Tecnológico de Monterrey

Actividad Semanal – 6 Visualización

CIENCIA Y ANALITICA DE DATOS
Profesor: María de La Paz Rico

Alumno: Guillermo Alfonso Muñiz Hermosillo - A01793101

ENLACE COLLAB:

https://colab.research.google.com/drive/1bZS_S9-fDIbprpYwuWUuw5vUvTlowyi-?usp=sharing

ENLACE GITHUB:

https://github.com/PosgradoMNA/actividades-de-aprendizaje-A01793101-GuillermoMuniz/tree/main/Acitividades%20Semanales/ActividadSemanal6-Visualizacion

Actividad Semanal -- 6 Visualización

CIENCIA Y ANALITICA DE DATOS

Profesor: María de La Paz Rico

Alumno: Guillermo Alfonso Muñiz Hermosillo - A01793101

COLLAB LINK

GITHUB LINK

Antes que comencemos el analisis, importamos nuestras librerias basicas como Numpy, Pandas, Matplolib y Seaborn.

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

import ssl
ssl._create_default_https_context = ssl._create_unverified_context
```

EJERCICIO 1

DESCARGA LOS DATOS Y CARGA EL DATASET

In [2]:
```
path = 'https://raw.githubusercontent.com/PosgradoMNA/Actividades_Aprendizaje-/
df = pd.read_csv(path)

## Para que fuera mas claro decidi remplazar el nombre de las columnas
newColumnNames = ['ID','Total_Credito', 'Sexo', 'Estudios', 'Estado_Civil', 'Ed
# PP = Pagos Pasados
# TR = Total del Recibo
# TPP = Total de Pagos Pasados
df.columns = newColumnNames # Remplazamos los nombres de las columnas
df.head()
```

Out[2]:

	ID	Total_Credito	Sexo	Estudios	Estado_Civil	Edad	PPSep2005	PPAgo2005	PPJul2005
0	1	20000	2.0	2.0	1.0	24.0	2.0	2.0	-1.0
1	2	120000	2.0	2.0	2.0	26.0	-1.0	2.0	0.0
2	3	90000	2.0	2.0	2.0	34.0	0.0	0.0	0.0
3	4	50000	2.0	2.0	1.0	37.0	0.0	0.0	0.0
4	5	50000	1.0	2.0	1.0	57.0	-1.0	0.0	-1.0

5 rows × 25 columns

Este conjunto de datos nos muestra informacion de un banco sobre tarjetas de credito e historial de pagos pasados asi como informacion biometrica y personal de los clientes del banco.

EJERCICIO 2

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

In [3]:
```
print('SHAPE: ', df.shape)
print('--------------------------')
```

```
SHAPE:  (30000, 25)
--------------------------
```

Existen 30,000 filas de 25 columnas en nuestro conjunto de datos

In [4]:
```
print('COLUMNAS: ', df.columns)
print('--------------------------')
```

```
COLUMNAS:  Index(['ID', 'Total_Credito', 'Sexo', 'Estudios', 'Estado_Civil',
'Edad',
       'PPSep2005', 'PPAgo2005', 'PPJul2005', 'PPJun2005', 'PPMay2005',
       'PPAbr2005', 'TRSep2005', 'TRAgo2005', 'TRJul2005', 'TRJun2005',
       'TRMay2005', 'TRAbr2005', 'TPPSep2005', 'TPPAgo2005', 'TPPJul2005',
       'TPPJun2005', 'TPPMay2005', 'TPPAbr2005', 'Y'],
      dtype='object')
--------------------------
```

In [5]:
```
print('TIPOS DE DATOS: ', df.dtypes)
print('--------------------------')
```

```
TIPOS DE DATOS:  ID                int64
Total_Credito      int64
Sexo             float64
Estudios         float64
Estado_Civil     float64
Edad             float64
PPSep2005        float64
PPAgo2005        float64
PPJul2005        float64
PPJun2005        float64
PPMay2005        float64
PPAbr2005        float64
TRSep2005        float64
TRAgo2005        float64
TRJul2005        float64
TRJun2005        float64
TRMay2005        float64
TRAbr2005        float64
TPPSep2005       float64
TPPAgo2005       float64
TPPJul2005       float64
TPPJun2005       float64
TPPMay2005       float64
TPPAbr2005       float64
Y                float64
dtype: object
---------------------------
```

In [6]:
```
print('INFO: ')
df.info()
print('---------------------------')
```

```
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   Total_Credito  30000 non-null  int64
 2   Sexo           29999 non-null  float64
 3   Estudios       29998 non-null  float64
 4   Estado_Civil   29998 non-null  float64
 5   Edad           29995 non-null  float64
 6   PPSep2005      29997 non-null  float64
 7   PPAgo2005      29995 non-null  float64
 8   PPJul2005      29993 non-null  float64
 9   PPJun2005      29991 non-null  float64
 10  PPMay2005      29984 non-null  float64
 11  PPAbr2005      29986 non-null  float64
 12  TRSep2005      29989 non-null  float64
 13  TRAgo2005      29989 non-null  float64
 14  TRJul2005      29987 non-null  float64
 15  TRJun2005      29985 non-null  float64
 16  TRMay2005      29983 non-null  float64
 17  TRAbr2005      29990 non-null  float64
 18  TPPSep2005     29992 non-null  float64
 19  TPPAgo2005     29991 non-null  float64
 20  TPPJul2005     29992 non-null  float64
 21  TPPJun2005     29989 non-null  float64
 22  TPPMay2005     29989 non-null  float64
 23  TPPAbr2005     29995 non-null  float64
 24  Y              29997 non-null  float64
dtypes: float64(23), int64(2)
memory usage: 5.7 MB
---------------------------
```

In [7]:
```
print('IN NA: ', df.isna())
print('---------------------------')
```

```
IN NA:            ID  Total_Credito   Sexo  Estudios  Estado_Civil   Edad  PPS
ep2005  \
0      False          False  False     False         False  False      False
1      False          False  False     False         False  False      False
2      False          False  False     False         False  False      False
3      False          False  False     False         False  False      False
4      False          False  False     False         False  False      False
...      ...            ...    ...       ...           ...    ...        ...
29995  False          False  False     False         False  False      False
29996  False          False  False     False         False  False      False
29997  False          False  False     False         False  False      False
29998  False          False  False     False         False  False      False
29999  False          False  False     False         False  False      False

       PPAgo2005  PPJul2005  PPJun2005  ...  TRJun2005  TRMay2005  TRAbr2005
\
0          False      False      False  ...      False      False      False
1          False      False      False  ...      False      False      False
2          False      False      False  ...      False      False      False
3          False      False      False  ...      False      False      False
4          False      False      False  ...      False      False      False
...          ...        ...        ...  ...        ...        ...        ...
29995      False      False      False  ...      False      False      False
29996      False      False      False  ...      False      False      False
29997      False      False      False  ...      False      False      False
29998      False      False      False  ...      False      False      False
29999      False      False      False  ...      False      False      False

       TPPSep2005  TPPAgo2005  TPPJul2005  TPPJun2005  TPPMay2005  TPPAbr2005
\
0           False       False       False       False       False       False
1           False       False       False       False       False       False
2           False       False       False       False       False       False
3           False       False       False       False       False       False
4           False       False       False       False       False       False
...           ...         ...         ...         ...         ...         ...
29995       False       False       False       False       False       False
29996       False       False       False       False       False       False
29997       False       False       False       False       False       False
29998       False       False       False       False       False       False
29999       False       False       False       False       False       False

          Y
0      False
1      False
2      False
3      False
4      False
...      ...
29995  False
29996  False
29997  False
29998  False
29999  False

[30000 rows x 25 columns]
--------------------------
```

In [8]:
```
print('HEAD DEL DATAFRAME: ')
df.head()
```

```
HEAD DEL DATAFRAME:
```

Out[8]:

	ID	Total_Credito	Sexo	Estudios	Estado_Civil	Edad	PPSep2005	PPAgo2005	PPJul2005
0	1	20000	2.0	2.0	1.0	24.0	2.0	2.0	-1.0
1	2	120000	2.0	2.0	2.0	26.0	-1.0	2.0	0.0
2	3	90000	2.0	2.0	2.0	34.0	0.0	0.0	0.0
3	4	50000	2.0	2.0	1.0	37.0	0.0	0.0	0.0
4	5	50000	1.0	2.0	1.0	57.0	-1.0	0.0	-1.0

5 rows × 25 columns

EJERCICIO 3

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

In [9]:
```
# Verificamos El numero de datos nulos
pd.DataFrame(df.isna().sum(), columns=['# NULOS'])
```

Out[9]:

	# NULOS
ID	0
Total_Credito	0
Sexo	1
Estudios	2
Estado_Civil	2
Edad	5
PPSep2005	3
PPAgo2005	5
PPJul2005	7
PPJun2005	9
PPMay2005	16
PPAbr2005	14
TRSep2005	11
TRAgo2005	11
TRJul2005	13
TRJun2005	15
TRMay2005	17
TRAbr2005	10
TPPSep2005	8
TPPAgo2005	9
TPPJul2005	8
TPPJun2005	11
TPPMay2005	11
TPPAbr2005	5
Y	3

In [10]:
```
# Copiamos nuestros datos a un dataframe nuevo, donde realizaremos las transfor
dfLimpio = df.copy()
print("ORIGINAL DATAFRAME SHAPE: ", df.shape)
print("DATAFRAME LIMPIO SHAPE ORIGINAL: ", dfLimpio.shape)
```

```
ORIGINAL DATAFRAME SHAPE:  (30000, 25)
DATAFRAME LIMPIO SHAPE ORIGINAL:  (30000, 25)
```

Limpieza de datos Nulos

- Variables Categoricas: Inputacion por Mediana
- Variables Numericas: Inputacion por Media y Escalamiento
- FILAS con Y faltante: DROP ya que son pocas (3) y son nuestra variable de prediccion.
- COLUMNA ID: DROP YA que no es necesario.

In [11]:
```
# Definimos nuestras variables Categoricas
categoricas = ['Sexo', 'Estudios', 'Estado_Civil', 'PPSep2005', 'PPAgo2005', 'P
# Definimos nuestras variables Numericas
numericas = ['Total_Credito','Edad','TRSep2005', 'TRAgo2005', 'TRJul2005','TRJu

print('TOTAL VARIABLES CATEGORICAS: ',len(categoricas))
print('TOTAL VARIABLES NUMERICAS: ',len(numericas))
```

```
TOTAL VARIABLES CATEGORICAS:  9
TOTAL VARIABLES NUMERICAS:  14
```

In [12]:
```
dfLimpio.head()
```

Out[12]:

	ID	Total_Credito	Sexo	Estudios	Estado_Civil	Edad	PPSep2005	PPAgo2005	PPJul2005
0	1	20000	2.0	2.0	1.0	24.0	2.0	2.0	-1.0
1	2	120000	2.0	2.0	2.0	26.0	-1.0	2.0	0.0
2	3	90000	2.0	2.0	2.0	34.0	0.0	0.0	0.0
3	4	50000	2.0	2.0	1.0	37.0	0.0	0.0	0.0
4	5	50000	1.0	2.0	1.0	57.0	-1.0	0.0	-1.0

5 rows × 25 columns

In [13]:
```
# Creacion de Imputers para hacer Limpieza de datos
from sklearn.impute import SimpleImputer
# Creamos un Imputer para las variables categoricas las cuales se sustituira po
# esto debido a que los valores de la moda pueden causar sesgos mayores al no e
imputer_numericas = SimpleImputer(strategy='median')

# Asi mismo creo un Imputer para las variables numericas, sustituyendo por la m
imputer_categoricas = SimpleImputer(strategy='most_frequent')
```

Procedemos Eliminando la columna ID ya que no es relevante para nuestro analisis

In [14]:
```
dfLimpio.drop(columns=['ID'], axis=1, inplace=True)
dfLimpio.head()
```

Out[14]:

	Total_Credito	Sexo	Estudios	Estado_Civil	Edad	PPSep2005	PPAgo2005	PPJul2005	PP
0	20000	2.0	2.0	1.0	24.0	2.0	2.0	-1.0	
1	120000	2.0	2.0	2.0	26.0	-1.0	2.0	0.0	
2	90000	2.0	2.0	2.0	34.0	0.0	0.0	0.0	
3	50000	2.0	2.0	1.0	37.0	0.0	0.0	0.0	
4	50000	1.0	2.0	1.0	57.0	-1.0	0.0	-1.0	

5 rows × 24 columns

Ahora eliminamos las filas faltantes en nuestra variable de prediccion Y

In [15]:
```
# Antes de Eliminar, observamos que hay 3 valores faltantes en la columna Y
dfLimpio['Y'].isna().sum()
```

Out[15]: 3

In [16]:
```
dfLimpio.dropna(subset=['Y'], inplace=True)
# Verificamos Los valores han sido Eliminados
print('Valores Nulos en la variable Y: ', dfLimpio['Y'].isna().sum())
```

```
Valores Nulos en la variable Y:  0
```

Finalmente usamos nuestro pipeline de inputaciones con las variables faltantes

In [17]:
```
dfLimpio[['Total_Credito']]
```

Out[17]:

	Total_Credito
0	20000
1	120000
2	90000
3	50000
4	50000
...	...
29995	220000
29996	150000
29997	30000
29998	80000
29999	50000

29997 rows × 1 columns

In [18]:
```
dfLimpio[categoricas] = imputer_categoricas.fit_transform(dfLimpio[categoricas])
dfLimpio[numericas] = imputer_numericas.fit_transform(dfLimpio[numericas])
# Verificamos que no haya mas valores nulos en nuestro conjunto de datos
dfLimpio.isna().sum()
```

Out[18]:
```
Total_Credito    0
Sexo             0
Estudios         0
Estado_Civil     0
Edad             0
PPSep2005        0
PPAgo2005        0
PPJul2005        0
PPJun2005        0
PPMay2005        0
PPAbr2005        0
TRSep2005        0
TRAgo2005        0
TRJul2005        0
TRJun2005        0
TRMay2005        0
TRAbr2005        0
TPPSep2005       0
TPPAgo2005       0
TPPJul2005       0
TPPJun2005       0
TPPMay2005       0
TPPAbr2005       0
Y                0
dtype: int64
```

In [19]:
```
dfLimpio.head()
```

Out[19]:

	Total_Credito	Sexo	Estudios	Estado_Civil	Edad	PPSep2005	PPAgo2005	PPJul2005	PP
0	20000.0	2.0	2.0	1.0	24.0	2.0	2.0	-1.0	
1	120000.0	2.0	2.0	2.0	26.0	-1.0	2.0	0.0	
2	90000.0	2.0	2.0	2.0	34.0	0.0	0.0	0.0	
3	50000.0	2.0	2.0	1.0	37.0	0.0	0.0	0.0	
4	50000.0	1.0	2.0	1.0	57.0	-1.0	0.0	-1.0	

5 rows × 24 columns

EJERCICIO 4

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

In [20]:
```
#Obtenemos la estadistica descriptiva
dfLimpio.describe()
```

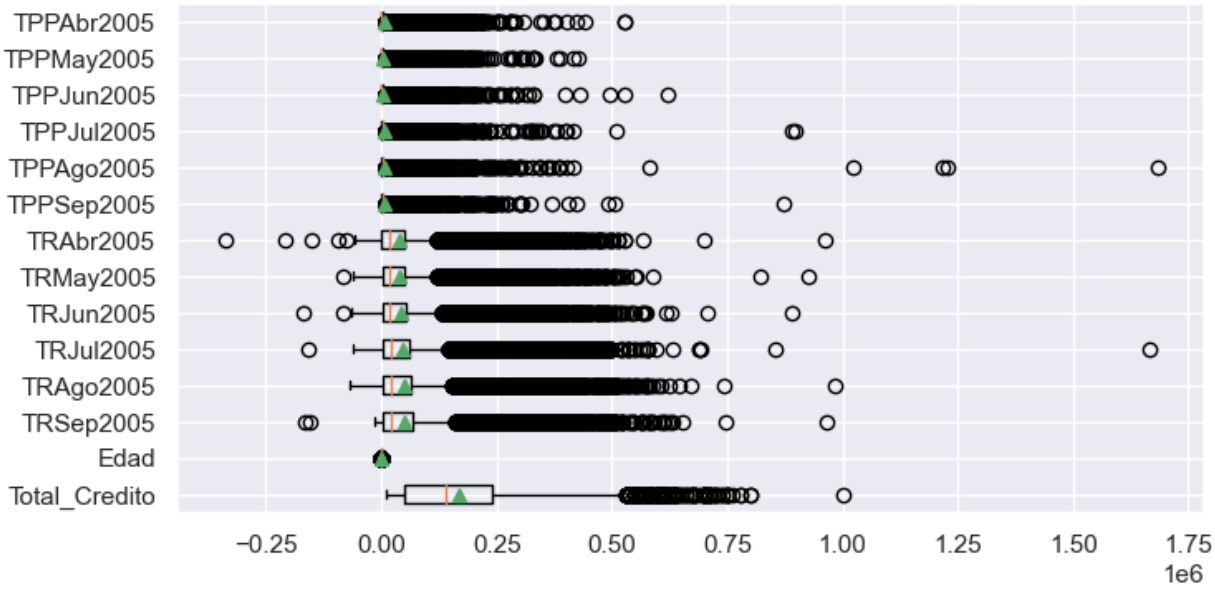
Out[20]:

	Total_Credito	Sexo	Estudios	Estado_Civil	Edad	PPSep200
count	29997.000000	29997.000000	29997.000000	29997.000000	29997.000000	29997.00000
mean	167496.072274	1.603794	1.853085	1.551955	35.483615	-0.01676
std	129748.803871	0.489116	0.790317	0.521963	9.217366	1.12370
min	10000.000000	1.000000	0.000000	0.000000	21.000000	-2.00000
25%	50000.000000	1.000000	1.000000	1.000000	28.000000	-1.00000
50%	140000.000000	2.000000	2.000000	2.000000	34.000000	0.00000
75%	240000.000000	2.000000	2.000000	2.000000	41.000000	0.00000
max	1000000.000000	2.000000	6.000000	3.000000	79.000000	8.00000

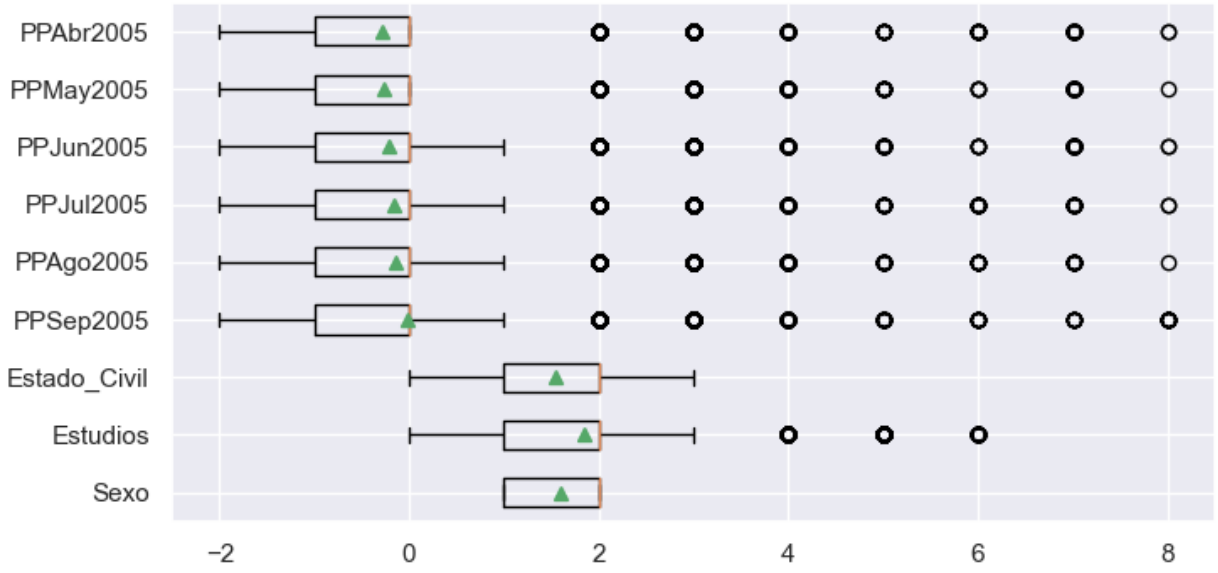
8 rows × 24 columns

Para explicar las medidas de dispersion actuales, procedere a mostrar un grafico de boxPlot, que nos muestra la distribucion actual de nuestros datos.

In [21]:
```
sns.set(rc={'figure.figsize':(8,4)})
plt.boxplot(dfLimpio[numericas], labels=dfLimpio[numericas].columns, showmeans=
plt.show()
```



In [22]:
```
sns.set(rc={'figure.figsize':(8,4)})
plt.boxplot(dfLimpio[categoricas], labels=dfLimpio[categoricas].columns, showme

plt.show()
```



Como podemos observar, Muchos de nuestros datos cuentan con muchos outliers, especialmente aquellas a las que llamamos variables numericas.

Esto se debe a la escala en la que se encuentran nuestros datos. Las variables numericas tienen valores desde 0 hasta los miles, mientras que las variables categoricas nos muestran valores muy pequeños, dentro solamente de el numero de categorias en el atributo.

Por lo que podriamos deducir que quiza sea necesario escalar nuestras variables, pero eso se realizara en el **Ejercicio 6

EJERCICIO 5

Realiza el conteo de las variables categóricas

In [23]:
```
for column in categoricas:
    print('-------------------------------------')
    print(pd.DataFrame(dfLimpio[column].value_counts()))
```

```
--------------------------------------
      Sexo
2.0  18112
1.0  11885
--------------------------------------
     Estudios
2.0     14030
1.0     10584
3.0      4915
5.0       280
4.0       123
6.0        51
0.0        14
--------------------------------------
     Estado_Civil
2.0         15965
1.0         13655
3.0           323
0.0            54
--------------------------------------
      PPSep2005
 0.0      14738
-1.0       5684
 1.0       3688
-2.0       2759
 2.0       2665
 3.0        322
 4.0         76
 5.0         26
 8.0         19
 6.0         11
 7.0          9
--------------------------------------
      PPAgo2005
 0.0      15732
-1.0       6047
 2.0       3925
-2.0       3782
 3.0        326
 4.0         99
 1.0         28
 5.0         25
 7.0         20
 6.0         12
 8.0          1
--------------------------------------
      PPJul2005
 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
------------------------------------
      PPJun2005
 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
------------------------------------
      PPMay2005
 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
------------------------------------
      PPAbr2005
 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
```

Ahora bien como vemos, ya no existen datos de registros faltantes. Sin embargo de acuerdo a la informacion brindade del conjunto de datos hay errores en los valores de los Registros.

- Estudios: Los valores de acuerdo a la guia son: (1 = graduate school; 2 = university; 3 = high school; 4 = others. Se puede observar que hay hasta 7 valores diferentes (0-6) por lo que conviene asumir que para este estudio asumiremos que los valores 0, 5, 6 se sumaran a 4 (Otros) ya que consideramos que de esta manera es como se diseño el modelo mencionado en el estudio.

- Estado Civil: Mismo caso que el anterior, los valores esperados en la guia son: (1 = married; 2 = single; 3 = others) Por lo que 1 y 2 permaneceran intactos, y los valores 0 los añadiremos a el valor 3 (Otros)

Por lo que Aplicaremos la transformación map() de Python para llevar a cabo la agrupación de los registros considerados como 'otros' y que tambien son los de menor frecuencia. Lo

haremos mediante un diccionario. En la notación a:b, el entero "a" es sustituido por "b".

In [24]:
```
# Sustituimos los valores 0, 5 y 6 por un 4 (Otros). Tenemos que mapear los vai
dfLimpio['Estudios'] = dfLimpio['Estudios'].map({0:4, 1:1, 2:2, 3:3, 4:4, 5:4,
# Sustitoimos el valor 0 por un 3(Otros). Tenemos que mapear los valores que no
dfLimpio['Estado_Civil'] = dfLimpio['Estado_Civil'].map({0:3, 1:1, 2:2, 3:3})

# Verificamos de nuevo que los cambios se hayan efectuado.
print('\nNumero de Registros en Estudios:\n',dfLimpio['Estudios'].value_counts(
print('\nNumero de Registros en Estado Civil:\n',dfLimpio['Estado_Civil'].value
```

```
Numero de Registros en Estudios:
 2    14030
1    10584
3     4915
4      468
Name: Estudios, dtype: int64

Numero de Registros en Estado Civil:
 2    15965
1    13655
3      377
Name: Estado_Civil, dtype: int64
```

EJERCICIO 6

Escala los datos, si consideras necesario

Como mencionamos arriba, creemos que si es necesario el escalamiento de las variables numericas, todo esto con la finalidad de que estas se encuentren en un rango similar sin perder sus atributos.

Por cuestiones de practicidad usaremos un StandarScaler para que la media sea igual a 0 siguiendo la distribucion normal estandar, esto nos ayudara mas adelante al obtener los componentes principales PCA

In [25]:
```
# Obtenemos las estadisticas describtivas antes de escalar
dfLimpio[numericas].describe()
```

Out[25]:

	Total_Credito	Edad	TRSep2005	TRAgo2005	TRJul2005	TR
count	29997.000000	29997.000000	29997.000000	29997.000000	2.999700e+04	29997
mean	167496.072274	35.483615	51228.534837	49182.573924	4.701637e+04	43265
std	129748.803871	9.217366	73637.071379	71175.530729	6.935104e+04	64334
min	10000.000000	21.000000	-165580.000000	-69777.000000	-1.572640e+05	-170000
25%	50000.000000	28.000000	3566.000000	2986.000000	2.671000e+03	2332
50%	140000.000000	34.000000	22385.500000	21205.000000	2.008900e+04	19052
75%	240000.000000	41.000000	67094.000000	64013.000000	6.016700e+04	54515
max	1000000.000000	79.000000	964511.000000	983931.000000	1.664089e+06	891586

In [26]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
dfLimpio[numericas] = scaler.fit_transform(dfLimpio[numericas])
dfLimpio.head()
```

Out[26]:

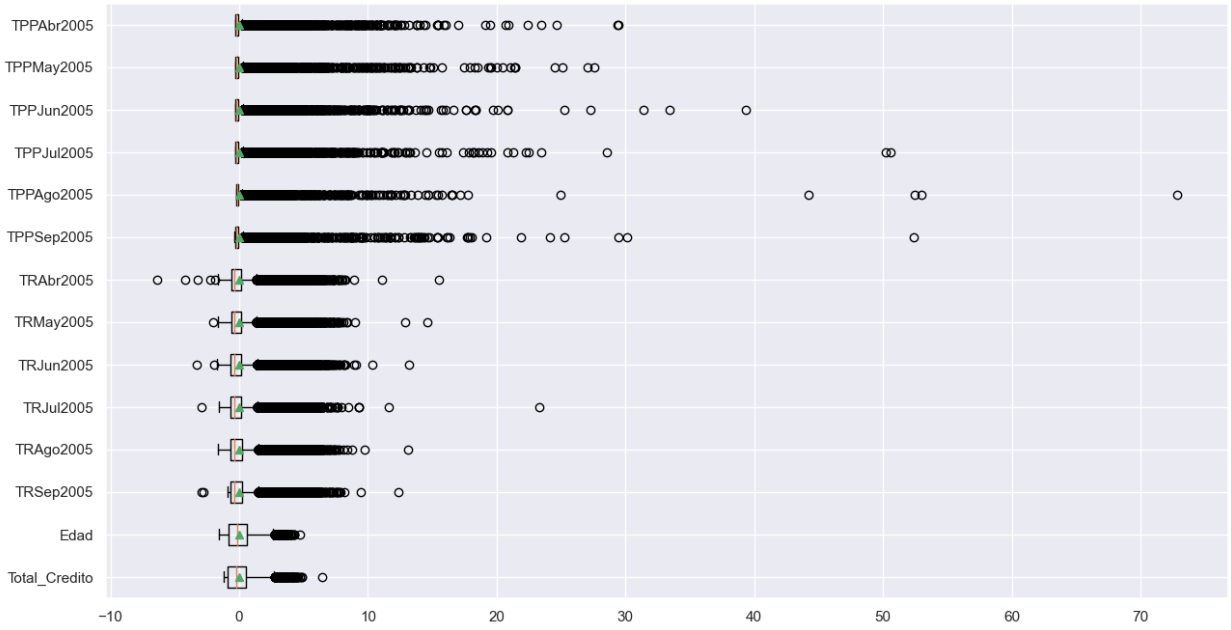
	Total_Credito	Sexo	Estudios	Estado_Civil	Edad	PPSep2005	PPAgo2005	PPJul2005
0	-1.136801	2.0	2	1	-1.245888	2.0	2.0	-1.0
1	-0.366068	2.0	2	2	-1.028903	-1.0	2.0	0.0
2	-0.597288	2.0	2	2	-0.160961	0.0	0.0	0.0
3	-0.905581	2.0	2	1	0.164517	0.0	0.0	0.0
4	-0.905581	1.0	2	1	2.334370	-1.0	0.0	-1.0

5 rows × 24 columns

Como podemos observar las variables Numericas han sido escaladas. Esto nos ayuda a obtener una mejor distribucion estandar en nuestros datos.

Lo podemos observar obteniendo las estadisticas describtivas de nuestro conjunto de datos o visualizando nuestro diagrama de caja con este nuevo escalamiento.

In [27]:
```
sns.set(rc={'figure.figsize':(15,8)})
plt.boxplot(dfLimpio[numericas], labels=dfLimpio[numericas].columns, showmeans=
plt.show()
```



Nuestros Outliers continuan ahi pero ahora las escalas son mas cercanas. Lo que nos traera un analisis mas eficiente que considere de manera optima las escalas de nuestras variables

EJERCICIO 7

Reduce las dimensiones con PCA, si consideras necesario.

- Indica la varianza de los datos explicada por cada componente seleccionado. Para actividades de exploración de los datos la varianza > 70%
- Indica la importancia de las variables en cada componente

Para comenzar como bien sabemos PCA funciona solo para variables continuas por lo que en nuestro caso, trabajaremos solo con las siguientes variables:

- Total_Credito
- Edad
- Total del Recibo
- Total de Pagos Pasados

Por lo que creare una copia de nuestro data frame con solo estas variables de interes. Tambien estaremos dejando fuera a la variable Y de prediccion.

In [28]:
```
dfPCA = dfLimpio.copy().drop(['Sexo', 'Estudios', 'Estado_Civil', 'PPSep2005','
dfPCA.head()
```

Out[28]:

	Total_Credito	Edad	TRSep2005	TRAgo2005	TRJul2005	TRJun2005	TRMay2005	TI
0	-1.136801	-1.245888	-0.642561	-0.647432	-0.668024	-0.672527	-0.663098	
1	-0.366068	-1.028903	-0.659279	-0.666779	-0.639285	-0.621667	-0.606269	
2	-0.597288	-0.160961	-0.298625	-0.493936	-0.482443	-0.449766	-0.417229	
3	-0.905581	0.164517	-0.057561	-0.013342	0.032799	-0.232414	-0.186772	
4	-0.905581	2.334370	-0.578679	-0.611352	-0.161231	-0.347035	-0.348179	

In [29]:
```
columnas = ['Total_Credito','Edad','TRSep2005', 'TRAgo2005','TRJul2005','TRJun2

summary = {
    "Varianzas: ": dfPCA[columnas].var().round(2),
    "Valor Min: ": dfPCA[columnas].min().round(2),
    "valor Max: ": dfPCA[columnas].max().round(2)
}
pd.DataFrame(summary, index = columnas).transpose()
```

Out[29]:

	Total_Credito	Edad	TRSep2005	TRAgo2005	TRJul2005	TRJun2005	TRMay2005
Varianzas:	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Valor Min:	-1.21	-1.57	-2.94	-1.67	-2.95	-3.32	-2.00
valor Max:	6.42	4.72	12.40	13.13	23.32	13.19	14.59

De la tabla anterior, podemos observar como los datos estan distribuidos entre las variables del Total_credito, total del recibo y Edad.

La magnitud de las variables despues de aplicar nuestro escalamiento ya es algo con lo que se puede trabajar ya que los rangos estan muy cercanos entre todas las variables.

Para comprobar esto buscaremos el % total de las varianzas de estas variables

A continuacion, veremos como se distribuye la varianza en cada variable que someteremos a estudio.

In [30]:
```
#Varianza total de todas las variables
varianzaTotal = dfPCA.var().sum()

#Calculamos la varianza para cada variable que estamos estudiando = df_num.calc
varVarianzas = []
for i, col in enumerate(columnas):
    varVarianzas.append({
        'Columna': col,
        'Varianza': dfPCA[col].var()
    })


#Porcentaje que representa la varianza de cada variable respecto al total.
print('Varianza Total: \t', varianzaTotal.round(2) )

sumaPorcentajes = 0
```

```
for var in varVarianzas:
    sumaPorcentajes += ((var['Varianza']/varianzaTotal)*100).round(2)
    print('Varianza ' + var['Columna'] + ': \t' ,  ((var['Varianza']/varianzaTo

print('Porcentaje de Varianza de Variables en Analisis: ', sumaPorcentajes)
```

```
Varianza Total:          14.0
Varianza Total_Credito:          7.14 %
Varianza Edad:   7.14 %
Varianza TRSep2005:      7.14 %
Varianza TRAgo2005:      7.14 %
Varianza TRJul2005:      7.14 %
Varianza TRJun2005:      7.14 %
Varianza TRMay2005:      7.14 %
Varianza TRAbr2005:      7.14 %
Porcentaje de Varianza de Variables en Analisis:  57.12
```

Con esto podemos comprobar que Ya contamos con un conjunto de datos de analisis balanceado. Ahora procederemos a aplicar nuestro PCA

Ahora, todas nuestras variables tienen el mismo porcentaje de varianza respecto al total.

En conjunto representan casi el **60% de la varianza.** de nuestro conjunto de datos

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

In [31]:
```
from sklearn.decomposition import PCA # importamos la libreria PCA de sklearn

pcs = PCA() # Instanciamos pcs con una nueva PCA()

pcs_t = pcs.fit_transform(dfPCA) # Hacemos fit y transform a nuestro conjunto d

pcsSummarydf = pd.DataFrame({
    '% Varianza explicada': np.round(pcs.explained_variance_ratio_,2) * 100, #
    '% Varianza acumulada': np.cumsum(pcs.explained_variance_ratio_) * 100 # Ob
})

nombreFila = [f'PC{i + 1}' for i in range(len(dfPCA.columns))]
pcsSummarydf.index = nombreFila

pcsSummarydf
```

Out[31]:

	% Varianza explicada	% Varianza acumulada
PC1	42.0	42.279613
PC2	12.0	54.537525
PC3	7.0	62.002910
PC4	7.0	68.609432
PC5	6.0	74.920995
PC6	6.0	81.154272
PC7	6.0	86.722351
PC8	5.0	91.912227
PC9	5.0	96.963529
PC10	2.0	98.852246
PC11	1.0	99.359496
PC12	0.0	99.652872
PC13	0.0	99.833989
PC14	0.0	100.000000

La variable pcsSummary a creado componentes de informacion que se derivan de las diferentes variables de nuestro conjunto de datos.

Nuestro PCA ha trabajado para generar un conjunto de componentes que mantengan el 100% de la informacion original, solo distribuida de manera diferente.

Para confirmar esto calcularemos la varianza total de nuestras variables originales, usando nuestro dataframe escalado y el de los componentes.

In [32]:
```
pcsdf = pd.DataFrame(pcs_t, columns = nombreFila)

print("Varianza total variables originales: ", dfPCA.var().sum().round(5))
print("Varianza total de los componentes: ", pcsdf.var().sum().round(5))
```

```
Varianza total variables originales:  14.00047
Varianza total de los componentes:  14.00047
```

En pcsSummarydf, La varianza acumulada nos indica que tanto porcentaje de la varianza se explica por cada componente.

Podriamos decir que el 86 % de la varianza total, esta explicada por los primeros 7 componentes o ampliando un poco casi el 97 se explica con los primeros 9 componentes.

Por lo tanto, nuestra seleccion anterior parece ser un buen punto de partida. **Los primeros 8 componentes explicariamos el 91% de la varianza total**

- Indica la importancia de las variables en cada componente

A continuacion, procedimos a visualizar la composicion de cada uno de los primeros 8 componentes.

In [33]:
```
summaryaux = {
    'Desviacion Estandar': np.sqrt(pcs.explained_variance_),
    'Proporcion de Varianza': pcs.explained_variance_ratio_,
    'Proporcion acumulativa': np.cumsum(pcs.explained_variance_ratio_)
}
pcsSummary = pd.DataFrame(summaryaux)[0:8].transpose()
pcsSummary = pcsSummary.round(2)
pcsSummary.columns = pcsdf.columns[0:8]
pcsSummary
```

Out[33]:

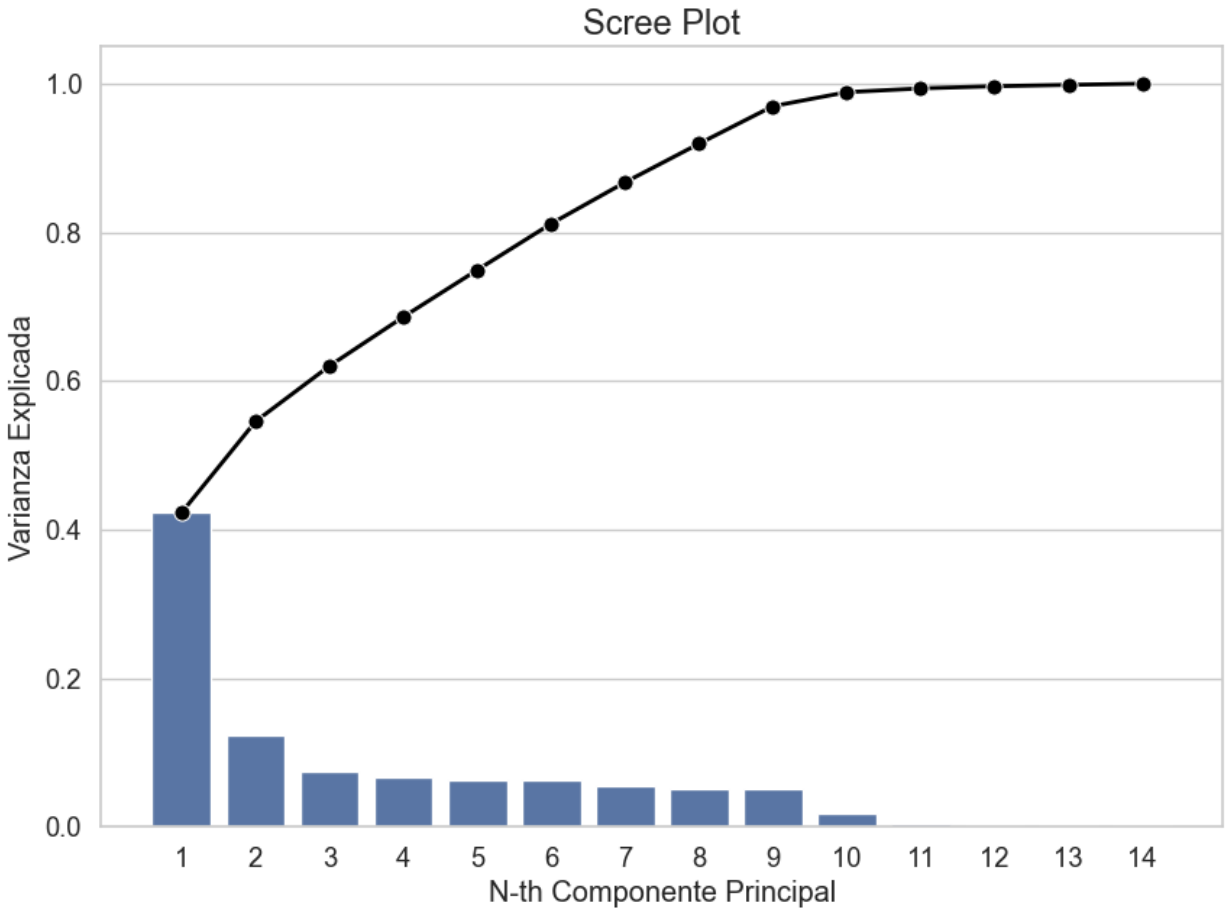
	PC1	PC2	PC3	PC4	PC5	PC6	PC7	PC8
Desviacion Estandar	2.43	1.31	1.02	0.96	0.94	0.93	0.88	0.85
Proporcion de Varianza	0.42	0.12	0.07	0.07	0.06	0.06	0.06	0.05
Proporcion acumulativa	0.42	0.55	0.62	0.69	0.75	0.81	0.87	0.92

A continuacion analizaremos la Desviacion estandar, la proporcion de la varianza y la varianza acumulativa. Para descubrir la magnitud de cada componente principal

Procederemos pues a crear nuestro SCREE PLOT para visualizar los eigenvalues de mayor a menor y visualizar la magnitud de estos.

In [34]:
```
# Generamos un arreglo con el numero de los componentes
pc_components = np.arange(pcs.n_components_) + 1

# El Acumulado del radio de la varianza en pcs
cusm = np.cumsum(pcs.explained_variance_ratio_)

# La variancia por cada componente principal
vartio = pcs.explained_variance_ratio_

scree = sns.set(style = 'whitegrid', font_scale=1.2)
fig, ax = plt.subplots(figsize = (10, 7))
scree = sns.barplot(x = pc_components, y = vartio, color='b')
scree = sns.lineplot(x = pc_components - 1,
                     y = cusm,
                     color = 'black',
                     linestyle = '-',
                     linewidth = 2,
                     marker = 'o',
                     markersize = 8)
scree.set_title('Scree Plot', fontsize = 17)
scree.set(xlabel='N-th Componente Principal', ylabel='Varianza Explicada')
```

Out[34]:
```
[Text(0.5, 0, 'N-th Componente Principal'), Text(0, 0.5, 'Varianza Explicad
a')]
```



A continuacion haremos una comparacion con las variables originales escaladas para ver cuantas necesitariamos para explicar el **91%** de la varianza a comparacion de las 8 que se necesitan para explicar el mismo porcentaje con los componentes de PCA

In [35]:
```
total_var = dfPCA.var().sum()

pd.DataFrame({
    "Porcentaje Varianza": (dfPCA.var()/ total_var) * 100,
    "Porcentaje Varianza Acumulado": (dfPCA.var().cumsum() / total_var) * 100
})
```

Out[35]:

	Porcentaje Varianza	Porcentaje Varianza Acumulado
Total_Credito	7.142857	7.142857
Edad	7.142857	14.285714
TRSep2005	7.142857	21.428571
TRAgo2005	7.142857	28.571429
TRJul2005	7.142857	35.714286
TRJun2005	7.142857	42.857143
TRMay2005	7.142857	50.000000
TRAbr2005	7.142857	57.142857
TPPSep2005	7.142857	64.285714
TPPAgo2005	7.142857	71.428571
TPPJul2005	7.142857	78.571429
TPPJun2005	7.142857	85.714286
TPPMay2005	7.142857	92.857143
TPPAbr2005	7.142857	100.000000

Como podemos observar, necesitariamos utilizar 13 variables del conjunto original para explicar el mismo porcentaje de varianza de solo 8 componentes de PCS. Por lo que podemos concluir que existe una reduccion de 5 atributos al usar PCA. Por lo que tambien podemos decir que se reduce alrededor del 30% de los atributos originales al usar solo 8 componentes para explicar este 92% de varianza.

A continuacion, procedemos a analizar los pesos de cada atributo en cada uno de los 8 componentes elegidos:

In [36]:
```
compsdf = pd.DataFrame(
    pcs.components_.round(2), # Traemos los pesos de cada uno de los componente
    columns = pcsdf.columns,
    index = dfPCA.columns) #Nombramos las filas como las columnas para una mejo

# Traemos los primeros 8 componentes solamente.
compsdf.iloc[:,:8]
```

Out[36]:

	PC1	PC2	PC3	PC4	PC5	PC6	PC7	PC8
Total_Credito	0.17	0.03	0.37	0.38	0.39	0.39	0.39	0.38
Edad	0.30	0.07	-0.19	-0.17	-0.13	-0.12	-0.11	-0.09
TRSep2005	-0.38	-0.87	-0.03	-0.00	0.03	0.03	0.03	0.02
TRAgo2005	-0.20	0.34	-0.06	0.01	0.06	0.07	0.04	-0.07
TRJul2005	0.03	-0.04	0.04	0.08	0.11	0.03	-0.11	-0.16
TRJun2005	-0.08	0.07	-0.04	-0.03	0.10	0.01	-0.10	0.07
TRMay2005	0.11	-0.08	0.01	-0.03	-0.12	0.13	-0.01	0.01
TRAbr2005	-0.05	0.03	0.01	-0.14	0.09	0.04	0.05	0.00
TPPSep2005	-0.82	0.33	0.01	0.02	-0.02	0.02	0.02	0.06
TPPAgo2005	-0.03	-0.01	0.57	0.39	0.12	-0.21	-0.42	-0.49
TPPJul2005	-0.01	0.00	0.42	0.04	-0.48	-0.52	0.07	0.51
TPPJun2005	0.02	-0.00	-0.43	0.34	0.50	-0.49	-0.25	0.34
TPPMay2005	-0.00	0.00	-0.18	0.33	-0.09	-0.36	0.72	-0.43
TPPAbr2005	0.00	0.00	-0.32	0.65	-0.53	0.35	-0.23	0.07

En esta tabla anterior, observamos que variable tiene mas peso para cada componente

Nota: Nos interesa el peso en valor absoluto por lo que a continuacion mostramos los maximos de cada componente por filas.

In [37]:
```
# Usamos idxmax para obtener el maximo valor por fila.
# Usamos abs para que sea el valor absoluto
pd.DataFrame(data = {'PC #': compsdf.iloc[:,:8].abs().idxmax().index,
                     'COLUMNA': compsdf.iloc[:,:8].abs().idxmax().values,
                    'VALOR': compsdf.iloc[:,:8].abs().max()})
```

Out[37]:

	PC #	COLUMNA	VALOR
PC1	PC1	TPPSep2005	0.82
PC2	PC2	TRSep2005	0.87
PC3	PC3	TPPAgo2005	0.57
PC4	PC4	TPPAbr2005	0.65
PC5	PC5	TPPAbr2005	0.53
PC6	PC6	TPPJul2005	0.52
PC7	PC7	TPPMay2005	0.72
PC8	PC8	TPPJul2005	0.51

En otras palabras, las variables con mas peso en cada componente son:

- Para PC1 los datos en Total de Pagos Pasados de Sep2005 tiene el mayor peso con el 82%

- Para PC2 los datos en Total de Recibo de Sep2005 tiene el mayor peso con el 87%
- Para PC3 los datos en Total de Pagos Pasados Ago2005 tiene el mayor peso con el 57%
- Para PC4 los datos en Total de Pagos Pasados Abr2005 tiene el mayor peso con el 65%
- Para PC5 los datos en Total de Pagos Pasados Abr2005 tiene el mayor peso con el 53%
- Para PC6 los datos en Total de Pagos Pasados Jul2005 tiene el mayor peso con el 52%
- Para PC7 los datos en Total de Pagos Pasados May2005 tiene el mayor peso con el 72%
- Para PC8 los datos en Total de Pagos Pasados Jul2005 tiene el mayor peso con el 51%

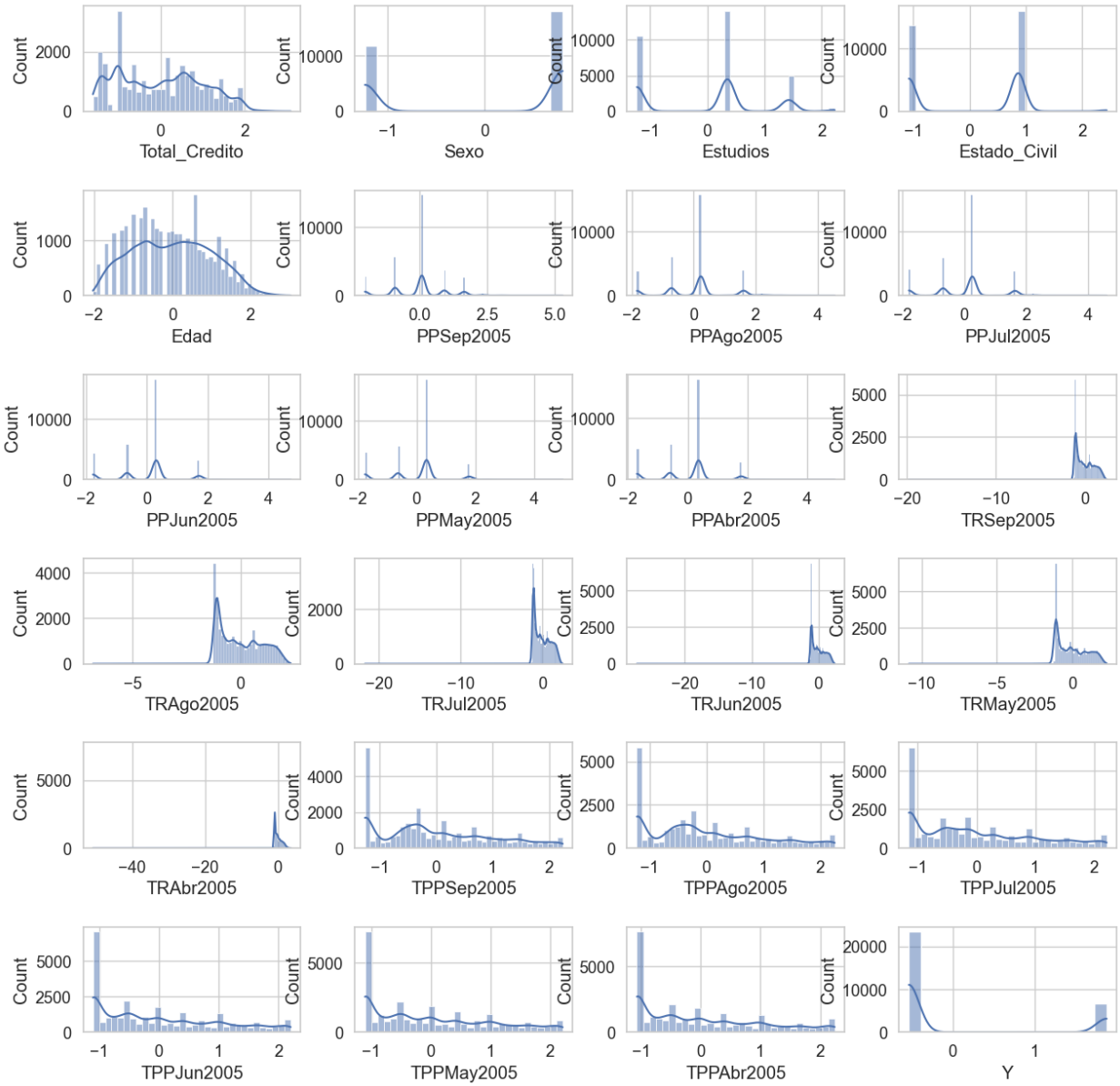
EJERCICIO 8

Elabora los histogramas de los atributos para visualizar su distribución

Antes de graficar en histogramas, escalare los datos usando un PowerTransform para tener datos mas parecidos a una forma Gaussiana y sea mas facil visualmente ver su distribucion

In [38]:
```
from sklearn.preprocessing import PowerTransformer

pt = PowerTransformer()
dfScaled = pd.DataFrame(pt.fit_transform(dfLimpio), columns=dfLimpio.columns)

fig, ax  = plt.subplots(6, 4, figsize=(15, 15))
plt.suptitle('Histogramas de Atributos')
plt.subplots_adjust(hspace=0.75, wspace=0.25)
for i, col in enumerate(dfScaled.columns):
    sns.histplot(data=dfScaled, x=col, ax=ax[i//4, i%4], kde=True).set(xlabel=c

plt.show()
```

Histogramas de Atributos



En estos histogramas podemos observar como se distribuyen los datos de cada variable. Como podemos observar para las variables categoricas, los valores en general estan centrados a la categoria a la que fueron escalados. Mientras que para las variables numericas observamos que hay diferentes tipos de distribucion, algunas son mas normalizadas otras sesgadas a la izquierda o derecha con rangos muy altos lo que indica un numero alto de outliers.

EJERCICIO 9 y 10

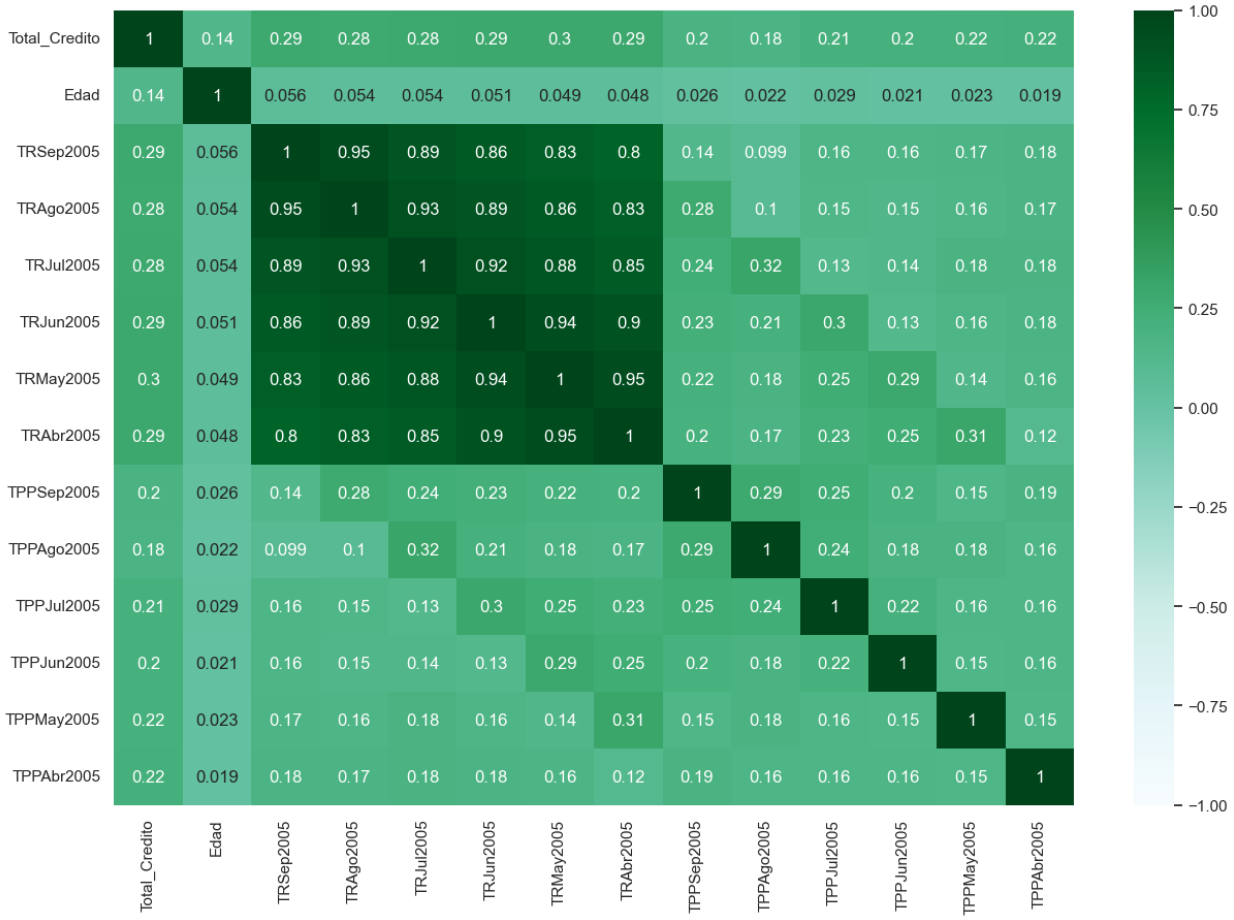
- Realiza la visualización de los datos usando por lo menos 3 gráficos que consideres adecuados: plot, scatter, jointplot, boxplot, areaplot, pie chart, pairplot, bar chart, etc.

- 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.

HEATMAP GRAPH DE CORRELACIONES DE VARIABLES PARA PCA

In [39]:
```
correlaciones = dfPCA.corr()

sns.set(rc = {'figure.figsize': (15,10)})
sns.heatmap(correlaciones, vmin = -1, vmax=1, cmap="BuGn", annot=True)
```

Out[39]: <AxesSubplot: >



Mediante este heatmap podemos observar 2 cosas:

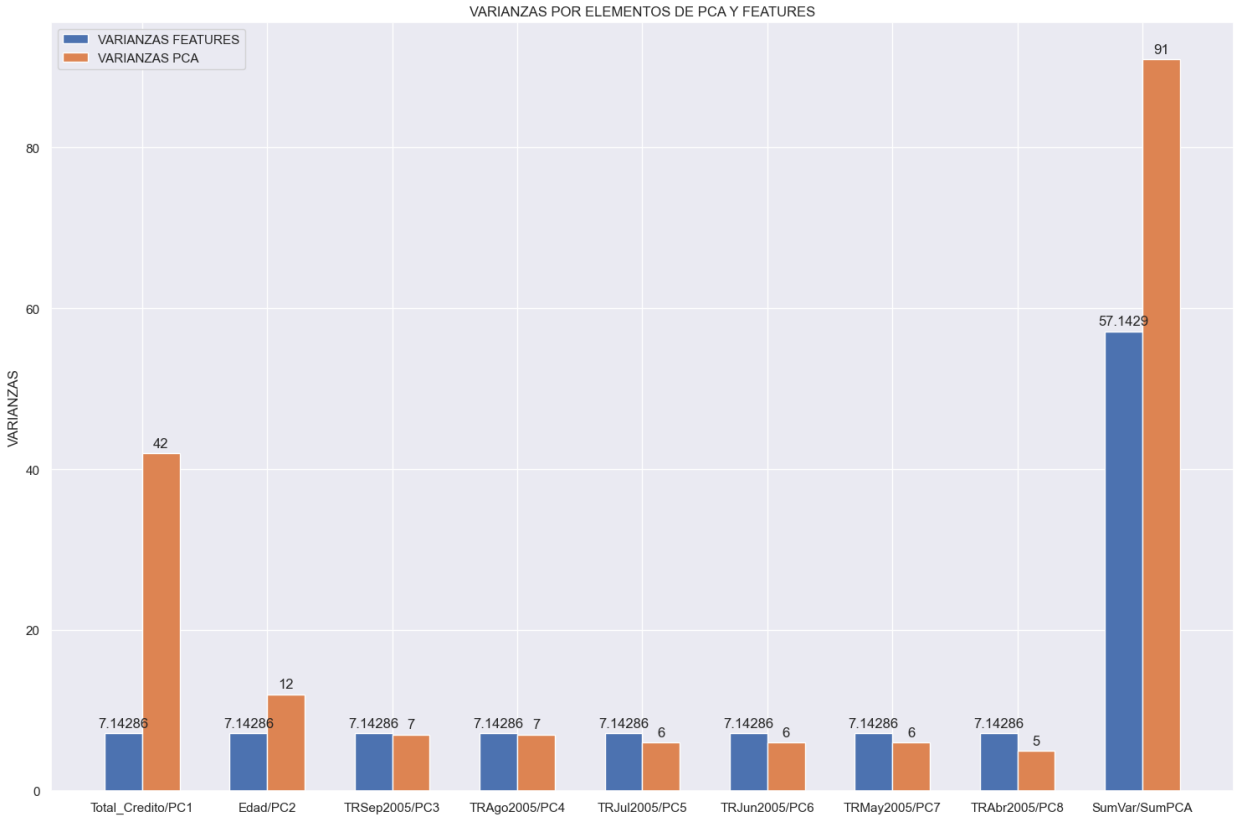
- Las columnas positivamente relacionadas son aquellas con un color verde obscuro. Mientras mas obscuro es el color verde, mas correlacionadas se encuentran. En nuestro conjunto de datos, podemos observar que las variables de Total del Recibo estan todas ellas correlacionadas fuertemente. Adicionalmente la edad y el total de credito entre 25 y 50% de correlacion con estas variables respectivamente, por lo que vamos a incluirlas en nuestro analisis
- Las columnas negativamente relacionadas tienen el color verde mas claro, para nuestro dataframe parece no haber correlaciones negativas.

Estas observaciones se ajustan a nuestro modelo PCA en el cual pudimos obtener que las variables con mas alto nivel de correlacion son las de Total del Recibo del año 2005 junto con el Total de credito y la edad de los sujetos de estudio.

BARCHART MOSTRANDO DIFERENCIAS ENTRE VARIANZAS DE PCA Y FEATURES

In [40]:
```
porcentaje = np.arange(0,100,14)
varianzasFeatures = np.array(((dfPCA.var()/ total_var) * 100))[0:8]
varianzasPCA = (np.round(pcs.explained_variance_ratio_,2) * 100)[0:8]

varianzasFeatures = np.append(varianzasFeatures, varianzasFeatures.sum())
varianzasPCA = np.append(varianzasPCA, varianzasPCA.sum())
```

In [41]:
```
labels = ['Total_Credito/PC1', 'Edad/PC2', 'TRSep2005/PC3', 'TRAgo2005/PC4', 'T

x = np.arange(len(labels))
width = 0.30
fig, ax = plt.subplots(figsize=(15, 10))

bars1 = ax.bar(x - width/2, varianzasFeatures, width, label='VARIANZAS FEATURES
bars2 = ax.bar(x + width/2, varianzasPCA, width, label='VARIANZAS PCA')

ax.set_ylabel('VARIANZAS')
ax.set_title('VARIANZAS POR ELEMENTOS DE PCA Y FEATURES')
ax.set_xticks(x, labels)
ax.legend()

ax.bar_label(bars1, padding=3)
ax.bar_label(bars2, padding=3)


fig.tight_layout()

plt.show()
```



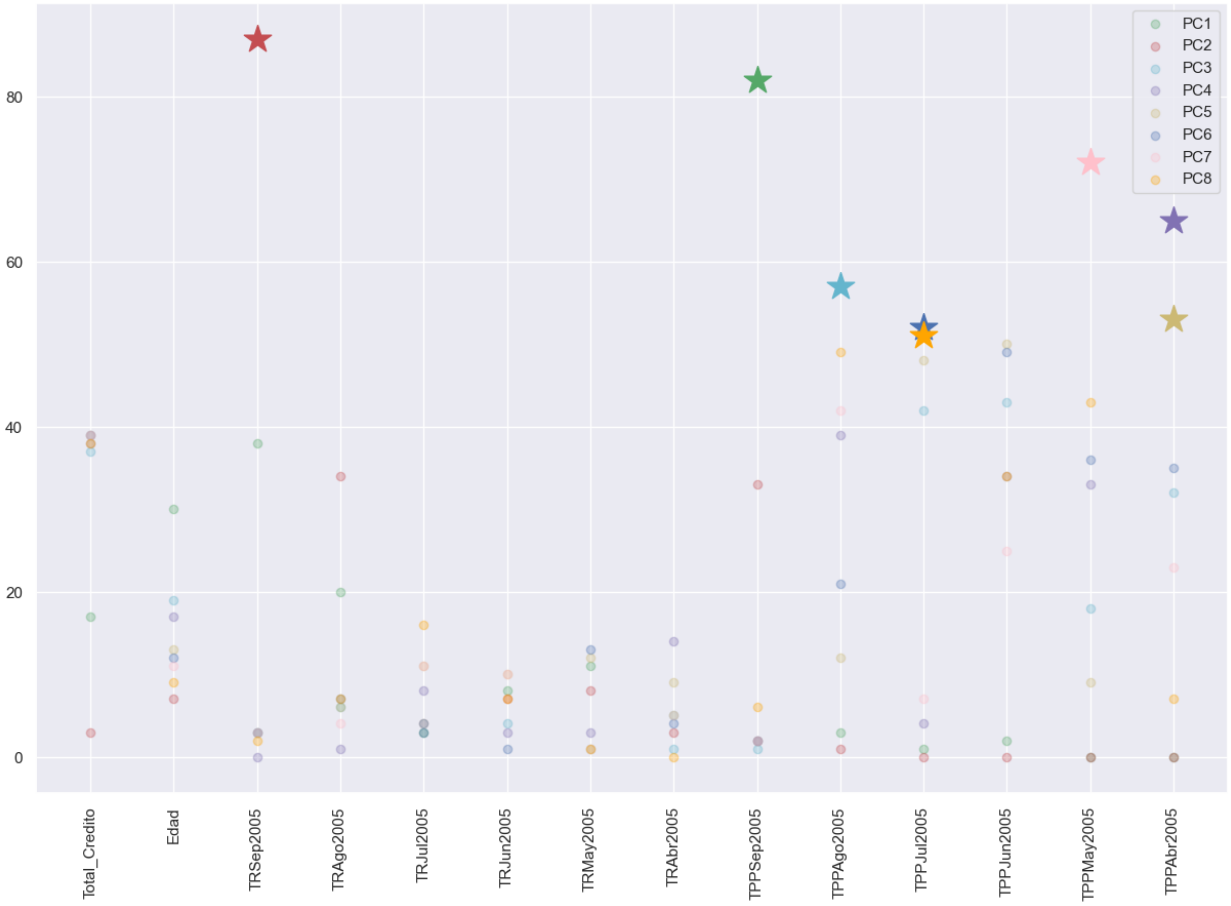
En este barchart podemos observar el porcentaje de varianza en nuestro conjunto de datos (hasta la variable 8) y nuestros 8 componentes principales. Como sabemos, necesitariamos utilizar 13 variables del conjunto original para explicar el mismo porcentaje de varianza de solo 8 componentes de PCS.

En este grafico observamos como con tan solo 8 componentes principales, explicamos el 91% de la varianza en los datos, mientras que con el mismo numero de features solo el 57%. Esto nos indica que nuestro modelo puede utilizar una transformacion a componentes principales para obtener ciertas conclusiones, sin perder valores estadisticos significativos, reduciendo el numero de variables y generando un modelo mas sencillo de trabajar con el.

SCATTER PLOT VISUALIZANDO LA MAGNITUD DE CADA VARIABLE EN CADA COMPONENTE PRINCIPAL

In [42]:
```
variables =  compsdf.index
pcNames = compsdf.columns[0:8]
fig, ax = plt.subplots()
colors = ['g', 'r', 'c', 'm', 'y', 'b', 'pink', 'orange']
for i, pcName in enumerate(pcNames):
    plt.xticks(rotation='vertical')
    maxVal = compsdf[[pcName]].abs().max().values[0]
    xval = compsdf.index[compsdf[pcName].abs() == maxVal]
    ax.scatter(compsdf.index, compsdf[[pcName]].abs() * 100, label = pcName, al
    ax.scatter( xval , maxVal * 100, color=colors[i], marker='*', s=400 )

ax.legend()
```

```
ax.grid(True)
plt.show()
```



En este scatter plot podemos observar de manera grafica la distribucion de pesos de cada variable en cada componente principal.Resaltando a su vez con un marker diferente la caracteristica con mayor peso en el componente.

Este grafico nos ayuda a comprender mejor, como funciona PCA obteniendo diferentes componentes principales usando las mismas variables de nuestro conjunto de datos pero transformadas, respetando los valores estadisticos originales como la varianza.