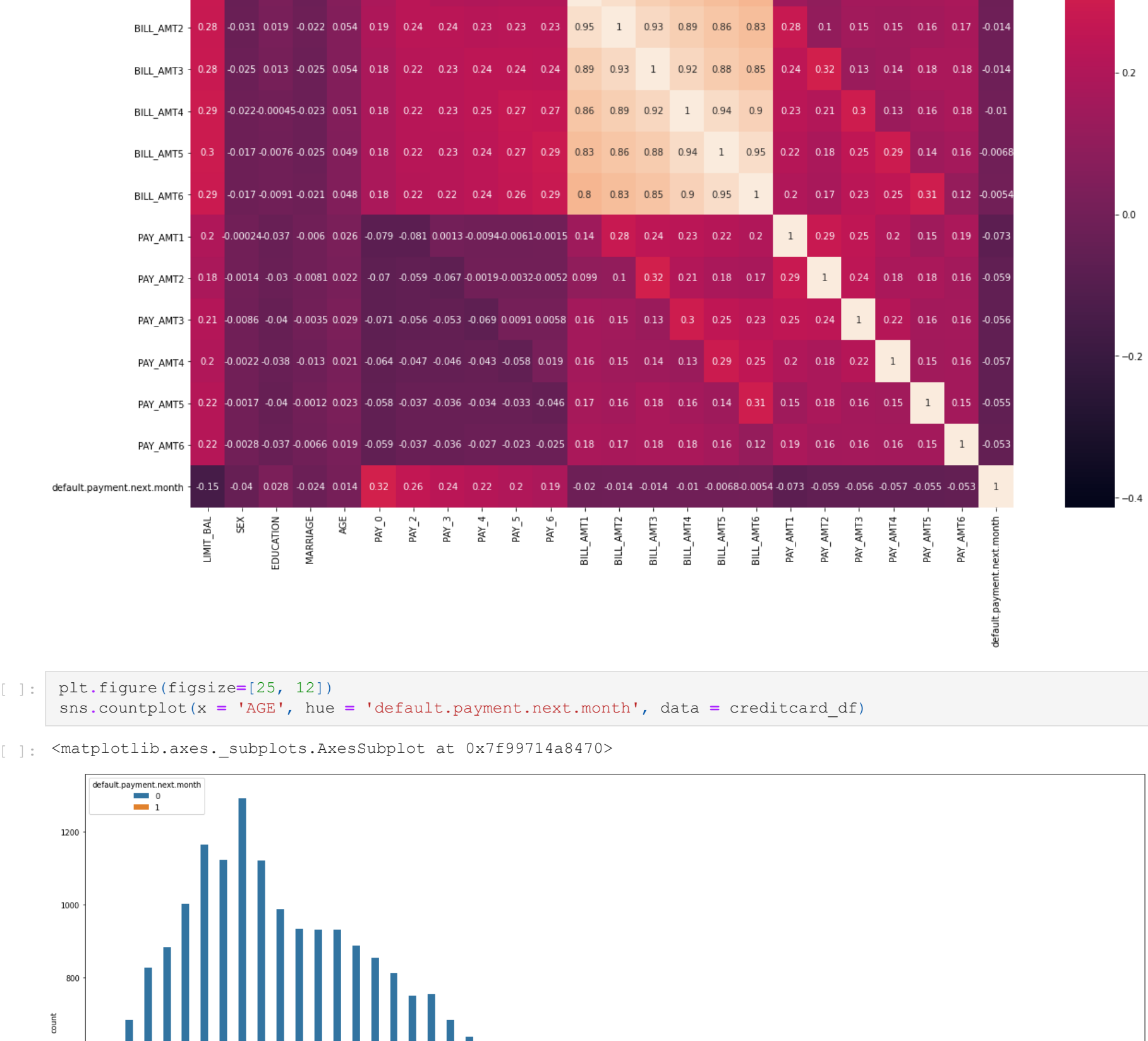
```
In [ ]: # Se compara la media y el error estándar de los clientes que cumplen y los que no
        cc_nodefault_df.describe()
```

Out[ ]:

	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_0	PAY_2	PAY_3	PAY_4
count	23364.000000	23364.000000	23364.000000	23364.000000	23364.000000	23364.000000	23364.000000	23364.000000	23364.000000
mean	178099.726074	1.614150	1.841337	1.558637	35.417266	-0.211222	-0.301917	-0.316256	-0.355633
std	131628.359660	0.486806	0.806780	0.520794	9.077355	0.952464	1.035191	1.048378	1.013162
min	10000.000000	1.000000	0.000000	0.000000	21.000000	-2.000000	-2.000000	-2.000000	-2.000000
25%	70000.000000	1.000000	1.000000	1.000000	28.000000	-1.000000	-1.000000	-1.000000	-1.000000
50%	150000.000000	2.000000	2.000000	2.000000	34.000000	0.000000	0.000000	0.000000	0.000000
75%	250000.000000	2.000000	2.000000	2.000000	41.000000	0.000000	0.000000	0.000000	0.000000
max	1000000.000000	2.000000	6.000000	3.000000	79.000000	8.000000	8.000000	8.000000	8.000000

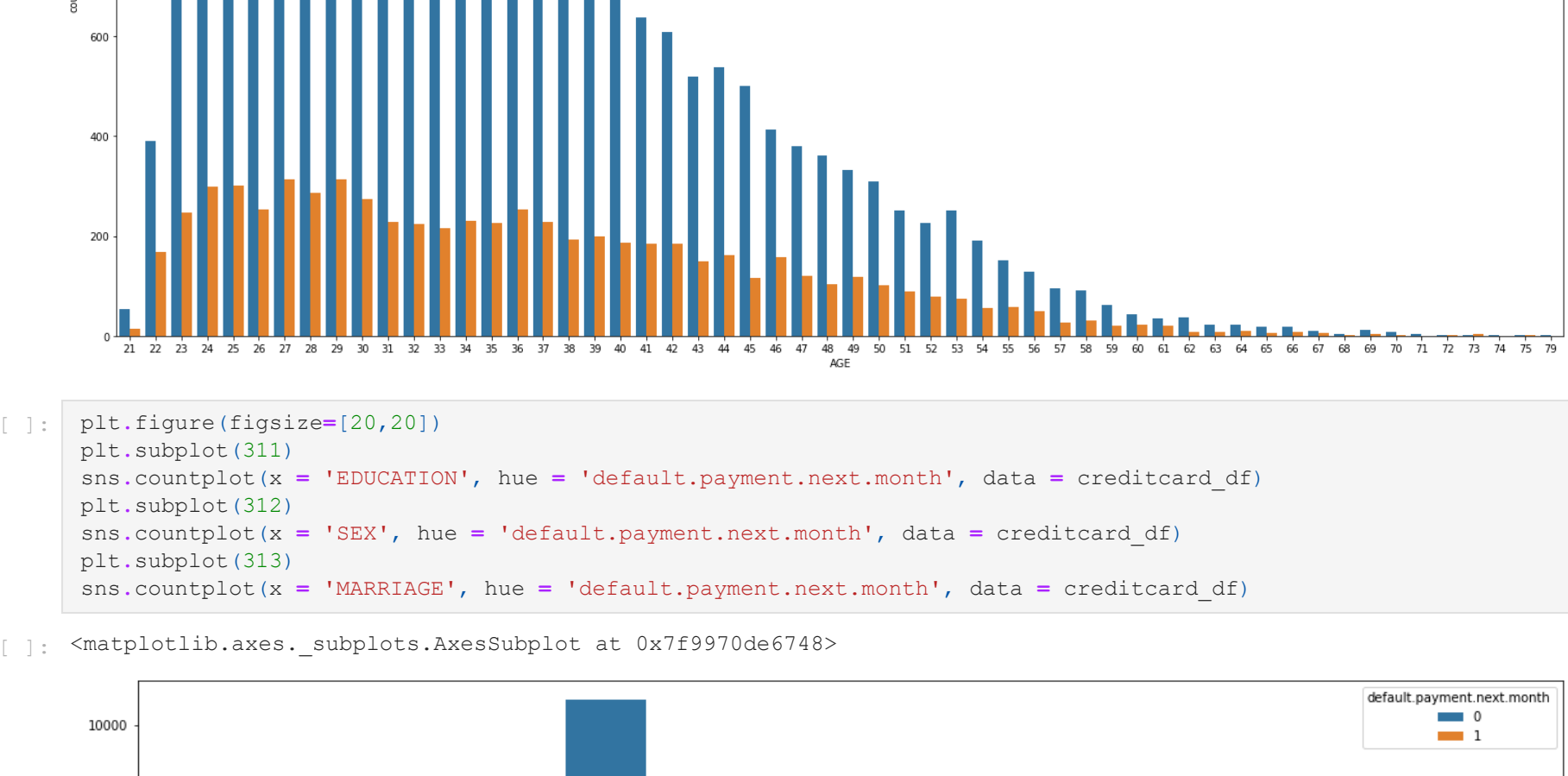
```
In [ ]: correlations = creditcard_df.corr()
        f, ax = plt.subplots(figsize = (20, 20))
        sns.heatmap(correlations, annot = True)
```

```
Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7f997381c8d0>
```



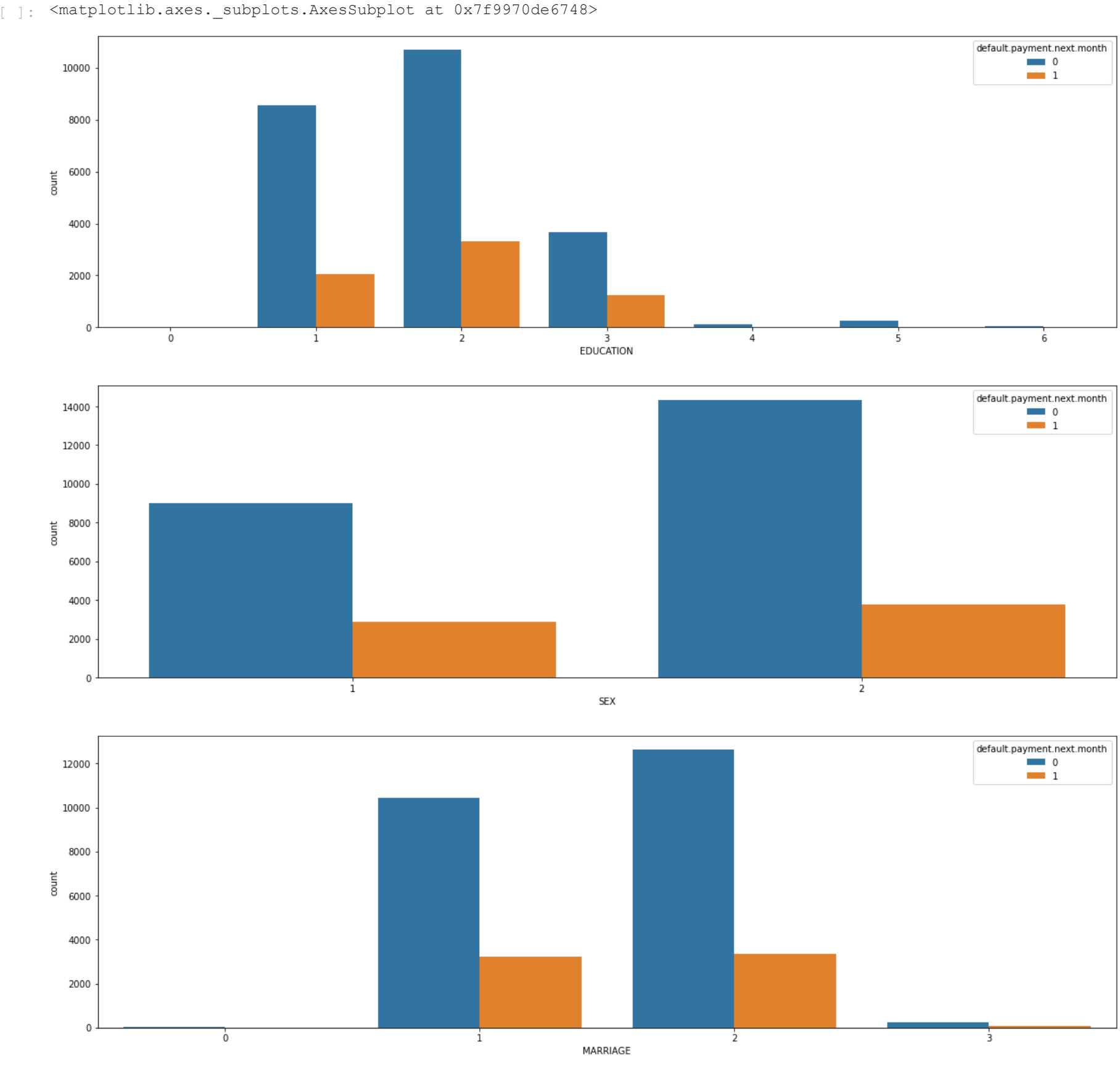
```
In [ ]: plt.figure(figsize=[25, 12])
        sns.countplot(x = 'AGE', hue = 'default.payment.next.month', data = creditcard_df)
```

```
Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7f99714a8470>
```



```
In [ ]: plt.figure(figsize=[20,20])
        plt.subplot(311)
        sns.countplot(x = 'EDUCATION', hue = 'default.payment.next.month', data = creditcard_df)
        plt.subplot(312)
        sns.countplot(x = 'SEX', hue = 'default.payment.next.month', data = creditcard_df)
        plt.subplot(313)
        sns.countplot(x = 'MARRIAGE', hue = 'default.payment.next.month', data = creditcard_df)
```

```
Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7f9970de6748>
```



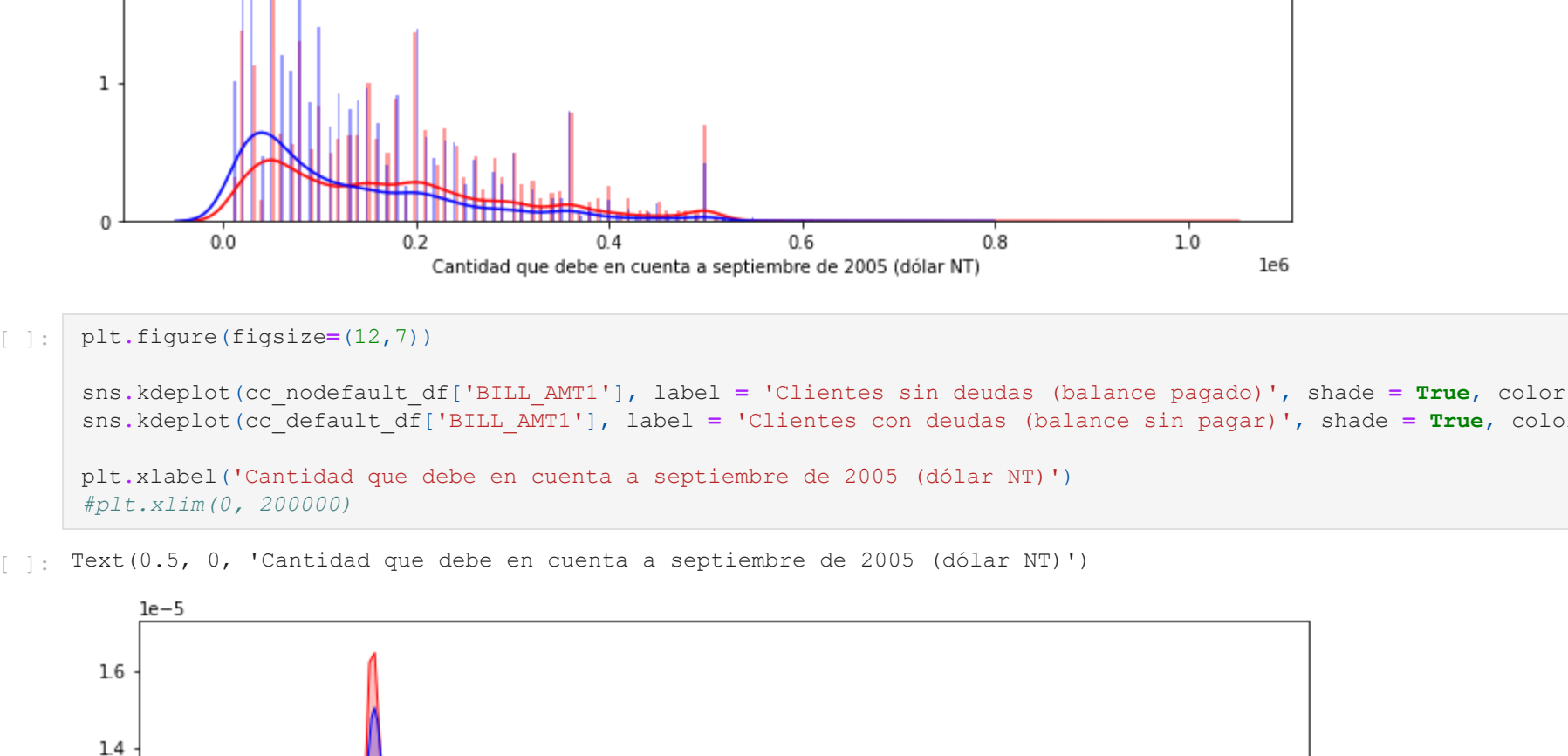
```
In [ ]: # KDE (Kernel Density Estimate) se utiliza para visualizar la densidad de probabilidad de una variable continu

        plt.figure(figsize=(12,7))

        sns.distplot(cc_nodefault_df['LIMIT_BAL'], bins = 250, color = 'r')
        sns.distplot(cc_default_df['LIMIT_BAL'], bins = 250, color = 'b')

        plt.xlabel('Cantidad que debe en cuenta a septiembre de 2005 (dólar NT)')
        #plt.xlim(0, 200000)
```

```
/usr/local/lib/python3.6/dist-packages/seaborn/distributions.py:2557: FutureWarning: `distplot` is a deprecated
function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-leve
l function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
/usr/local/lib/python3.6/dist-packages/seaborn/distributions.py:2557: FutureWarning: `distplot` is a deprecated
function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-leve
l function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

```
Out[ ]: Text(0.5, 0, 'Cantidad que debe en cuenta a septiembre de 2005 (dólar NT)')
```

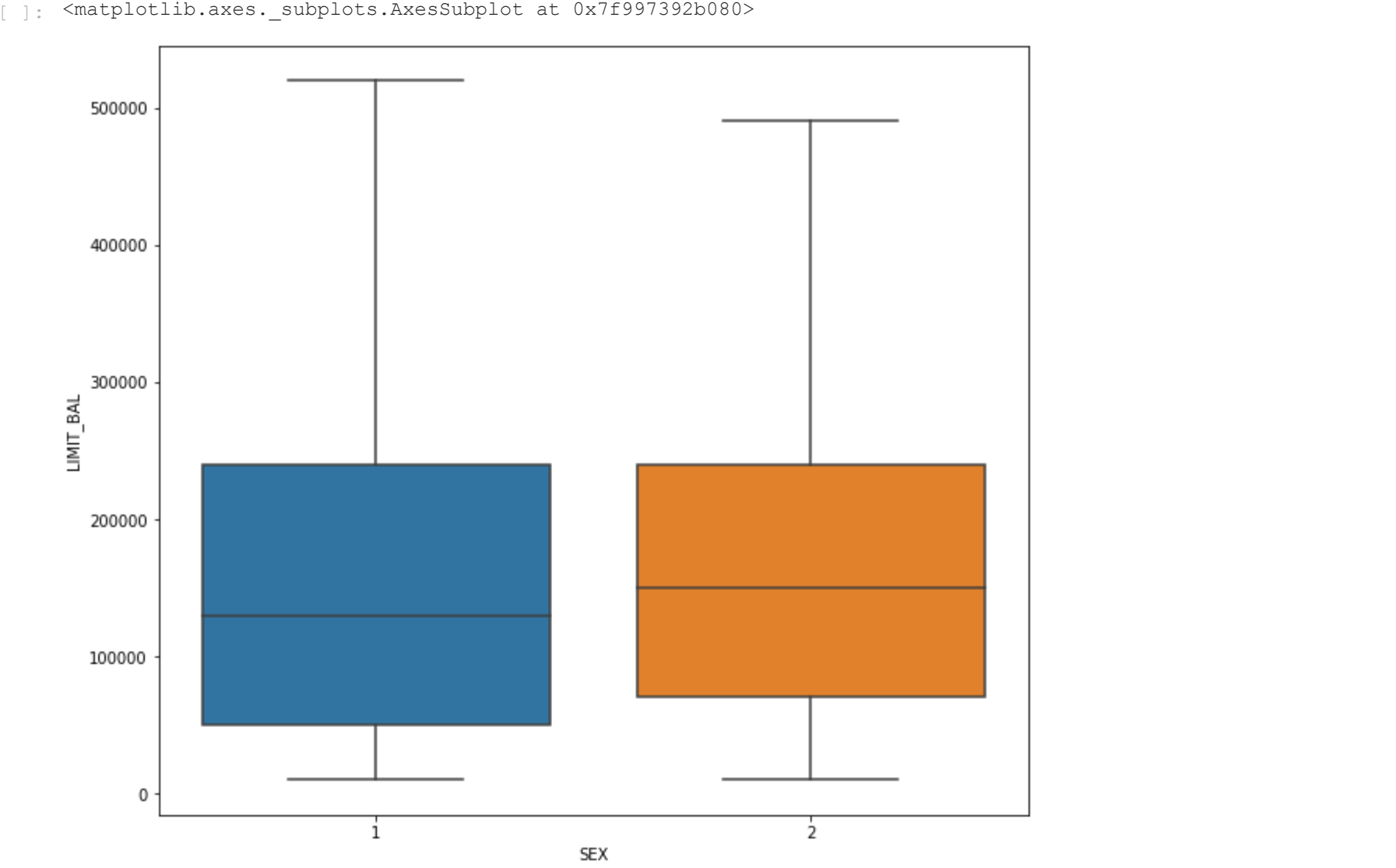
```
In [ ]: plt.figure(figsize=(12,7))

        sns.kdeplot(cc_nodefault_df['BILL_AMT1'], label = 'Clientes sin deudas (balance pagado)', shade = True, color =
        sns.kdeplot(cc_default_df['BILL_AMT1'], label = 'Clientes con deudas (balance sin pagar)', shade = True, color

        plt.xlabel('Cantidad que debe en cuenta a septiembre de 2005 (dólar NT)')
        #plt.xlim(0, 200000)
```

```
Out[ ]: Text(0.5, 0, 'Cantidad que debe en cuenta a septiembre de 2005 (dólar NT)')
```



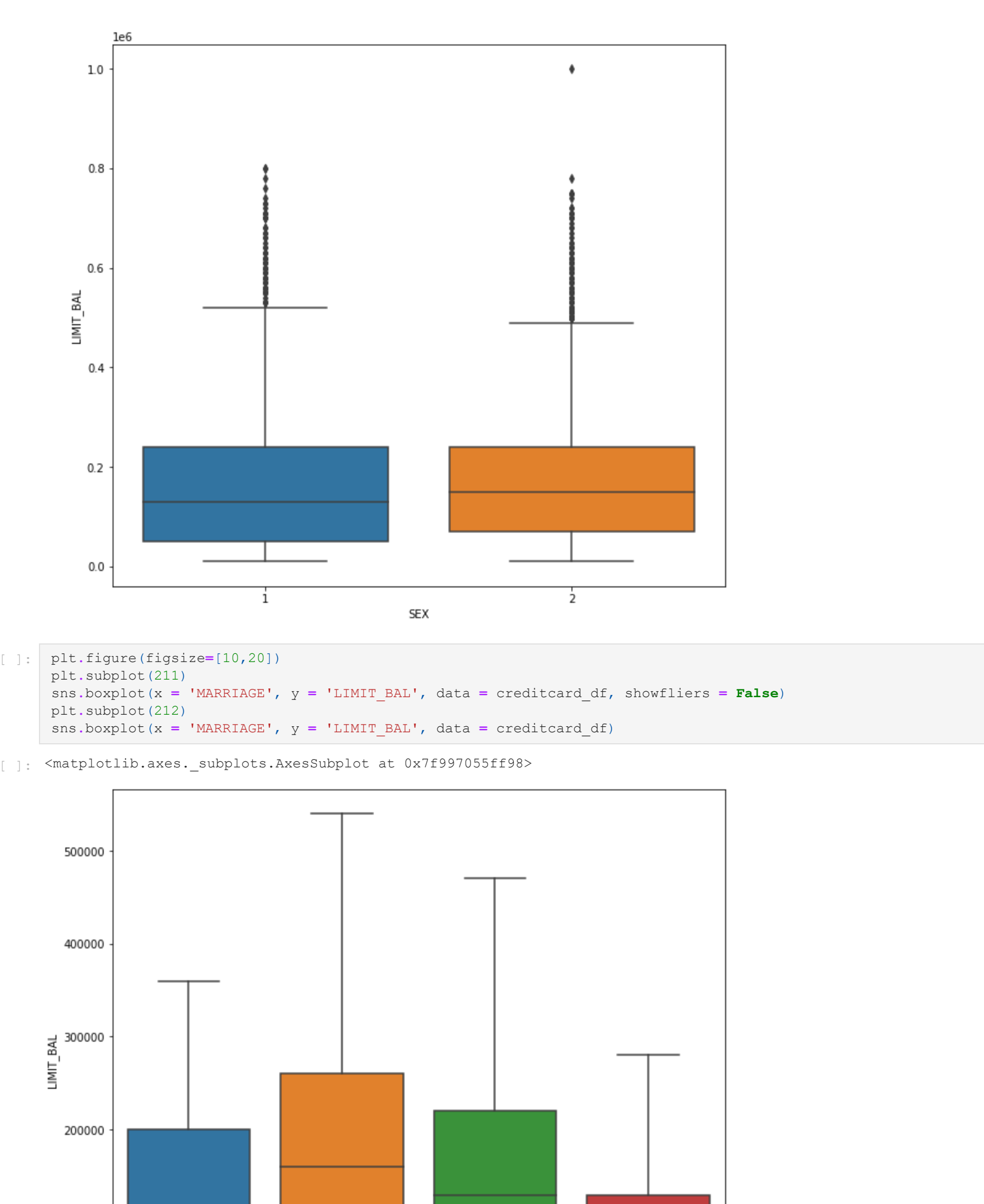
```
In [ ]: plt.figure(figsize=(12,7))

        sns.kdeplot(cc_nodefault_df['PAY_AMT1'], label = 'Clientes sin deudas (balance pagado)', shade = True, color =
        sns.kdeplot(cc_default_df['PAY_AMT1'], label = 'Clientes con deudas (balance sin pagar)', shade = True, color

        plt.xlabel('PAY_AMT1: Cantidad del pago anterior en septiembre de 2005 (dólar NT)')
        plt.xlim(0, 200000)
```

```
Out[ ]: (0.0, 200000.0)
```



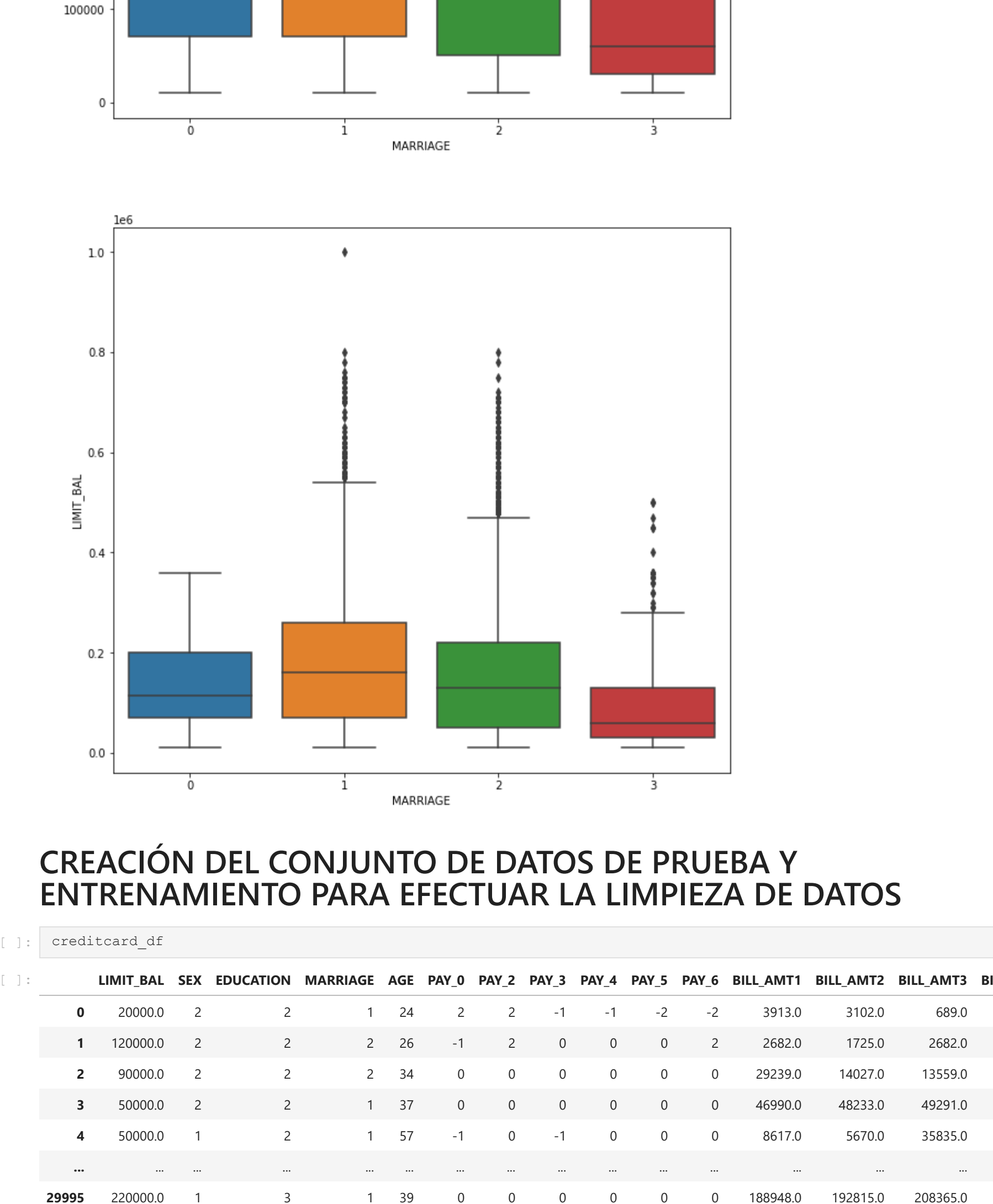
```
In [ ]: # Comprobamos si el género del individuo tiene efecto en el límite del balance

        plt.figure(figsize=[10,20])
        plt.subplot(211)
        sns.boxplot(x = 'SEX', y = 'LIMIT_BAL', data = creditcard_df, showfliers = False)
        plt.subplot(212)
        sns.boxplot(x = 'SEX', y = 'LIMIT_BAL', data = creditcard_df)
```

```
Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7f997392b080>
```



```
In [ ]: plt.figure(figsize=[10,20])
        plt.subplot(211)
        sns.boxplot(x = 'MARRIAGE', y = 'LIMIT_BAL', data = creditcard_df, showfliers = False)
        plt.subplot(212)
        sns.boxplot(x = 'MARRIAGE', y = 'LIMIT_BAL', data = creditcard_df)
```

```
Out[ ]: <matplotlib.axes._subplots.AxesSubplot at 0x7f997055ff98>
```



CREACIÓN DEL CONJUNTO DE DATOS DE PRUEBA Y ENTRENAMIENTO PARA EFECTUAR LA LIMPIEZA DE DATOS

```
In [ ]: creditcard_df
```

Out[ ]:

	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_0	PAY_2	PAY_3	PAY_4	PAY_5	PAY_6	BILL_AMT1	BILL_AMT2	BILL_AMT3
0	20000.0	2	2	1	24	2	2	-1	-1	-2	-2	3913.0	3102.0	689.0
1	120000.0	2	2	2	26	-1	2	0	0	0	2	2682.0	1725.0	2682.0
2	90000.0	2	2	2	34	0	0	0	0	0	0	29239.0	14027.0	13559.0
3	50000.0	2	2	1	37	0	0	0	0	0	0	46990.0	48233.0	49291.0
4	50000.0	1	2	1	57	-1	0	-1	0	0	0	8617.0	5670.0	35835.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
29995	220000.0	1	3	1	39	0	0	0	0	0	0	188948.0	192815.0	208365.0
29996	150000.0	1	3	2	43	-1	-1	-1	-1	0	0	1683.0	1828.0	3502.0
29997	30000.0	1	2	2	37	4	3	2	-1	0	0	3565.0	3356.0	2758.0
29998	80000.0	1	3	1	41	1	-1	0	0	0	-1	-1645.0	78379.0	76304.0
29999	50000.0	1	2	1	46	0	0	0	0	0	0	47929.0	48905.0	49764.0

30000 rows × 24 columns

```
In [ ]: X_cat = creditcard_df[['SEX', 'EDUCATION', 'MARRIAGE']]
        X_cat
```

Out[ ]:

	SEX	EDUCATION	MARRIAGE
0	2	2	1
1	2	2	2
2	2	2	2
3	2	2	1
4	1	2	1
...	...	...	...
29995	1	3	1
29996	1	3	2
29997	1	2	2
29998	1	3	1
29999	1	2	1

30000 rows × 3 columns

```
In [ ]: from sklearn.preprocessing import OneHotEncoder
        onehotencoder = OneHotEncoder()
        X_cat = onehotencoder.fit_transform(X_cat).toarray()
```

```
In [ ]: X_cat.shape
```

```
Out[ ]: (30000, 13)
```

```
In [ ]: X_cat = pd.DataFrame(X_cat)
```

```
In [ ]: X_cat
```

Out[ ]:

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0
1	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0
2	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0
3	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0
4	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...
29995	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0
29996	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0
29997	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0
29998	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0
29999	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0

30000 rows × 13 columns

```
In [ ]: # Se elimina la variable objetivo 'default.payment.next.month'
        X_numerical = creditcard_df[['LIMIT_BAL', 'AGE', 'PAY_0', 'PAY_2', 'PAY_3', 'PAY_4', 'PAY_5', 'PAY_6',
                                     'BILL_AMT1', 'BILL_AMT2', 'BILL_AMT3', 'BILL_AMT4', 'BILL_AMT5', 'BILL_AMT6',
                                     'PAY_AMT1', 'PAY_AMT2', 'PAY_AMT3', 'PAY_AMT4', 'PAY_AMT5', 'PAY_AMT6']]
        X_numerical
```

Out[ ]:

	LIMIT_BAL	AGE	PAY_0	PAY_2	PAY_3	PAY_4	PAY_5	PAY_6	BILL_AMT1	BILL_AMT2	BILL_AMT3	BILL_AMT4	BILL_AMT5	BILL_AMT6
0	20000.0	24	2	2	-1	-1	-2	-2	3913.0	3102.0	689.0	0.0	0.0	0.0
1	120000.0	26	-1	2	0	0	0	2	2682.0	1725.0	2682.0	3272.0	3455.0	3261.0
2	90000.0	34	0	0	0	0	0	0	29239.0	14027.0	13559.0	14331.0	14948.0	15549.0
3	50000.0	37	0	0	0	0	0	0	46990.0	48233.0	49291.0	28314.0	28959.0	29547.0
4	50000.0	57	-1	0	-1	0	0	0	8617.0	5670.0	35835.0	20940.0	19146.0	19131.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
29995	220000.0	39	0	0	0	0	0	0	188948.0	192815.0	208365.0	88004.0	31237.0	15980.0
29996	150000.0	43	-1	-1	-1	-1	0	0	1683.0	1828.0	3502.0	8979.0	5190.0	0.0
29997	30000.0	37	4	3	2	-1	0	0	3565.0	3356.0	2758.0	20878.0	20582.0	19357.0
29998	80000.0	41	1	-1	0	0	0	-1	-1645.0	78379.0	76304.0	52774.0	11855.0	48944.0
29999	50000.0	46	0	0	0	0	0	0	47929.0	48905.0	49764.0	36535.0	32428.0	15313.0

30000 rows × 20 columns

```
In [ ]: X_all = pd.concat([X_cat, X_numerical], axis = 1)
        X_all
```

	0	1	2	3	4	5	6	7	8	9	10	11	12	LIMIT_BAL	AGE	PAY_0	PAY_2	PAY_3	PAY_4	PAY_5	PAY_6	BILL_AMT1
0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	20000.0	24	2	2	-1	-1	-2	-2	3913.0
1	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	120000.0	26	-1	2	0	0	0	2	2682.0
2	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	90000.0	34	0	0	0	0	0	0	29239.0
3	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	50000.0	37	0	0	0	0	0	0	46990.0
4	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	50000.0	57	-1	0	-1	0	0	0	8617.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
29995	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	220000.0	39	0	0	0	0	0	0	188948.0
29996	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	150000.0	43	-1	-1	-1	-1	0	0	1683.0
29997	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	30000.0	37	4	3	2	-1	0	0	3565.0
29998	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	80000.0	41	1	-1	0	0	0	-1	-1645.0
29999	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	50000.0	46	0	0	0	0	0	0	47929.0

30000 rows × 33 columns

In [ ]:
```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
X = scaler.fit_transform(X_all)
```

In [ ]:
```
y = creditcard_df['default.payment.next.month']
y
```

Out[ ]:
```
0        1
1        1
2        0
3        0
4        0
        ..
29995    0
29996    0
29997    1
29998    1
29999    1
Name: default.payment.next.month, Length: 30000, dtype: int64
```

ENTRENAR Y EVALUAR UN CLASIFICADOR XGBOOST (EN LOCAL)

In [ ]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25)
```

In [ ]:
```
X_train.shape
```

Out[ ]:
```
(22500, 33)
```

In [ ]:
```
X_test.shape
```

Out[ ]:
```
(7500, 33)
```

In [ ]:
```
!pip install xgboost
```

```
Requirement already satisfied: xgboost in /usr/local/lib/python3.6/dist-packages (0.90)
Requirement already satisfied: scipy in /usr/local/lib/python3.6/dist-packages (from xgboost) (1.4.1)
Requirement already satisfied: numpy in /usr/local/lib/python3.6/dist-packages (from xgboost) (1.19.5)
```

In [ ]:
```
# Entrenar un modelo de regresión con XGBoost

import xgboost as xgb


model = xgb.XGBClassifier(objective ='reg:squarederror', learning_rate = 0.1, max_depth = 5, n_estimators = 10

model.fit(X_train, y_train)
```

Out[ ]:
```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0,
              learning_rate=0.1, max_delta_step=0, max_depth=5,
              min_child_weight=1, missing=None, n_estimators=100, n_jobs=1,
              nthread=None, objective='reg:squarederror', random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
              silent=None, subsample=1, verbosity=1)
```

In [ ]:
```
from sklearn.metrics import accuracy_score

y_pred = model.predict(X_test)
```

In [ ]:
```
y_pred
```

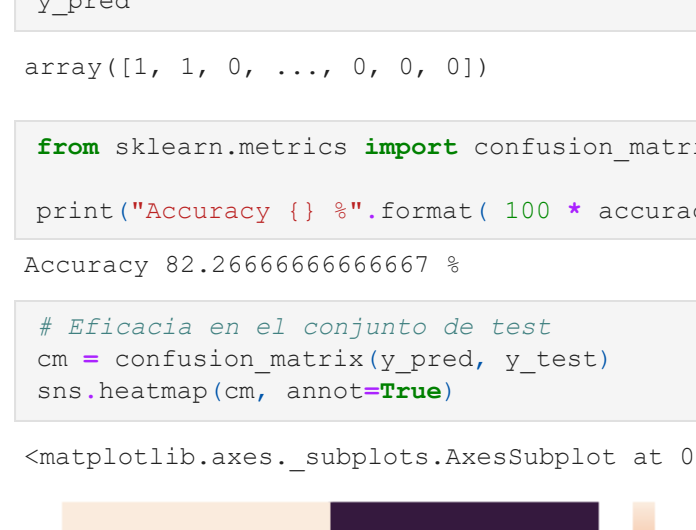
Out[ ]:
```
array([1, 1, 0, ..., 0, 0, 0])
```

In [ ]:
```
from sklearn.metrics import confusion_matrix, classification_report

print("Accuracy {} %".format( 100 * accuracy_score(y_pred, y_test)))
```

```
Accuracy 82.26666666666667 %
```

In [ ]:
```
# Eficacia en el conjunto de test
cm = confusion_matrix(y_pred, y_test)
sns.heatmap(cm, annot=True)
```

Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f996e180be0>
```



In [ ]:
```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.84      0.95      0.89      5883
           1       0.67      0.36      0.47      1617

    accuracy                           0.82      7500
   macro avg       0.75      0.65      0.68      7500
weighted avg       0.80      0.82      0.80      7500
```

OPTIMIZAR LOS HIPERPARÁMETROS DE XGBOOST REALIZANDO UN GRID SEARCH

In [ ]:
```
param_grid = {
        'gamma': [0.5, 1, 5],    # parámetro de regularización
        'subsample': [0.6, 0.8, 1.0], # % de filas que usamos para construir cada árbol
        'colsample_bytree': [0.6, 0.8, 1.0], # % de columnas usadas por cada árbol
        'max_depth': [3, 4, 5] # profundidad de cada árbol
        }
```

In [ ]:
```
from xgboost import XGBClassifier

xgb_model = XGBClassifier(learning_rate=0.01, n_estimators=100, objective='binary:logistic')
from sklearn.model_selection import GridSearchCV
grid = GridSearchCV(xgb_model, param_grid, refit = True, verbose = 4)
grid.fit(X_train, y_train)
```

In [ ]:
```
# Boto3 es el kit de desarrollo de software (SDK) de Amazon Web Services (AWS) para Python

import sagemaker
import boto3

# Crear una sesión de sagemaker
sagemaker_session = sagemaker.Session()

# S3 Bucket y prefijo que queremos usar
bucket = 'sagemaker-practical-1'
prefix = 'XGBoost-Regressor'
key = 'XGBoost-Regressor'
# Los roles dan acceso de aprendizaje y al alojamiento a nuestros datos
# Esto se especifica al abrir la instancia de sagemakers en "Crear un rol de IAM"
role = sagemaker.get_execution_role()
```

In [ ]:
```
print(role)
```

In [ ]:
```
# Leer los datos del archivo csv y luego cargar los datos en el depósito s3
import os
with open('train.csv','rb') as f:
    # El siguiente código carga los datos en el bucket de S3 para acceder más tarde al entrenamiento
    boto3.Session().resource('s3').Bucket(bucket).Object(os.path.join(prefix, 'train', key)).upload_fileobj(f)

# Ubicación de los datos de entrenamiento en s3
s3_train_data = 's3://{}/{}/train/{}'.format(bucket, prefix, key)
print('Ubicación de los datos de entrenamiento cargados: {}'.format(s3_train_data))
```

In [ ]:
```
with open('validation.csv','rb') as f:
    # Código de carga los datos en el bucket de S3 para acceder más tarde al entrenamiento
    boto3.Session().resource('s3').Bucket(bucket).Object(os.path.join(prefix, 'validation', key)).upload_fileob

# Ubicación de los datos de validación en s3
s3_validation_data = 's3://{}/{}/validation/{}'.format(bucket, prefix, key)
print('uploaded validation data location: {}'.format(s3_validation_data))
```

In [ ]:
```
# Placeholder de salida en el bucket S3 para almacenar la salida

output_location = 's3://{}/{}/output'.format(bucket, prefix)
print('los artefactos de entrenamiento se cargarán en: {}'.format(output_location))
```

In [ ]:
```
# Este código se usa para obtener el contenedor de entrenamiento de los algoritmos integrados de sagemaker
# todo lo que tenemos que especificar el nombre del algoritmo que queremos usar

# Obtengamos una referencia a la imagen del contenedor XGBoost

from sagemaker.amazon.amazon_estimator import get_image_uri

container = get_image_uri(boto3.Session().region_name, 'xgboost','0.90-2') # Latest version of XGboost
```

In [ ]:
```
# Especificar el tipo de instancia que nos gustaría usar para el entrenamiento.
# ruta de salida y sesión de sagemaker en el Estimador.
# También podemos especificar cuántas instancias nos gustaría utilizar para el entrenamiento

# Recordemos que XGBoost funciona combinando un conjunto de modelos débiles para generar resultados precisos /
# Los modelos débiles son aleatorios para evitar el sobreajuste

# num_round: el número de rondas para ejecutar el entrenamiento.

# Alfa: Término de regularización L1 sobre pesos. Incrementar este valor hace que los modelos sean más conserva

# colsample_by_tree: fracción de características que se usarán para entrenar cada árbol.

# eta: Reducción del tamaño del paso que se utiliza en las actualizaciones para evitar el sobreajuste.
# Después de cada paso de boosting, el parámetro eta reduce los pesos de las funciones para hacer que el proce


Xgboost_regressor1 = sagemaker.estimator.Estimator(container,
                                       role,
                                       train_instance_count = 1,
                                       train_instance_type = 'ml.m5.2xlarge',
                                       output_path = output_location,
                                       sagemaker_session = sagemaker_session)

# Podemos ajustar los hiperparámetros para mejorar el rendimiento del modelo

Xgboost_regressor1.set_hyperparameters(max_depth = 10,
                           objective = 'reg:linear',
                           colsample_bytree = 0.3,
                           alpha = 10,
                           eta = 0.1,
                           num_round = 100
                           )
```

In [ ]:
```
# Creamos los canales "train", "validation" para entregar los datos al modelo

train_input = sagemaker.session.s3_input(s3_data = s3_train_data, content_type='csv',s3_data_type = 'S3Prefix')
valid_input = sagemaker.session.s3_input(s3_data = s3_validation_data, content_type='csv',s3_data_type = 'S3Pre


data_channels = {'train': train_input,'validation': valid_input}


Xgboost_regressor1.fit(data_channels)
```

DESPLIEGUE EL MODELO PARA REALIZAR LA INFERENCIA

In [ ]:
```
# Implementar el modelo para realizar inferencias

Xgboost_regressor = Xgboost_regressor1.deploy(initial_instance_count = 1, instance_type = 'ml.m5.2xlarge')
```

In [ ]:
```
'''
El content_type anula los datos que se pasarán al modelo implementado, ya que el modelo implementado espera dat
en formato texto / csv, lo especificamos como tipo de contenido.

El serializador acepta un solo argumento, los datos de entrada, y devuelve una secuencia de bytes en el conteni
tipo

Referencia: https://sagemaker.readthedocs.io/en/stable/predictors.html
'''
from sagemaker.predictor import csv_serializer, json_deserializer

Xgboost_regressor.content_type = 'text/csv'
Xgboost_regressor.serializer = csv_serializer
Xgboost_regressor.deserializer = None
```

In [ ]:
```
X_test.shape
```

In [ ]:
```
# Predicción

predictions1 = Xgboost_regressor.predict(X_test[0:10000])
```

In [ ]:
```
predictions2 = Xgboost_regressor.predict(X_test[10000:20000])
```

In [ ]:
```
predictions3 = Xgboost_regressor.predict(X_test[20000:30000])
```

In [ ]:
```
predictions4 = Xgboost_regressor.predict(X_test[30000:31618])
```

In [ ]:
```
# código para convertir los valores en formato de bytes a una matriz

def bytes_2_array(x):

    # realiza la predicción completa como un string y lo divide en función de ','
    l = str(x).split(',')

    # Dado que el primer elemento contiene caracteres no deseados como (b, ',') los eliminamos
    l[0] = l[0][2:]
    # lo mismo que arriba elimina el último carácter no deseado (')
    l[-1] = l[-1][:-1]

    # iterando a través de la lista de cadenas y convirtiéndolas en tipo float
    for i in range(len(l)):
        l[i] = float(l[i])

    # convertimos la lista a array
    l = np.array(l).astype('float32')

    # redimensionamos la matriz unidimensional a una matriz bidimensional
    return l.reshape(-1,1)
```

In [ ]:
```
predicted_values_1 = bytes_2_array(predictions1)
```

In [ ]:
```
predicted_values_1.shape
```

In [ ]:
```
predicted_values_2 = bytes_2_array(predictions2)
predicted_values_2.shape
```

In [ ]:
```
predicted_values_3 = bytes_2_array(predictions3)
predicted_values_3.shape
```

In [ ]:
```
predicted_values_4 = bytes_2_array(predictions4)
predicted_values_4.shape
```

In [ ]:
```
predicted_values = np.concatenate((predicted_values_1, predicted_values_2, predicted_values_3, predicted_values
```

In [ ]:
```
predicted_values.shape
```

In [ ]:
```
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error
from math import sqrt
k = X_test.shape[1]
n = len(X_test)
RMSE = float(format(np.sqrt(mean_squared_error(y_test, predicted_values)),'.3f'))
MSE = mean_squared_error(y_test, predicted_values)
MAE = mean_absolute_error(y_test, predicted_values)
r2 = r2_score(y_test, predicted_values)
adj_r2 = 1-(1-r2)*(n-1)/(n-k-1)

print('RMSE =',RMSE, '\nMSE =',MSE, '\nMAE =',MAE, '\nR2 =', r2, '\nAdjusted R2 =', adj_r2)
```

In [ ]:
```
# Eliminamos el end-point

Xgboost_regressor.delete_endpoint()
```