

```
# This is formatted as code
```

▼ Actividad 6 Visualización A01793659

Actividad Semanal 6- Visualizacion

Ciencia y Analítica de datos

Jhon Edison Muñoz Burgos A01793659

**1. Descarga los datos a un sitio externo. y carga el dataset en tu libreta. **

```
input= "https://raw.githubusercontent.com/PosgradoMNA/Actividades_Aprendizaje-/main/default%2
import pandas as pd
import numpy as np
import matplotlib as plt
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

#se importa como csv el archivo a trabajar
df= pd.read_csv(input)
df
```



ID	X1	X2	X3	X4	X5	X6	X7	X8	X9	...	X15	X16	
----	----	----	----	----	----	----	----	----	----	-----	-----	-----	--

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

4 5 90000 2.0 2.0 2.0 34.0 0.0 0.0 0.0 0.0 ... 14331.0 14948.0 1554

df.columns #Validamos las columnas de dataframe

```
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')
```

df.head()

	ID	X1	X2	X3	X4	X5	X6	X7	X8	X9	...	X15	X16	X17	
0	1	20000	2.0	2.0	1.0	24.0	2.0	2.0	-1.0	-1.0	...	0.0	0.0	0.0	
1	2	120000	2.0	2.0	2.0	26.0	-1.0	2.0	0.0	0.0	...	3272.0	3455.0	3261.0	
2	3	90000	2.0	2.0	2.0	34.0	0.0	0.0	0.0	0.0	...	14331.0	14948.0	15549.0	151
3	4	50000	2.0	2.0	1.0	37.0	0.0	0.0	0.0	0.0	...	28314.0	28959.0	29547.0	200
4	5	50000	1.0	2.0	1.0	57.0	-1.0	0.0	-1.0	0.0	...	20940.0	19146.0	19131.0	200

5 rows × 25 columns



df.dtypes

```
ID      int64
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

```

```
df.info()
```

```

<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    X1       30000 non-null   int64   
 2    X2       29999 non-null   float64  
 3    X3       29998 non-null   float64  
 4    X4       29998 non-null   float64  
 5    X5       29995 non-null   float64  
 6    X6       29997 non-null   float64  
 7    X7       29995 non-null   float64  
 8    X8       29993 non-null   float64  
 9    X9       29991 non-null   float64  
10   X10      29984 non-null   float64  
11   X11      29986 non-null   float64  
12   X12      29989 non-null   float64  
13   X13      29989 non-null   float64  
14   X14      29987 non-null   float64  
15   X15      29985 non-null   float64  
16   X16      29983 non-null   float64  
17   X17      29990 non-null   float64  
18   X18      29992 non-null   float64  
19   X19      29991 non-null   float64  
20   X20      29992 non-null   float64  
21   X21      29989 non-null   float64  
22   X22      29989 non-null   float64  
23   X23      29995 non-null   float64  
24   Y        29997 non-null   float64  
dtypes: float64(23), int64(2)
memory usage: 5.7 MB

```

```

# se revisa si existe algún dato vacio en el archivo
df.isnull().values.any()

```

```
True
```

```
print("El DataFrame tiene {} valores nulos".format(df.isna().sum().sum())) #validamos cantidad
```

```
El DataFrame tiene 196 valores nulos
```

```

# se valida si en la columna hay datos perdidos
df.isnull().any()

```

```

ID      False
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

```

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

Hasta el día de hoy conocemos 2 posibles soluciones a realizar para limpiar la base de datos
 # una de ellas es eliminar las entradas con valores nulos
 # y la otra es colocar valores estadísticos del mismo archivo para tratar de no modificar las
 # para este ejercicio se tomará la solución 2, la cual es reemplazar los valores nulos con va

#This research employed a binary variable, default payment (Yes = 1, No = 0), as the response
 #X1: Amount of the given credit (NT dollar): it includes both the individual consumer credit
 #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 t
 #X12-X17: Amount of bill statement (NT dollar). X12 = amount of bill statement in September,
 #X18-X23: Amount of previous payment (NT dollar). X18 = amount paid in September, 2005; X19 =

```

#copia del archivo original
#ndf = df.copy()
#primero se utilizo la solución 1, de eliminar filas totalmnete cuando la salida Y no estaba
#si no se cuenta con ella no vale la pena utilizar esa información
ndf = df[df['Y'].notna()]

```

```
#Los archivos se rellenan con la media
#X1 no tiene valores vacios.

##X2: Gender (1 = male; 2 = female).
ndf['X2'].fillna(value = ndf.X2.mean(), inplace = True)

#X3 X3: Education (1 = graduate school; 2 = university; 3 = high school; 4 = others).
ndf['X3'].fillna(value = ndf.X3.mean(), inplace = True)

#X4: Marital status (1 = married; 2 = single; 3 = others).
ndf['X4'].fillna(value = ndf.X4.mean(), inplace = True)

#X5: Age (year). Se castea como entero para tener edades cerradas
aux=ndf.X5.mean()
ndf['X5'].fillna(value = int(aux), inplace = True)

#X6 - X11: Se castea como entero para tener edades cerradas
aux=ndf.X6.mean()
ndf['X6'].fillna(value = int(aux), inplace = True)
aux=ndf.X7.mean()
ndf['X7'].fillna(value = int(aux), inplace = True)
aux=ndf.X8.mean()
ndf['X8'].fillna(value = int(aux), inplace = True)
aux=ndf.X9.mean()
ndf['X9'].fillna(value = int(aux), inplace = True)
aux=ndf.X10.mean()
ndf['X10'].fillna(value = int(aux), inplace = True)
aux=ndf.X11.mean()
ndf['X11'].fillna(value = int(aux), inplace = True)

#X12-X17: es un valor float
aux=ndf.X12.mean()
ndf['X12'].fillna(value = aux, inplace = True)
aux=ndf.X13.mean()
ndf['X13'].fillna(value = aux, inplace = True)
aux=ndf.X14.mean()
ndf['X14'].fillna(value = aux, inplace = True)
aux=ndf.X15.mean()
ndf['X15'].fillna(value = aux, inplace = True)
aux=ndf.X16.mean()
ndf['X16'].fillna(value = aux, inplace = True)
aux=ndf.X17.mean()
ndf['X17'].fillna(value = aux, inplace = True)

#X18-X23: es un valor float

aux=ndf.X18.mean()
ndf['X18'].fillna(value = aux, inplace = True)
aux=ndf.X19.mean()
ndf['X19'].fillna(value = aux, inplace = True)
aux=ndf.X20.mean()
```

```
ndf['X20'].fillna(value = aux, inplace = True)
aux=ndf.X21.mean()
ndf['X21'].fillna(value = aux, inplace = True)
aux=ndf.X22.mean()
ndf['X22'].fillna(value = aux, inplace = True)
aux=ndf.X23.mean()
ndf['X23'].fillna(value = aux, inplace = True)
```

```
ndf.isnull().any() # se verifica si no existen datos perdidos.
```

```
/usr/local/lib/python3.7/dist-packages/pandas/core/generic.py:6392: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user
```

```
return self._update_inplace(result)
```

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

```
ndf
```

	ID	X1	X2	X3	X4	X5	X6	X7	X8	X9	...	X15	X16	
0	1	20000	2.0	2.0	1.0	24.0	2.0	2.0	-1.0	-1.0	...	0.0	0.0	
1	2	120000	2.0	2.0	2.0	26.0	-1.0	2.0	0.0	0.0	...	3272.0	3455.0	326
2	3	90000	2.0	2.0	2.0	34.0	0.0	0.0	0.0	0.0	...	14331.0	14948.0	1554
3	4	50000	2.0	2.0	1.0	37.0	0.0	0.0	0.0	0.0	...	28314.0	28959.0	2954
4	5	50000	1.0	2.0	1.0	57.0	-1.0	0.0	-1.0	0.0	...	20940.0	19146.0	1913
...
29995	29996	220000	1.0	3.0	1.0	39.0	0.0	0.0	0.0	0.0	...	88004.0	31237.0	1598
29996	29997	150000	1.0	3.0	2.0	43.0	-1.0	-1.0	-1.0	-1.0	...	8979.0	5190.0	
29997	29998	200000	1.0	3.0	2.0	37.0	1.0	2.0	0.0	1.0	...	20070.0	20500.0	1005

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

```
Aux_df=ndf.copy()
Aux_df.head()

Aux_df.info()

<class 'pandas.core.frame.DataFrame'>
Int64Index: 29997 entries, 0 to 29999
Data columns (total 25 columns):
#   Column      Non-Null Count  Dtype
---  -
0    ID          29997 non-null   int64
1    X1          29997 non-null   int64
2    X2          29997 non-null   float64
3    X3          29997 non-null   float64
4    X4          29997 non-null   float64
5    X5          29997 non-null   float64
6    X6          29997 non-null   float64
7    X7          29997 non-null   float64
8    X8          29997 non-null   float64
9    X9          29997 non-null   float64
10   X10         29997 non-null   float64
11   X11         29997 non-null   float64
12   X12         29997 non-null   float64
13   X13         29997 non-null   float64
14   X14         29997 non-null   float64
15   X15         29997 non-null   float64
16   X16         29997 non-null   float64
17   X17         29997 non-null   float64
18   X18         29997 non-null   float64
19   X19         29997 non-null   float64
20   X20         29997 non-null   float64
21   X21         29997 non-null   float64
22   X22         29997 non-null   float64
```

```

23  X23      29997 non-null float64
24  Y        29997 non-null float64
dtypes: float64(23), int64(2)
memory usage: 6.0 MB

```

Aux_df.info() #para conocer el tipo de datos que hay en cada columna, la mayoría o todos son

#23 variables tipo float y 2 variables tipo int, todas las variables son numéricas

```

<class 'pandas.core.frame.DataFrame'>
Int64Index: 29997 entries, 0 to 29999
Data columns (total 25 columns):
#   Column  Non-Null Count  Dtype
---  -
0   ID      29997 non-null    int64
1   X1      29997 non-null    int64
2   X2      29997 non-null    float64
3   X3      29997 non-null    float64
4   X4      29997 non-null    float64
5   X5      29997 non-null    float64
6   X6      29997 non-null    float64
7   X7      29997 non-null    float64
8   X8      29997 non-null    float64
9   X9      29997 non-null    float64
10  X10     29997 non-null    float64
11  X11     29997 non-null    float64
12  X12     29997 non-null    float64
13  X13     29997 non-null    float64
14  X14     29997 non-null    float64
15  X15     29997 non-null    float64
16  X16     29997 non-null    float64
17  X17     29997 non-null    float64
18  X18     29997 non-null    float64
19  X19     29997 non-null    float64
20  X20     29997 non-null    float64
21  X21     29997 non-null    float64
22  X22     29997 non-null    float64
23  X23     29997 non-null    float64
24  Y       29997 non-null    float64
dtypes: float64(23), int64(2)
memory usage: 6.0 MB

```

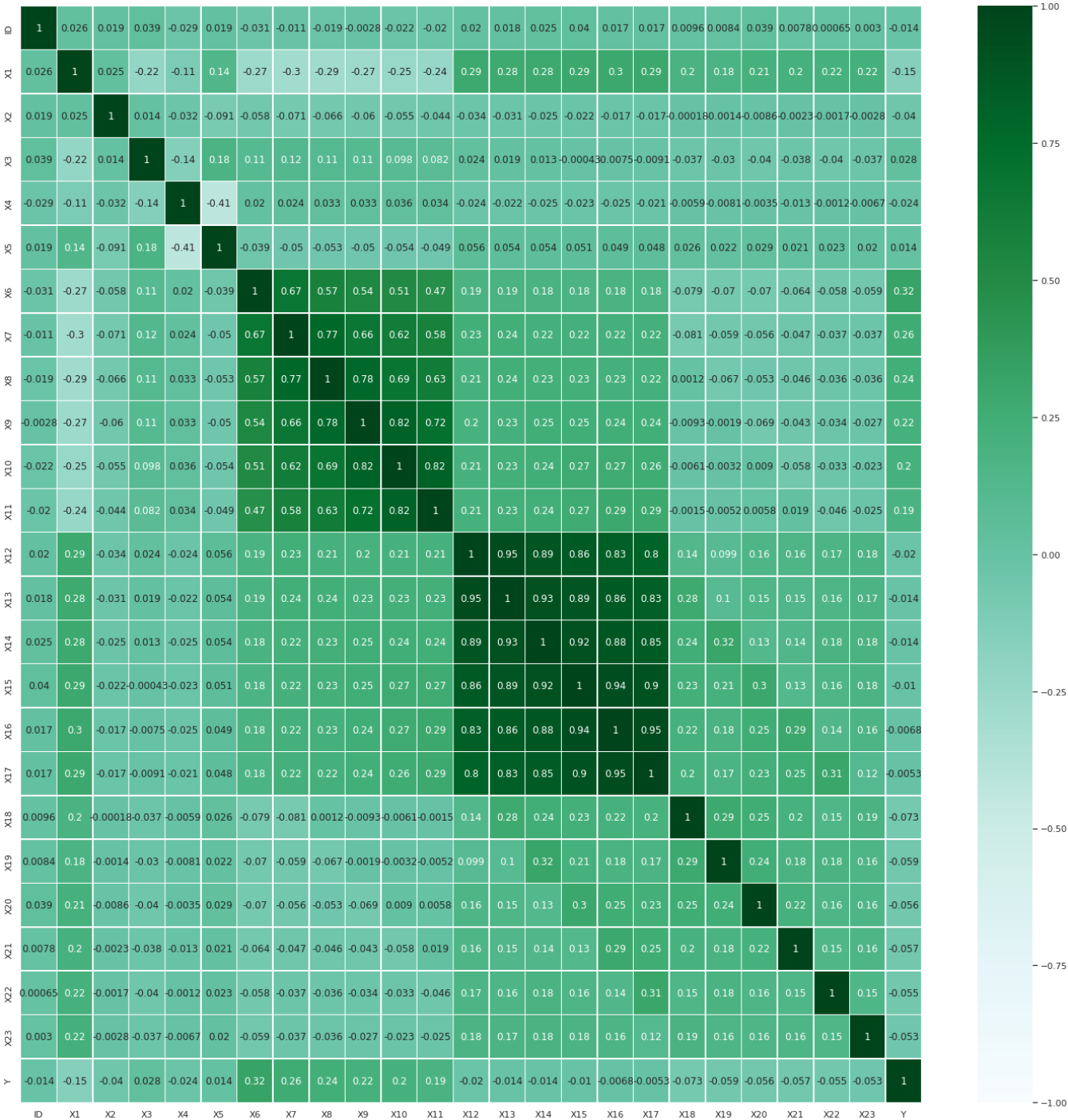
#al ser todas las variables numéricas no se necesita hacer una selección de variables para poder
#se efectúa la correlación entre variables

```

correlacion =Aux_df.corr()
sns.set(rc = {'figure.figsize':(25,25)})
sns.heatmap(correlacion, vmin = -1, vmax = 1, cmap = "BuGn", annot= True,linewidths=.5)

```


<matplotlib.axes._subplots.AxesSubplot at 0x7f26152e3c10>



```

# Se encuentra correlacion entre algunas columnas, utilizaremos +/-0.9 de correlacion como marg
# Estas son las columnas que consideramos altamente correlacionadas
# X12 con X13 = 0.95
# X13 con X14 = 0.93
# X14 con X15 = 0.92
# X15 con X16 = 0.94
# X16 con X17 = 0.95

columnasDeInteres=['X12', 'X13', 'X14', 'X15', 'X16']

resumeDeColumnas = {
    "varianza": Aux_df[columnasDeInteres].var(),
    "minimo": Aux_df[columnasDeInteres].min(),
    "maximo": Aux_df[columnasDeInteres].max()
}

pd.DataFrame(resumeDeColumnas, index=columnasDeInteres).transpose()

```

	X12	X13	X14	X15	X16
varianza	5.422169e+09	5.065721e+09	4.809326e+09	4.138687e+09	3.696128e+09
minimo	-1.655800e+05	-6.977700e+04	-1.572640e+05	-1.700000e+05	-8.133400e+04
maximo	9.645110e+05	9.839310e+05	1.664089e+06	8.915860e+05	9.271710e+05



Por medio del mapa de calor observamos una correlacion entre las variables X12,X13,X14,X15 Y X16 por lo tanto stas seran nuestras columnas de interes, Cuando se revisa el data frame observams que la varianza entre estas columnas esta entre 3.7 y 5.4 para cada columna. Todas las columnas tienen una varianza negativa como valor minimo y como valor maximo de estas columnas la mas alta es la X13 con 9.8

```
Aux_df.describe()
```

	ID	X1	X2	X3	X4	X5
count	29997.000000	29997.000000	29997.000000	29997.000000	29997.000000	29997.000000
mean	14999.803847	167496.072274	1.603781	1.853076	1.551925	35.48371
std	8659.837419	129748.803871	0.489111	0.790316	0.521950	9.2173
min	1.000000	10000.000000	1.000000	0.000000	0.000000	21.00000
25%	7501.000000	50000.000000	1.000000	1.000000	1.000000	28.00000
50%	15000.000000	140000.000000	2.000000	2.000000	2.000000	34.00000
75%	22499.000000	240000.000000	2.000000	2.000000	2.000000	41.00000
max	30000.000000	1000000.000000	2.000000	6.000000	3.000000	79.00000

8 rows × 25 columns



Acontinuación se explica la medidas de tendencia Central y Dispersión

1. Count: Cantidad de datos sin valores nulos por columna para este caso sera de 29997
2. Mean: Es la suma de todos los datos dividido entre la cantidad de datos.
3. Std: Desviacion estandar, nos perte ver que tan dispersos estan nuestros datos.
4. Min: Valor minimo de cada columna.
5. 25% Cantidad de valores que se encuentren en el primer cuartil.
6. 50% Cantidad de valores que se encuentren correspondientes al 0.50
7. 75% Cantidad de valores que se encuentren correspondientes al 0.75
8. max. Valor Maximo apra cada columna.

```
Aux_df.columns
```

```
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')
```

5. Realiza el conteo de las variables categóricas

Definimos como variables categoricas las siguientes.

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.

```
categoricas = ['X2', 'X3', 'X4', 'X6', 'X7', 'X8', 'X9', 'X10', 'X11', 'Y'],
```

```
i=0
```

```
for i in categoricas[i]:
```

```
    print(ndf.value_counts([i]),'\n')
```

```
    0.0    15767
```

```
   -1.0     5935
```

```
   -2.0     4085
```

```
    2.0     3817
```

```
    3.0       240
```

```
    4.0        76
```

```
    7.0         27
```

```
    6.0         23
```

```
    5.0         21
```

```
    1.0          4
```

```
    8.0          2
```

```
dtype: int64
```

```
X9
```

```
    0.0    16457
```

```
   -1.0     5685
```

```
   -2.0     4348
```

```
    2.0     3156
```

```
    3.0      180
```

```
    4.0        69
```

```
    7.0         58
```

```
    5.0         35
```

```
    6.0          5
```

```
    1.0          2
```

```
    8.0          2
```

```
dtype: int64
```

```
X10
```

```
    0.0    16951
```

```
   -1.0     5535
```

```
   -2.0     4546
```

```
    2.0     2623
```

```
    3.0      178
```

```
    4.0        84
```

```
    7.0         58
```

```
    5.0         17
```

```
    6.0          4
```

```
    8.0          1
```

```
dtype: int64
```

```
dtype: int64
```

```
X11
```

```
0.0    16290
-1.0    5735
-2.0    4895
 2.0    2764
 3.0     184
 4.0      49
 7.0      46
 6.0      19
 5.0      13
 8.0       2
```

```
dtype: int64
```

```
Y
```

```
0.0    23362
1.0     6635
dtype: int64
```

6.Escala los datos, si consideras necesario

los valores numericos obtenidos son muy grandes y dificil de interpretar por lo que se reali

```
scaler = StandardScaler()      # Nos permite que todos los valores esten en la misma escala
scaled = scaler.fit_transform(Aux_df)
```

```
df_escalado= pd.DataFrame(scaled,columns=Aux_df.columns) #Para organizar los datos obtenidos
```

```
df_escalado.head()
```

	ID	X1	X2	X3	X4	X5	X6	X7	
0	-1.732025	-1.136801	0.810094	0.185909	-1.057446	-1.245909	1.794773	1.782533	-0.69
1	-1.731909	-0.366068	0.810094	0.185909	0.858477	-1.028923	-0.875003	1.782533	0.13
2	-1.731794	-0.597288	0.810094	0.185909	0.858477	-0.160980	0.014923	0.111784	0.13
3	-1.731678	-0.905581	0.810094	0.185909	-1.057446	0.164499	0.014923	0.111784	0.13
4	-1.731563	-0.905581	-1.234465	0.185909	-1.057446	2.334357	-0.875003	0.111784	-0.69

```
5 rows × 25 columns
```

Se aplicara el metodo PCa de sckikit learn al df transformado


```
pcs = PCA()
pcs_transformado= pcs.fit_transform(df_escalado)

#se pone nombre a los componentes
pcs_labels = []
for i in range(len(df_escalado.columns)):
    pcs_labels.append(f'PC{i + 1}')

# se crea un dataframe con el PCS y lo nombres de los componentes
df_pcs = pd.DataFrame (pcs_transformado, columns=pcs_labels)

#Se imprime la varianza ´pr componente de PCA
df_resumen_pcs = pd.DataFrame({
'% varianza componente': np.round(pcs.explained_variance_ratio_,4 )* 100,
'% varianza acumulada': np.cumsum(pcs.explained_variance_ratio_) * 100
})

df_resumen_pcs.index = pcs_labels
df_resumen_pcs
```

	% varianza componente	% varianza acumulada	
PC1	26.21	26.206842	
PC2	16.81	43.016573	
PC3	6.22	49.236856	
PC4	5.91	55.142659	
PC5	4.24	59.387260	
PC6	3.94	63.329130	
PC7	3.88	67.209618	
PC8	3.66	70.870152	
---	---	---	

#Se comprueba que la varianza obtenida del dataframe escalado y la varianza del dataframe cre

#Se puede decir que existe la misma informacion

```
print("Varianza total variables originales: ", df_escalado.var().sum())
```

```
print("Varianza total de los componentes: ", df_pcs.var().sum())
```

Varianza total variables originales: 25.00083344445926

Varianza total de los componentes: 25.000833444459257

PC11

2.51

90.084017

df_resumen_pcs

	% varianza componente	% varianza acumulada
PC1	26.21	26.206842
PC2	16.81	43.016573
PC3	6.22	49.236856
PC4	5.91	55.142659
PC5	4.24	59.387260
PC6	3.94	63.329130
PC7	3.88	67.209618
PC8	3.66	70.870152
PC9	3.63	74.495372
PC10	3.54	78.039299
PC11	2.10	80.139275



#de la tabla anterior podemos determinar que con un 11 componentes de PCA tenemos el 80 % d

total_var =df_escalado.var().sum()

pd.DataFrame({

"Porcentaje Varianza por componente": (df_escalado.var()/ total_var) * 100,

"Porcentaje Varianza Acumulado": (df_escalado.var().cumsum() / total_var) * 100

})

	Porcentaje Varianza por componente	Porcentaje Varianza Acumulado
ID	4.0	4.0
X1	4.0	8.0
X2	4.0	12.0
X3	4.0	16.0
X4	4.0	20.0
X5	4.0	24.0
X6	4.0	28.0
X7	4.0	32.0
X8	4.0	36.0
X9	4.0	40.0
X10	4.0	44.0
X11	4.0	48.0

```
# De las variables originales necesitaríamos 19 variables para obtener 80% de la varianza
# mientras que con PPCA solo necesitamos 11 componentes, casi la mitad
# ahora vamos a analizar cual es el componente principal de cada uno de los 11 componentes a
df_componentes = pd.DataFrame(
    pcs.components_.round(4), # pcs.components nos da los pesos de cada uno de los componentes
    columns = df_pcs.columns, #Son 13 columnas, las cuales hacen referencia a cada uno de los com
    index = df_escalado.columns) #Las filas son 13 también, las cuales hacen referencia a las var
#Recuerda, el código siguiente dice que me de todas las filas pero solo las primeras 7 column
#La estructura es [filas, columnas]
df_componentes.iloc[:, :11]
```

	PC1	PC2	PC3	PC4	PC5	PC6	PC7	PC8	PC9	PC10	
ID	0.0062	0.0657	-0.0221	0.0199	-0.0056	0.0140	0.1644	0.1972	0.2032	0.2098	
X1	0.0228	0.3116	0.0309	-0.0882	-0.0399	0.0629	-0.2967	-0.3280	-0.3349	-0.3344	-
X2	-0.0596	0.0098	0.0267	-0.3239	0.4739	-0.4833	-0.0205	0.0173	0.0617	0.0888	
X3	0.0524	0.0759	-0.0757	0.2230	-0.4188	0.4356	0.0190	0.0486	0.0815	0.1065	
X4	0.5012	-0.1857	0.6088	0.4060	0.0204	-0.1604	-0.1051	-0.0407	-0.0027	0.0373	
X5	0.7054	-0.0603	-0.6536	0.0150	0.1777	0.0071	-0.0146	0.0019	-0.0135	-0.0182	-
X6	-0.4501	-0.2599	-0.3730	0.4178	0.1599	0.0120	-0.1475	-0.0715	-0.0090	0.0212	
X7	-0.1101	-0.2764	0.0139	0.4546	0.1964	-0.1402	0.2151	0.0457	-0.0844	-0.1931	-
X8	-0.0945	0.0135	0.0867	-0.2650	-0.1056	0.0388	-0.0075	-0.0795	-0.0534	-0.0306	
X9	-0.0467	-0.0338	-0.0407	0.0594	0.0093	0.0079	-0.0340	0.0025	-0.0103	-0.0176	
X10	0.0325	0.0271	0.0147	-0.1600	-0.0558	0.0468	-0.0333	-0.0326	-0.0098	0.0475	
X11	-0.1212	0.0088	0.0312	0.0750	0.0377	0.0147	-0.0073	-0.0024	-0.0263	-0.0567	
X12	-0.0103	0.0152	-0.0153	0.0140	-0.0037	-0.0165	-0.0054	0.0029	-0.1100	0.0116	
X13	0.0090	-0.1248	0.0064	-0.1951	-0.2292	-0.0922	0.5103	0.3986	0.1528	-0.1513	-

```
# La tabla anterior muestra la composicion de los componentes de PCa con respecto a las variables
# vamos a seleccionar las variables principales de cada componente para
# quedarnos con las variables mas importantes
df_componentes.iloc[:, :25].abs().idxmax()
```

```
PC1      X5
PC2      X15
PC3      X5
PC4      X7
PC5      X14
PC6      X14
PC7      X16
PC8      X17
PC9      X19
PC10     X17
PC11     X20
PC12     X17
PC13     X18
PC14      Y
PC15      Y
PC16     X21
PC17     X23
PC18     X21
PC19     X12
PC20     X12
PC21     X11
PC22     X9
```

```
PC23      X8
PC24      X10
PC25      X7
dtype: object
```

```
# Por lo tanto las variables mas importantes de los datos son
# X5, X7, X14, X15, X16, X17, X19, X20
```

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

```
df_componentes.columns
```

```
Index(['PC1', 'PC2', 'PC3', 'PC4', 'PC5', 'PC6', 'PC7', 'PC8', 'PC9', 'PC10',
      'PC11', 'PC12', 'PC13', 'PC14', 'PC15', 'PC16', 'PC17', 'PC18', 'PC19',
      'PC20', 'PC21', 'PC22', 'PC23', 'PC24', 'PC25'],
      dtype='object')
```

```
PC_components = np.arange(pcs.n_components_) + 1
#PC_components
```

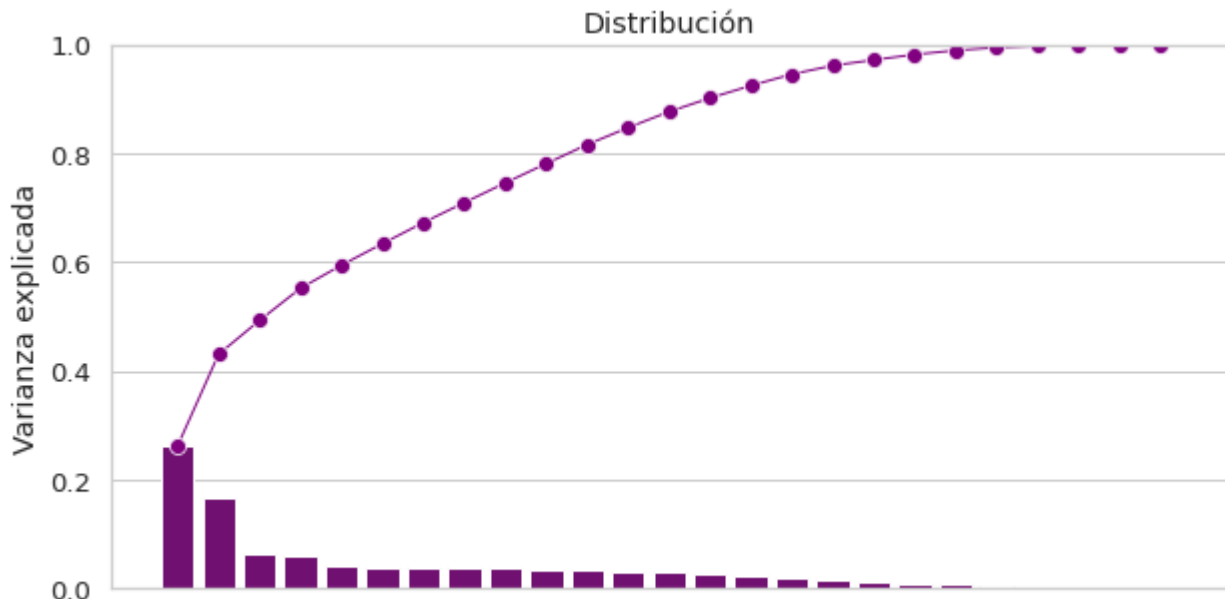
```
_ = sns.set(style = 'whitegrid',
            font_scale = 1.2
            )
```

```
fig, ax = plt.subplots(figsize=(10, 5))
```

```
_ = sns.barplot(x = PC_components,
                y = pcs.explained_variance_ratio_,
                color = 'purple'
                )
```

```
_ = sns.lineplot(x = PC_components-1,
                 y = np.cumsum(pcs.explained_variance_ratio_),
                 color = 'purple',
                 linestyle = '- ',
                 linewidth = 1,
                 marker = 'o',
                 markersize = 8
                 )
```

```
plt.title('Distribución')
plt.xlabel('Principales Componentes')
plt.ylabel('Varianza explicada')
plt.ylim(0, 1)
plt.show()
```



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.

Se encuentra correlación entre algunas columnas,
 utilizaremos ± 0.9 de correlación como margen

Estas son las columnas que consideramos altamente correlacionadas

X12 con X13 = 0.95

X13 con X14 = 0.93

X14 con X15 = 0.92

X15 con X16 = 0.94

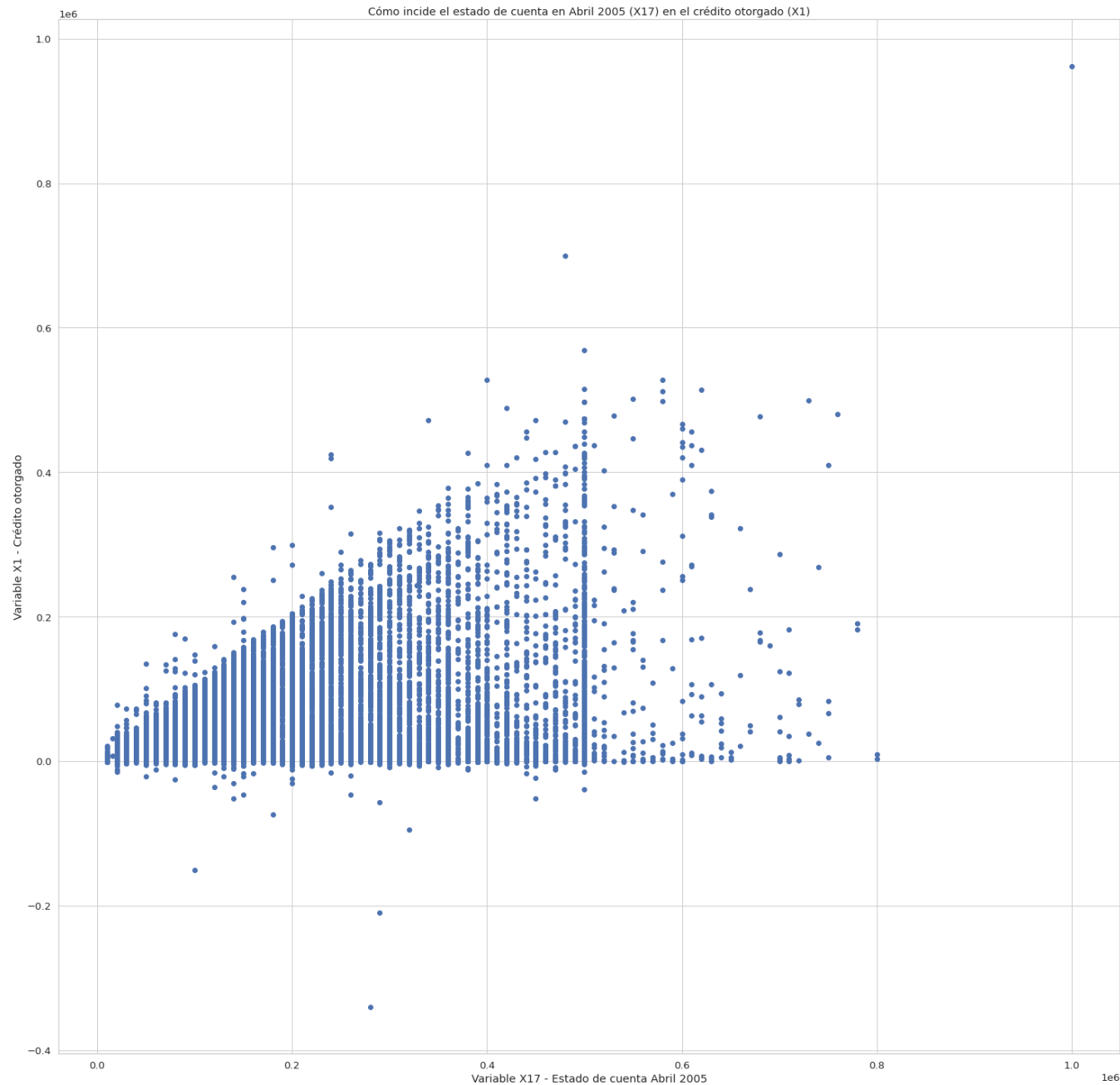
X16 con X17 = 0.95

```
plt.scatter(Aux_df.X12, Aux_df.X13)
```

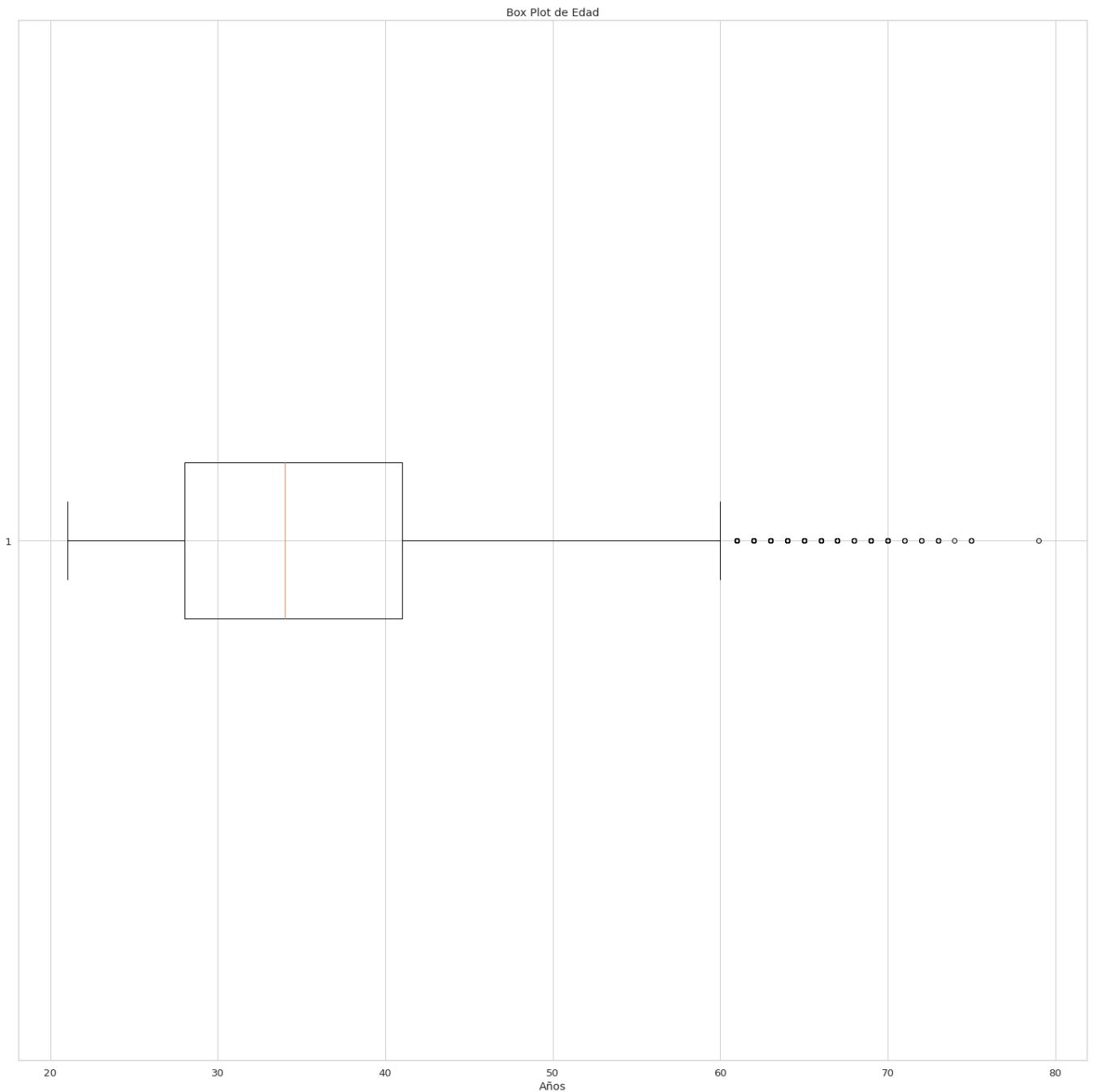
```
plt.title("monto del estado de cuenta en agosto de 2005 (X13) Importe del estado de cuenta' (  
plt.xlabel('Variable X13 - monto del estado de cuenta en agosto de 2005')  
plt.ylabel('Variable X12 - Importe del estado de cuenta')  
plt.show()
```



```
plt.scatter(Aux_df.X1, Aux_df.X17)
plt.title("Cómo incide el estado de cuenta en Abril 2005 (X17) en el crédito otorgado (X1)")
plt.xlabel('Variable X17 - Estado de cuenta Abril 2005')
plt.ylabel('Variable X1 - Crédito otorgado')
plt.show()
```



```
plt.boxplot(Aux_df.X5, vert=False)
plt.title('Box Plot de Edad')
plt.xlabel('Años')
plt.show()
```

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.

El PCA nos permite hacer una reducción de datos sin impactar en la información, PCA nos permite mejorar el rendimiento del modelo en la mayoría de escenarios. En el rendimiento se debe tener en cuenta que los datos tengan correlación de características, al igual es útil eliminando información redundante lo cual mejora el modelo de regresión, además permite una visualización de grandes cantidades de datos.