Modelo de Detección de fraude Bancario

May 27, 2021

0.1 MODELO PARA LA DETECCION DE FRAUDE BANCARIO

- Modelo desarrollado con Deep learning para detectar casos de fraude interbancario

Los pasos a seguir son:

- Obtencion de los datos
- Preparacion de datos
  - Exploracion de datos
  - Transformacion de datos
- Construccion del modelo
- Evaluacion del modelo

Ciclo del machine learning aplicado a este proyecto

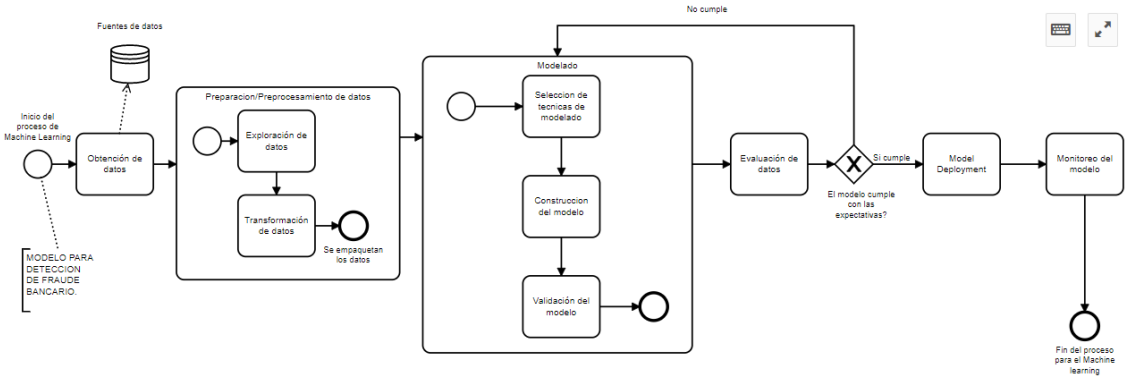


Diagrama de procesos desarrollado en BPMN

0.1.1 Obtencion de los datos

- Analizar las fuentes de datos
- Colecta de datos
- Integracion de las fuentes de datos

Como primer paso importamos las librerias para obtener los datos

[1]:
```
from tensorflow.keras.utils import get_file
import os

path_file=get_file(
    fname=os.path.join(os.getcwd(),"creditcard.csv"),
```

```
    origin="https://storage.googleapis.com/download.tensorflow.org/data/
↪creditcard.csv",
    )
```

[2]:
```
%matplotlib inline

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns; sns.set()
```

Establecer una semilla (seed) para que los resultados sean reproducibles

[3]:
```
seed=42
np.random.seed(seed)
```

Procedemos con la carga de los datos

[4]:
```
data=pd.read_csv(path_file)
```

[5]:
```
data.head()
```

[5]:
```
   Time        V1        V2        V3        V4        V5        V6        V7  \
0   0.0 -1.359807 -0.072781  2.536347  1.378155 -0.338321  0.462388  0.239599
1   0.0  1.191857  0.266151  0.166480  0.448154  0.060018 -0.082361 -0.078803
2   1.0 -1.358354 -1.340163  1.773209  0.379780 -0.503198  1.800499  0.791461
3   1.0 -0.966272 -0.185226  1.792993 -0.863291 -0.010309  1.247203  0.237609
4   2.0 -1.158233  0.877737  1.548718  0.403034 -0.407193  0.095921  0.592941

         V8        V9  …       V21       V22       V23       V24       V25  \
0  0.098698  0.363787  … -0.018307  0.277838 -0.110474  0.066928  0.128539
1  0.085102 -0.255425  … -0.225775 -0.638672  0.101288 -0.339846  0.167170
2  0.247676 -1.514654  …  0.247998  0.771679  0.909412 -0.689281 -0.327642
3  0.377436 -1.387024  … -0.108300  0.005274 -0.190321 -1.175575  0.647376
4 -0.270533  0.817739  … -0.009431  0.798278 -0.137458  0.141267 -0.206010

        V26       V27       V28  Amount  Class
0 -0.189115  0.133558 -0.021053  149.62      0
1  0.125895 -0.008983  0.014724    2.69      0
2 -0.139097 -0.055353 -0.059752  378.66      0
3 -0.221929  0.062723  0.061458  123.50      0
4  0.502292  0.219422  0.215153   69.99      0

[5 rows x 31 columns]
```

0.1.2 Preparacion de datos

- Exploracion de datos

- Transformacion de datos (Data wrangling, data cleaning, feature selection y feature extraction)

En esta seccion primero exploramos las informacion de los datos

[6]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 284807 entries, 0 to 284806
Data columns (total 31 columns):
 #   Column  Non-Null Count   Dtype  
---  ------  --------------   -----  
 0   Time    284807 non-null  float64
 1   V1      284807 non-null  float64
 2   V2      284807 non-null  float64
 3   V3      284807 non-null  float64
 4   V4      284807 non-null  float64
 5   V5      284807 non-null  float64
 6   V6      284807 non-null  float64
 7   V7      284807 non-null  float64
 8   V8      284807 non-null  float64
 9   V9      284807 non-null  float64
 10  V10     284807 non-null  float64
 11  V11     284807 non-null  float64
 12  V12     284807 non-null  float64
 13  V13     284807 non-null  float64
 14  V14     284807 non-null  float64
 15  V15     284807 non-null  float64
 16  V16     284807 non-null  float64
 17  V17     284807 non-null  float64
 18  V18     284807 non-null  float64
 19  V19     284807 non-null  float64
 20  V20     284807 non-null  float64
 21  V21     284807 non-null  float64
 22  V22     284807 non-null  float64
 23  V23     284807 non-null  float64
 24  V24     284807 non-null  float64
 25  V25     284807 non-null  float64
 26  V26     284807 non-null  float64
 27  V27     284807 non-null  float64
 28  V28     284807 non-null  float64
 29  Amount  284807 non-null  float64
 30  Class   284807 non-null  int64  
dtypes: float64(30), int64(1)
memory usage: 67.4 MB
```

Observamos los datos de *float* e *int* para la variable objetivo

- Ahora vamos a obtener los estadisticos de los datos rapidamente

[7]:
```
data.describe().T
```

[7]:

	count	mean	std	min	25%	\
Time	284807.0	9.481386e+04	47488.145955	0.000000	54201.500000	
V1	284807.0	3.919560e-15	1.958696	-56.407510	-0.920373	
V2	284807.0	5.688174e-16	1.651309	-72.715728	-0.598550	
V3	284807.0	-8.769071e-15	1.516255	-48.325589	-0.890365	
V4	284807.0	2.782312e-15	1.415869	-5.683171	-0.848640	
V5	284807.0	-1.552563e-15	1.380247	-113.743307	-0.691597	
V6	284807.0	2.010663e-15	1.332271	-26.160506	-0.768296	
V7	284807.0	-1.694249e-15	1.237094	-43.557242	-0.554076	
V8	284807.0	-1.927028e-16	1.194353	-73.216718	-0.208630	
V9	284807.0	-3.137024e-15	1.098632	-13.434066	-0.643098	
V10	284807.0	1.768627e-15	1.088850	-24.588262	-0.535426	
V11	284807.0	9.170318e-16	1.020713	-4.797473	-0.762494	
V12	284807.0	-1.810658e-15	0.999201	-18.683715	-0.405571	
V13	284807.0	1.693438e-15	0.995274	-5.791881	-0.648539	
V14	284807.0	1.479045e-15	0.958596	-19.214325	-0.425574	
V15	284807.0	3.482336e-15	0.915316	-4.498945	-0.582884	
V16	284807.0	1.392007e-15	0.876253	-14.129855	-0.468037	
V17	284807.0	-7.528491e-16	0.849337	-25.162799	-0.483748	
V18	284807.0	4.328772e-16	0.838176	-9.498746	-0.498850	
V19	284807.0	9.049732e-16	0.814041	-7.213527	-0.456299	
V20	284807.0	5.085503e-16	0.770925	-54.497720	-0.211721	
V21	284807.0	1.537294e-16	0.734524	-34.830382	-0.228395	
V22	284807.0	7.959909e-16	0.725702	-10.933144	-0.542350	
V23	284807.0	5.367590e-16	0.624460	-44.807735	-0.161846	
V24	284807.0	4.458112e-15	0.605647	-2.836627	-0.354586	
V25	284807.0	1.453003e-15	0.521278	-10.295397	-0.317145	
V26	284807.0	1.699104e-15	0.482227	-2.604551	-0.326984	
V27	284807.0	-3.660161e-16	0.403632	-22.565679	-0.070840	
V28	284807.0	-1.206049e-16	0.330083	-15.430084	-0.052960	
Amount	284807.0	8.834962e+01	250.120109	0.000000	5.600000	
Class	284807.0	1.727486e-03	0.041527	0.000000	0.000000	

	50%	75%	max
Time	84692.000000	139320.500000	172792.000000
V1	0.018109	1.315642	2.454930
V2	0.065486	0.803724	22.057729
V3	0.179846	1.027196	9.382558
V4	-0.019847	0.743341	16.875344
V5	-0.054336	0.611926	34.801666
V6	-0.274187	0.398565	73.301626
V7	0.040103	0.570436	120.589494
V8	0.022358	0.327346	20.007208
V9	-0.051429	0.597139	15.594995
V10	-0.092917	0.453923	23.745136

```
V11        -0.032757       0.739593      12.018913
V12         0.140033       0.618238       7.848392
V13        -0.013568       0.662505       7.126883
V14         0.050601       0.493150      10.526766
V15         0.048072       0.648821       8.877742
V16         0.066413       0.523296      17.315112
V17        -0.065676       0.399675       9.253526
V18        -0.003636       0.500807       5.041069
V19         0.003735       0.458949       5.591971
V20        -0.062481       0.133041      39.420904
V21        -0.029450       0.186377      27.202839
V22         0.006782       0.528554      10.503090
V23        -0.011193       0.147642      22.528412
V24         0.040976       0.439527       4.584549
V25         0.016594       0.350716       7.519589
V26        -0.052139       0.240952       3.517346
V27         0.001342       0.091045      31.612198
V28         0.011244       0.078280      33.847808
Amount     22.000000      77.165000   25691.160000
Class       0.000000       0.000000       1.000000
```

Verificamos la presencia de datos nulos en los datos

[8]:
```
data.isnull().sum(axis=0)
```

[8]:
```
Time      0
V1        0
V2        0
V3        0
V4        0
V5        0
V6        0
V7        0
V8        0
V9        0
V10       0
V11       0
V12       0
V13       0
V14       0
V15       0
V16       0
V17       0
V18       0
V19       0
V20       0
V21       0
V22       0
```

```
V23       0
V24       0
V25       0
V26       0
V27       0
V28       0
Amount    0
Class     0
dtype: int64
```

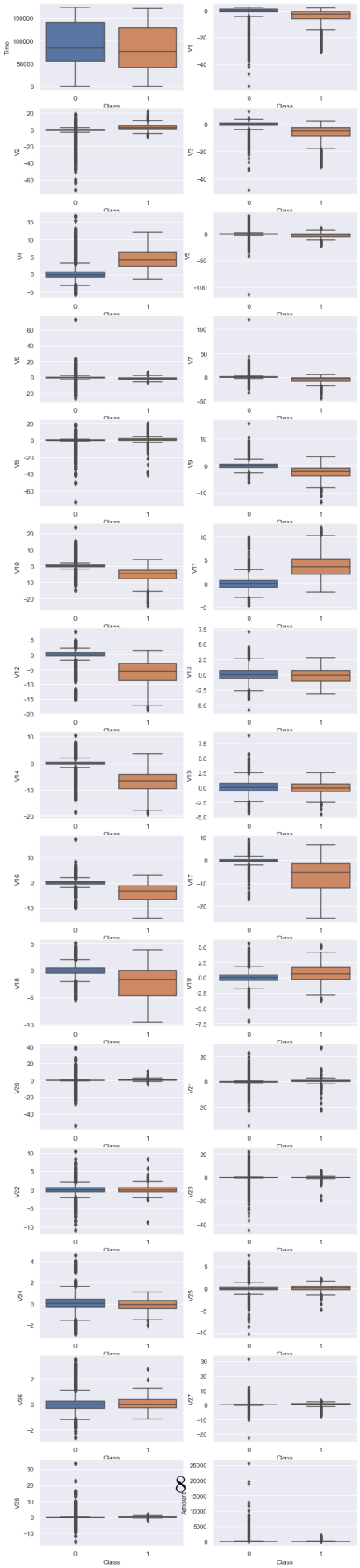
Por cierto **no hay precencia de datos nulos** en dichos datos, entonces vamos a observar su comportamiento

- La columna tiempo no guarda informacion significativa por lo cual la excluiremos del analisis
- Observamos la distribucion de los datos en base a un histograna

[9]:
```
fig,axes=plt.subplots(10,3,sharex=False,figsize=(30,30))
columns_tabla=data.columns

for i, ax in enumerate(axes.flat):
    sns.histplot(data.iloc[:,i],bins=10,kde=True,ax=ax)
    ax.set_title(f"Columna {columns_tabla[i]}")
plt.show()
```

- Observamos las anomalias de los datos. excluyendo la variable dependiente que pertenece a la columna

[10]:
```
fig,axes=plt.subplots(15,2,figsize=(10,50))
columns_tabla=data.columns

for i,ax in enumerate(axes.flat):
    sns.boxplot(x="Class",y=columns_tabla[i],data=data,ax=ax)
```


Time
V1
V2
V3
V4
V5
V6
V7
V8
V9
V10
V11
V12
V13
V14
V15
V16
V17
V18
V19
V20
V21
V22
V23
V24
V25
V26
V27
V28
Amount
Class

La mayoria de los datos posee anomalias pero en estos casos sabemos que no son malas debido a que tambien corresponden a los casos donde hubo fraude.

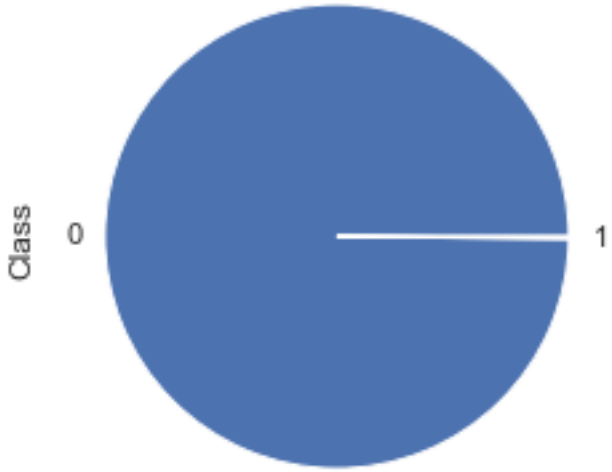
Debemos analizar mas detenidamiente estas muestras de anomalias

- Observamos la distribucion de las clases de los datos a taves de todos el set de informacion

[11]:
```
conteo=data.Class.value_counts(normalize=True)
display(conteo.map(lambda x: f"{x*100:0.2f}%"))
plt.rcParams["figure.figsize"]=(4,4)
conteo.plot(kind="pie",)
```

```
0    99.83%
1     0.17%
Name: Class, dtype: object
```

[11]: <AxesSubplot:ylabel='Class'>

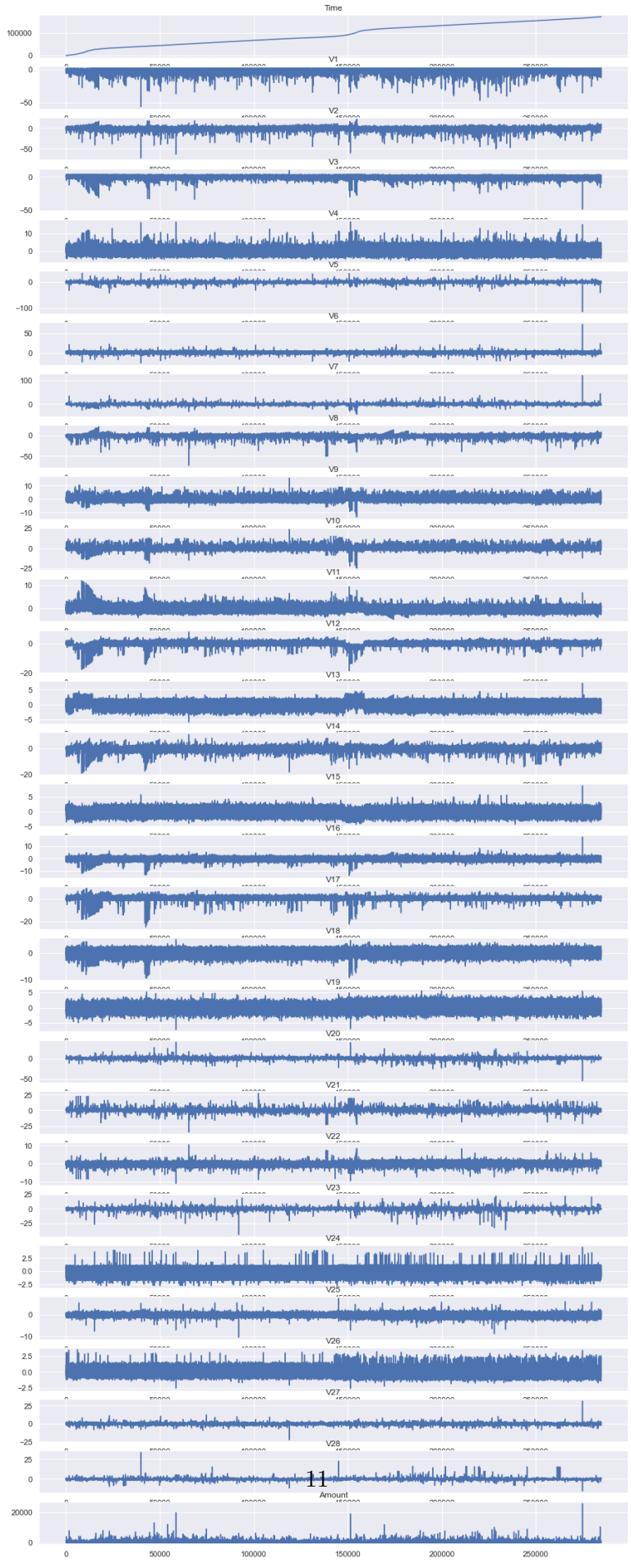


Tenemos como clase mayoritaria dominante los casos que no fueron fraude financiero con un 99.83% del total de datos, mientras que la minoria se corresponde con lo que queremos obtener (0.17%).

Este es un verdadero problema de desbalanceamiento de clases.Debido que el modelo aprendera mas de los datos de la clase mayoritaria que la de la clase minoritaria correspondiente a los fraude detectados.

- Observamos las tendencias de los datos

[12]:
```
fig,axes=plt.subplots(30,1,figsize=(15,40))

for i , ax in enumerate(axes.flat):
    ax.plot(data.iloc[:,i])
    ax.set_title(data.columns[i])
plt.show()
```

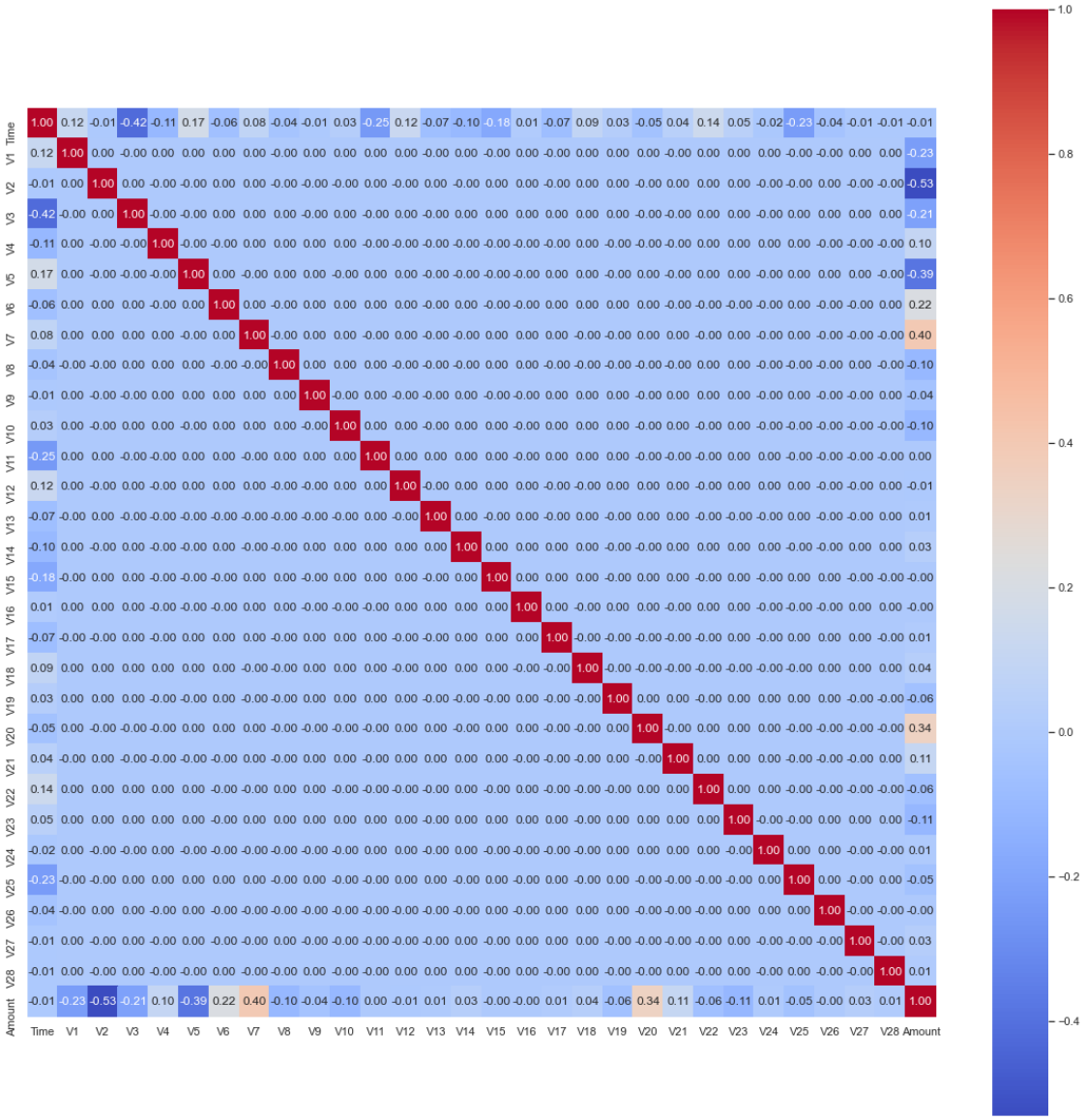

Time
V1
V2
V3
V4
V5
V6
V7
V8
V9
V10
V11
V12
V13
V14
V15
V16
V17
V18
V19
V20
V21
V22
V23
V24
V25
V26
V27
V28
Amount
0
50000
100000
150000
200000
250000

[13]:
```
plt.rcParams["figure.figsize"] = plt.rcParamsDefault["figure.figsize"]␣
↪#reseteamos la configuracion del tamaño del grafico
```

- Analizamos la correlacion de las variables. Esto es muy importante para evitar la colinealidad en los datos

[14]:
```
corr=data.iloc[:,:-1].corr(method="pearson")

plt.figure(figsize=(20,20))
sns.heatmap(corr.T,fmt="0.2f",annot=True,square=True,cmap="coolwarm")
```

[14]: <AxesSubplot:>



Buscamos la **colinealidad** entre las variables, esto se da cuando dos variables estan correlacionadas entre si. Por otro lado tambien podemos encontrar la **Multicolinealidad** cuando mas de una variable indepediente se encuentra correlacionada

Las correlaciones encontradas fueron las siguientes:

- **V1** y **Amount** estan levemente correlacionadas con un coeficiente de correlacion de **-0.23**. Esto es una correlacion inversa que significa que el aumento de una tiene una disminucion en el valor del otro par.

[15]:
```
from scipy.stats import pearsonr
```

Creamos una funcion que calcule la correlacion entre dos variables y nos entregue a traves de un nivel de significancia si la correlacion es estadisticamente siginficativa

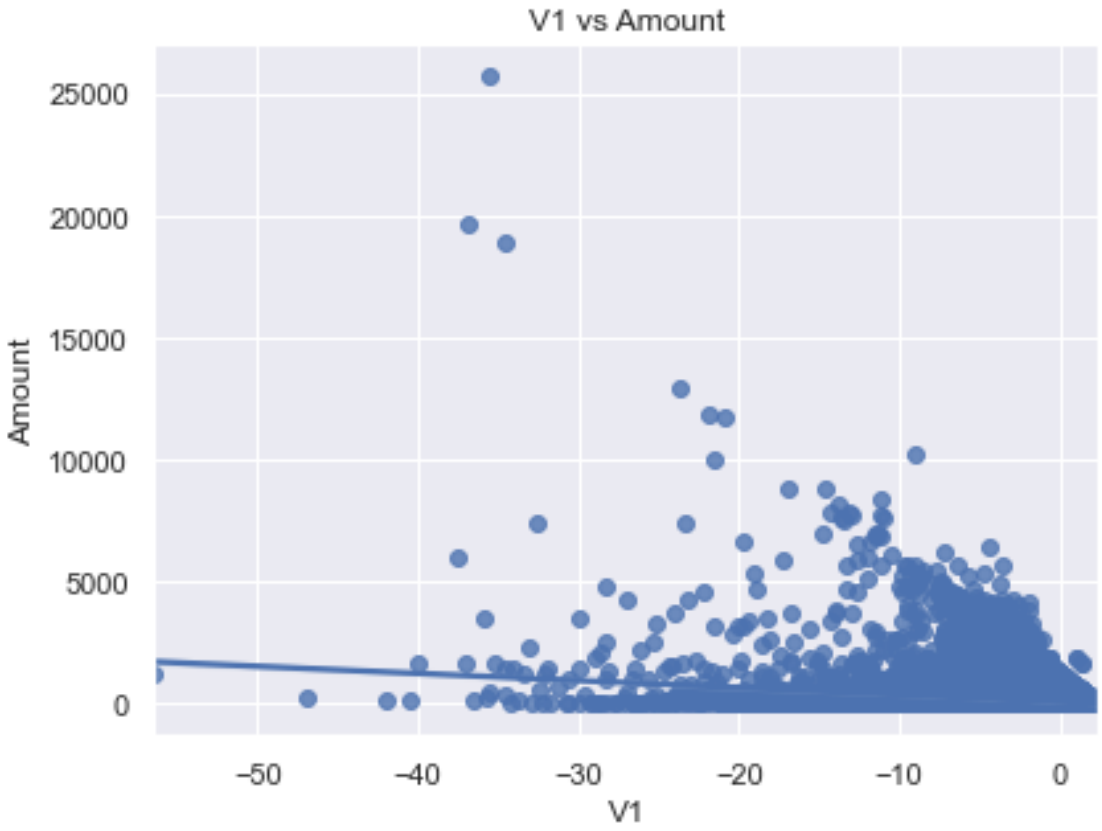
[16]:
```
def test_correlation(X1,X2,alpha=0.05):
    corr,p_valor=pearsonr(X1,X2) #comprobamos la correlacion entre X1 y X2
    #Utilizamos el p_valor para determinar si el coeficiente de correlacion es␣
↪estadisticamente significativo
    #si el p_valor (alfa) es menor o igual al nivel de significacia 0.05␣
↪entonces la correlaciones estadisiticamente significativa.

    print(f"COEF. correlacion: {corr}   | p-valor : {p_valor}")
    print("La correlacion es estadisticamente significativa" if p_valor<=alpha␣
↪else "La correlacion NO es significativa")
```

[17]:
```
#ejecutamos el test de correlacion
test_correlation(data.V1,data.Amount)
```

```
COEF. correlacion: -0.2277086529224042   | p-valor : 0.0
La correlacion es estadisticamente significativa
```

[18]:
```
sns.regplot(x="V1",y="Amount",data=data)
plt.title("V1 vs Amount")
```

[18]:
```
Text(0.5, 1.0, 'V1 vs Amount')
```



Concluimos que V1 y Amount tienen una correlacion estadisticamente significativa

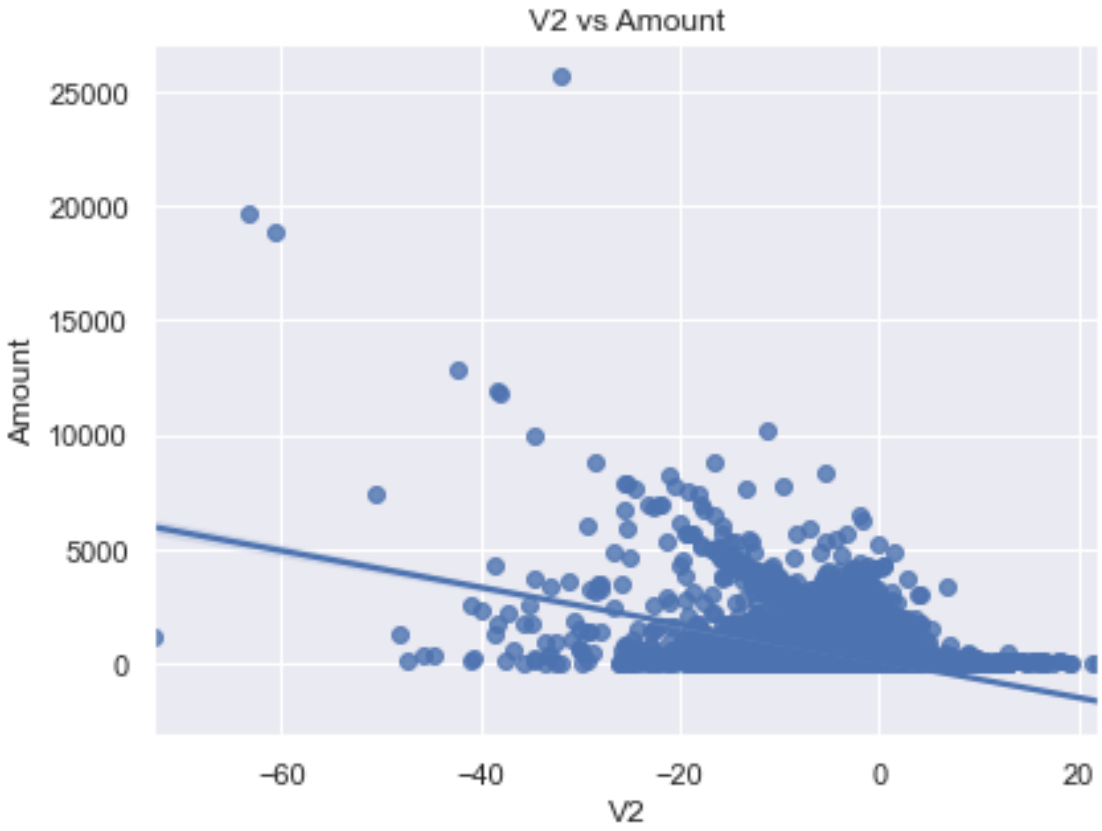
- **V2** y **Amount** estan moderadamente correlacionadas con un coeficiente de correlacion negativa de -0.53. Por lo cual tambien demostramos que estan significativamente correlacionados

[19]:
```
test_correlation(data.V2,data.Amount)
```

```
COEF. correlacion: -0.5314089393280333   | p-valor : 0.0
La correlacion es estadisticamente significativa
```

[20]:
```
sns.regplot(x="V2",y="Amount",data=data)
plt.title("V2 vs Amount")
```

[20]: `Text(0.5, 1.0, 'V2 vs Amount')`



- Tenemos la correlacion entre las variable independiente V3 con Time (-0.42) y Amount (-0.21)
- Descartamos la variable Time de este analisis debido a que el tiempo desde ya no aporta al modelo y sera exluido mas adelante

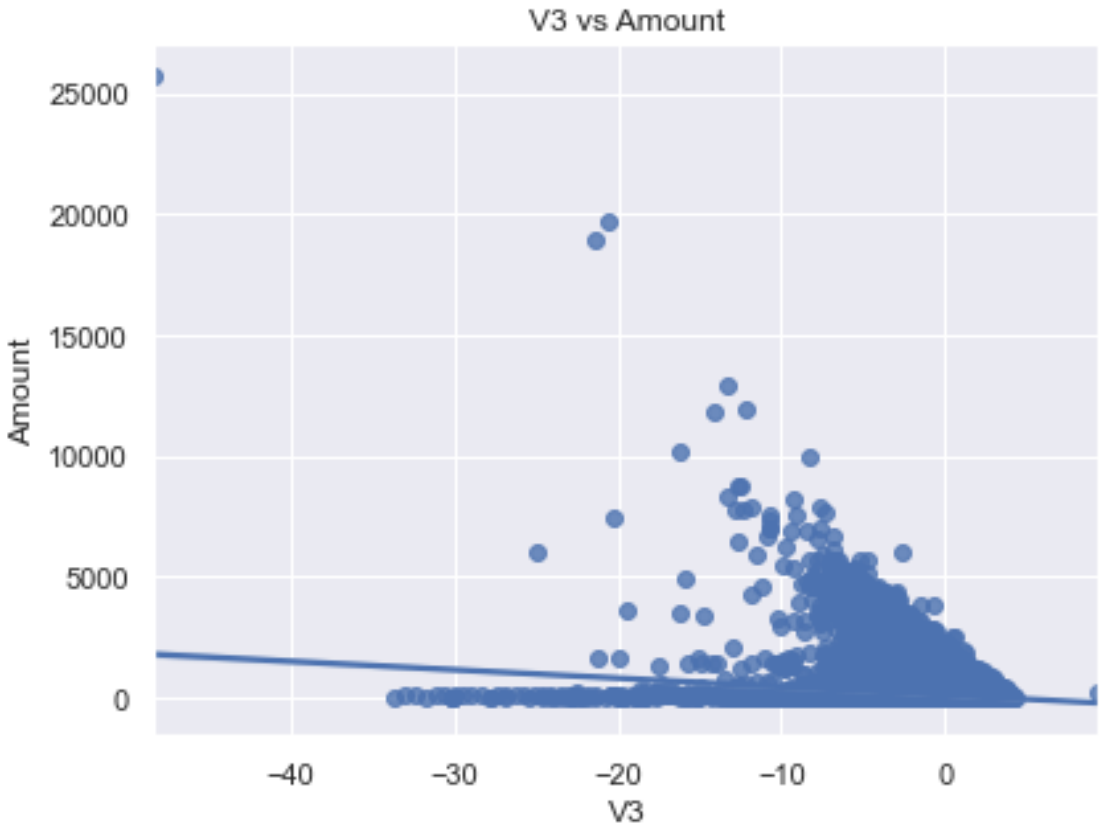
Por lo tanto probamos la correlacion entre **V3** y **Amount** y concluimos tambien que tienen una correlacion significativa

[21]:
```
test_correlation(data.V3,data.Amount)
```

```
COEF. correlacion: -0.21088047528990986   | p-valor : 0.0
La correlacion es estadisticamente significativa
```

[22]:
```
sns.regplot(x="V3",y="Amount",data=data)
plt.title("V3 vs Amount")
```

[22]:
```
Text(0.5, 1.0, 'V3 vs Amount')
```



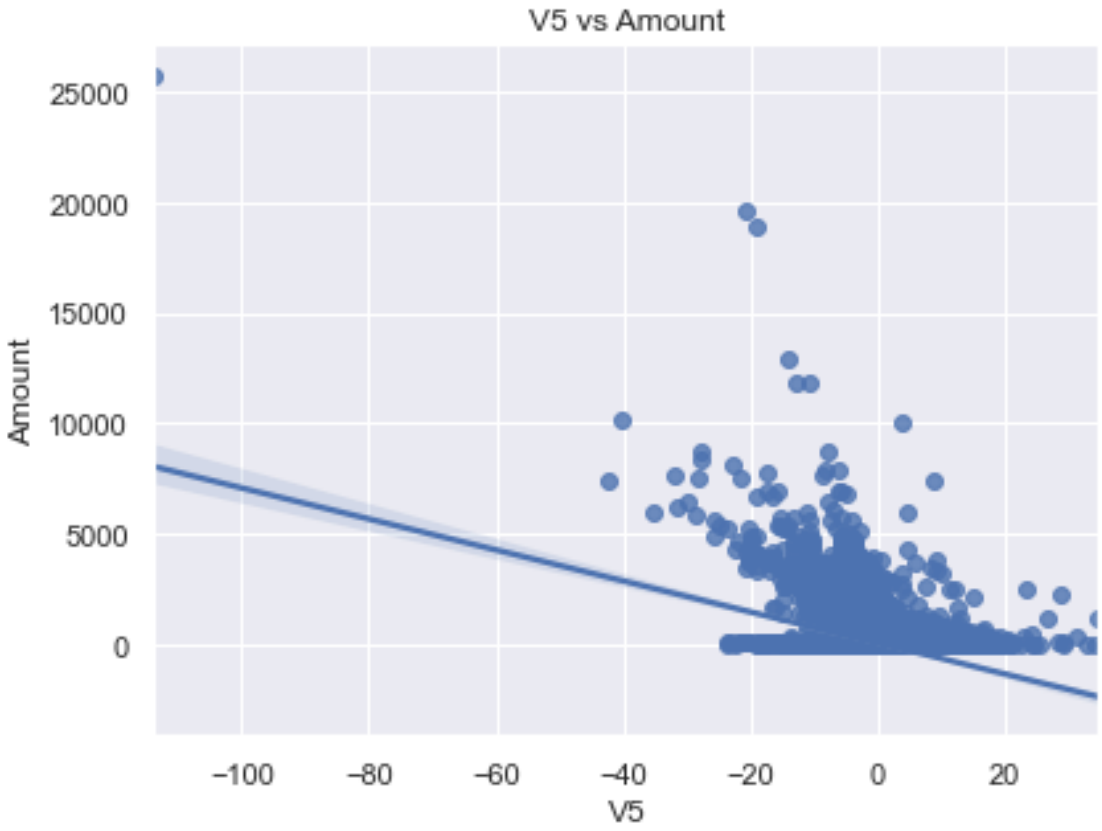
- La **V4** no se encuentra muy correlacionada con las demas variables, esta variable puede ser buena
- La **V5** se encuentra correlacionada con **Amount**

[23]:
```
test_correlation(data.V5,data.Amount)
```

```
COEF. correlacion: -0.38635625616306024   | p-valor : 0.0
La correlacion es estadisticamente significativa
```

[24]:
```
sns.regplot(x="V5",y="Amount",data=data)
plt.title("V5 vs Amount")
```

[24]:
```
Text(0.5, 1.0, 'V5 vs Amount')
```



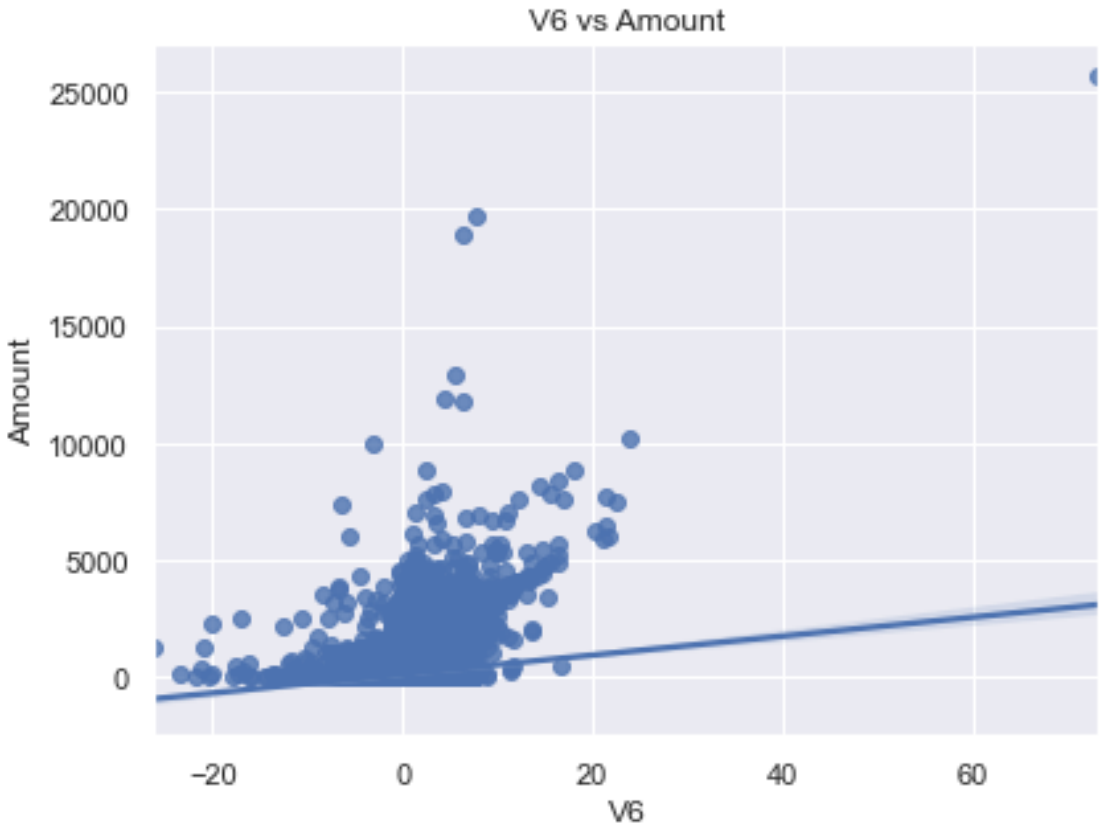
- La **V6** se encuentra correlacionada con **Amount**

[25]:
```
test_correlation(data.V6,data.Amount)
```

```
COEF. correlacion: 0.2159811802252792   | p-valor : 0.0
La correlacion es estadisticamente significativa
```

[26]:
```
sns.regplot(x="V6",y="Amount",data=data)
plt.title("V6 vs Amount")
```

[26]: `Text(0.5, 1.0, 'V6 vs Amount')`



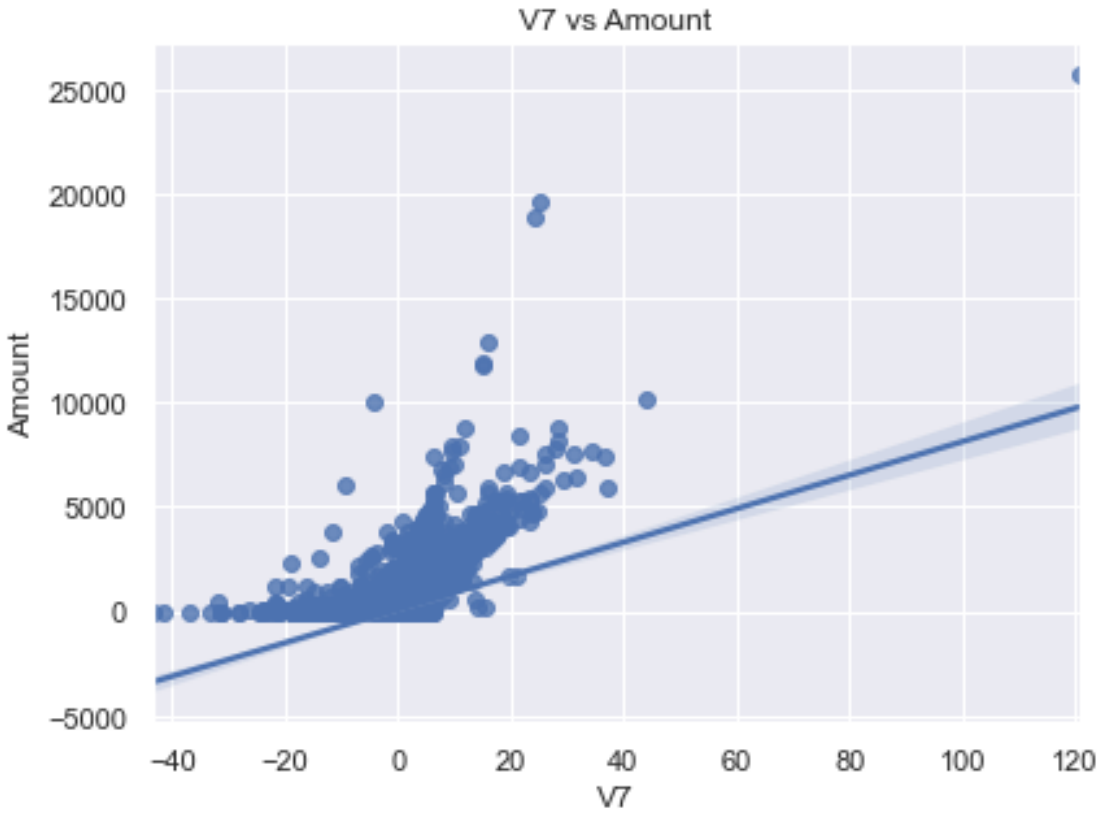
- La **V7** se encuentra correlacionada con **Amount**

[27]:
```
test_correlation(data.V7,data.Amount)
```

```
COEF. correlacion: 0.39731127818168943   | p-valor : 0.0
La correlacion es estadisticamente significativa
```

[28]:
```
sns.regplot(x="V7",y="Amount",data=data)
plt.title("V7 vs Amount")
```

[28]:
```
Text(0.5, 1.0, 'V7 vs Amount')
```



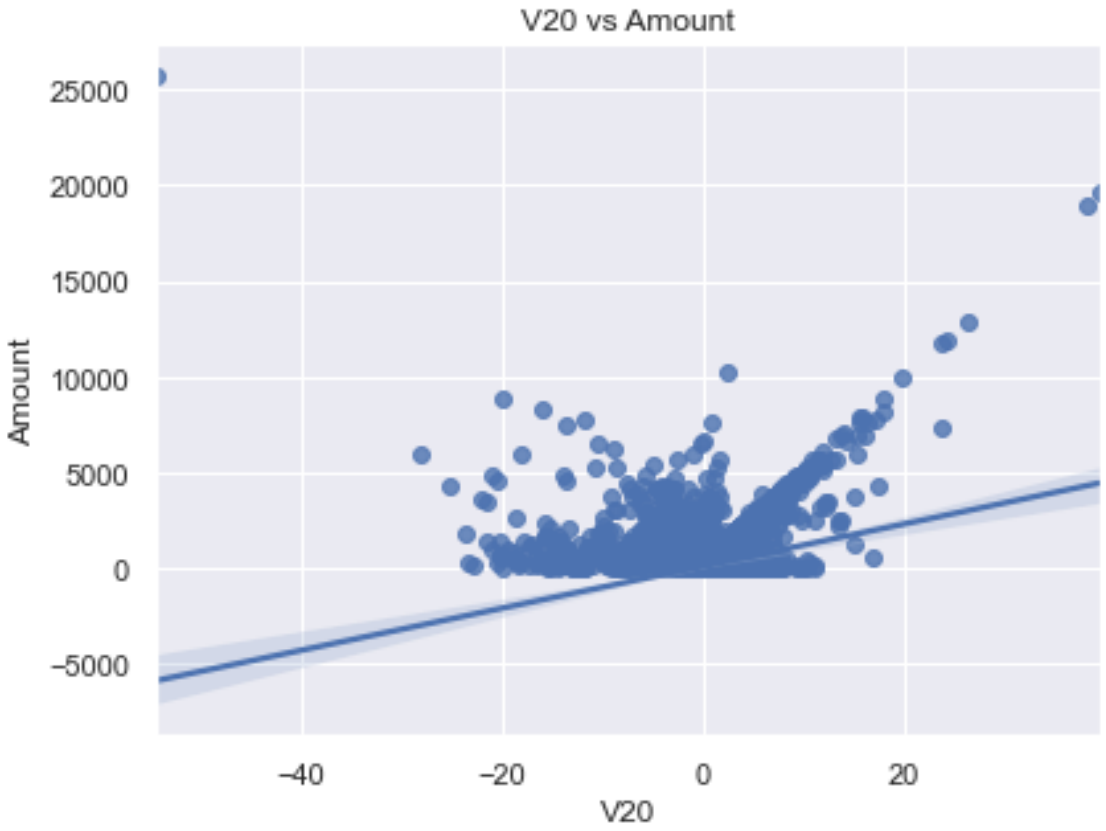
- Por ultimo tambien notamos una correlacion significativa entre **V20** y **Amount**

[29]:
```
test_correlation(data.V20,data.Amount)
```

```
COEF. correlacion: 0.33940340454617346   | p-valor : 0.0
La correlacion es estadisticamente significativa
```

[30]:
```
sns.regplot(x="V20",y="Amount",data=data)
plt.title("V20 vs Amount")
```

[30]: `Text(0.5, 1.0, 'V20 vs Amount')`



- El resto de variables se excluyeron de este analisis debido a que en la matriz de correlacion no tuvieron coeficientes de correlacion muy buenos con respecto a otras variables

Ahora realizaremos analisis orientados a una preparacion de datos

[31]:
```
#analizamos el rango de cada variable en el dataset esto es para ver sus␣
↪limites y poder realizar un escalamiento de datos mas adelanta
data.aggregate([np.min,np.max]).T
```

[31]:

	amin	amax
Time	0.000000	172792.000000
V1	-56.407510	2.454930
V2	-72.715728	22.057729
V3	-48.325589	9.382558
V4	-5.683171	16.875344
V5	-113.743307	34.801666
V6	-26.160506	73.301626
V7	-43.557242	120.589494
V8	-73.216718	20.007208
V9	-13.434066	15.594995
V10	-24.588262	23.745136

```
V11      -4.797473      12.018913
V12     -18.683715       7.848392
V13      -5.791881       7.126883
V14     -19.214325      10.526766
V15      -4.498945       8.877742
V16     -14.129855      17.315112
V17     -25.162799       9.253526
V18      -9.498746       5.041069
V19      -7.213527       5.591971
V20     -54.497720      39.420904
V21     -34.830382      27.202839
V22     -10.933144      10.503090
V23     -44.807735      22.528412
V24      -2.836627       4.584549
V25     -10.295397       7.519589
V26      -2.604551       3.517346
V27     -22.565679      31.612198
V28     -15.430084      33.847808
Amount    0.000000   25691.160000
Class     0.000000       1.000000
```

Podemos observar que el rango de datos se encuentra muy disperso como tambien existen columnas como **Time** y **Amount** que poseen rango mas anchos debido a los dato que manejan. Esto amerita que luego que los datos esten preparados se escalen los datos para que posean un rango comun

- Podemos esforzarnos y buscar **outliers** pero en este contexto de fraude bancario a aveces las anomalias en casos de fraude son las que nuestro modelo debe predecir para ajustarnos correctamente a los datos. Ahora tambien con respecto a los casos de no fraude, es normal que los clientes retiren cantidades infimas y luego cantidades grandes despues de un periodo largo de tiempo. En este contexto penalizar los outliers no es una buena idea.

Ahora examinemos otro problema comun en los datos. Los **valores duplicados**.

[32]:
```
mask_duplicated=data.duplicated(subset=None) #devuelve la cantidad de valores␣
 ↪duplicados en funcion de todas las columnas
print(mask_duplicated.sum())
```

```
1081
```

Como parte de nuestro analisis podemos saber cuales son esas filas duplicadas, para ellos hacemos uso de la matriz booleana de enmascaramiento devuelta como resultado

[33]:
```
data[mask_duplicated].head() #mostramos los 5 primeras filas
```

[33]:
```
     Time        V1        V2        V3        V4        V5        V6  \
33   26.0 -0.529912  0.873892  1.347247  0.145457  0.414209  0.100223
35   26.0 -0.535388  0.865268  1.351076  0.147575  0.433680  0.086983
113  74.0  1.038370  0.127486  0.184456  1.109950  0.441699  0.945283
114  74.0  1.038370  0.127486  0.184456  1.109950  0.441699  0.945283
115  74.0  1.038370  0.127486  0.184456  1.109950  0.441699  0.945283
```

```
           V7        V8        V9  …       V21       V22       V23  \
33   0.711206  0.176066 -0.286717  …  0.046949  0.208105 -0.185548
35   0.693039  0.179742 -0.285642  …  0.049526  0.206537 -0.187108
113 -0.036715  0.350995  0.118950  …  0.102520  0.605089  0.023092
114 -0.036715  0.350995  0.118950  …  0.102520  0.605089  0.023092
115 -0.036715  0.350995  0.118950  …  0.102520  0.605089  0.023092

           V24       V25       V26       V27       V28  Amount  Class
33   0.001031  0.098816 -0.552904 -0.073288  0.023307    6.14      0
35   0.000753  0.098117 -0.553471 -0.078306  0.025427    1.77      0
113 -0.626463  0.479120 -0.166937  0.081247  0.001192    1.18      0
114 -0.626463  0.479120 -0.166937  0.081247  0.001192    1.18      0
115 -0.626463  0.479120 -0.166937  0.081247  0.001192    1.18      0

[5 rows x 31 columns]
```

- Es sorprendente saber de la existencia de **1081 datos duplicados**. Estos se eliminaran en la fase de procesamiento de datos

Ahora veamos la distribucion de de clases contenidas en dichos datos duplicados

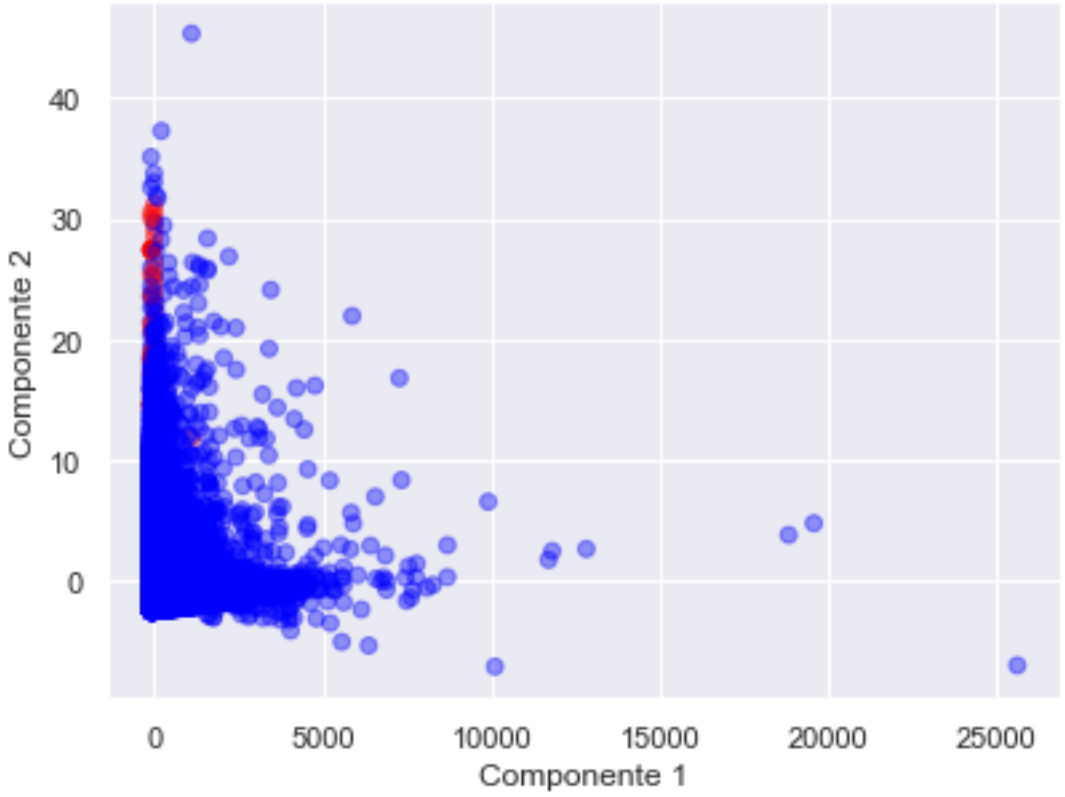
[34]:
```
data[mask_duplicated].Class.value_counts()
```

[34]:
```
0    1062
1      19
Name: Class, dtype: int64
```

Ahora bien, puede que estas transacciones no vengan de un mismo cliente y que sea casualidad que tengan las mismas caracteristicas, como tambien de que hubiera una fuga en la base de datos y se duplicaron algunas transacciones a la misma hora. En ambos casos se debe retirar del conjunto de datos mas adelante

[56]:
```
#observamos la dispersion de los datos a traves del analisis de componentes␣
↪principales
from sklearn.decomposition import PCA

#creo una funcion para mostrar la dispersion de clases por cada dataset

def plot_scatter_by_class(ax=None,data=None):
    pca=PCA(n_components=2)
    XY=pca.fit_transform(data.iloc[:,1:-1]) #excluimos el Time y la columna␣
↪Class

    color_map=np.array(["blue","red"])[data.Class] #Los No fraudes son Azules,y␣
↪fraudes rojos
    if ax is None:
        _,(ax)=plt.subplots(1,1)
```

```
    ax.scatter(XY[:,0],XY[:,1],color=color_map,alpha=0.4)
    ax.set_ylabel("Componente 2")
    ax.set_xlabel("Componente 1")


plot_scatter_by_class(data=data)
```



Otros pasos

Las caracteristicas de entrada son continuas y la salida es categorica. Por lo tanto no necesitamos realizar algun procesamiento adicional de condificacion en caliente u ordinales

0.1.3 PROCESAMIENTO DE DATOS

- Ya estamos listos para aplicar tecnicas de procesamiento de datos.

En primer lugar desde un inicio notamos que la columna tiempo no aporta valor, ya que solo apunta a la hora donde se realizo dicha transaccion.

[44]:
```
df_cleaned=data.copy() #los datos de analisis los copiamos al dataframe de␣
↪limpeza para empezar con el proceso
```

[45]:
```
#procedemos a eliminar las columnas con valors duplicados detectados en la␣
↪exploracion de datos

mask_duplicated=df_cleaned.duplicated(subset=None) #subset=None para considerar␣
↪todas las columnas para evaluar la duplicidad
mask_duplicated.sum()
```

[45]:
```
1081
```

[46]:
```
df_cleaned=df_cleaned.drop_duplicates(subset=None) #borramos los duplicados
```

[47]:
```
df_cleaned.drop("Time",axis=1,inplace=True) #borramos la columna tiempo. No␣
↪aportaba al modelo
```

Las variables **V1,V2,V3,V5,V6,V7 ... V20**. Estaba correlacionadas con Amount pero no a un alto nivel mayor a 0.70, debido a que es normal que las caracteristicas flutuen generando un poco de correlacion, no podemos eliminarlas por ser colineales porque su coeficiente no era muy elevado como para eliminar una y dejar otra, cada quien tenia sus particularidades.

Estas variables no se eliminan en esta fase (*)

(*) Si el modelo no tiene mejoras podemos regresar a la fase de exploracion y analisis para considerar eliminar dichas variables. El proceso de machine learning es iterativo

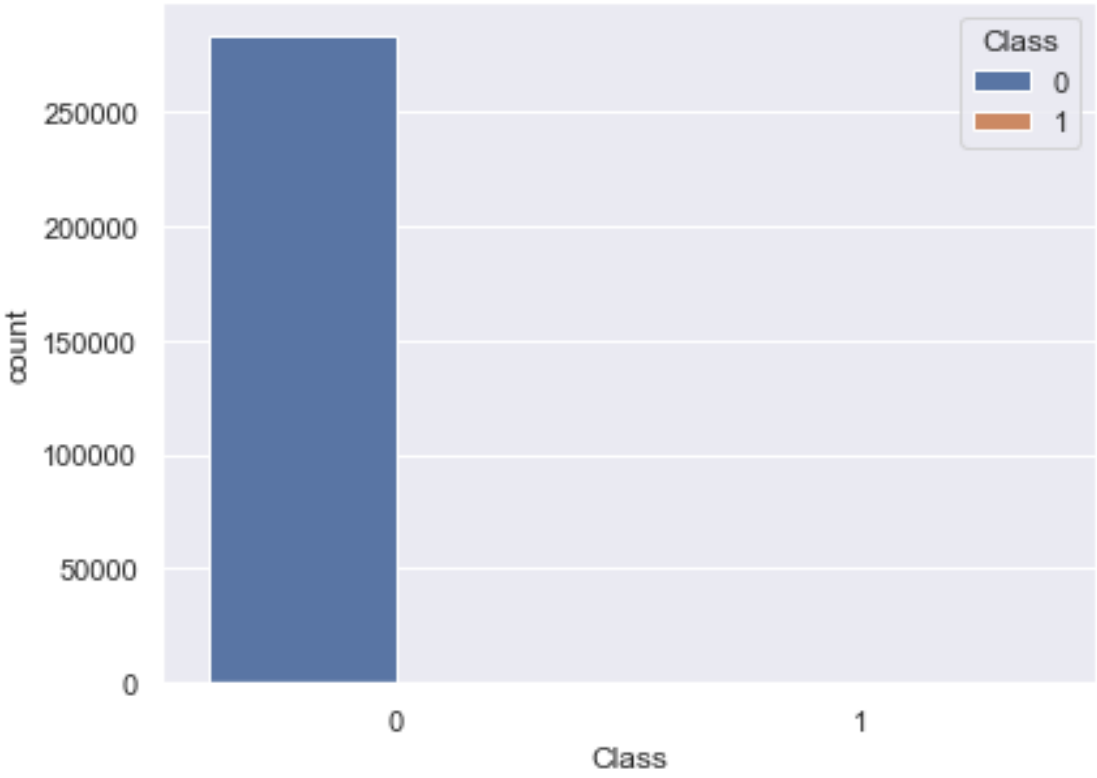
El problema de datos desbalanceados (99+% No fraude | 1% Si Fraude) Tener datos desbalanceados hace sesgar los resultados del modelo, debido a que se familiarizara mas con la clase mayoritaria que acabara prediciendo muy buenos datos sobre esta, que en lugar de la clase minoritaria.

A continuacion tenemos la proporcion actual

[48]:
```
print("====Conteo de Clases====")
display(df_cleaned.Class.value_counts())
print("====Proporcion de clases====")
display(df_cleaned.Class.value_counts(normalize=True).map(lambda x: f"{x:0.
↪10f}%"))
```

```
====Conteo de Clases====

0    283253
1       473
Name: Class, dtype: int64

====Proporcion de clases====

0    0.9983328986%
1    0.0016671014%
Name: Class, dtype: object
```

[49]:
```
ax=sns.countplot(x="Class",hue="Class",data=df_cleaned)
```



Primer camino: Submuestreo Esta es una tecnica para reducir la proporcion de la clase mayoritaria para que este al nivel de la minoritaria, tal que al final el proceso, los datos se reducen al nivel de esta ultima.

[50]:
```
#Nuestro conjunto de datos estaba desequilibrado con el 99% de clase negativa y␣
↪1% positiva en fraude.
#En este caso tenemos que hacer un submuestreo en favor a la clase␣
↪minoritaria(nuestro objetivo), para que se equilibren los datos y el modelo␣
↪no nos arroje metricas sesgadas

from imblearn.under_sampling import RandomUnderSampler

undersampling=RandomUnderSampler(random_state=42)
X_sample,y_sample=undersampling.fit_resample(df_cleaned.iloc[:,:-1],df_cleaned.
↪iloc[:,-1]) #allfeatures , labels
```

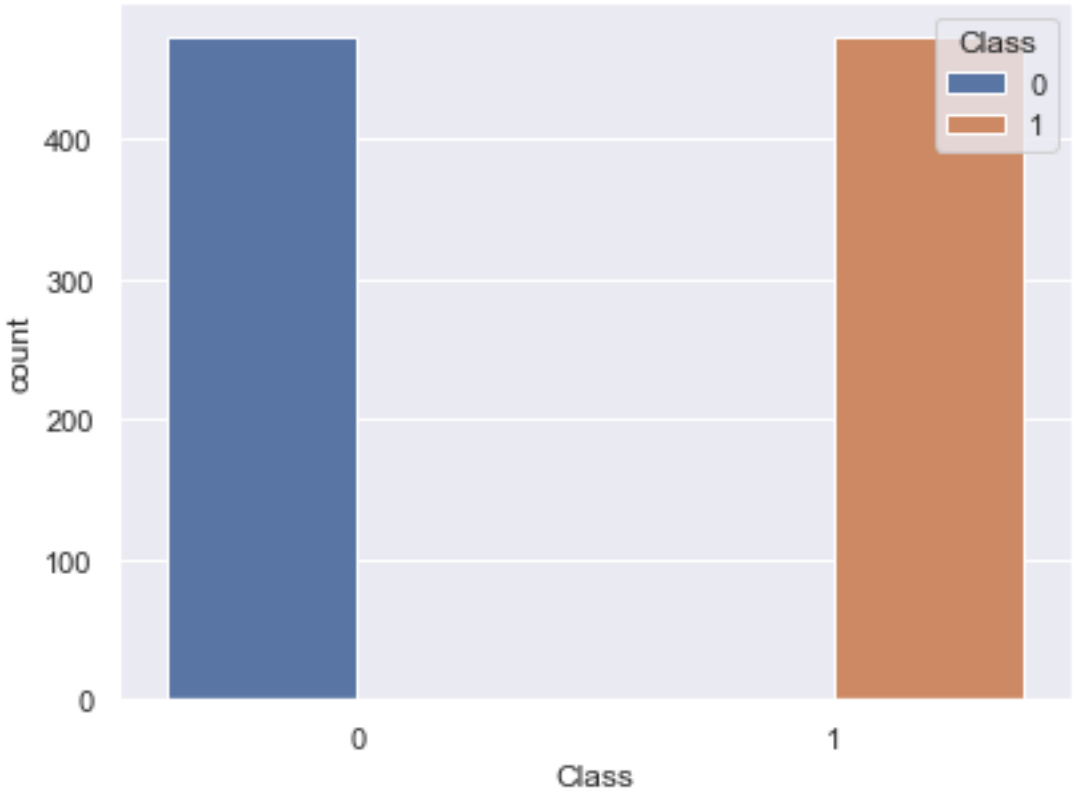
[51]:
```
y_sample.value_counts() #proporcion de datos luego de submuestreo
```

[51]:
```
1    473
0    473
Name: Class, dtype: int64
```

Concatenamos los conjuntos de datos en un solo set

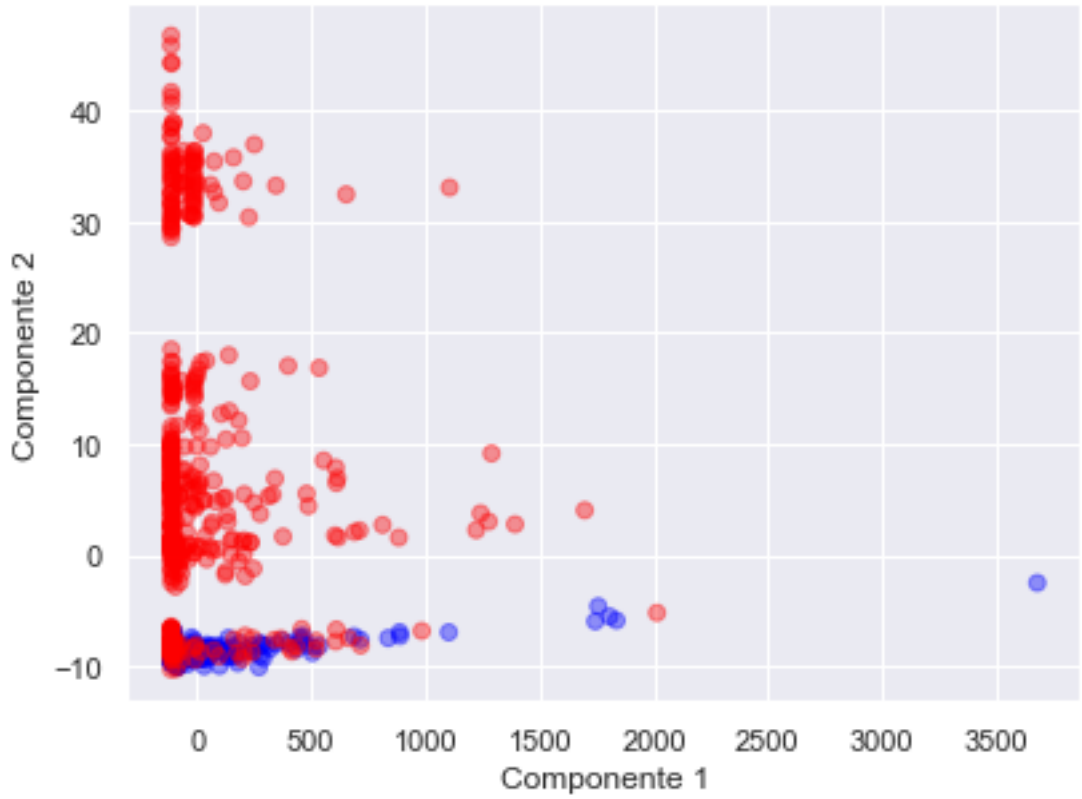
[53]:
```
data_undersampled=pd.concat([X_sample,y_sample],axis=1)
```

[54]:
```
ax=sns.countplot(x="Class",hue="Class",data=data_undersampled)
```



Mostramos la dispersion de estos datos

[57]:
```
plot_scatter_by_class(data=data_undersampled)
```



Guardamos los datos procesados en entrenamiento/test. OJO: No se realizo el escalado de datos

[68]:
```
from sklearn.model_selection import train_test_split

def save_balanced_data(data,name=None):
    import os
    columns=data.columns
    x_train,x_test,y_train,y_test=train_test_split(data.values[:,:-1],data.
 ↪values[:,-1],test_size=0.3,stratify=data.values[:,-1])
    train_data=np.concatenate([x_train,y_train.reshape(-1,1)],axis=1)
    test_data=np.concatenate([x_test,y_test.reshape(-1,1)],axis=1)
    #los transformo en dataframe y los guardo
    os.makedirs(f"./{name}",exist_ok=True)
    pd.DataFrame(train_data,columns=columns).to_csv(f"./{name}/train.
 ↪csv",index=False)
    pd.DataFrame(test_data,columns=columns).to_csv(f"./{name}/test.
 ↪csv",index=False)
    print(f"./{name}/train.csv Guardado correctamente")
    print(f"./{name}/test.csv Guardado correctamente")

save_balanced_data(data_undersampled,name="creditcard_undersampling")
```

```
./creditcard_undersampling/train.csv Guardado correctamente
./creditcard_undersampling/test.csv Guardado correctamente
```

Segundo Camino: Mantener la proporcion de datos y centrarse en las metricas Aqui dejamos los datos sin balancear para poder utilizarlos directamente en el modelo y evaluarlos con metricas como **Recall, F1, Precision**.

[69]:
```
save_balanced_data(df_cleaned,name="creditcard_imbalanced")
```

```
./creditcard_imbalanced/train.csv Guardado correctamente
./creditcard_imbalanced/test.csv Guardado correctamente
```

Los conjuntos balanceados como no balanceados se guardaron correctamente. Existen ahora dos posibles cambinos ya mencionados.

- Si el modelo no tiene buenos resultados en la evaluacion con un conjunto,entonces tenemos otro. De esta manera medimos el impacto de la transformacion de datos en el modelo

3ER Camino: Sobremuestreo de datos

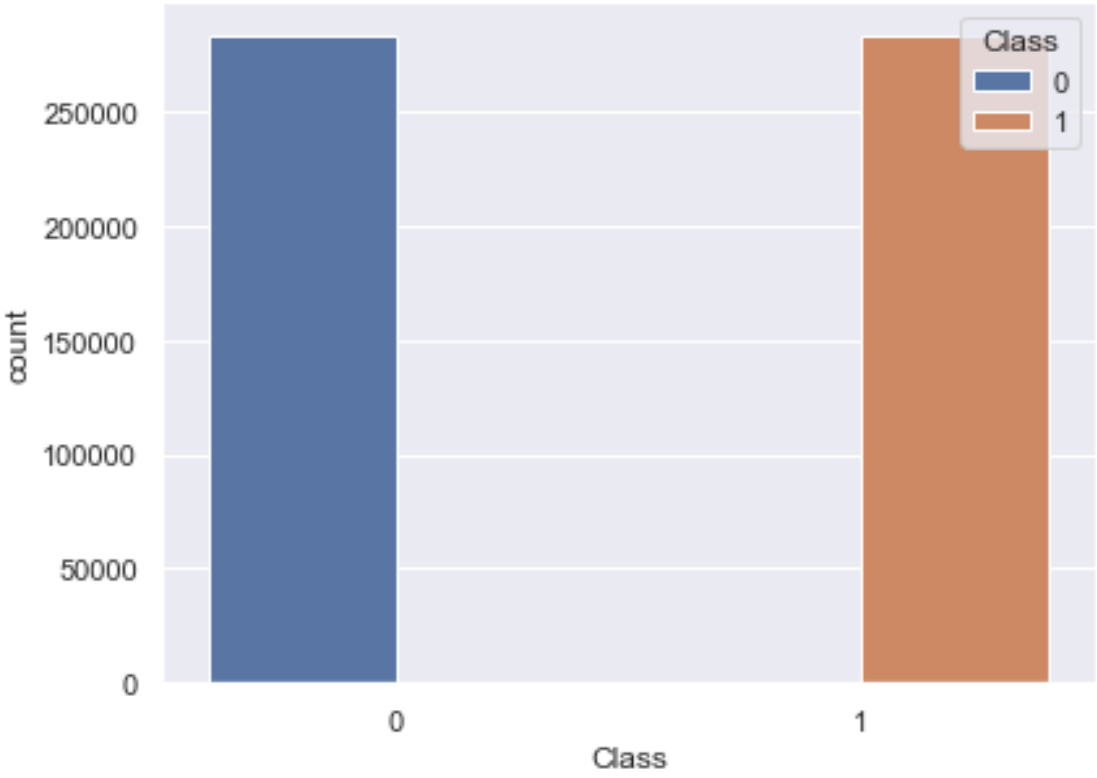
- Para ello haremos un sobremuestreo usando la funcion SMOTE de la libreria imbalanced learn

[70]:
```
from imblearn.over_sampling import SMOTE

smt=SMOTE(random_state=42)
x_sampled,y_sampled=smt.fit_resample(df_cleaned.iloc[:,:-1],df_cleaned.iloc[:
 ↪,-1])
np.unique(y_sampled,return_counts=True)
```

[70]:
```
(array([0, 1], dtype=int64), array([283253, 283253], dtype=int64))
```

[71]:
```
data_oversampled=pd.concat([x_sampled,y_sampled],axis=1)
```

[73]:
```
ax=sns.countplot(x="Class",hue="Class",data=data_oversampled)
```



Como podemos ver los datos se emparejaron los de mayor muestra, por lo que tambien decidimos guardar una muestra

[74]:
```
save_balanced_data(data_oversampled,name="creditcard_oversampling")
```

```
./creditcard_oversampling/train.csv Guardado correctamente
./creditcard_oversampling/test.csv Guardado correctamente
```

0.2 Modelado

- Esta fase se realiza en otro notebook, pero por temas de presentacion se realizara en el mismo.

Cuando queremos empezar a modelar, primero cargamos los datos.

- Definir la tecnica de modelado : Redes neuronales MLP
- Construir el modelo
- Validar el modelo

[75]:
```
import numpy as np
import pandas as pd
#establecer una semilla para reprudicibilidad de resultados
seed=42
seed=np.random.seed(seed)
```

[4]:
```
def load_data(name=None):

    data=pd.read_csv(name).values
    return data[:,:-1],data[:,-1] #retornamos los conjuntos (features,labels)
```

[93]:
```
x_train,y_train=load_data("./creditcard_imbalanced/train.csv") #cargamos los␣
↪datos de entrenamiento
```

Dividimos los datos en pliegues de entrenamiento y validacion; para luego escalarlos

[92]:
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler


def partition_data(X,y,with_scale=False,split_size=0.2):
    """
    Esta es una funcion generica que divide los datos en dos sets balanceados y␣
↪luego da la opcion de escalarlos.
    El motivo de esta funcion es para ahorrar lineas de codigo en este proyecto␣
↪complejo y usarlas con facilidad

    X: Conjunto de caracteristicas en matrices numpy
    y: Etiquetas de prediccion en matrices numpy

    Return:
    (x_train,x_val,y_train,y_val) si with_scale=False
    (scale,x_train,x_val,y_train,y_val) si with_scale=True, devolviendo el␣
↪objeto escalador 'scaler' """


  ␣
↪x_train,x_val,y_train,y_val=train_test_split(X,y,test_size=split_size,random_state=42,shuff

    if with_scale:
        scaler=MinMaxScaler(feature_range=(0,1))
        scaler.fit(x_train,y_train);

        x_train=scaler.transform(x_train)
        x_val=scaler.transform(x_val)

        return scaler,x_train,x_val,y_train,y_val

    return x_train,x_val,y_train,y_val
```

[94]:
```
scaler_imb,x_train,x_val,y_train,y_val=partition_data(x_train,y_train,with_scale=True,split_si
↪2) #20% de datos de validacion
```

A continuacion comenzamos a construir el modelo, para ellos cargamos las librearias inciales y herramientas para su implementacion

[95]:
```
%load_ext tensorboard
import tensorflow as tf
```

[96]:
```
#primero cargamos las metricas del modelo
metrics=[
        tf.keras.metrics.BinaryAccuracy(name="accuracy"),
        tf.keras.metrics.Recall(name="recall"),
    ]
```

0.2.1 CONSTRUCCION DEL MODELO BASE

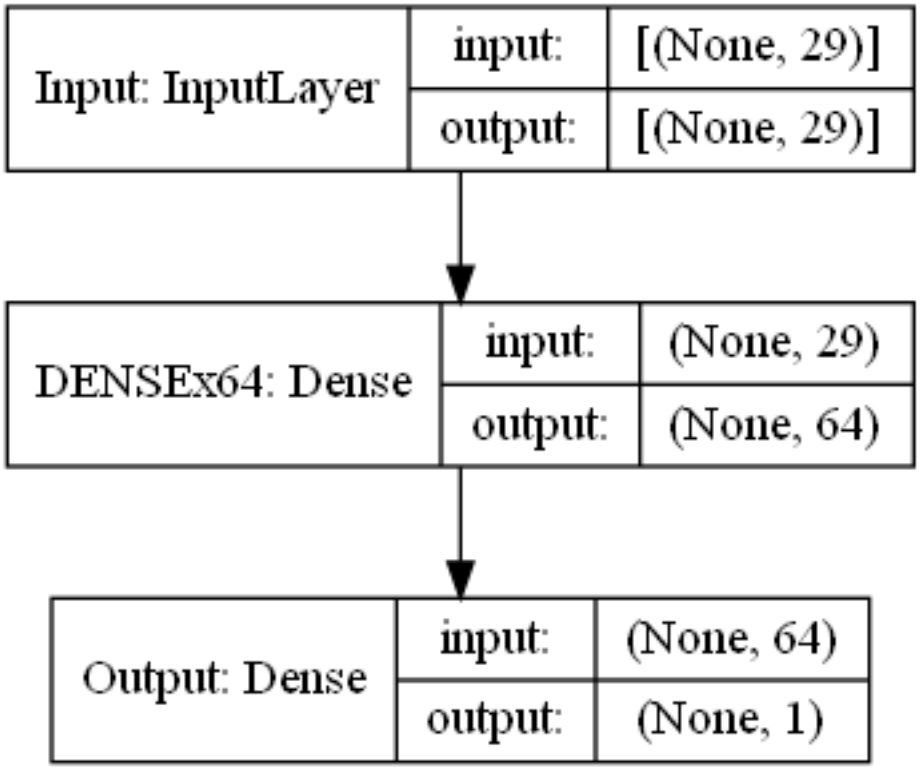
El modelo base tiene como objetivo mostrar una prediccion inicial de los datos, para que luego un modelo posterior se desarrolle teniendo en cuenta que tiene que tener un puntaje mayor al obtenido por el base

[103]:
```
n_features=x_train.shape[1] #obtenemos el numero de caracteristicas

def build_model(metrics=metrics):

    #arquitectura del modelo

    model=tf.keras.models.Sequential(name="MODEL_SEQUENTIAL")

    model.add(tf.keras.Input(shape=(n_features,),name="Input"))
    model.add(tf.keras.layers.Dense(64,activation="relu",name="DENSEx64"))␣
 ↪#para evitar cuellos de botella se elige un valor mayor al de la dimension␣
 ↪de entrada

    model.add(tf.keras.layers.Dense(1,activation="sigmoid",name="Output"))

    #la configuracion de compilacion del modelo
    model.compile(optimizer=tf.keras.optimizers.Adam(lr=1e-3),
                  loss=tf.keras.losses.BinaryCrossentropy(),
                  metrics=metrics
                 )
    return model
```

Mostramos la arquitectura del modelo base.

[98]:
```
from tensorflow.keras.utils import plot_model

base_model=build_model()
plot_model(base_model,"base_model.png",show_shapes=True)
```

[98]:



Vemos la descripcion de la capas y el numero de parametros entrenables

[99]:
```
base_model.summary()
```

```
Model: "MODEL_SEQUENTIAL"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
DENSEx64 (Dense)             (None, 64)                1920
_________________________________________________________________
Output (Dense)               (None, 1)                 65
=================================================================
Total params: 1,985
Trainable params: 1,985
Non-trainable params: 0
_________________________________________________________________
```

Ahora establecemos **callbacks** tanto para detener el proceso de entrenamiento si no se obtienen buenos resultados por un numero determinado de epocas. Como tambien para obtener graficas del progeso del modelo durante el entrenamiento segun las epocas

[100]:
```
early_stop=tf.keras.callbacks.EarlyStopping(
```

```
                                                monitor="val_loss", #monitorizamos␣
↪la funcion de perdida
                                                patience=10,
                                    )
```

[104]:
```
#callbacks para las graficas
def get_callback_tensorboard(name=None):
    import time

    name= int(time.time()) if name is None else f"{name}_{int(time.time())}"␣
↪#guardo el nombre y el tiempo
    log_dir=f"./tensorboard/fraud_detection/{name}"
    tb=tf.keras.callbacks.TensorBoard(log_dir=log_dir)
    return tb
```

[106]:
```
BATCH_SIZE=64
base_model=build_model()
tensorboard_callbacks=get_callback_tensorboard("baseline")
history=base_model.
↪fit(x_train,y_train,epochs=100,batch_size=BATCH_SIZE,validation_data=(x_val,y_val),callback
```

```
Epoch 1/100
2483/2483 [==============================] - 5s 2ms/step - loss: 0.0282 -
accuracy: 0.9988 - recall: 0.2991 - val_loss: 0.0078 - val_accuracy: 0.9984 -
val_recall: 0.0303
Epoch 2/100
2483/2483 [==============================] - 3s 1ms/step - loss: 0.0076 -
accuracy: 0.9984 - recall: 0.1698 - val_loss: 0.0052 - val_accuracy: 0.9989 -
val_recall: 0.4242
Epoch 3/100
2483/2483 [==============================] - 3s 1ms/step - loss: 0.0057 -
accuracy: 0.9987 - recall: 0.3551 - val_loss: 0.0044 - val_accuracy: 0.9991 -
val_recall: 0.5303
Epoch 4/100
2483/2483 [==============================] - 3s 1ms/step - loss: 0.0039 -
accuracy: 0.9991 - recall: 0.5738 - val_loss: 0.0043 - val_accuracy: 0.9991 -
val_recall: 0.5303
Epoch 5/100
2483/2483 [==============================] - 3s 1ms/step - loss: 0.0040 -
accuracy: 0.9993 - recall: 0.6681 - val_loss: 0.0039 - val_accuracy: 0.9992 -
val_recall: 0.5909
Epoch 6/100
2483/2483 [==============================] - 3s 1ms/step - loss: 0.0036 -
accuracy: 0.9992 - recall: 0.6530 - val_loss: 0.0037 - val_accuracy: 0.9993 -
val_recall: 0.6818
Epoch 7/100
2483/2483 [==============================] - 3s 1ms/step - loss: 0.0037 -
accuracy: 0.9991 - recall: 0.6285 - val_loss: 0.0038 - val_accuracy: 0.9993 -
```

```
val_recall: 0.7424
Epoch 8/100
2483/2483 [==============================] - 3s 1ms/step - loss: 0.0028 -
accuracy: 0.9994 - recall: 0.7156 - val_loss: 0.0036 - val_accuracy: 0.9994 -
val_recall: 0.7424
Epoch 9/100
2483/2483 [==============================] - 3s 1ms/step - loss: 0.0038 -
accuracy: 0.9993 - recall: 0.7088 - val_loss: 0.0035 - val_accuracy: 0.9994 -
val_recall: 0.7273
Epoch 10/100
2483/2483 [==============================] - 3s 1ms/step - loss: 0.0031 -
accuracy: 0.9994 - recall: 0.7266 - val_loss: 0.0035 - val_accuracy: 0.9994 -
val_recall: 0.7424
Epoch 11/100
2483/2483 [==============================] - 4s 1ms/step - loss: 0.0032 -
accuracy: 0.9994 - recall: 0.7251 - val_loss: 0.0034 - val_accuracy: 0.9994 -
val_recall: 0.7424
Epoch 12/100
2483/2483 [==============================] - 3s 1ms/step - loss: 0.0035 -
accuracy: 0.9993 - recall: 0.7605 - val_loss: 0.0037 - val_accuracy: 0.9994 -
val_recall: 0.7121
Epoch 13/100
2483/2483 [==============================] - 3s 1ms/step - loss: 0.0032 -
accuracy: 0.9993 - recall: 0.7661 - val_loss: 0.0035 - val_accuracy: 0.9994 -
val_recall: 0.7424
Epoch 14/100
2483/2483 [==============================] - 3s 1ms/step - loss: 0.0036 -
accuracy: 0.9993 - recall: 0.7293 - val_loss: 0.0035 - val_accuracy: 0.9993 -
val_recall: 0.7576
Epoch 15/100
2483/2483 [==============================] - 3s 1ms/step - loss: 0.0027 -
accuracy: 0.9995 - recall: 0.7892 - val_loss: 0.0034 - val_accuracy: 0.9994 -
val_recall: 0.7424
Epoch 16/100
2483/2483 [==============================] - 3s 1ms/step - loss: 0.0037 -
accuracy: 0.9993 - recall: 0.7277 - val_loss: 0.0035 - val_accuracy: 0.9994 -
val_recall: 0.7424
Epoch 17/100
2483/2483 [==============================] - 3s 1ms/step - loss: 0.0032 -
accuracy: 0.9995 - recall: 0.7589 - val_loss: 0.0035 - val_accuracy: 0.9994 -
val_recall: 0.7424
Epoch 18/100
2483/2483 [==============================] - 3s 1ms/step - loss: 0.0034 -
accuracy: 0.9994 - recall: 0.7408 - val_loss: 0.0033 - val_accuracy: 0.9994 -
val_recall: 0.7576
Epoch 19/100
2483/2483 [==============================] - 3s 1ms/step - loss: 0.0029 -
accuracy: 0.9994 - recall: 0.7793 - val_loss: 0.0034 - val_accuracy: 0.9994 -
```

```
val_recall: 0.7424
Epoch 20/100
2483/2483 [==============================] - 4s 1ms/step - loss: 0.0032 -
accuracy: 0.9994 - recall: 0.7677 - val_loss: 0.0035 - val_accuracy: 0.9994 -
val_recall: 0.7424
Epoch 21/100
2483/2483 [==============================] - 3s 1ms/step - loss: 0.0030 -
accuracy: 0.9994 - recall: 0.7624 - val_loss: 0.0035 - val_accuracy: 0.9994 -
val_recall: 0.7424
Epoch 22/100
2483/2483 [==============================] - 3s 1ms/step - loss: 0.0036 -
accuracy: 0.9994 - recall: 0.7579 - val_loss: 0.0033 - val_accuracy: 0.9994 -
val_recall: 0.7576
Epoch 23/100
2483/2483 [==============================] - 3s 1ms/step - loss: 0.0029 -
accuracy: 0.9995 - recall: 0.8062 - val_loss: 0.0036 - val_accuracy: 0.9993 -
val_recall: 0.7727
Epoch 24/100
2483/2483 [==============================] - 3s 1ms/step - loss: 0.0030 -
accuracy: 0.9994 - recall: 0.8028 - val_loss: 0.0034 - val_accuracy: 0.9994 -
val_recall: 0.7424
Epoch 25/100
2483/2483 [==============================] - 3s 1ms/step - loss: 0.0031 -
accuracy: 0.9994 - recall: 0.7903 - val_loss: 0.0033 - val_accuracy: 0.9994 -
val_recall: 0.7727
Epoch 26/100
2483/2483 [==============================] - 3s 1ms/step - loss: 0.0036 -
accuracy: 0.9993 - recall: 0.7570 - val_loss: 0.0033 - val_accuracy: 0.9994 -
val_recall: 0.7424
Epoch 27/100
2483/2483 [==============================] - 3s 1ms/step - loss: 0.0037 -
accuracy: 0.9993 - recall: 0.7327 - val_loss: 0.0033 - val_accuracy: 0.9993 -
val_recall: 0.7727
Epoch 28/100
2483/2483 [==============================] - 3s 1ms/step - loss: 0.0028 -
accuracy: 0.9995 - recall: 0.7866 - val_loss: 0.0032 - val_accuracy: 0.9994 -
val_recall: 0.7576
Epoch 29/100
2483/2483 [==============================] - 3s 1ms/step - loss: 0.0024 -
accuracy: 0.9995 - recall: 0.8040 - val_loss: 0.0032 - val_accuracy: 0.9994 -
val_recall: 0.7576
Epoch 30/100
2483/2483 [==============================] - 3s 1ms/step - loss: 0.0029 -
accuracy: 0.9995 - recall: 0.8231 - val_loss: 0.0032 - val_accuracy: 0.9994 -
val_recall: 0.7576
Epoch 31/100
2483/2483 [==============================] - 3s 1ms/step - loss: 0.0029 -
accuracy: 0.9994 - recall: 0.7851 - val_loss: 0.0036 - val_accuracy: 0.9993 -
```

```
val_recall: 0.7727
Epoch 32/100
2483/2483 [==============================] - 3s 1ms/step - loss: 0.0026 -
accuracy: 0.9995 - recall: 0.8092 - val_loss: 0.0031 - val_accuracy: 0.9994 -
val_recall: 0.7576
Epoch 33/100
2483/2483 [==============================] - 3s 1ms/step - loss: 0.0029 -
accuracy: 0.9994 - recall: 0.7788 - val_loss: 0.0031 - val_accuracy: 0.9994 -
val_recall: 0.7576
Epoch 34/100
2483/2483 [==============================] - 3s 1ms/step - loss: 0.0027 -
accuracy: 0.9994 - recall: 0.8041 - val_loss: 0.0033 - val_accuracy: 0.9994 -
val_recall: 0.7576
Epoch 35/100
2483/2483 [==============================] - 4s 1ms/step - loss: 0.0027 -
accuracy: 0.9994 - recall: 0.8118 - val_loss: 0.0031 - val_accuracy: 0.9994 -
val_recall: 0.7576
Epoch 36/100
2483/2483 [==============================] - 3s 1ms/step - loss: 0.0033 -
accuracy: 0.9994 - recall: 0.7855 - val_loss: 0.0031 - val_accuracy: 0.9994 -
val_recall: 0.7576
Epoch 37/100
2483/2483 [==============================] - 3s 1ms/step - loss: 0.0024 -
accuracy: 0.9995 - recall: 0.8238 - val_loss: 0.0031 - val_accuracy: 0.9994 -
val_recall: 0.7576
Epoch 38/100
2483/2483 [==============================] - 3s 1ms/step - loss: 0.0032 -
accuracy: 0.9994 - recall: 0.7550 - val_loss: 0.0032 - val_accuracy: 0.9994 -
val_recall: 0.7576
Epoch 39/100
2483/2483 [==============================] - 3s 1ms/step - loss: 0.0028 -
accuracy: 0.9994 - recall: 0.7657 - val_loss: 0.0034 - val_accuracy: 0.9994 -
val_recall: 0.7424
Epoch 40/100
2483/2483 [==============================] - 3s 1ms/step - loss: 0.0032 -
accuracy: 0.9994 - recall: 0.7716 - val_loss: 0.0031 - val_accuracy: 0.9994 -
val_recall: 0.7576
Epoch 41/100
2483/2483 [==============================] - 3s 1ms/step - loss: 0.0031 -
accuracy: 0.9995 - recall: 0.7686 - val_loss: 0.0031 - val_accuracy: 0.9994 -
val_recall: 0.7576
Epoch 42/100
2483/2483 [==============================] - 3s 1ms/step - loss: 0.0029 -
accuracy: 0.9993 - recall: 0.8042 - val_loss: 0.0030 - val_accuracy: 0.9994 -
val_recall: 0.7576
Epoch 43/100
2483/2483 [==============================] - 3s 1ms/step - loss: 0.0029 -
accuracy: 0.9995 - recall: 0.8035 - val_loss: 0.0031 - val_accuracy: 0.9995 -
```

```
val_recall: 0.7424
Epoch 44/100
2483/2483 [==============================] - 4s 1ms/step - loss: 0.0033 -
accuracy: 0.9993 - recall: 0.7645 - val_loss: 0.0031 - val_accuracy: 0.9994 -
val_recall: 0.7576
Epoch 45/100
2483/2483 [==============================] - 3s 1ms/step - loss: 0.0026 -
accuracy: 0.9995 - recall: 0.8132 - val_loss: 0.0030 - val_accuracy: 0.9994 -
val_recall: 0.7576
Epoch 46/100
2483/2483 [==============================] - 3s 1ms/step - loss: 0.0023 -
accuracy: 0.9995 - recall: 0.8124 - val_loss: 0.0031 - val_accuracy: 0.9994 -
val_recall: 0.7576
Epoch 47/100
2483/2483 [==============================] - 3s 1ms/step - loss: 0.0032 -
accuracy: 0.9994 - recall: 0.7731 - val_loss: 0.0034 - val_accuracy: 0.9994 -
val_recall: 0.7273
Epoch 48/100
2483/2483 [==============================] - 3s 1ms/step - loss: 0.0030 -
accuracy: 0.9994 - recall: 0.7545 - val_loss: 0.0030 - val_accuracy: 0.9995 -
val_recall: 0.7424
Epoch 49/100
2483/2483 [==============================] - 3s 1ms/step - loss: 0.0031 -
accuracy: 0.9994 - recall: 0.7575 - val_loss: 0.0034 - val_accuracy: 0.9994 -
val_recall: 0.7424
Epoch 50/100
2483/2483 [==============================] - 3s 1ms/step - loss: 0.0026 -
accuracy: 0.9994 - recall: 0.7885 - val_loss: 0.0030 - val_accuracy: 0.9995 -
val_recall: 0.7424
Epoch 51/100
2483/2483 [==============================] - 3s 1ms/step - loss: 0.0027 -
accuracy: 0.9994 - recall: 0.7813 - val_loss: 0.0030 - val_accuracy: 0.9995 -
val_recall: 0.7576
Epoch 52/100
2483/2483 [==============================] - 3s 1ms/step - loss: 0.0033 -
accuracy: 0.9994 - recall: 0.7995 - val_loss: 0.0038 - val_accuracy: 0.9992 -
val_recall: 0.7727
Epoch 53/100
2483/2483 [==============================] - 3s 1ms/step - loss: 0.0026 -
accuracy: 0.9995 - recall: 0.7607 - val_loss: 0.0032 - val_accuracy: 0.9994 -
val_recall: 0.7576
Epoch 54/100
2483/2483 [==============================] - 3s 1ms/step - loss: 0.0029 -
accuracy: 0.9994 - recall: 0.7302 - val_loss: 0.0031 - val_accuracy: 0.9995 -
val_recall: 0.7424
Epoch 55/100
2483/2483 [==============================] - 3s 1ms/step - loss: 0.0035 -
accuracy: 0.9994 - recall: 0.7408 - val_loss: 0.0035 - val_accuracy: 0.9994 -
```

```
val_recall: 0.7424
Epoch 56/100
2483/2483 [==============================] - 3s 1ms/step - loss: 0.0024 -
accuracy: 0.9996 - recall: 0.7988 - val_loss: 0.0029 - val_accuracy: 0.9994 -
val_recall: 0.7576
Epoch 57/100
2483/2483 [==============================] - 3s 1ms/step - loss: 0.0032 -
accuracy: 0.9994 - recall: 0.7400 - val_loss: 0.0033 - val_accuracy: 0.9995 -
val_recall: 0.7424
Epoch 58/100
2483/2483 [==============================] - 3s 1ms/step - loss: 0.0026 -
accuracy: 0.9995 - recall: 0.7961 - val_loss: 0.0030 - val_accuracy: 0.9994 -
val_recall: 0.7576
Epoch 59/100
2483/2483 [==============================] - 3s 1ms/step - loss: 0.0027 -
accuracy: 0.9994 - recall: 0.7496 - val_loss: 0.0031 - val_accuracy: 0.9995 -
val_recall: 0.7576
Epoch 60/100
2483/2483 [==============================] - 3s 1ms/step - loss: 0.0027 -
accuracy: 0.9995 - recall: 0.7745 - val_loss: 0.0035 - val_accuracy: 0.9995 -
val_recall: 0.7424
Epoch 61/100
2483/2483 [==============================] - 3s 1ms/step - loss: 0.0029 -
accuracy: 0.9994 - recall: 0.7671 - val_loss: 0.0032 - val_accuracy: 0.9994 -
val_recall: 0.7576
Epoch 62/100
2483/2483 [==============================] - 3s 1ms/step - loss: 0.0028 -
accuracy: 0.9995 - recall: 0.7843 - val_loss: 0.0029 - val_accuracy: 0.9994 -
val_recall: 0.7576
Epoch 63/100
2483/2483 [==============================] - 3s 1ms/step - loss: 0.0032 -
accuracy: 0.9993 - recall: 0.7371 - val_loss: 0.0031 - val_accuracy: 0.9994 -
val_recall: 0.7576
Epoch 64/100
2483/2483 [==============================] - 3s 1ms/step - loss: 0.0038 -
accuracy: 0.9992 - recall: 0.6999 - val_loss: 0.0032 - val_accuracy: 0.9995 -
val_recall: 0.7576
Epoch 65/100
2483/2483 [==============================] - 3s 1ms/step - loss: 0.0030 -
accuracy: 0.9994 - recall: 0.7800 - val_loss: 0.0034 - val_accuracy: 0.9995 -
val_recall: 0.7424
Epoch 66/100
2483/2483 [==============================] - 3s 1ms/step - loss: 0.0029 -
accuracy: 0.9994 - recall: 0.7620 - val_loss: 0.0032 - val_accuracy: 0.9994 -
val_recall: 0.7424
Epoch 67/100
2483/2483 [==============================] - 3s 1ms/step - loss: 0.0028 -
accuracy: 0.9995 - recall: 0.8107 - val_loss: 0.0034 - val_accuracy: 0.9993 -
```

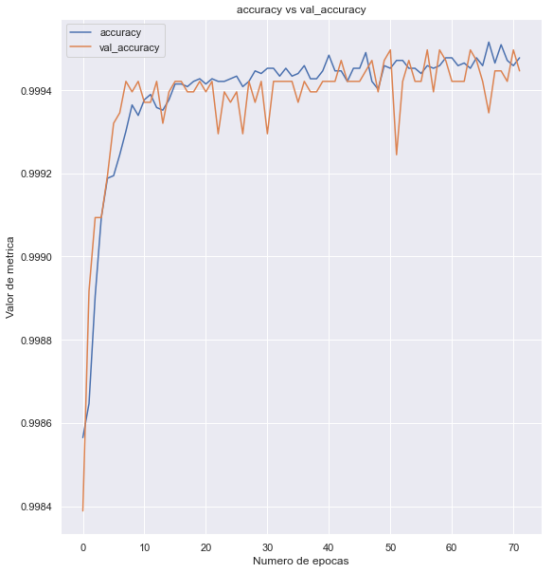
```
val_recall: 0.7727
Epoch 68/100
2483/2483 [==============================] - 3s 1ms/step - loss: 0.0031 -
accuracy: 0.9994 - recall: 0.7189 - val_loss: 0.0031 - val_accuracy: 0.9994 -
val_recall: 0.7273
Epoch 69/100
2483/2483 [==============================] - 3s 1ms/step - loss: 0.0025 -
accuracy: 0.9996 - recall: 0.8038 - val_loss: 0.0033 - val_accuracy: 0.9994 -
val_recall: 0.7273
Epoch 70/100
2483/2483 [==============================] - 3s 1ms/step - loss: 0.0025 -
accuracy: 0.9995 - recall: 0.7832 - val_loss: 0.0032 - val_accuracy: 0.9994 -
val_recall: 0.7576
Epoch 71/100
2483/2483 [==============================] - 3s 1ms/step - loss: 0.0028 -
accuracy: 0.9994 - recall: 0.7767 - val_loss: 0.0032 - val_accuracy: 0.9995 -
val_recall: 0.7576
Epoch 72/100
2483/2483 [==============================] - 3s 1ms/step - loss: 0.0024 -
accuracy: 0.9995 - recall: 0.8394 - val_loss: 0.0030 - val_accuracy: 0.9994 -
val_recall: 0.7273
```

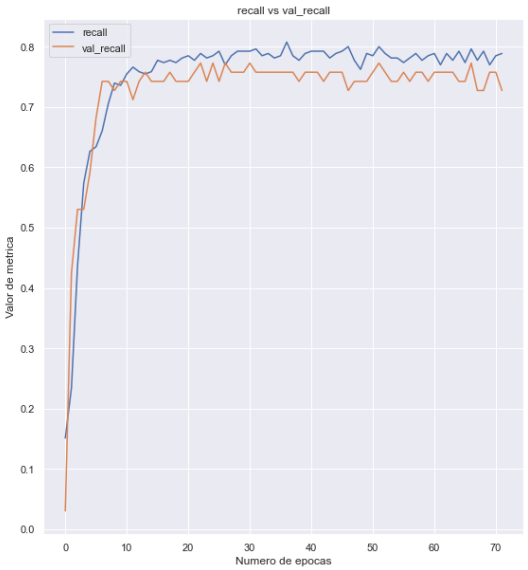
Obtenemos algunas graficas de rendimiento tanto en recall como loss. Para ver mas graficas tenemos a **tensorboard** que es mucho mejor que una version estatica de matplotlib.

[121]:
```
def␣
 ↪plot_metrics(history,metrics=["accuracy","val_accuracy"],kind_type="line",ax=None):
 ↪
    import pandas as pd #por si no esta previamente importada
    data_metrics=pd.DataFrame(history)
    if ax is None:
        _,(ax)=plt.subplots(1,1)
    ax.set_title(" vs ".join(metrics))
    ax.set_xlabel("Numero de epocas")
    ax.set_ylabel("Valor de metrica")
    data_metrics[metrics].plot(kind=kind_type,ax=ax)
```

- Veamos el contraste de el accuracy en entrenamiento y validacion

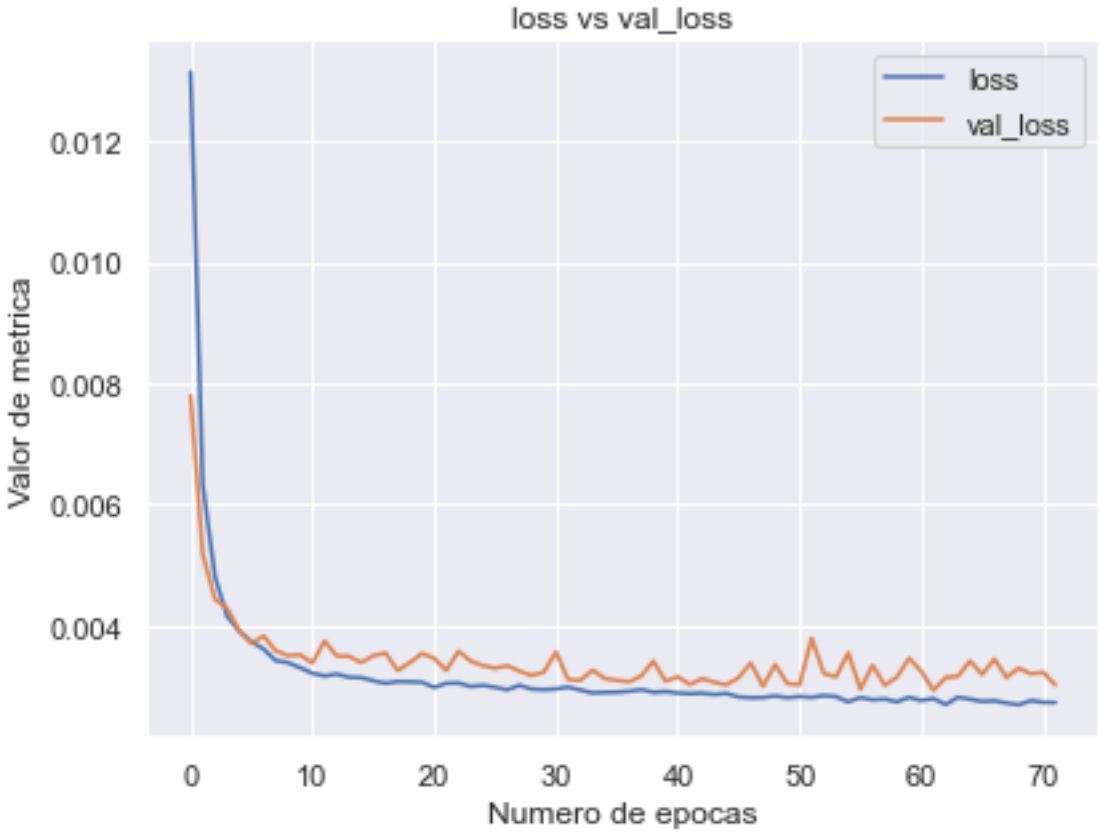
[127]:
```
fig,(ax1,ax2)=plt.subplots(1,2,figsize=(20,10))

plot_metrics(history.
 ↪history,metrics=["accuracy","val_accuracy"],kind_type="line",ax=ax1)
plot_metrics(history.
 ↪history,metrics=["recall","val_recall"],kind_type="line",ax=ax2)
```





Veamos la evolucion de la funcion de perdida

[126]:
```
plot_metrics(history.history,metrics=["loss","val_loss"])
```



Validacion: MODELO BASE

[149]:
```
base_model.evaluate(x_val,y_val)
```

```
1242/1242 [==============================] - 1s 763us/step - loss: 0.0030 -
accuracy: 0.9994 - recall: 0.7273
```

[149]:
```
[0.003015657654032111, 0.9994461536407471, 0.7272727489471436]
```

Evaluamos el modelo en el conjunto de validacion y obtenemos un **accuracy 99.99%** y **recall 72%**

Le agregamos metricas classificacion para ver el **rendimiento sobre los datos de validacion**

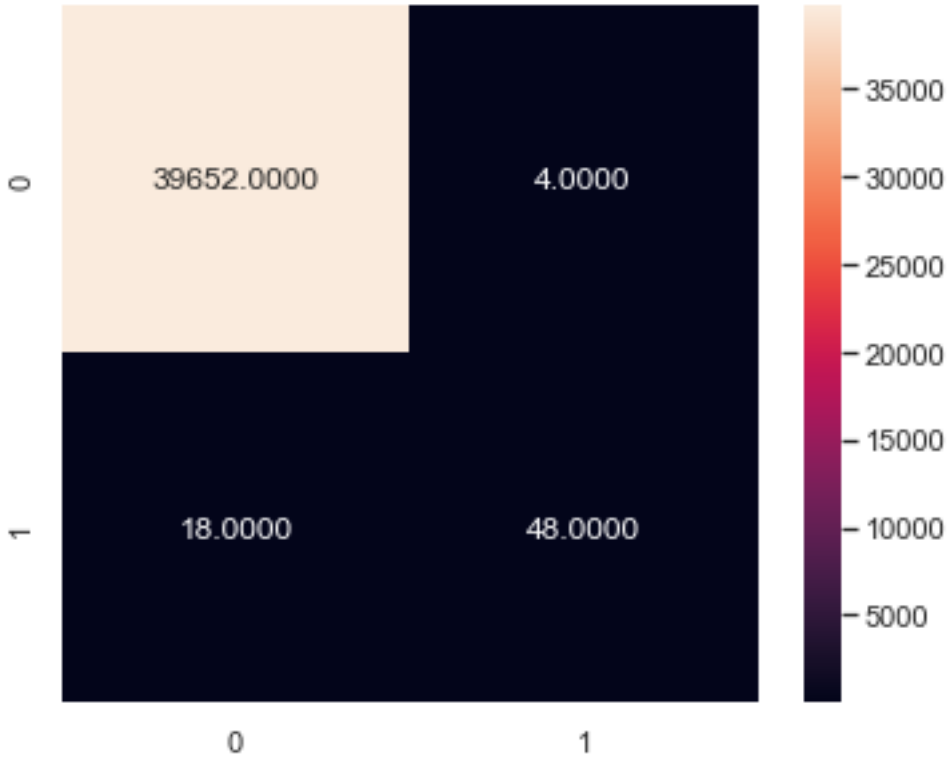
[150]:
```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
import seaborn as sns; sns.set()

#ahora creamos una funcion que nos muestra la matriz de confusion y el reporte␣
 ↪de clasificacion

def display_report_and_matrix_confusion(model,X,y):

    y_pred=model.predict(X)
    y_pred=np.where(y_pred>=0.5,1,0) #establecemos el umbral como 0.5
    report=classification_report(y,y_pred)
    print("REPORTE DE CLASIFICACION")
    print(report)
    print("\n Matriz de confusion")
    mat=confusion_matrix(y,y_pred)
    sns.heatmap(mat,annot=True,square=True,fmt="0.4f")
    plt.show()
```

vemos la matriz de confusion para ver como van los resultados

[151]:
```
display_report_and_matrix_confusion(base_model,x_val,y_val)
```

```
REPORTE DE CLASIFICACION
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00     39656
         1.0       0.92      0.73      0.81        66

    accuracy                           1.00     39722
   macro avg       0.96      0.86      0.91     39722
weighted avg       1.00      1.00      1.00     39722
```

```
Matriz de confusion
```



El modelo base tiene buenos resultados en el conjunto de validacion. Al ser bueno lo llevaremos a la semifinal de modelos si logra pasar las expectativas en el conjunto de **Test**.

Entonces para ellos cargamos los datos de **Test** para evaluarlos y ver la realidad

EVALUACION DEL MODELO: MODELO BASE

- Esta etapa solo es necesaria si es que el modelo tiene buen rendemiento en los datos de validacion y es definitivo, en caso contrario se necesita explorar nuevas configuraciones y arquitecturas del modelo.Hasta lograr un puntaje decente que conlleve al equilibrio entre el subajuste y sobreajuste

[136]:
```
x_test,y_test=load_data("./creditcard_imbalanced/test.csv")
```

[137]:
```
x_test=scaler_imb.transform(x_test) #los escalamos con los datos de␣
↪entrenamiento
```

Evaluamos el modelo

[139]:
```
base_model.evaluate(x_test,y_test)
```

```
2660/2660 [==============================] - 2s 819us/step - loss: 0.0034 -
accuracy: 0.9993 - recall: 0.6549
```

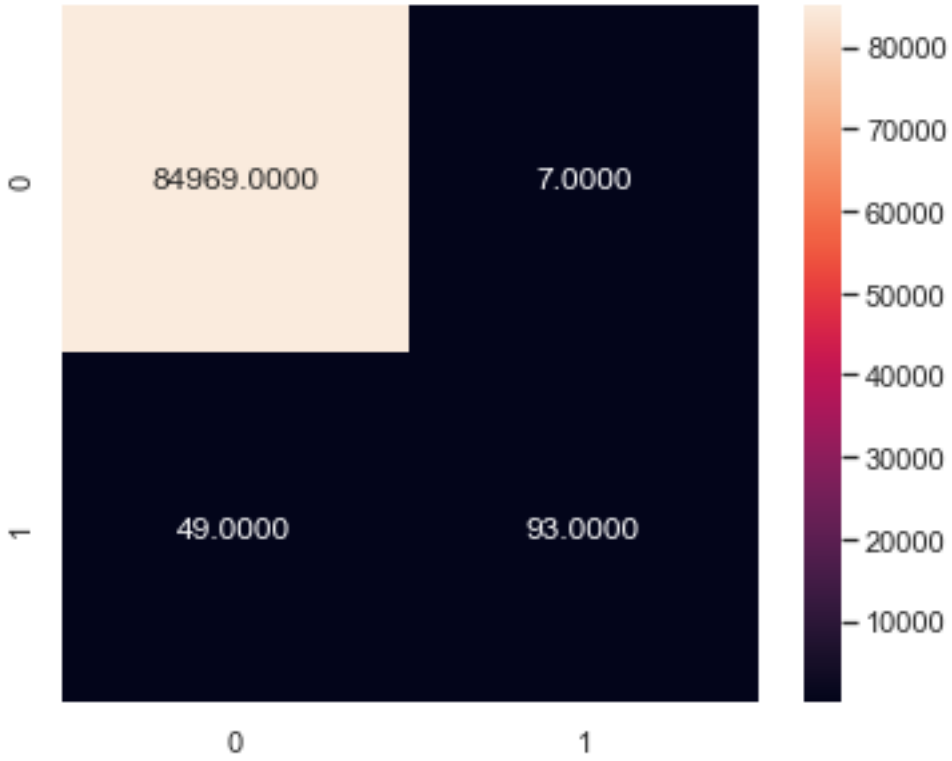
[139]:
```
[0.003442350309342146, 0.9993420839309692, 0.6549295783042908]
```

Estamos viendo como tiene una **precision del 99%** pero al final un **recall de 65%**

[152]:
```
display_report_and_matrix_confusion(base_model,x_test,y_test)
```

```
REPORTE DE CLASIFICACION
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00     84976
         1.0       0.93      0.65      0.77       142

    accuracy                           1.00     85118
   macro avg       0.96      0.83      0.88     85118
weighted avg       1.00      1.00      1.00     85118
```

Matriz de confusion



Nada mal para ser el modelo base con solo 49 falsos negativos luego de 80 000 muestas de test.

Los modelos que se implementen despues tienen que ser mucho mejor que este.

- Guardamos el modelo con el formato **SavedModel** de tensorflow para contrastarlo luego con los demas modelo

[154]:
```
tf.saved_model.save(base_model,"./best_endpoints/base_model")
```

```
INFO:tensorflow:Assets written to: ./best_endpoints/base_model\assets
```

Guardamos las medidas de escalamiento

[178]:
```
import joblib

joblib.dump(scaler_imb,"./best_endpoints/scaler_imbalanced.pkl")
```

[178]:
```
['./best_endpoints/scaler_imbalanced.pkl']
```

0.2.2 MODELO V1.0 CON DATOS SOBREMUESTREADOS

- Una vez listo el modelo base podemos buscar manera de equilibrar el conjunto de datos, por lo que haremos un submuestreo de datos en favor a la clase minoritaria

[155]:
```
x_train,y_train=load_data("./creditcard_oversampling/train.csv")
```

- Luego de los datos balanceados a partir del conjunto de entrenamiento, ahora podemos separar los datos de validacion y train y escalarlos

[156]:
```
scale_over,x_train,x_val,y_train,y_val=partition_data(x_train,y_train,with_scale=True)
```

- Ya equilibramos los datos. ahora realizaremos la busqueda de los mejores hiperparametros para la capa oculta del modelo con **keras tunner**

[157]:
```
import kerastuner as kt
import tensorflow as tf
n_features=x_train.shape[1] #el numero de columnas como dimension de entrada
STEP_UNITS=16 #las pasos de unidades de capa densa para probar los mejores␣
 ↪parametros
def build_model_tuning(hp):
    model=tf.keras.models.Sequential()
    model.add(tf.keras.Input(shape=(n_features,),name="Input"))
    model.add(tf.keras.layers.Dense(
                      units=hp.
 ↪Int("units",min_value=32,max_value=512,step=STEP_UNITS),
                      activation="relu",name="HIDDEN_LAYER"
        ))
    model.add(tf.keras.layers.Dense(1,activation="sigmoid",name="output"))
    #configuracion del modelo

    model.compile(optimizer=tf.keras.optimizers.Adam(lr=1e-3),
                  loss=tf.keras.losses.BinaryCrossentropy(),
                  metrics=metrics
                 )
```

```
    return model
```

Creamos una instancia del sintonizador y realizamos el hipertuning

[158]:
```
tuning=kt.Hyperband(
            build_model_tuning,
            objective=kt.Objective("val_recall", direction="max"),
            max_epochs=10,
            factor=3,
            directory="./hipertuning/",
            project_name="creditcard_tuner",
            overwrite=True
        )
```

Realizamos el hipertuning del modelo

[159]:
```
early_stop=tf.keras.callbacks.EarlyStopping(monitor="val_loss",patience=5)
tensorboard_callbacks=get_callback_tensorboard("oversample_model")
tuning.
 ↪search(x_train,y_train,epochs=100,batch_size=64,validation_data=(x_val,y_val),callbacks=[ea
```

```
Trial 30 Complete [00h 01m 15s]
val_recall: 0.979019045829773

Best val_recall So Far: 0.9882486462593079
Total elapsed time: 00h 15m 32s
INFO:tensorflow:Oracle triggered exit
```

Luego de la busqueda exhaustiva de hiperparametros, nuestro sintonizador nos devuelve los mejores hiperarametros.

[160]:
```
best_hps=tuning.get_best_hyperparameters(num_trials=1)[0]
print("El mejor numero de neuronas para la capa densa es de {0}".
 ↪format(best_hps.get("units")))
```

```
El mejor numero de neuronas para la capa densa es de 176
```

- Entrene el modelo con los datos obtenidos de mejores hiperametros, esto con la finalidad de que ahora se obtengan el mejor numero de epocas de entrenamiento que tienen el **recall** mas alto

[161]:
```
best_model=tuning.hypermodel.build(best_hps)
early_stop=tf.keras.callbacks.EarlyStopping(monitor="val_loss",patience=5) #se␣
 ↪detiene cuando la perdida de validacion no mejora
history=best_model.
 ↪fit(x_train,y_train,epochs=100,batch_size=64,validation_data=(x_val,y_val),callbacks=[early
 ↪ #mantenemos los callbacks para ahorrar tiempo en caso las metricas no␣
 ↪mejoren a medida que avancen los epochs
```

```
Epoch 1/100
4957/4957 [==============================] - 7s 1ms/step - loss: 0.2066 -
accuracy: 0.9508 - recall: 0.9234 - val_loss: 0.1487 - val_accuracy: 0.9460 -
val_recall: 0.9001
Epoch 2/100
4957/4957 [==============================] - 7s 1ms/step - loss: 0.1263 -
accuracy: 0.9497 - recall: 0.9244 - val_loss: 0.1133 - val_accuracy: 0.9540 -
val_recall: 0.9287
Epoch 3/100
4957/4957 [==============================] - 7s 1ms/step - loss: 0.1145 -
accuracy: 0.9547 - recall: 0.9333 - val_loss: 0.1044 - val_accuracy: 0.9598 -
val_recall: 0.9485
Epoch 4/100
4957/4957 [==============================] - 7s 1ms/step - loss: 0.1061 -
accuracy: 0.9582 - recall: 0.9406 - val_loss: 0.0976 - val_accuracy: 0.9619 -
val_recall: 0.9562
Epoch 5/100
4957/4957 [==============================] - 7s 1ms/step - loss: 0.1003 -
accuracy: 0.9610 - recall: 0.9460 - val_loss: 0.0946 - val_accuracy: 0.9648 -
val_recall: 0.9637
Epoch 6/100
4957/4957 [==============================] - 7s 1ms/step - loss: 0.0945 -
accuracy: 0.9636 - recall: 0.9497 - val_loss: 0.0852 - val_accuracy: 0.9680 -
val_recall: 0.9598
Epoch 7/100
4957/4957 [==============================] - 7s 1ms/step - loss: 0.0902 -
accuracy: 0.9658 - recall: 0.9533 - val_loss: 0.0850 - val_accuracy: 0.9652 -
val_recall: 0.9433
Epoch 8/100
4957/4957 [==============================] - 7s 1ms/step - loss: 0.0863 -
accuracy: 0.9669 - recall: 0.9552 - val_loss: 0.0771 - val_accuracy: 0.9699 -
val_recall: 0.9574
Epoch 9/100
4957/4957 [==============================] - 7s 1ms/step - loss: 0.0828 -
accuracy: 0.9681 - recall: 0.9572 - val_loss: 0.0804 - val_accuracy: 0.9675 -
val_recall: 0.9429
Epoch 10/100
4957/4957 [==============================] - 7s 1ms/step - loss: 0.0782 -
accuracy: 0.9695 - recall: 0.9592 - val_loss: 0.0850 - val_accuracy: 0.9698 -
val_recall: 0.9819
Epoch 11/100
4957/4957 [==============================] - 7s 1ms/step - loss: 0.0750 -
accuracy: 0.9710 - recall: 0.9616 - val_loss: 0.0828 - val_accuracy: 0.9660 -
val_recall: 0.9375
Epoch 12/100
4957/4957 [==============================] - 7s 1ms/step - loss: 0.0719 -
accuracy: 0.9723 - recall: 0.9631 - val_loss: 0.0661 - val_accuracy: 0.9726 -
val_recall: 0.9573
```

```
Epoch 13/100
4957/4957 [==============================] - 7s 1ms/step - loss: 0.0684 -
accuracy: 0.9732 - recall: 0.9650 - val_loss: 0.0639 - val_accuracy: 0.9779 -
val_recall: 0.9766
Epoch 14/100
4957/4957 [==============================] - 7s 1ms/step - loss: 0.0657 -
accuracy: 0.9746 - recall: 0.9672 - val_loss: 0.0644 - val_accuracy: 0.9760 -
val_recall: 0.9748
Epoch 15/100
4957/4957 [==============================] - 6s 1ms/step - loss: 0.0627 -
accuracy: 0.9755 - recall: 0.9688 - val_loss: 0.0595 - val_accuracy: 0.9748 -
val_recall: 0.9596
Epoch 16/100
4957/4957 [==============================] - 7s 1ms/step - loss: 0.0609 -
accuracy: 0.9767 - recall: 0.9703 - val_loss: 0.0523 - val_accuracy: 0.9807 -
val_recall: 0.9794
Epoch 17/100
4957/4957 [==============================] - 6s 1ms/step - loss: 0.0561 -
accuracy: 0.9784 - recall: 0.9732 - val_loss: 0.0486 - val_accuracy: 0.9825 -
val_recall: 0.9769
Epoch 18/100
4957/4957 [==============================] - 6s 1ms/step - loss: 0.0527 -
accuracy: 0.9799 - recall: 0.9756 - val_loss: 0.0462 - val_accuracy: 0.9821 -
val_recall: 0.9799
Epoch 19/100
4957/4957 [==============================] - 7s 1ms/step - loss: 0.0502 -
accuracy: 0.9805 - recall: 0.9762 - val_loss: 0.0426 - val_accuracy: 0.9830 -
val_recall: 0.9781
Epoch 20/100
4957/4957 [==============================] - 6s 1ms/step - loss: 0.0493 -
accuracy: 0.9813 - recall: 0.9778 - val_loss: 0.0446 - val_accuracy: 0.9809 -
val_recall: 0.9711
Epoch 21/100
4957/4957 [==============================] - 6s 1ms/step - loss: 0.0467 -
accuracy: 0.9822 - recall: 0.9791 - val_loss: 0.0524 - val_accuracy: 0.9748 -
val_recall: 0.9542
Epoch 22/100
4957/4957 [==============================] - 6s 1ms/step - loss: 0.0451 -
accuracy: 0.9827 - recall: 0.9796 - val_loss: 0.0393 - val_accuracy: 0.9859 -
val_recall: 0.9888
Epoch 23/100
4957/4957 [==============================] - 6s 1ms/step - loss: 0.0432 -
accuracy: 0.9834 - recall: 0.9814 - val_loss: 0.0402 - val_accuracy: 0.9875 -
val_recall: 0.9936
Epoch 24/100
4957/4957 [==============================] - 6s 1ms/step - loss: 0.0406 -
accuracy: 0.9849 - recall: 0.9834 - val_loss: 0.0355 - val_accuracy: 0.9866 -
val_recall: 0.9862
```

```
Epoch 25/100
4957/4957 [==============================] - 6s 1ms/step - loss: 0.0399 -
accuracy: 0.9853 - recall: 0.9839 - val_loss: 0.0351 - val_accuracy: 0.9872 -
val_recall: 0.9831
Epoch 26/100
4957/4957 [==============================] - 6s 1ms/step - loss: 0.0391 -
accuracy: 0.9855 - recall: 0.9843 - val_loss: 0.0344 - val_accuracy: 0.9890 -
val_recall: 0.9957
Epoch 27/100
4957/4957 [==============================] - 6s 1ms/step - loss: 0.0384 -
accuracy: 0.9857 - recall: 0.9841 - val_loss: 0.0353 - val_accuracy: 0.9884 -
val_recall: 0.9961
Epoch 28/100
4957/4957 [==============================] - 6s 1ms/step - loss: 0.0360 -
accuracy: 0.9869 - recall: 0.9860 - val_loss: 0.0309 - val_accuracy: 0.9884 -
val_recall: 0.9846
Epoch 29/100
4957/4957 [==============================] - 6s 1ms/step - loss: 0.0345 -
accuracy: 0.9872 - recall: 0.9864 - val_loss: 0.0336 - val_accuracy: 0.9874 -
val_recall: 0.9799
Epoch 30/100
4957/4957 [==============================] - 6s 1ms/step - loss: 0.0331 -
accuracy: 0.9878 - recall: 0.9874 - val_loss: 0.0286 - val_accuracy: 0.9899 -
val_recall: 0.9873
Epoch 31/100
4957/4957 [==============================] - 6s 1ms/step - loss: 0.0337 -
accuracy: 0.9877 - recall: 0.9876 - val_loss: 0.0392 - val_accuracy: 0.9845 -
val_recall: 0.9839
Epoch 32/100
4957/4957 [==============================] - 6s 1ms/step - loss: 0.0338 -
accuracy: 0.9879 - recall: 0.9875 - val_loss: 0.0274 - val_accuracy: 0.9913 -
val_recall: 0.9969
Epoch 33/100
4957/4957 [==============================] - 6s 1ms/step - loss: 0.0317 -
accuracy: 0.9888 - recall: 0.9890 - val_loss: 0.0249 - val_accuracy: 0.9921 -
val_recall: 0.9923
Epoch 34/100
4957/4957 [==============================] - 6s 1ms/step - loss: 0.0313 -
accuracy: 0.9888 - recall: 0.9885 - val_loss: 0.0431 - val_accuracy: 0.9794 -
val_recall: 0.9611
Epoch 35/100
4957/4957 [==============================] - 6s 1ms/step - loss: 0.0308 -
accuracy: 0.9892 - recall: 0.9894 - val_loss: 0.0344 - val_accuracy: 0.9848 -
val_recall: 0.9739
Epoch 36/100
4957/4957 [==============================] - 7s 1ms/step - loss: 0.0290 -
accuracy: 0.9899 - recall: 0.9905 - val_loss: 0.0382 - val_accuracy: 0.9844 -
val_recall: 0.9731
```

```
Epoch 37/100
4957/4957 [==============================] - 7s 1ms/step - loss: 0.0286 -
accuracy: 0.9901 - recall: 0.9909 - val_loss: 0.0255 - val_accuracy: 0.9924 -
val_recall: 0.9914
Epoch 38/100
4957/4957 [==============================] - 7s 1ms/step - loss: 0.0282 -
accuracy: 0.9900 - recall: 0.9907 - val_loss: 0.0215 - val_accuracy: 0.9940 -
val_recall: 0.9971
Epoch 39/100
4957/4957 [==============================] - 7s 1ms/step - loss: 0.0269 -
accuracy: 0.9907 - recall: 0.9916 - val_loss: 0.0229 - val_accuracy: 0.9929 -
val_recall: 0.9950
Epoch 40/100
4957/4957 [==============================] - 7s 1ms/step - loss: 0.0260 -
accuracy: 0.9912 - recall: 0.9918 - val_loss: 0.0233 - val_accuracy: 0.9929 -
val_recall: 0.9950
Epoch 41/100
4957/4957 [==============================] - 7s 1ms/step - loss: 0.0263 -
accuracy: 0.9911 - recall: 0.9918 - val_loss: 0.0206 - val_accuracy: 0.9935 -
val_recall: 0.9919
Epoch 42/100
4957/4957 [==============================] - 7s 1ms/step - loss: 0.0255 -
accuracy: 0.9910 - recall: 0.9916 - val_loss: 0.0260 - val_accuracy: 0.9894 -
val_recall: 0.9822
Epoch 43/100
4957/4957 [==============================] - 6s 1ms/step - loss: 0.0256 -
accuracy: 0.9910 - recall: 0.9916 - val_loss: 0.0383 - val_accuracy: 0.9834 -
val_recall: 0.9705
Epoch 44/100
4957/4957 [==============================] - 7s 1ms/step - loss: 0.0246 -
accuracy: 0.9914 - recall: 0.9921 - val_loss: 0.0190 - val_accuracy: 0.9943 -
val_recall: 0.9961
Epoch 45/100
4957/4957 [==============================] - 7s 1ms/step - loss: 0.0247 -
accuracy: 0.9911 - recall: 0.9922 - val_loss: 0.0253 - val_accuracy: 0.9913 -
val_recall: 0.9978
Epoch 46/100
4957/4957 [==============================] - 7s 1ms/step - loss: 0.0237 -
accuracy: 0.9918 - recall: 0.9928 - val_loss: 0.0211 - val_accuracy: 0.9940 -
val_recall: 0.9924
Epoch 47/100
4957/4957 [==============================] - 8s 2ms/step - loss: 0.0226 -
accuracy: 0.9924 - recall: 0.9936 - val_loss: 0.0261 - val_accuracy: 0.9888 -
val_recall: 0.9821
Epoch 48/100
4957/4957 [==============================] - 6s 1ms/step - loss: 0.0224 -
accuracy: 0.9924 - recall: 0.9933 - val_loss: 0.0192 - val_accuracy: 0.9940 -
val_recall: 0.9970
```

```
Epoch 49/100
4957/4957 [==============================] - 7s 1ms/step - loss: 0.0221 -
accuracy: 0.9925 - recall: 0.9935 - val_loss: 0.0196 - val_accuracy: 0.9933 -
val_recall: 0.9900
```

Una vez culminado el entrenamiento procedemos a identificar el **epoch** donde obtuvo mas puntaje la metrica **RECALL** que es recuperacion de precision. Esto calcula pares de recuperacion/precision para diferentes umbrales de probabilidad.

Observacion: En el entrenamiento anterior el entrenamiento se detuvo en el epochs=49 debido a que **val_loss** no mejoraba por mas de 10 epocas

[162]:
```
n_epochs=np.argmax(history.history["val_recall"])+1 #busca la ubicacion que␣
↪posee el valor mas alto basandose en su indice+1
print(f"El numero adecuado de epocas de entrenamiento es de : {n_epochs}")
```

```
El numero adecuado de epocas de entrenamiento es de : 45
```

Esta vez continuamos el entrenamiento sin callbacks EarlyStpping debido a que no tenemos nada que monitorear para la mejora. Tenemos los mejores hiperparametros y el numero de epocas apropiado. Ahora si, a realizar un entrenamiento limpio.

[163]:
```
tensorboard_callback=get_callback_tensorboard("oversample_model") #incluimos␣
↪tensorboard para mostrar la grafica asociada al entrenamiento
best_model=tuning.hypermodel.build(best_hps)
history=best_model.
↪fit(x_train,y_train,epochs=n_epochs,batch_size=64,validation_data=(x_val,y_val),callbacks=[
```

```
Epoch 1/45
4957/4957 [==============================] - 8s 2ms/step - loss: 0.2092 -
accuracy: 0.9571 - recall: 0.9264 - val_loss: 0.1283 - val_accuracy: 0.9501 -
val_recall: 0.9358
Epoch 2/45
4957/4957 [==============================] - 7s 1ms/step - loss: 0.1277 -
accuracy: 0.9498 - recall: 0.9244 - val_loss: 0.1177 - val_accuracy: 0.9541 -
val_recall: 0.9455
Epoch 3/45
4957/4957 [==============================] - 7s 1ms/step - loss: 0.1160 -
accuracy: 0.9540 - recall: 0.9325 - val_loss: 0.1025 - val_accuracy: 0.9581 -
val_recall: 0.9387
Epoch 4/45
4957/4957 [==============================] - 7s 1ms/step - loss: 0.1057 -
accuracy: 0.9591 - recall: 0.9410 - val_loss: 0.1013 - val_accuracy: 0.9597 -
val_recall: 0.9284
Epoch 5/45
4957/4957 [==============================] - 7s 1ms/step - loss: 0.0985 -
accuracy: 0.9613 - recall: 0.9463 - val_loss: 0.0898 - val_accuracy: 0.9654 -
val_recall: 0.9456
Epoch 6/45
4957/4957 [==============================] - 7s 1ms/step - loss: 0.0923 -
```

```
accuracy: 0.9644 - recall: 0.9513 - val_loss: 0.0932 - val_accuracy: 0.9630 -
val_recall: 0.9335
Epoch 7/45
4957/4957 [==============================] - 7s 1ms/step - loss: 0.0878 -
accuracy: 0.9659 - recall: 0.9536 - val_loss: 0.0798 - val_accuracy: 0.9678 -
val_recall: 0.9491
Epoch 8/45
4957/4957 [==============================] - 7s 1ms/step - loss: 0.0818 -
accuracy: 0.9682 - recall: 0.9580 - val_loss: 0.0851 - val_accuracy: 0.9647 -
val_recall: 0.9377
Epoch 9/45
4957/4957 [==============================] - 7s 1ms/step - loss: 0.0783 -
accuracy: 0.9702 - recall: 0.9604 - val_loss: 0.0724 - val_accuracy: 0.9709 -
val_recall: 0.9536
Epoch 10/45
4957/4957 [==============================] - 7s 1ms/step - loss: 0.0734 -
accuracy: 0.9713 - recall: 0.9624 - val_loss: 0.0648 - val_accuracy: 0.9750 -
val_recall: 0.9682
Epoch 11/45
4957/4957 [==============================] - 7s 1ms/step - loss: 0.0702 -
accuracy: 0.9728 - recall: 0.9648 - val_loss: 0.0745 - val_accuracy: 0.9735 -
val_recall: 0.9878
Epoch 12/45
4957/4957 [==============================] - 7s 1ms/step - loss: 0.0639 -
accuracy: 0.9751 - recall: 0.9681 - val_loss: 0.0601 - val_accuracy: 0.9752 -
val_recall: 0.9619
Epoch 13/45
4957/4957 [==============================] - 7s 1ms/step - loss: 0.0616 -
accuracy: 0.9762 - recall: 0.9697 - val_loss: 0.0581 - val_accuracy: 0.9757 -
val_recall: 0.9638
Epoch 14/45
4957/4957 [==============================] - 7s 1ms/step - loss: 0.0570 -
accuracy: 0.9780 - recall: 0.9725 - val_loss: 0.0547 - val_accuracy: 0.9801 -
val_recall: 0.9842
Epoch 15/45
4957/4957 [==============================] - 7s 1ms/step - loss: 0.0545 -
accuracy: 0.9791 - recall: 0.9745 - val_loss: 0.0474 - val_accuracy: 0.9821 -
val_recall: 0.9776
Epoch 16/45
4957/4957 [==============================] - 7s 1ms/step - loss: 0.0524 -
accuracy: 0.9799 - recall: 0.9757 - val_loss: 0.1185 - val_accuracy: 0.9552 -
val_recall: 0.9118
Epoch 17/45
4957/4957 [==============================] - 7s 1ms/step - loss: 0.0507 -
accuracy: 0.9804 - recall: 0.9764 - val_loss: 0.0439 - val_accuracy: 0.9848 -
val_recall: 0.9877
Epoch 18/45
4957/4957 [==============================] - 7s 1ms/step - loss: 0.0481 -
```

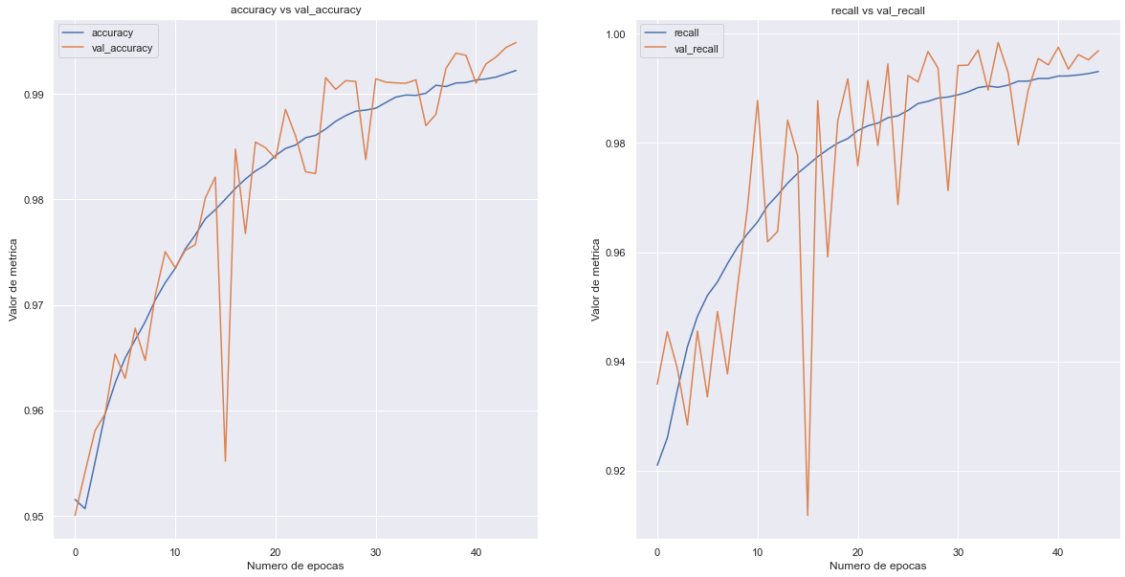
```
accuracy: 0.9813 - recall: 0.9779 - val_loss: 0.0523 - val_accuracy: 0.9768 -
val_recall: 0.9591
Epoch 19/45
4957/4957 [==============================] - 7s 1ms/step - loss: 0.0463 -
accuracy: 0.9820 - recall: 0.9790 - val_loss: 0.0385 - val_accuracy: 0.9854 -
val_recall: 0.9840
Epoch 20/45
4957/4957 [==============================] - 7s 1ms/step - loss: 0.0427 -
accuracy: 0.9837 - recall: 0.9814 - val_loss: 0.0434 - val_accuracy: 0.9849 -
val_recall: 0.9917
Epoch 21/45
4957/4957 [==============================] - 7s 1ms/step - loss: 0.0421 -
accuracy: 0.9844 - recall: 0.9824 - val_loss: 0.0410 - val_accuracy: 0.9839 -
val_recall: 0.9758
Epoch 22/45
4957/4957 [==============================] - 7s 1ms/step - loss: 0.0398 -
accuracy: 0.9848 - recall: 0.9826 - val_loss: 0.0342 - val_accuracy: 0.9885 -
val_recall: 0.9914
Epoch 23/45
4957/4957 [==============================] - 7s 1ms/step - loss: 0.0402 -
accuracy: 0.9848 - recall: 0.9830 - val_loss: 0.0352 - val_accuracy: 0.9860 -
val_recall: 0.9795
Epoch 24/45
4957/4957 [==============================] - 7s 1ms/step - loss: 0.0376 -
accuracy: 0.9858 - recall: 0.9845 - val_loss: 0.0480 - val_accuracy: 0.9826 -
val_recall: 0.9945
Epoch 25/45
4957/4957 [==============================] - 7s 1ms/step - loss: 0.0379 -
accuracy: 0.9857 - recall: 0.9846 - val_loss: 0.0403 - val_accuracy: 0.9825 -
val_recall: 0.9687
Epoch 26/45
4957/4957 [==============================] - 7s 1ms/step - loss: 0.0367 -
accuracy: 0.9862 - recall: 0.9854 - val_loss: 0.0270 - val_accuracy: 0.9916 -
val_recall: 0.9923
Epoch 27/45
4957/4957 [==============================] - 7s 1ms/step - loss: 0.0334 -
accuracy: 0.9876 - recall: 0.9874 - val_loss: 0.0283 - val_accuracy: 0.9904 -
val_recall: 0.9912
Epoch 28/45
4957/4957 [==============================] - 7s 1ms/step - loss: 0.0329 -
accuracy: 0.9878 - recall: 0.9873 - val_loss: 0.0287 - val_accuracy: 0.9913 -
val_recall: 0.9967
Epoch 29/45
4957/4957 [==============================] - 7s 1ms/step - loss: 0.0325 -
accuracy: 0.9881 - recall: 0.9880 - val_loss: 0.0291 - val_accuracy: 0.9912 -
val_recall: 0.9937
Epoch 30/45
4957/4957 [==============================] - 7s 1ms/step - loss: 0.0314 -
```

```
accuracy: 0.9886 - recall: 0.9884 - val_loss: 0.0369 - val_accuracy: 0.9838 -
val_recall: 0.9713
Epoch 31/45
4957/4957 [==============================] - 7s 1ms/step - loss: 0.0316 -
accuracy: 0.9883 - recall: 0.9885 - val_loss: 0.0258 - val_accuracy: 0.9914 -
val_recall: 0.9942
Epoch 32/45
4957/4957 [==============================] - 7s 1ms/step - loss: 0.0294 -
accuracy: 0.9892 - recall: 0.9895 - val_loss: 0.0266 - val_accuracy: 0.9911 -
val_recall: 0.9942
Epoch 33/45
4957/4957 [==============================] - 7s 1ms/step - loss: 0.0287 -
accuracy: 0.9896 - recall: 0.9900 - val_loss: 0.0266 - val_accuracy: 0.9910 -
val_recall: 0.9970
Epoch 34/45
4957/4957 [==============================] - 7s 1ms/step - loss: 0.0292 -
accuracy: 0.9894 - recall: 0.9899 - val_loss: 0.0253 - val_accuracy: 0.9910 -
val_recall: 0.9897
Epoch 35/45
4957/4957 [==============================] - 7s 1ms/step - loss: 0.0273 -
accuracy: 0.9900 - recall: 0.9904 - val_loss: 0.0269 - val_accuracy: 0.9913 -
val_recall: 0.9984
Epoch 36/45
4957/4957 [==============================] - 7s 1ms/step - loss: 0.0275 -
accuracy: 0.9901 - recall: 0.9905 - val_loss: 0.0354 - val_accuracy: 0.9870 -
val_recall: 0.9928
Epoch 37/45
4957/4957 [==============================] - 7s 1ms/step - loss: 0.0269 -
accuracy: 0.9905 - recall: 0.9907 - val_loss: 0.0283 - val_accuracy: 0.9881 -
val_recall: 0.9796
Epoch 38/45
4957/4957 [==============================] - 7s 1ms/step - loss: 0.0260 -
accuracy: 0.9907 - recall: 0.9912 - val_loss: 0.0215 - val_accuracy: 0.9924 -
val_recall: 0.9897
Epoch 39/45
4957/4957 [==============================] - 7s 1ms/step - loss: 0.0259 -
accuracy: 0.9907 - recall: 0.9914 - val_loss: 0.0200 - val_accuracy: 0.9939 -
val_recall: 0.9955
Epoch 40/45
4957/4957 [==============================] - 7s 1ms/step - loss: 0.0250 -
accuracy: 0.9912 - recall: 0.9921 - val_loss: 0.0213 - val_accuracy: 0.9937 -
val_recall: 0.9943
Epoch 41/45
4957/4957 [==============================] - 7s 1ms/step - loss: 0.0252 -
accuracy: 0.9910 - recall: 0.9918 - val_loss: 0.0265 - val_accuracy: 0.9910 -
val_recall: 0.9975
Epoch 42/45
4957/4957 [==============================] - 7s 1ms/step - loss: 0.0252 -
```

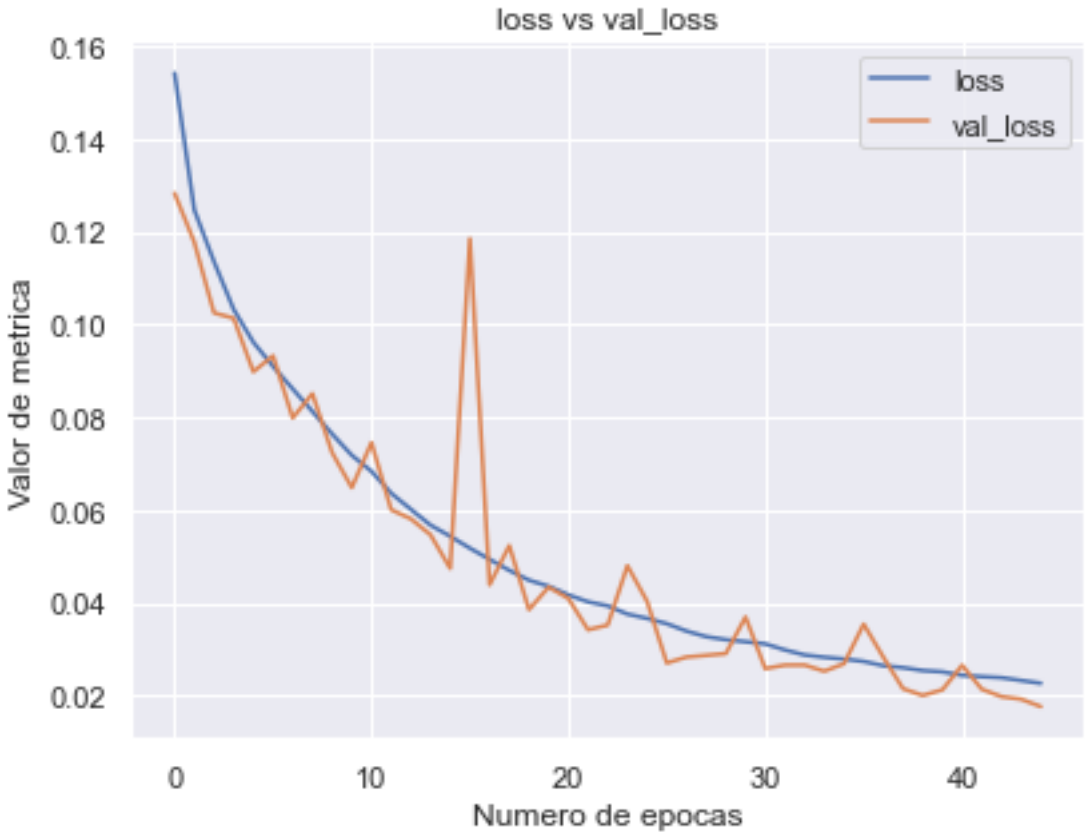
```
accuracy: 0.9910 - recall: 0.9920 - val_loss: 0.0214 - val_accuracy: 0.9929 -
val_recall: 0.9935
Epoch 43/45
4957/4957 [==============================] - 8s 2ms/step - loss: 0.0245 -
accuracy: 0.9912 - recall: 0.9922 - val_loss: 0.0198 - val_accuracy: 0.9935 -
val_recall: 0.9962
Epoch 44/45
4957/4957 [==============================] - 7s 1ms/step - loss: 0.0238 -
accuracy: 0.9919 - recall: 0.9929 - val_loss: 0.0192 - val_accuracy: 0.9944 -
val_recall: 0.9952
Epoch 45/45
4957/4957 [==============================] - 7s 2ms/step - loss: 0.0217 -
accuracy: 0.9927 - recall: 0.9939 - val_loss: 0.0176 - val_accuracy: 0.9949 -
val_recall: 0.9969
```

Ahora analizamos las metricas de desempeño del modelo a traves del entrenamiento

[165]:
```
fig,(ax1,ax2)=plt.subplots(1,2,figsize=(20,10))

plot_metrics(history.history,metrics=["accuracy","val_accuracy"],ax=ax1)
plot_metrics(history.history,metrics=["recall","val_recall"],ax=ax2)
```



[166]:
```
#tenemos la grafica de la funcion de perdida
plot_metrics(history.history,metrics=["loss","val_loss"])
```



Una vez entrenado el mejor modelo procedemos a realizar la **validacion del modelo**

VALIDACION DEL MODELO: MODELO SOBREMUESTREADO

- Veamos como se comporto el modelo durante el entrenamiento y su contraste con los datos de validacion

[169]:
```
best_model.evaluate(x_val,y_val)
```

```
2479/2479 [==============================] - 2s 834us/step - loss: 0.0176 -
accuracy: 0.9949 - recall: 0.9969
```

[169]:
```
[0.01762423664331436, 0.9948809146881104, 0.9968730211257935]
```

El modelo esta **generalizando muy bien en los datos de validacion** y es algo normal debido a que se entrenó y sintonizó con dichos datos. Pero a diferencia del **modelo base**, esta mucho mejor.

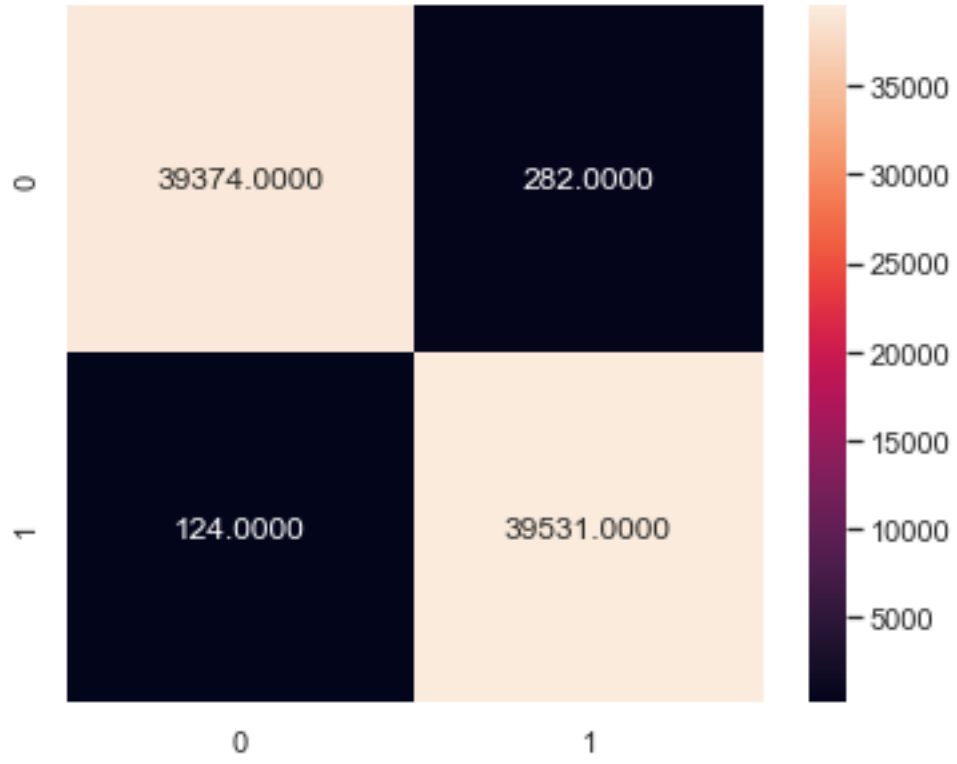
A continuacion veamos las metricas relacionadas a los reportes y la matriz de confusion

[168]:
```
display_report_and_matrix_confusion(best_model,x_val,y_val)
```

```
REPORTE DE CLASIFICACION
              precision    recall  f1-score   support
```

```
              precision    recall  f1-score   support

         0.0       1.00      0.99      0.99     39656
         1.0       0.99      1.00      0.99     39655

    accuracy                           0.99     79311
   macro avg       0.99      0.99      0.99     79311
weighted avg       0.99      0.99      0.99     79311
```

```
Matriz de confusion
```



Como era de esperar, los datos se balancearon y por ello incremento el factor de errores y aciertos. Ahora tenemos 124 falsos negativos (transacciones fraudulentas que dejamos ir), como tambien tenemos mas de **35 000 transacciones que identificamos con exito** !

Por otro lado algo de resaltar en este conjunto de validacion es que tenemos mas Falsos positivos que negativos y esta bien.

EVALUACION DEL MODELO CON SOBREMUESTREO

- Para ello cargamos los datos de test guardados en otro archivo ./creditcard_oversampling/test.cav

[170]:
```
x_test,y_test=load_data("./creditcard_oversampling/test.csv")
```

[171]:
```
x_test=scale_over.transform(x_test)
```

[172]:
```
best_model.evaluate(x_test,y_test)
```

```
5311/5311 [==============================] - 6s 1ms/step - loss: 0.0186 -
accuracy: 0.9949 - recall: 0.9970
```

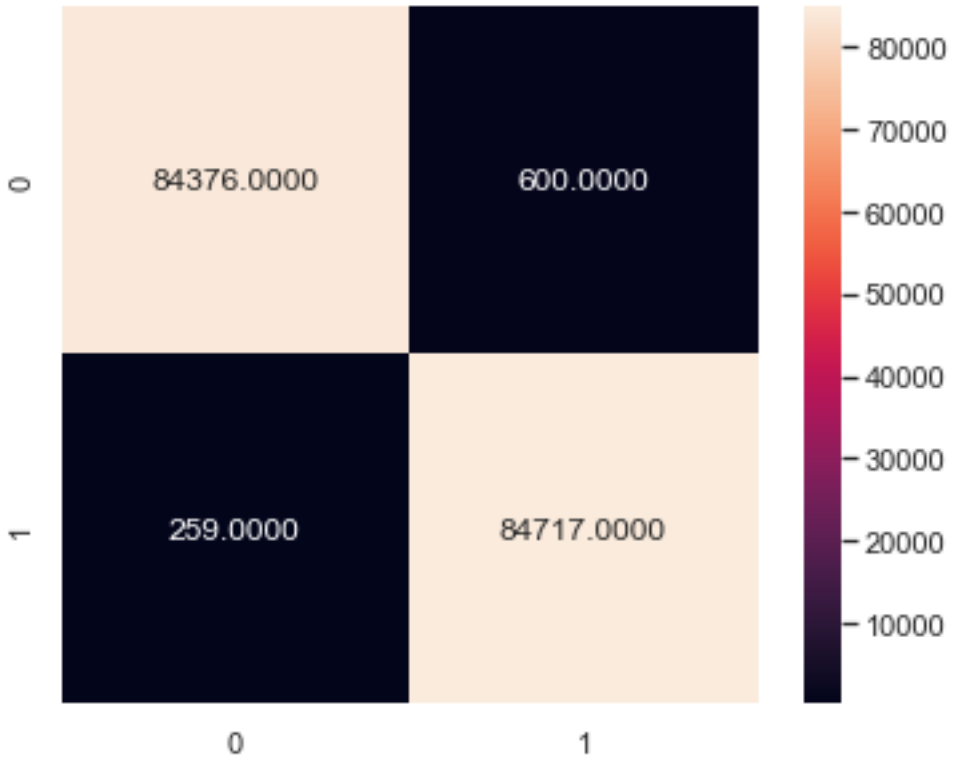
[172]:
```
[0.01861482672393322, 0.9949456453323364, 0.9969520568847656]
```

Excelente!. Tenemos una puntuacion de **Accuracy 99.49%** sobre datos nunca antes vistos y un **Recall de 99.7%**. Desde luego unas excelentes metricas sobre datos de Test

[173]:
```
display_report_and_matrix_confusion(best_model,x_test,y_test)
```

```
REPORTE DE CLASIFICACION
              precision    recall  f1-score   support

         0.0       1.00      0.99      0.99     84976
         1.0       0.99      1.00      0.99     84976

    accuracy                           0.99    169952
   macro avg       0.99      0.99      0.99    169952
weighted avg       0.99      0.99      0.99    169952
```

```
 Matriz de confusion
```



- **TP** (verdadero positivo) : 84376 datos predecidos como positivos correctamente
- **FP** (Falso positivo) : 600 datos predecidos falsamente como positivos
- **FN** (Falso Negativo) : 259 datos predecidos como negativos incorrectamente
- **TN** (verdadero positivo) : 84717 datos predecidos como positivos correctamente

El modelo da resultados excelentes. De mas de 16 000 muestras sobremuestreadas. solo 250 no se detectaron a tiempo, pero en su lugar mas de 80 000 muestras fraudulentas fueron atrapadas exitosamente.

- El modelo tiene una precision del **99% y tasa de recall de 99%**. Desde luego fue un mejor modelo

[175]:
```
tf.saved_model.save(best_model,"./best_endpoints/model_oversampling")␣
↪#guardamos el modelo
```

```
INFO:tensorflow:Assets written to: ./best_endpoints/model_oversampling\assets
```

Las medidas de escalamiento tambien se guardan

[176]:
```
import joblib

joblib.dump(scale_over,"./best_endpoints/scaler_oversampling.pkl")
```

[176]:
```
['./best_endpoints/scaler_oversampling.pkl']
```

0.2.3 MODELO v2.0 CON DATOS SIN BALANCEAR

- Esta vez se intentara mejorar el modelo base sin balancear a traves de un sintonizador de hiperparametros

Para ello realizaremos los mismos pasos de carga de datos, separacion y normalizacion de datos

[196]:
```
x_train,y_train=load_data("./creditcard_imbalanced/train.csv")
scaler_imbv2,x_train,x_val,y_train,y_val=partition_data(x_train,y_train,with_scale=True,split_
↪2) #20% de datos de validacion
```

Una vez obtenido los datos, llamo al sintonizador de hiperparametros,y lo configuro

[198]:
```
tuning=kt.Hyperband(
            build_model_tuning,
            objective=kt.Objective("val_recall", direction="max"),
            max_epochs=10,
            factor=3,
            directory="./hipertuning/",
            project_name="creditcard_tuner",
            overwrite=True
        )
```

[199]:
```
early_stopping=tf.keras.callbacks.EarlyStopping(monitor="val_loss",patience=10)␣
↪#si el entrenamiento no mejora siginificativamente durante 10 epocas lo␣
↪detengo
```

[200]:
```
tuning.
↪search(x_train,y_train,epochs=100,batch_size=64,validation_data=(x_val,y_val),callbacks=[ea
```

```
Trial 30 Complete [00h 00m 36s]
val_recall: 0.7424242496490479

Best val_recall So Far: 0.7727272510528564
Total elapsed time: 00h 08m 11s
INFO:tensorflow:Oracle triggered exit
```

[201]:
```
best_hps=tuning.get_best_hyperparameters(num_trials=1)[0]
print(f"Las mejores neuronas para la capa densa son: {best_hps.get('units')}")
```

```
Las mejores neuronas para la capa densa son: 304
```

Ahora construimos el modelo usando el mejor modelo encontrado

[202]:
```
model_imbalanced=tuning.get_best_models(num_models=1)[0]
```

[203]:
```
model_imbalanced.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
HIDDEN_LAYER (Dense)         (None, 304)               9120
_________________________________________________________________
output (Dense)               (None, 1)                 305
=================================================================
Total params: 9,425
Trainable params: 9,425
Non-trainable params: 0
_________________________________________________________________
```

VALIDACION DEL MODELO: MODELO DESBALANCEADO BASE MEJORADO

Validamos el modelo

[204]:
```
loss_val,acc_val,recall_val=model_imbalanced.evaluate(x_val,y_val)
```

```
1242/1242 [==============================] - 1s 843us/step - loss: 0.0044 -
accuracy: 0.9993 - recall: 0.6756
```

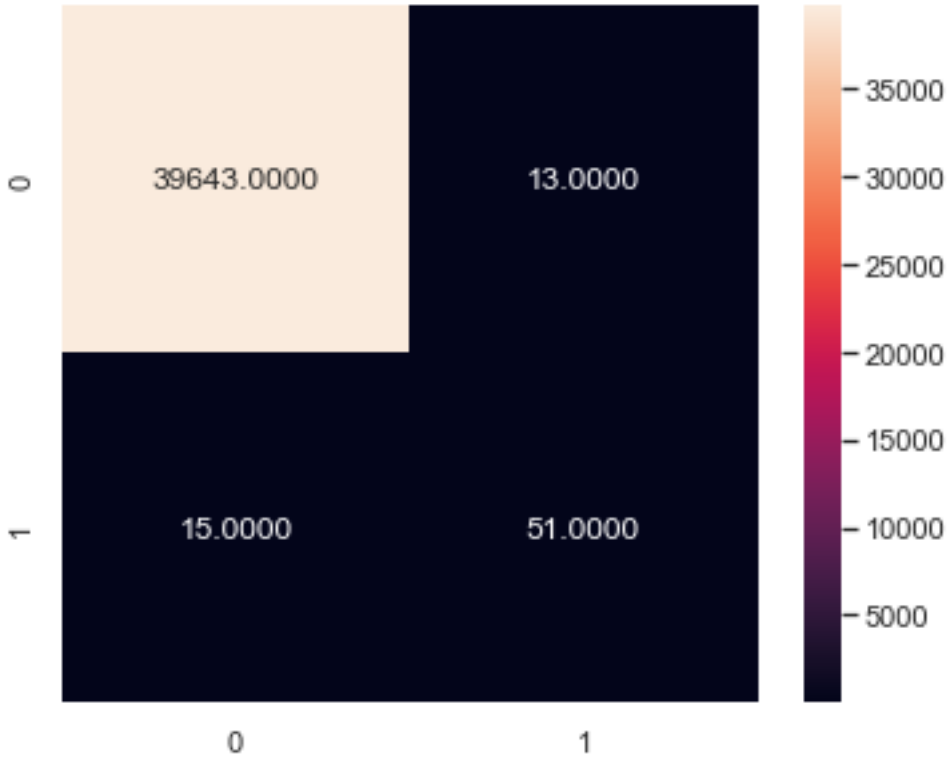
[205]:
```
print(f"Accuracy {acc_val:0.10f}")
print(f"Recall {recall_val:0.10f}")
```

```
Accuracy 0.9993076921
Recall 0.7196969986
```

[206]:
```
display_report_and_matrix_confusion(model_imbalanced,x_val,y_val)
```

```
REPORTE DE CLASIFICACION
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00     39656
         1.0       0.80      0.77      0.78        66

    accuracy                           1.00     39722
   macro avg       0.90      0.89      0.89     39722
weighted avg       1.00      1.00      1.00     39722
```

Matriz de confusion



- Tenemos un **accuracy 99.84%** y un recall (que es nuestro objetivo) de **77%** en los datos de validacion. Mejor que el modelo base en la etapa de validacion

EVALUACION DEL MODELO V2

[207]:
```
#primero cargamos los datos y escalamos
x_test,y_test=load_data("./creditcard_imbalanced/test.csv")
x_test=scaler_imbv2.transform(x_test)
```

[208]:
```
model_imbalanced.evaluate(x_test,y_test)
```

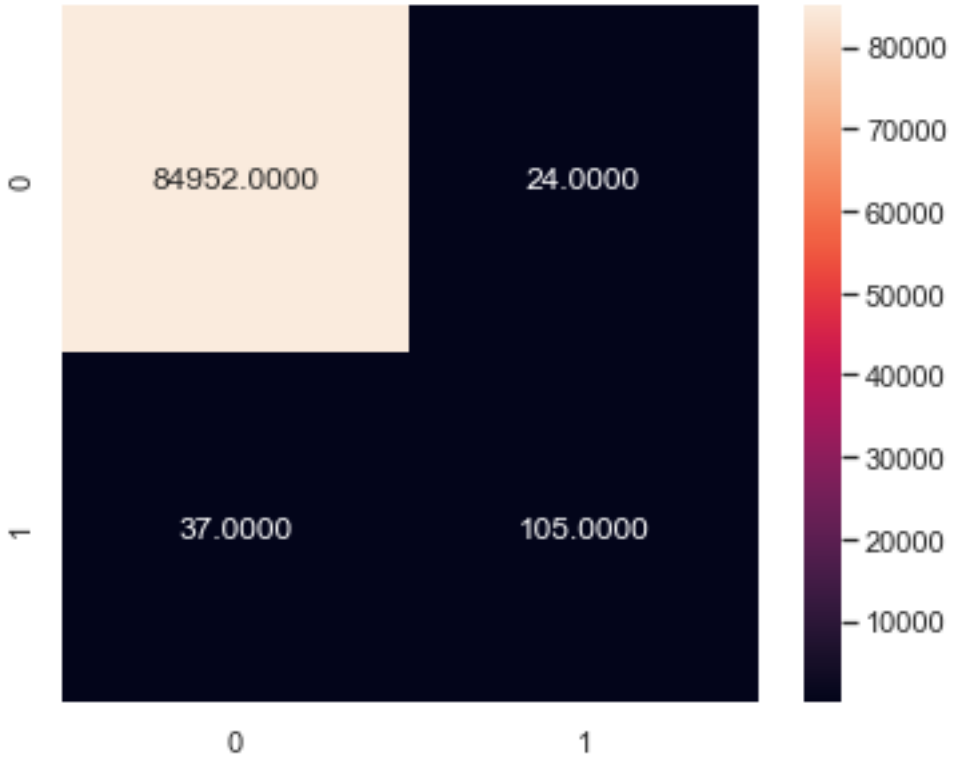
```
2660/2660 [==============================] - 2s 826us/step - loss: 0.0039 -
accuracy: 0.9993 - recall: 0.7394
```

[208]:
```
[0.003884269157424569, 0.9992833733558655, 0.7394366264343262]
```

[209]:
```
display_report_and_matrix_confusion(model_imbalanced,x_test,y_test)
```

```
REPORTE DE CLASIFICACION
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00     84976
         1.0       0.81      0.74      0.77       142

    accuracy                           1.00     85118
   macro avg       0.91      0.87      0.89     85118
weighted avg       1.00      1.00      1.00     85118
```

Matriz de confusion



En comparacion con el modelo base tuvo un **accuracy de 99.9% y un recall superior de 73%** , es mejor que el modelo base.

Obtuvo menor Falsos negativos, pero gano mas falsos positivos, que haran que una transaccion comun sea considerada fraude en al gunos clientes. Es mas sensible al fraude a un costo de ser desconfiado de las verdaderas transacciones confiables

[210]:
```
tf.saved_model.save(model_imbalanced,"./best_endpoints/best_model_imbalanced")
```

```
INFO:tensorflow:Assets written to: ./best_endpoints/best_model_imbalanced\assets
```

[213]:
```
#los datos de escalamiento tambien se guardan
import joblib
joblib.dump(scaler_imbv2,"./best_endpoints/scaler_best_imbalanced.pkl")
```

[213]:
```
['./best_endpoints/scaler_best_imbalanced.pkl']
```

0.2.4 MODELO v3.0 CON DATOS BALANCEADOS POR METODO DE SUB-MUESTREO

- Esto nos permite tener la proporcion de clases pertenecientes a fraude y No fraude equilibradas mediante la reduccion de la cantidad de muestras mayoritarias al nivel de clases Minoritarias. Tomelo como el efecto inverso de un Sobre muestreo

[214]:
```
#cargamos los datos del sobremuestreo obtenidos en la fase de transformacion de␣
↪datos/empacado
x_train,y_train=load_data("./creditcard_undersampling/train.csv")
scaler_usmp,x_train,x_val,y_train,y_val=partition_data(x_train,y_train,with_scale=True,split_s
↪2)
```

Teniendo los datos listos, crearemos el modelo y su sintonizador para obtener los mejores parametros para el entrenamiento

[215]:
```
BATCH_SIZE=42
STEP_UNITS=2 #Esta variable se encuentra dentro del las funcion que construye␣
↪el modelo
```

[222]:
```
early_stopping=tf.keras.callbacks.EarlyStopping(monitor="val_loss",patience=10)

tuning=kt.Hyperband(
            build_model_tuning,
            objective=kt.Objective("val_recall", direction="max"),
            max_epochs=10,
            factor=3,
            directory="./hipertuning/",
            project_name="creditcard_tuner",
            overwrite=True
```

```
        )
tuning.
 ↪search(x_train,y_train,epochs=150,batch_size=BATCH_SIZE,validation_data=(x_val,y_val),callb
```

```
Trial 30 Complete [00h 00m 01s]
val_recall: 0.8181818127632141

Best val_recall So Far: 1.0
Total elapsed time: 00h 00m 36s
INFO:tensorflow:Oracle triggered exit
```

[223]:
```
best_hps=tuning.get_best_hyperparameters(num_trials=1)[0]
```

[224]:
```
print("El mejor numero de neuronas para la capa oculta es de",best_hps.
 ↪get("units"))
```

```
El mejor numero de neuronas para la capa oculta es de 294
```

Entonces obtenemos la mejor combinacion de hiperparametros y reconstruimos el modelo, ahora con la intencion de obtener el mejor numero de **epochs** para el modelo

[225]:
```
#existen dos maneras de hacerlo
model_unsampling=tuning.hypermodel.build(best_hps) #a traves de la construccion␣
 ↪de sus mejores hiperparametros
#model_unsampling=tuning.get_best_models(num_models=1)[0] #obteniendo el mejor␣
 ↪modelo
```

La configuracion para la version 3 del modelo es:

[226]:
```
model_unsampling.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
HIDDEN_LAYER (Dense)         (None, 294)               8820
_________________________________________________________________
output (Dense)               (None, 1)                 295
=================================================================
Total params: 9,115
Trainable params: 9,115
Non-trainable params: 0
_________________________________________________________________
```

- Obtenemos el mejor epochs

[227]:
```
early_stopping=tf.keras.callbacks.EarlyStopping(monitor="val_loss",patience=10)
history=model_unsampling.
 ↪fit(x_train,y_train,epochs=150,batch_size=BATCH_SIZE,validation_data=(x_val,y_val),callback
```

```
Epoch 1/150
13/13 [==============================] - 1s 21ms/step - loss: 0.6647 - accuracy:
0.7110 - recall: 0.4197 - val_loss: 0.6129 - val_accuracy: 0.7444 - val_recall:
0.8636
Epoch 2/150
13/13 [==============================] - 0s 6ms/step - loss: 0.5921 - accuracy:
0.8615 - recall: 0.8106 - val_loss: 0.5504 - val_accuracy: 0.8045 - val_recall:
0.6061
Epoch 3/150
13/13 [==============================] - 0s 5ms/step - loss: 0.5445 - accuracy:
0.8314 - recall: 0.6734 - val_loss: 0.5070 - val_accuracy: 0.8346 - val_recall:
0.6667
Epoch 4/150
13/13 [==============================] - 0s 5ms/step - loss: 0.4942 - accuracy:
0.8079 - recall: 0.6125 - val_loss: 0.4667 - val_accuracy: 0.8872 - val_recall:
0.7879
Epoch 5/150
13/13 [==============================] - 0s 4ms/step - loss: 0.4391 - accuracy:
0.8898 - recall: 0.7779 - val_loss: 0.4259 - val_accuracy: 0.8797 - val_recall:
0.7576
Epoch 6/150
13/13 [==============================] - 0s 5ms/step - loss: 0.4090 - accuracy:
0.8943 - recall: 0.7827 - val_loss: 0.3945 - val_accuracy: 0.8872 - val_recall:
0.7879
Epoch 7/150
13/13 [==============================] - 0s 5ms/step - loss: 0.3848 - accuracy:
0.8977 - recall: 0.7827 - val_loss: 0.3651 - val_accuracy: 0.8947 - val_recall:
0.7879
Epoch 8/150
13/13 [==============================] - 0s 5ms/step - loss: 0.3622 - accuracy:
0.8931 - recall: 0.7908 - val_loss: 0.3422 - val_accuracy: 0.8872 - val_recall:
0.7727
Epoch 9/150
13/13 [==============================] - 0s 5ms/step - loss: 0.3550 - accuracy:
0.8757 - recall: 0.7639 - val_loss: 0.3302 - val_accuracy: 0.8797 - val_recall:
0.7576
Epoch 10/150
13/13 [==============================] - 0s 6ms/step - loss: 0.3219 - accuracy:
0.8961 - recall: 0.7914 - val_loss: 0.3059 - val_accuracy: 0.8947 - val_recall:
0.8030
Epoch 11/150
13/13 [==============================] - 0s 5ms/step - loss: 0.3017 - accuracy:
0.9068 - recall: 0.8140 - val_loss: 0.2982 - val_accuracy: 0.9023 - val_recall:
0.8182
Epoch 12/150
13/13 [==============================] - 0s 5ms/step - loss: 0.2815 - accuracy:
0.9115 - recall: 0.8261 - val_loss: 0.2798 - val_accuracy: 0.8947 - val_recall:
0.8030
```

```
Epoch 13/150
13/13 [==============================] - 0s 5ms/step - loss: 0.2616 - accuracy:
0.9070 - recall: 0.8162 - val_loss: 0.2686 - val_accuracy: 0.9023 - val_recall:
0.8182
Epoch 14/150
13/13 [==============================] - 0s 5ms/step - loss: 0.2613 - accuracy:
0.9087 - recall: 0.8219 - val_loss: 0.2596 - val_accuracy: 0.9023 - val_recall:
0.8182
Epoch 15/150
13/13 [==============================] - 0s 5ms/step - loss: 0.2749 - accuracy:
0.8970 - recall: 0.7940 - val_loss: 0.2514 - val_accuracy: 0.9023 - val_recall:
0.8182
Epoch 16/150
13/13 [==============================] - 0s 5ms/step - loss: 0.2365 - accuracy:
0.9190 - recall: 0.8437 - val_loss: 0.2443 - val_accuracy: 0.9023 - val_recall:
0.8182
Epoch 17/150
13/13 [==============================] - 0s 5ms/step - loss: 0.2614 - accuracy:
0.8987 - recall: 0.7997 - val_loss: 0.2437 - val_accuracy: 0.8947 - val_recall:
0.8030
Epoch 18/150
13/13 [==============================] - 0s 5ms/step - loss: 0.2378 - accuracy:
0.9081 - recall: 0.8212 - val_loss: 0.2356 - val_accuracy: 0.9323 - val_recall:
0.8788
Epoch 19/150
13/13 [==============================] - 0s 5ms/step - loss: 0.2431 - accuracy:
0.9173 - recall: 0.8442 - val_loss: 0.2314 - val_accuracy: 0.9323 - val_recall:
0.8788
Epoch 20/150
13/13 [==============================] - 0s 5ms/step - loss: 0.2145 - accuracy:
0.9337 - recall: 0.8776 - val_loss: 0.2287 - val_accuracy: 0.9323 - val_recall:
0.8788
Epoch 21/150
13/13 [==============================] - 0s 5ms/step - loss: 0.2112 - accuracy:
0.9380 - recall: 0.8799 - val_loss: 0.2189 - val_accuracy: 0.9173 - val_recall:
0.8485
Epoch 22/150
13/13 [==============================] - 0s 5ms/step - loss: 0.2054 - accuracy:
0.9358 - recall: 0.8786 - val_loss: 0.2192 - val_accuracy: 0.9098 - val_recall:
0.8333
Epoch 23/150
13/13 [==============================] - 0s 5ms/step - loss: 0.2043 - accuracy:
0.9342 - recall: 0.8683 - val_loss: 0.2305 - val_accuracy: 0.9398 - val_recall:
0.9091
Epoch 24/150
13/13 [==============================] - 0s 5ms/step - loss: 0.2144 - accuracy:
0.9369 - recall: 0.8824 - val_loss: 0.2097 - val_accuracy: 0.9323 - val_recall:
0.8788
```

```
Epoch 25/150
13/13 [==============================] - 0s 5ms/step - loss: 0.2156 - accuracy:
0.9287 - recall: 0.8635 - val_loss: 0.2051 - val_accuracy: 0.9248 - val_recall:
0.8636
Epoch 26/150
13/13 [==============================] - 0s 5ms/step - loss: 0.1769 - accuracy:
0.9482 - recall: 0.8981 - val_loss: 0.2063 - val_accuracy: 0.9323 - val_recall:
0.8788
Epoch 27/150
13/13 [==============================] - 0s 5ms/step - loss: 0.2195 - accuracy:
0.9228 - recall: 0.8500 - val_loss: 0.2037 - val_accuracy: 0.9248 - val_recall:
0.8636
Epoch 28/150
13/13 [==============================] - 0s 5ms/step - loss: 0.1990 - accuracy:
0.9239 - recall: 0.8645 - val_loss: 0.2064 - val_accuracy: 0.9248 - val_recall:
0.8636
Epoch 29/150
13/13 [==============================] - 0s 5ms/step - loss: 0.1959 - accuracy:
0.9354 - recall: 0.8740 - val_loss: 0.2034 - val_accuracy: 0.9398 - val_recall:
0.8939
Epoch 30/150
13/13 [==============================] - 0s 5ms/step - loss: 0.1925 - accuracy:
0.9429 - recall: 0.8896 - val_loss: 0.1956 - val_accuracy: 0.9323 - val_recall:
0.8788
Epoch 31/150
13/13 [==============================] - 0s 5ms/step - loss: 0.1908 - accuracy:
0.9327 - recall: 0.8758 - val_loss: 0.1989 - val_accuracy: 0.9398 - val_recall:
0.8939
Epoch 32/150
13/13 [==============================] - 0s 5ms/step - loss: 0.1923 - accuracy:
0.9339 - recall: 0.9058 - val_loss: 0.1897 - val_accuracy: 0.9323 - val_recall:
0.8788
Epoch 33/150
13/13 [==============================] - 0s 5ms/step - loss: 0.1774 - accuracy:
0.9403 - recall: 0.8907 - val_loss: 0.1894 - val_accuracy: 0.9323 - val_recall:
0.8788
Epoch 34/150
13/13 [==============================] - 0s 5ms/step - loss: 0.1928 - accuracy:
0.9351 - recall: 0.8845 - val_loss: 0.1937 - val_accuracy: 0.9323 - val_recall:
0.8788
Epoch 35/150
13/13 [==============================] - 0s 5ms/step - loss: 0.1785 - accuracy:
0.9393 - recall: 0.8790 - val_loss: 0.1864 - val_accuracy: 0.9323 - val_recall:
0.8788
Epoch 36/150
13/13 [==============================] - 0s 5ms/step - loss: 0.1953 - accuracy:
0.9227 - recall: 0.8537 - val_loss: 0.1860 - val_accuracy: 0.9323 - val_recall:
0.8788
```

```
Epoch 37/150
13/13 [==============================] - 0s 5ms/step - loss: 0.1865 - accuracy:
0.9402 - recall: 0.8882 - val_loss: 0.1850 - val_accuracy: 0.9323 - val_recall:
0.8788
Epoch 38/150
13/13 [==============================] - 0s 4ms/step - loss: 0.1720 - accuracy:
0.9416 - recall: 0.9076 - val_loss: 0.1830 - val_accuracy: 0.9323 - val_recall:
0.8788
Epoch 39/150
13/13 [==============================] - 0s 5ms/step - loss: 0.1662 - accuracy:
0.9371 - recall: 0.8810 - val_loss: 0.1852 - val_accuracy: 0.9398 - val_recall:
0.8939
Epoch 40/150
13/13 [==============================] - 0s 5ms/step - loss: 0.1878 - accuracy:
0.9376 - recall: 0.8884 - val_loss: 0.1826 - val_accuracy: 0.9323 - val_recall:
0.8788
Epoch 41/150
13/13 [==============================] - 0s 4ms/step - loss: 0.1608 - accuracy:
0.9418 - recall: 0.8899 - val_loss: 0.1805 - val_accuracy: 0.9323 - val_recall:
0.8788
Epoch 42/150
13/13 [==============================] - 0s 4ms/step - loss: 0.1591 - accuracy:
0.9447 - recall: 0.8953 - val_loss: 0.1850 - val_accuracy: 0.9398 - val_recall:
0.8939
Epoch 43/150
13/13 [==============================] - 0s 5ms/step - loss: 0.1938 - accuracy:
0.9236 - recall: 0.8918 - val_loss: 0.1934 - val_accuracy: 0.9323 - val_recall:
0.8788
Epoch 44/150
13/13 [==============================] - 0s 5ms/step - loss: 0.1937 - accuracy:
0.9297 - recall: 0.8692 - val_loss: 0.1776 - val_accuracy: 0.9323 - val_recall:
0.8788
Epoch 45/150
13/13 [==============================] - 0s 7ms/step - loss: 0.1725 - accuracy:
0.9499 - recall: 0.9073 - val_loss: 0.1768 - val_accuracy: 0.9323 - val_recall:
0.8788
Epoch 46/150
13/13 [==============================] - 0s 5ms/step - loss: 0.1539 - accuracy:
0.9538 - recall: 0.9170 - val_loss: 0.1930 - val_accuracy: 0.9398 - val_recall:
0.9091
Epoch 47/150
13/13 [==============================] - 0s 5ms/step - loss: 0.1861 - accuracy:
0.9187 - recall: 0.8789 - val_loss: 0.1781 - val_accuracy: 0.9323 - val_recall:
0.8788
Epoch 48/150
13/13 [==============================] - 0s 4ms/step - loss: 0.1794 - accuracy:
0.9303 - recall: 0.8960 - val_loss: 0.1849 - val_accuracy: 0.9323 - val_recall:
0.8788
```

```
Epoch 49/150
13/13 [==============================] - 0s 4ms/step - loss: 0.2035 - accuracy:
0.9210 - recall: 0.8556 - val_loss: 0.1761 - val_accuracy: 0.9398 - val_recall:
0.8939
Epoch 50/150
13/13 [==============================] - 0s 5ms/step - loss: 0.1601 - accuracy:
0.9500 - recall: 0.9049 - val_loss: 0.1752 - val_accuracy: 0.9398 - val_recall:
0.8939
Epoch 51/150
13/13 [==============================] - 0s 4ms/step - loss: 0.1701 - accuracy:
0.9463 - recall: 0.9052 - val_loss: 0.1730 - val_accuracy: 0.9323 - val_recall:
0.8788
Epoch 52/150
13/13 [==============================] - 0s 5ms/step - loss: 0.1498 - accuracy:
0.9493 - recall: 0.9022 - val_loss: 0.1756 - val_accuracy: 0.9398 - val_recall:
0.8939
Epoch 53/150
13/13 [==============================] - 0s 5ms/step - loss: 0.1755 - accuracy:
0.9354 - recall: 0.8864 - val_loss: 0.1761 - val_accuracy: 0.9323 - val_recall:
0.8788
Epoch 54/150
13/13 [==============================] - 0s 4ms/step - loss: 0.1744 - accuracy:
0.9365 - recall: 0.8925 - val_loss: 0.1722 - val_accuracy: 0.9323 - val_recall:
0.8788
Epoch 55/150
13/13 [==============================] - 0s 5ms/step - loss: 0.1741 - accuracy:
0.9357 - recall: 0.8980 - val_loss: 0.1722 - val_accuracy: 0.9323 - val_recall:
0.8788
Epoch 56/150
13/13 [==============================] - 0s 5ms/step - loss: 0.1720 - accuracy:
0.9375 - recall: 0.8759 - val_loss: 0.1863 - val_accuracy: 0.9474 - val_recall:
0.9242
Epoch 57/150
13/13 [==============================] - 0s 4ms/step - loss: 0.1610 - accuracy:
0.9391 - recall: 0.9179 - val_loss: 0.1722 - val_accuracy: 0.9323 - val_recall:
0.8788
Epoch 58/150
13/13 [==============================] - 0s 4ms/step - loss: 0.1856 - accuracy:
0.9359 - recall: 0.8863 - val_loss: 0.1723 - val_accuracy: 0.9323 - val_recall:
0.8788
Epoch 59/150
13/13 [==============================] - 0s 5ms/step - loss: 0.1710 - accuracy:
0.9367 - recall: 0.8721 - val_loss: 0.1851 - val_accuracy: 0.9474 - val_recall:
0.9242
Epoch 60/150
13/13 [==============================] - 0s 4ms/step - loss: 0.1831 - accuracy:
0.9333 - recall: 0.8957 - val_loss: 0.1708 - val_accuracy: 0.9323 - val_recall:
0.8788
```

```
Epoch 61/150
13/13 [==============================] - 0s 4ms/step - loss: 0.1862 - accuracy:
0.9216 - recall: 0.8785 - val_loss: 0.2061 - val_accuracy: 0.9323 - val_recall:
0.8636
Epoch 62/150
13/13 [==============================] - 0s 5ms/step - loss: 0.1562 - accuracy:
0.9455 - recall: 0.8988 - val_loss: 0.1854 - val_accuracy: 0.9474 - val_recall:
0.9242
Epoch 63/150
13/13 [==============================] - 0s 4ms/step - loss: 0.1705 - accuracy:
0.9340 - recall: 0.8860 - val_loss: 0.1748 - val_accuracy: 0.9323 - val_recall:
0.8788
Epoch 64/150
13/13 [==============================] - 0s 4ms/step - loss: 0.1613 - accuracy:
0.9432 - recall: 0.8898 - val_loss: 0.1709 - val_accuracy: 0.9398 - val_recall:
0.8939
Epoch 65/150
13/13 [==============================] - 0s 4ms/step - loss: 0.1762 - accuracy:
0.9361 - recall: 0.8880 - val_loss: 0.1704 - val_accuracy: 0.9323 - val_recall:
0.8788
Epoch 66/150
13/13 [==============================] - 0s 5ms/step - loss: 0.1586 - accuracy:
0.9413 - recall: 0.8914 - val_loss: 0.1742 - val_accuracy: 0.9323 - val_recall:
0.8939
Epoch 67/150
13/13 [==============================] - 0s 4ms/step - loss: 0.1566 - accuracy:
0.9449 - recall: 0.9062 - val_loss: 0.1693 - val_accuracy: 0.9323 - val_recall:
0.8788
Epoch 68/150
13/13 [==============================] - 0s 5ms/step - loss: 0.1727 - accuracy:
0.9278 - recall: 0.8631 - val_loss: 0.1689 - val_accuracy: 0.9323 - val_recall:
0.8788
Epoch 69/150
13/13 [==============================] - 0s 4ms/step - loss: 0.1810 - accuracy:
0.9284 - recall: 0.8919 - val_loss: 0.1720 - val_accuracy: 0.9323 - val_recall:
0.8788
Epoch 70/150
13/13 [==============================] - 0s 5ms/step - loss: 0.1854 - accuracy:
0.9262 - recall: 0.8696 - val_loss: 0.1689 - val_accuracy: 0.9398 - val_recall:
0.8939
Epoch 71/150
13/13 [==============================] - 0s 5ms/step - loss: 0.1715 - accuracy:
0.9402 - recall: 0.8869 - val_loss: 0.1690 - val_accuracy: 0.9398 - val_recall:
0.8939
Epoch 72/150
13/13 [==============================] - 0s 5ms/step - loss: 0.1714 - accuracy:
0.9412 - recall: 0.8877 - val_loss: 0.1696 - val_accuracy: 0.9323 - val_recall:
0.8788
```

```
Epoch 73/150
13/13 [==============================] - 0s 5ms/step - loss: 0.1579 - accuracy:
0.9424 - recall: 0.8937 - val_loss: 0.1752 - val_accuracy: 0.9323 - val_recall:
0.8939
Epoch 74/150
13/13 [==============================] - 0s 5ms/step - loss: 0.1680 - accuracy:
0.9410 - recall: 0.9037 - val_loss: 0.1743 - val_accuracy: 0.9323 - val_recall:
0.8788
Epoch 75/150
13/13 [==============================] - 0s 5ms/step - loss: 0.1504 - accuracy:
0.9445 - recall: 0.8968 - val_loss: 0.1882 - val_accuracy: 0.9323 - val_recall:
0.9394
Epoch 76/150
13/13 [==============================] - 0s 5ms/step - loss: 0.1758 - accuracy:
0.9335 - recall: 0.9233 - val_loss: 0.1708 - val_accuracy: 0.9323 - val_recall:
0.8788
Epoch 77/150
13/13 [==============================] - 0s 5ms/step - loss: 0.1361 - accuracy:
0.9458 - recall: 0.9031 - val_loss: 0.1665 - val_accuracy: 0.9398 - val_recall:
0.8939
Epoch 78/150
13/13 [==============================] - 0s 4ms/step - loss: 0.1446 - accuracy:
0.9552 - recall: 0.9192 - val_loss: 0.1706 - val_accuracy: 0.9323 - val_recall:
0.8939
Epoch 79/150
13/13 [==============================] - 0s 4ms/step - loss: 0.1609 - accuracy:
0.9465 - recall: 0.9150 - val_loss: 0.1671 - val_accuracy: 0.9398 - val_recall:
0.8939
Epoch 80/150
13/13 [==============================] - 0s 4ms/step - loss: 0.1662 - accuracy:
0.9429 - recall: 0.9072 - val_loss: 0.1663 - val_accuracy: 0.9398 - val_recall:
0.8939
Epoch 81/150
13/13 [==============================] - 0s 4ms/step - loss: 0.1485 - accuracy:
0.9411 - recall: 0.9135 - val_loss: 0.1736 - val_accuracy: 0.9323 - val_recall:
0.8788
Epoch 82/150
13/13 [==============================] - 0s 5ms/step - loss: 0.1593 - accuracy:
0.9433 - recall: 0.9049 - val_loss: 0.1656 - val_accuracy: 0.9398 - val_recall:
0.8939
Epoch 83/150
13/13 [==============================] - 0s 5ms/step - loss: 0.1713 - accuracy:
0.9298 - recall: 0.8760 - val_loss: 0.1714 - val_accuracy: 0.9323 - val_recall:
0.8939
Epoch 84/150
13/13 [==============================] - 0s 5ms/step - loss: 0.1678 - accuracy:
0.9412 - recall: 0.9107 - val_loss: 0.1719 - val_accuracy: 0.9323 - val_recall:
0.8788
```

```
Epoch 85/150
13/13 [==============================] - 0s 4ms/step - loss: 0.1472 - accuracy:
0.9464 - recall: 0.9010 - val_loss: 0.1651 - val_accuracy: 0.9398 - val_recall:
0.8939
Epoch 86/150
13/13 [==============================] - 0s 5ms/step - loss: 0.1602 - accuracy:
0.9396 - recall: 0.9090 - val_loss: 0.1670 - val_accuracy: 0.9323 - val_recall:
0.8788
Epoch 87/150
13/13 [==============================] - 0s 4ms/step - loss: 0.1567 - accuracy:
0.9416 - recall: 0.9079 - val_loss: 0.1731 - val_accuracy: 0.9398 - val_recall:
0.9091
Epoch 88/150
13/13 [==============================] - 0s 4ms/step - loss: 0.1642 - accuracy:
0.9334 - recall: 0.9046 - val_loss: 0.1666 - val_accuracy: 0.9323 - val_recall:
0.8788
Epoch 89/150
13/13 [==============================] - 0s 4ms/step - loss: 0.1490 - accuracy:
0.9472 - recall: 0.9014 - val_loss: 0.1641 - val_accuracy: 0.9398 - val_recall:
0.8939
Epoch 90/150
13/13 [==============================] - 0s 5ms/step - loss: 0.1581 - accuracy:
0.9393 - recall: 0.8955 - val_loss: 0.1652 - val_accuracy: 0.9323 - val_recall:
0.8939
Epoch 91/150
13/13 [==============================] - 0s 4ms/step - loss: 0.1644 - accuracy:
0.9385 - recall: 0.9233 - val_loss: 0.1742 - val_accuracy: 0.9323 - val_recall:
0.8788
Epoch 92/150
13/13 [==============================] - 0s 4ms/step - loss: 0.1418 - accuracy:
0.9503 - recall: 0.9120 - val_loss: 0.1634 - val_accuracy: 0.9398 - val_recall:
0.8939
Epoch 93/150
13/13 [==============================] - 0s 4ms/step - loss: 0.1670 - accuracy:
0.9309 - recall: 0.8713 - val_loss: 0.1651 - val_accuracy: 0.9323 - val_recall:
0.8939
Epoch 94/150
13/13 [==============================] - 0s 5ms/step - loss: 0.1371 - accuracy:
0.9523 - recall: 0.9146 - val_loss: 0.1631 - val_accuracy: 0.9398 - val_recall:
0.8939
Epoch 95/150
13/13 [==============================] - 0s 5ms/step - loss: 0.1760 - accuracy:
0.9349 - recall: 0.8969 - val_loss: 0.1641 - val_accuracy: 0.9398 - val_recall:
0.8939
Epoch 96/150
13/13 [==============================] - 0s 5ms/step - loss: 0.1647 - accuracy:
0.9380 - recall: 0.9029 - val_loss: 0.1697 - val_accuracy: 0.9323 - val_recall:
0.8788
```

```
Epoch 97/150
13/13 [==============================] - 0s 5ms/step - loss: 0.1559 - accuracy:
0.9489 - recall: 0.9086 - val_loss: 0.1644 - val_accuracy: 0.9323 - val_recall:
0.8788
Epoch 98/150
13/13 [==============================] - 0s 5ms/step - loss: 0.1550 - accuracy:
0.9416 - recall: 0.8976 - val_loss: 0.1622 - val_accuracy: 0.9398 - val_recall:
0.8939
Epoch 99/150
13/13 [==============================] - 0s 4ms/step - loss: 0.1595 - accuracy:
0.9377 - recall: 0.9032 - val_loss: 0.1811 - val_accuracy: 0.9398 - val_recall:
0.8788
Epoch 100/150
13/13 [==============================] - 0s 4ms/step - loss: 0.1835 - accuracy:
0.9335 - recall: 0.8685 - val_loss: 0.1627 - val_accuracy: 0.9398 - val_recall:
0.8939
Epoch 101/150
13/13 [==============================] - 0s 5ms/step - loss: 0.1530 - accuracy:
0.9432 - recall: 0.8975 - val_loss: 0.1621 - val_accuracy: 0.9398 - val_recall:
0.8939
Epoch 102/150
13/13 [==============================] - 0s 5ms/step - loss: 0.1341 - accuracy:
0.9568 - recall: 0.9160 - val_loss: 0.1729 - val_accuracy: 0.9474 - val_recall:
0.9242
Epoch 103/150
13/13 [==============================] - 0s 5ms/step - loss: 0.1581 - accuracy:
0.9378 - recall: 0.9203 - val_loss: 0.1620 - val_accuracy: 0.9398 - val_recall:
0.8939
Epoch 104/150
13/13 [==============================] - 0s 6ms/step - loss: 0.1475 - accuracy:
0.9465 - recall: 0.9155 - val_loss: 0.1610 - val_accuracy: 0.9398 - val_recall:
0.8939
Epoch 105/150
13/13 [==============================] - 0s 6ms/step - loss: 0.1265 - accuracy:
0.9525 - recall: 0.9186 - val_loss: 0.1625 - val_accuracy: 0.9323 - val_recall:
0.8788
Epoch 106/150
13/13 [==============================] - 0s 5ms/step - loss: 0.1246 - accuracy:
0.9552 - recall: 0.9137 - val_loss: 0.1819 - val_accuracy: 0.9323 - val_recall:
0.9394
Epoch 107/150
13/13 [==============================] - 0s 5ms/step - loss: 0.1590 - accuracy:
0.9473 - recall: 0.9155 - val_loss: 0.1624 - val_accuracy: 0.9323 - val_recall:
0.8788
Epoch 108/150
13/13 [==============================] - 0s 5ms/step - loss: 0.1577 - accuracy:
0.9505 - recall: 0.9172 - val_loss: 0.1605 - val_accuracy: 0.9398 - val_recall:
0.8939
```

```
Epoch 109/150
13/13 [==============================] - 0s 4ms/step - loss: 0.1403 - accuracy:
0.9478 - recall: 0.9106 - val_loss: 0.1697 - val_accuracy: 0.9474 - val_recall:
0.9242
Epoch 110/150
13/13 [==============================] - 0s 4ms/step - loss: 0.1837 - accuracy:
0.9410 - recall: 0.9279 - val_loss: 0.1936 - val_accuracy: 0.9398 - val_recall:
0.8788
Epoch 111/150
13/13 [==============================] - 0s 5ms/step - loss: 0.2143 - accuracy:
0.9184 - recall: 0.8561 - val_loss: 0.1737 - val_accuracy: 0.9398 - val_recall:
0.8788
Epoch 112/150
13/13 [==============================] - 0s 5ms/step - loss: 0.1787 - accuracy:
0.9301 - recall: 0.8855 - val_loss: 0.1716 - val_accuracy: 0.9323 - val_recall:
0.8788
Epoch 113/150
13/13 [==============================] - 0s 5ms/step - loss: 0.1639 - accuracy:
0.9352 - recall: 0.8845 - val_loss: 0.1623 - val_accuracy: 0.9323 - val_recall:
0.8788
Epoch 114/150
13/13 [==============================] - 0s 5ms/step - loss: 0.1782 - accuracy:
0.9410 - recall: 0.8866 - val_loss: 0.1694 - val_accuracy: 0.9474 - val_recall:
0.9242
Epoch 115/150
13/13 [==============================] - 0s 6ms/step - loss: 0.1522 - accuracy:
0.9437 - recall: 0.9113 - val_loss: 0.1598 - val_accuracy: 0.9398 - val_recall:
0.8939
Epoch 116/150
13/13 [==============================] - 0s 6ms/step - loss: 0.1455 - accuracy:
0.9455 - recall: 0.9134 - val_loss: 0.1593 - val_accuracy: 0.9398 - val_recall:
0.8939
Epoch 117/150
13/13 [==============================] - 0s 6ms/step - loss: 0.1313 - accuracy:
0.9602 - recall: 0.9293 - val_loss: 0.1715 - val_accuracy: 0.9474 - val_recall:
0.9242
Epoch 118/150
13/13 [==============================] - 0s 6ms/step - loss: 0.1748 - accuracy:
0.9411 - recall: 0.9016 - val_loss: 0.1609 - val_accuracy: 0.9398 - val_recall:
0.8939
Epoch 119/150
13/13 [==============================] - 0s 6ms/step - loss: 0.1411 - accuracy:
0.9460 - recall: 0.9065 - val_loss: 0.1658 - val_accuracy: 0.9398 - val_recall:
0.9091
Epoch 120/150
13/13 [==============================] - 0s 5ms/step - loss: 0.1420 - accuracy:
0.9536 - recall: 0.9180 - val_loss: 0.1610 - val_accuracy: 0.9323 - val_recall:
0.8788
```

```
Epoch 121/150
13/13 [==============================] - 0s 6ms/step - loss: 0.1414 - accuracy:
0.9435 - recall: 0.8954 - val_loss: 0.1588 - val_accuracy: 0.9398 - val_recall:
0.8939
Epoch 122/150
13/13 [==============================] - 0s 6ms/step - loss: 0.1630 - accuracy:
0.9368 - recall: 0.8981 - val_loss: 0.1586 - val_accuracy: 0.9398 - val_recall:
0.8939
Epoch 123/150
13/13 [==============================] - 0s 5ms/step - loss: 0.1386 - accuracy:
0.9520 - recall: 0.9194 - val_loss: 0.1763 - val_accuracy: 0.9323 - val_recall:
0.9394
Epoch 124/150
13/13 [==============================] - 0s 6ms/step - loss: 0.1641 - accuracy:
0.9380 - recall: 0.9214 - val_loss: 0.1581 - val_accuracy: 0.9398 - val_recall:
0.8939
Epoch 125/150
13/13 [==============================] - 0s 5ms/step - loss: 0.1649 - accuracy:
0.9425 - recall: 0.9156 - val_loss: 0.1600 - val_accuracy: 0.9398 - val_recall:
0.8939
Epoch 126/150
13/13 [==============================] - 0s 5ms/step - loss: 0.1639 - accuracy:
0.9473 - recall: 0.9234 - val_loss: 0.1810 - val_accuracy: 0.9398 - val_recall:
0.8788
Epoch 127/150
13/13 [==============================] - 0s 5ms/step - loss: 0.1612 - accuracy:
0.9414 - recall: 0.8844 - val_loss: 0.1606 - val_accuracy: 0.9323 - val_recall:
0.8939
Epoch 128/150
13/13 [==============================] - 0s 5ms/step - loss: 0.1742 - accuracy:
0.9280 - recall: 0.8888 - val_loss: 0.1582 - val_accuracy: 0.9398 - val_recall:
0.8939
Epoch 129/150
13/13 [==============================] - 0s 5ms/step - loss: 0.1312 - accuracy:
0.9568 - recall: 0.9315 - val_loss: 0.1586 - val_accuracy: 0.9398 - val_recall:
0.8939
Epoch 130/150
13/13 [==============================] - 0s 5ms/step - loss: 0.1524 - accuracy:
0.9427 - recall: 0.9070 - val_loss: 0.1576 - val_accuracy: 0.9398 - val_recall:
0.8939
Epoch 131/150
13/13 [==============================] - 0s 4ms/step - loss: 0.1302 - accuracy:
0.9562 - recall: 0.9204 - val_loss: 0.1612 - val_accuracy: 0.9398 - val_recall:
0.9091
Epoch 132/150
13/13 [==============================] - 0s 4ms/step - loss: 0.1495 - accuracy:
0.9534 - recall: 0.9482 - val_loss: 0.1576 - val_accuracy: 0.9398 - val_recall:
0.8939
```

```
Epoch 133/150
13/13 [==============================] - 0s 5ms/step - loss: 0.1426 - accuracy:
0.9530 - recall: 0.9326 - val_loss: 0.1628 - val_accuracy: 0.9323 - val_recall:
0.8788
Epoch 134/150
13/13 [==============================] - 0s 4ms/step - loss: 0.1609 - accuracy:
0.9389 - recall: 0.8959 - val_loss: 0.1630 - val_accuracy: 0.9323 - val_recall:
0.8788
Epoch 135/150
13/13 [==============================] - 0s 5ms/step - loss: 0.1735 - accuracy:
0.9347 - recall: 0.8945 - val_loss: 0.1638 - val_accuracy: 0.9323 - val_recall:
0.8788
Epoch 136/150
13/13 [==============================] - 0s 5ms/step - loss: 0.1612 - accuracy:
0.9430 - recall: 0.8913 - val_loss: 0.1711 - val_accuracy: 0.9323 - val_recall:
0.9242
Epoch 137/150
13/13 [==============================] - 0s 4ms/step - loss: 0.1502 - accuracy:
0.9572 - recall: 0.9333 - val_loss: 0.1569 - val_accuracy: 0.9398 - val_recall:
0.8939
Epoch 138/150
13/13 [==============================] - 0s 4ms/step - loss: 0.1360 - accuracy:
0.9486 - recall: 0.9191 - val_loss: 0.1567 - val_accuracy: 0.9398 - val_recall:
0.8939
Epoch 139/150
13/13 [==============================] - 0s 4ms/step - loss: 0.1558 - accuracy:
0.9508 - recall: 0.9330 - val_loss: 0.1566 - val_accuracy: 0.9398 - val_recall:
0.8939
Epoch 140/150
13/13 [==============================] - 0s 5ms/step - loss: 0.1470 - accuracy:
0.9476 - recall: 0.9139 - val_loss: 0.1668 - val_accuracy: 0.9474 - val_recall:
0.9242
Epoch 141/150
13/13 [==============================] - 0s 5ms/step - loss: 0.1437 - accuracy:
0.9508 - recall: 0.9425 - val_loss: 0.1695 - val_accuracy: 0.9398 - val_recall:
0.8788
Epoch 142/150
13/13 [==============================] - 0s 5ms/step - loss: 0.1468 - accuracy:
0.9477 - recall: 0.9046 - val_loss: 0.1594 - val_accuracy: 0.9323 - val_recall:
0.8788
Epoch 143/150
13/13 [==============================] - 0s 5ms/step - loss: 0.1459 - accuracy:
0.9492 - recall: 0.9077 - val_loss: 0.1738 - val_accuracy: 0.9323 - val_recall:
0.9394
Epoch 144/150
13/13 [==============================] - 0s 5ms/step - loss: 0.1504 - accuracy:
0.9520 - recall: 0.9404 - val_loss: 0.1621 - val_accuracy: 0.9398 - val_recall:
0.8788
```

```
Epoch 145/150
13/13 [==============================] - 0s 5ms/step - loss: 0.1413 - accuracy:
0.9501 - recall: 0.9042 - val_loss: 0.1571 - val_accuracy: 0.9398 - val_recall:
0.8939
Epoch 146/150
13/13 [==============================] - 0s 5ms/step - loss: 0.1527 - accuracy:
0.9479 - recall: 0.9046 - val_loss: 0.1562 - val_accuracy: 0.9323 - val_recall:
0.8939
Epoch 147/150
13/13 [==============================] - 0s 5ms/step - loss: 0.1721 - accuracy:
0.9267 - recall: 0.8899 - val_loss: 0.1578 - val_accuracy: 0.9398 - val_recall:
0.8939
Epoch 148/150
13/13 [==============================] - 0s 5ms/step - loss: 0.1458 - accuracy:
0.9511 - recall: 0.9135 - val_loss: 0.1549 - val_accuracy: 0.9398 - val_recall:
0.8939
Epoch 149/150
13/13 [==============================] - 0s 4ms/step - loss: 0.1602 - accuracy:
0.9447 - recall: 0.9168 - val_loss: 0.1568 - val_accuracy: 0.9398 - val_recall:
0.8939
Epoch 150/150
13/13 [==============================] - 0s 5ms/step - loss: 0.1586 - accuracy:
0.9392 - recall: 0.9052 - val_loss: 0.1613 - val_accuracy: 0.9398 - val_recall:
0.8788
```

[229]:
```
#a continuacion extraemos el indice de mayor rendimiento para obtener el epoch
n_epochs=np.argmax(history.history["val_recall"])+1
print("El numero de epochs adecuado es ",n_epochs)
```

```
El numero de epochs adecuado es  75
```

[230]:
```
model_unsampling=tuning.hypermodel.build(best_hps) #reconstruimos el modelo
tensorboard_callback=tf.keras.callbacks.TensorBoard(log_dir="./graph/
 ↪deteccion_fraude/best_model_unsampling_last/")
history=model_unsampling.
 ↪fit(x_train,y_train,epochs=n_epochs,batch_size=BATCH_SIZE,validation_data=(x_val,y_val),cal
```

```
Epoch 1/75
13/13 [==============================] - 80s 7s/step - loss: 0.6656 - accuracy:
0.7277 - recall: 0.6944 - val_loss: 0.6042 - val_accuracy: 0.8271 - val_recall:
0.6515
Epoch 2/75
13/13 [==============================] - 0s 6ms/step - loss: 0.5887 - accuracy:
0.7933 - recall: 0.5448 - val_loss: 0.5598 - val_accuracy: 0.9098 - val_recall:
0.8333
Epoch 3/75
13/13 [==============================] - 0s 5ms/step - loss: 0.5392 - accuracy:
0.8993 - recall: 0.8024 - val_loss: 0.5082 - val_accuracy: 0.8421 - val_recall:
0.6818
```

```
Epoch 4/75
13/13 [==============================] - 0s 5ms/step - loss: 0.4984 - accuracy:
0.8339 - recall: 0.6646 - val_loss: 0.4642 - val_accuracy: 0.8797 - val_recall:
0.7576
Epoch 5/75
13/13 [==============================] - 0s 5ms/step - loss: 0.4566 - accuracy:
0.8680 - recall: 0.7219 - val_loss: 0.4282 - val_accuracy: 0.9023 - val_recall:
0.8030
Epoch 6/75
13/13 [==============================] - 0s 5ms/step - loss: 0.4133 - accuracy:
0.8876 - recall: 0.7703 - val_loss: 0.3921 - val_accuracy: 0.8797 - val_recall:
0.7576
Epoch 7/75
13/13 [==============================] - 0s 4ms/step - loss: 0.3764 - accuracy:
0.9003 - recall: 0.7976 - val_loss: 0.3717 - val_accuracy: 0.9023 - val_recall:
0.8182
Epoch 8/75
13/13 [==============================] - 0s 5ms/step - loss: 0.3679 - accuracy:
0.8859 - recall: 0.7763 - val_loss: 0.3451 - val_accuracy: 0.9023 - val_recall:
0.8182
Epoch 9/75
13/13 [==============================] - 0s 5ms/step - loss: 0.3396 - accuracy:
0.9054 - recall: 0.8060 - val_loss: 0.3217 - val_accuracy: 0.9023 - val_recall:
0.8030
Epoch 10/75
13/13 [==============================] - 0s 4ms/step - loss: 0.3145 - accuracy:
0.8893 - recall: 0.7688 - val_loss: 0.3091 - val_accuracy: 0.9023 - val_recall:
0.8182
Epoch 11/75
13/13 [==============================] - 0s 5ms/step - loss: 0.3056 - accuracy:
0.9026 - recall: 0.7969 - val_loss: 0.2977 - val_accuracy: 0.9098 - val_recall:
0.8333
Epoch 12/75
13/13 [==============================] - 0s 5ms/step - loss: 0.3152 - accuracy:
0.9037 - recall: 0.8119 - val_loss: 0.2876 - val_accuracy: 0.8947 - val_recall:
0.7879
Epoch 13/75
13/13 [==============================] - 0s 5ms/step - loss: 0.2990 - accuracy:
0.8978 - recall: 0.7931 - val_loss: 0.2788 - val_accuracy: 0.8947 - val_recall:
0.7879
Epoch 14/75
13/13 [==============================] - 0s 5ms/step - loss: 0.2811 - accuracy:
0.8964 - recall: 0.7978 - val_loss: 0.2621 - val_accuracy: 0.9098 - val_recall:
0.8182
Epoch 15/75
13/13 [==============================] - 0s 5ms/step - loss: 0.2588 - accuracy:
0.9027 - recall: 0.8047 - val_loss: 0.2703 - val_accuracy: 0.9323 - val_recall:
0.8788
```

```
Epoch 16/75
13/13 [==============================] - 0s 4ms/step - loss: 0.2475 - accuracy:
0.9266 - recall: 0.8580 - val_loss: 0.2581 - val_accuracy: 0.9323 - val_recall:
0.8788
Epoch 17/75
13/13 [==============================] - 0s 4ms/step - loss: 0.2566 - accuracy:
0.9253 - recall: 0.8813 - val_loss: 0.2483 - val_accuracy: 0.9023 - val_recall:
0.8030
Epoch 18/75
13/13 [==============================] - 0s 5ms/step - loss: 0.2621 - accuracy:
0.9122 - recall: 0.8328 - val_loss: 0.2423 - val_accuracy: 0.9023 - val_recall:
0.8030
Epoch 19/75
13/13 [==============================] - 0s 5ms/step - loss: 0.2422 - accuracy:
0.8915 - recall: 0.7886 - val_loss: 0.2350 - val_accuracy: 0.9248 - val_recall:
0.8636
Epoch 20/75
13/13 [==============================] - 0s 5ms/step - loss: 0.2249 - accuracy:
0.9227 - recall: 0.8558 - val_loss: 0.2293 - val_accuracy: 0.9248 - val_recall:
0.8636
Epoch 21/75
13/13 [==============================] - 0s 5ms/step - loss: 0.2121 - accuracy:
0.9283 - recall: 0.8628 - val_loss: 0.2298 - val_accuracy: 0.9323 - val_recall:
0.8788
Epoch 22/75
13/13 [==============================] - 0s 5ms/step - loss: 0.2229 - accuracy:
0.9304 - recall: 0.8714 - val_loss: 0.2274 - val_accuracy: 0.9173 - val_recall:
0.8333
Epoch 23/75
13/13 [==============================] - 0s 5ms/step - loss: 0.2340 - accuracy:
0.9178 - recall: 0.8402 - val_loss: 0.2177 - val_accuracy: 0.9248 - val_recall:
0.8636
Epoch 24/75
13/13 [==============================] - 0s 5ms/step - loss: 0.2148 - accuracy:
0.9243 - recall: 0.8563 - val_loss: 0.2156 - val_accuracy: 0.9248 - val_recall:
0.8636
Epoch 25/75
13/13 [==============================] - 0s 5ms/step - loss: 0.2168 - accuracy:
0.9237 - recall: 0.8647 - val_loss: 0.2256 - val_accuracy: 0.9023 - val_recall:
0.8030
Epoch 26/75
13/13 [==============================] - 0s 5ms/step - loss: 0.2034 - accuracy:
0.9187 - recall: 0.8366 - val_loss: 0.2159 - val_accuracy: 0.9323 - val_recall:
0.8788
Epoch 27/75
13/13 [==============================] - 0s 5ms/step - loss: 0.2218 - accuracy:
0.9290 - recall: 0.8909 - val_loss: 0.2064 - val_accuracy: 0.9248 - val_recall:
0.8636
```

```
Epoch 28/75
13/13 [==============================] - 0s 5ms/step - loss: 0.1760 - accuracy:
0.9442 - recall: 0.8930 - val_loss: 0.2158 - val_accuracy: 0.9248 - val_recall:
0.8788
Epoch 29/75
13/13 [==============================] - 0s 5ms/step - loss: 0.2155 - accuracy:
0.9351 - recall: 0.8877 - val_loss: 0.2027 - val_accuracy: 0.9248 - val_recall:
0.8636
Epoch 30/75
13/13 [==============================] - 0s 5ms/step - loss: 0.2050 - accuracy:
0.9189 - recall: 0.8540 - val_loss: 0.1996 - val_accuracy: 0.9323 - val_recall:
0.8788
Epoch 31/75
13/13 [==============================] - 0s 5ms/step - loss: 0.1766 - accuracy:
0.9455 - recall: 0.8908 - val_loss: 0.1980 - val_accuracy: 0.9323 - val_recall:
0.8788
Epoch 32/75
13/13 [==============================] - 0s 5ms/step - loss: 0.2018 - accuracy:
0.9249 - recall: 0.8712 - val_loss: 0.2029 - val_accuracy: 0.9323 - val_recall:
0.8636
Epoch 33/75
13/13 [==============================] - 0s 5ms/step - loss: 0.1992 - accuracy:
0.9283 - recall: 0.8527 - val_loss: 0.1980 - val_accuracy: 0.9323 - val_recall:
0.8788
Epoch 34/75
13/13 [==============================] - 0s 5ms/step - loss: 0.1922 - accuracy:
0.9363 - recall: 0.8763 - val_loss: 0.1928 - val_accuracy: 0.9323 - val_recall:
0.8788
Epoch 35/75
13/13 [==============================] - 0s 4ms/step - loss: 0.1755 - accuracy:
0.9419 - recall: 0.8866 - val_loss: 0.2062 - val_accuracy: 0.9398 - val_recall:
0.9091
Epoch 36/75
13/13 [==============================] - 0s 5ms/step - loss: 0.1790 - accuracy:
0.9442 - recall: 0.9082 - val_loss: 0.1912 - val_accuracy: 0.9323 - val_recall:
0.8788
Epoch 37/75
13/13 [==============================] - 0s 5ms/step - loss: 0.1888 - accuracy:
0.9355 - recall: 0.8704 - val_loss: 0.1881 - val_accuracy: 0.9323 - val_recall:
0.8788
Epoch 38/75
13/13 [==============================] - 0s 4ms/step - loss: 0.2106 - accuracy:
0.9176 - recall: 0.8515 - val_loss: 0.1930 - val_accuracy: 0.9323 - val_recall:
0.8788
Epoch 39/75
13/13 [==============================] - 0s 5ms/step - loss: 0.2049 - accuracy:
0.9165 - recall: 0.8460 - val_loss: 0.1986 - val_accuracy: 0.9323 - val_recall:
0.8636
```

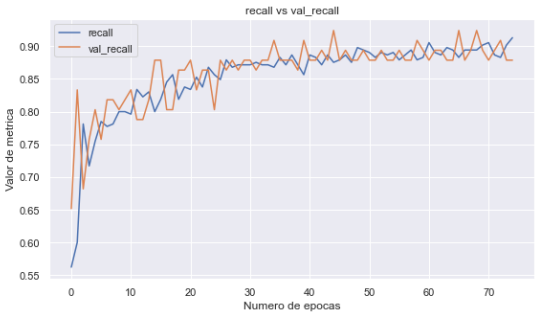
```
Epoch 40/75
13/13 [==============================] - 0s 5ms/step - loss: 0.1743 - accuracy:
0.9249 - recall: 0.8646 - val_loss: 0.2010 - val_accuracy: 0.9398 - val_recall:
0.9091
Epoch 41/75
13/13 [==============================] - 0s 5ms/step - loss: 0.2043 - accuracy:
0.9272 - recall: 0.8782 - val_loss: 0.1892 - val_accuracy: 0.9323 - val_recall:
0.8788
Epoch 42/75
13/13 [==============================] - 0s 5ms/step - loss: 0.1716 - accuracy:
0.9404 - recall: 0.8916 - val_loss: 0.1833 - val_accuracy: 0.9323 - val_recall:
0.8788
Epoch 43/75
13/13 [==============================] - 0s 5ms/step - loss: 0.1708 - accuracy:
0.9397 - recall: 0.8875 - val_loss: 0.1900 - val_accuracy: 0.9323 - val_recall:
0.8939
Epoch 44/75
13/13 [==============================] - 0s 5ms/step - loss: 0.1661 - accuracy:
0.9441 - recall: 0.9108 - val_loss: 0.1815 - val_accuracy: 0.9323 - val_recall:
0.8788
Epoch 45/75
13/13 [==============================] - ETA: 0s - loss: 0.1480 - accuracy:
0.9762 - recall: 0.941 - 0s 5ms/step - loss: 0.1640 - accuracy: 0.9505 - recall:
0.9009 - val_loss: 0.2168 - val_accuracy: 0.9173 - val_recall: 0.9242
Epoch 46/75
13/13 [==============================] - 0s 4ms/step - loss: 0.2035 - accuracy:
0.9226 - recall: 0.9050 - val_loss: 0.1800 - val_accuracy: 0.9323 - val_recall:
0.8788
Epoch 47/75
13/13 [==============================] - 0s 5ms/step - loss: 0.1721 - accuracy:
0.9458 - recall: 0.9118 - val_loss: 0.1818 - val_accuracy: 0.9398 - val_recall:
0.8939
Epoch 48/75
13/13 [==============================] - 0s 4ms/step - loss: 0.1744 - accuracy:
0.9314 - recall: 0.8860 - val_loss: 0.1798 - val_accuracy: 0.9323 - val_recall:
0.8788
Epoch 49/75
13/13 [==============================] - 0s 5ms/step - loss: 0.1851 - accuracy:
0.9373 - recall: 0.9045 - val_loss: 0.1778 - val_accuracy: 0.9323 - val_recall:
0.8788
Epoch 50/75
13/13 [==============================] - 0s 4ms/step - loss: 0.1534 - accuracy:
0.9476 - recall: 0.9075 - val_loss: 0.1796 - val_accuracy: 0.9398 - val_recall:
0.8939
Epoch 51/75
13/13 [==============================] - 0s 5ms/step - loss: 0.1795 - accuracy:
0.9318 - recall: 0.8767 - val_loss: 0.1811 - val_accuracy: 0.9323 - val_recall:
0.8788
```

```
Epoch 52/75
13/13 [==============================] - 0s 5ms/step - loss: 0.1727 - accuracy:
0.9319 - recall: 0.8720 - val_loss: 0.1763 - val_accuracy: 0.9323 - val_recall:
0.8788
Epoch 53/75
13/13 [==============================] - 0s 6ms/step - loss: 0.1825 - accuracy:
0.9332 - recall: 0.8822 - val_loss: 0.1796 - val_accuracy: 0.9323 - val_recall:
0.8939
Epoch 54/75
13/13 [==============================] - 0s 5ms/step - loss: 0.1859 - accuracy:
0.9323 - recall: 0.8906 - val_loss: 0.1774 - val_accuracy: 0.9323 - val_recall:
0.8788
Epoch 55/75
13/13 [==============================] - 0s 6ms/step - loss: 0.1593 - accuracy:
0.9399 - recall: 0.8893 - val_loss: 0.1747 - val_accuracy: 0.9323 - val_recall:
0.8788
Epoch 56/75
13/13 [==============================] - 0s 6ms/step - loss: 0.1664 - accuracy:
0.9406 - recall: 0.8867 - val_loss: 0.1782 - val_accuracy: 0.9323 - val_recall:
0.8939
Epoch 57/75
13/13 [==============================] - 0s 6ms/step - loss: 0.1792 - accuracy:
0.9279 - recall: 0.8845 - val_loss: 0.1771 - val_accuracy: 0.9323 - val_recall:
0.8788
Epoch 58/75
13/13 [==============================] - 0s 5ms/step - loss: 0.1716 - accuracy:
0.9330 - recall: 0.8801 - val_loss: 0.1766 - val_accuracy: 0.9323 - val_recall:
0.8788
Epoch 59/75
13/13 [==============================] - 0s 5ms/step - loss: 0.1770 - accuracy:
0.9387 - recall: 0.8826 - val_loss: 0.1860 - val_accuracy: 0.9398 - val_recall:
0.9091
Epoch 60/75
13/13 [==============================] - 0s 6ms/step - loss: 0.1591 - accuracy:
0.9442 - recall: 0.9032 - val_loss: 0.1790 - val_accuracy: 0.9323 - val_recall:
0.8939
Epoch 61/75
13/13 [==============================] - 0s 5ms/step - loss: 0.1655 - accuracy:
0.9458 - recall: 0.9118 - val_loss: 0.1728 - val_accuracy: 0.9323 - val_recall:
0.8788
Epoch 62/75
13/13 [==============================] - 0s 6ms/step - loss: 0.1430 - accuracy:
0.9518 - recall: 0.9087 - val_loss: 0.1725 - val_accuracy: 0.9398 - val_recall:
0.8939
Epoch 63/75
13/13 [==============================] - 0s 6ms/step - loss: 0.1603 - accuracy:
0.9412 - recall: 0.8859 - val_loss: 0.1717 - val_accuracy: 0.9398 - val_recall:
0.8939
```

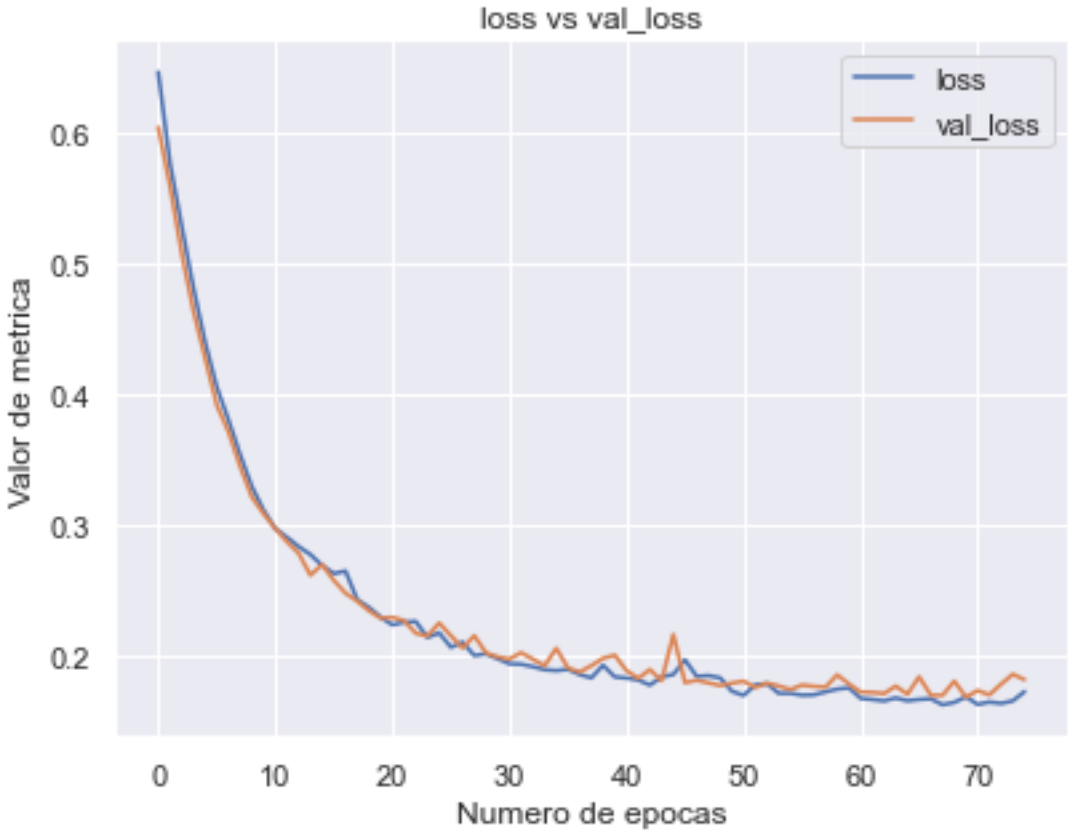
```
Epoch 64/75
13/13 [==============================] - 0s 5ms/step - loss: 0.1650 - accuracy:
0.9374 - recall: 0.9014 - val_loss: 0.1774 - val_accuracy: 0.9323 - val_recall:
0.8788
Epoch 65/75
13/13 [==============================] - 0s 5ms/step - loss: 0.1945 - accuracy:
0.9306 - recall: 0.8726 - val_loss: 0.1711 - val_accuracy: 0.9323 - val_recall:
0.8788
Epoch 66/75
13/13 [==============================] - 0s 4ms/step - loss: 0.1669 - accuracy:
0.9393 - recall: 0.8932 - val_loss: 0.1845 - val_accuracy: 0.9474 - val_recall:
0.9242
Epoch 67/75
13/13 [==============================] - 0s 4ms/step - loss: 0.1834 - accuracy:
0.9366 - recall: 0.9084 - val_loss: 0.1704 - val_accuracy: 0.9323 - val_recall:
0.8788
Epoch 68/75
13/13 [==============================] - 0s 5ms/step - loss: 0.1635 - accuracy:
0.9432 - recall: 0.8943 - val_loss: 0.1703 - val_accuracy: 0.9398 - val_recall:
0.8939
Epoch 69/75
13/13 [==============================] - 0s 5ms/step - loss: 0.1622 - accuracy:
0.9498 - recall: 0.9017 - val_loss: 0.1812 - val_accuracy: 0.9474 - val_recall:
0.9242
Epoch 70/75
13/13 [==============================] - 0s 5ms/step - loss: 0.1656 - accuracy:
0.9462 - recall: 0.9225 - val_loss: 0.1689 - val_accuracy: 0.9398 - val_recall:
0.8939
Epoch 71/75
13/13 [==============================] - 0s 5ms/step - loss: 0.1652 - accuracy:
0.9341 - recall: 0.8931 - val_loss: 0.1740 - val_accuracy: 0.9323 - val_recall:
0.8788
Epoch 72/75
13/13 [==============================] - 0s 4ms/step - loss: 0.1697 - accuracy:
0.9349 - recall: 0.8820 - val_loss: 0.1707 - val_accuracy: 0.9398 - val_recall:
0.8939
Epoch 73/75
13/13 [==============================] - 0s 5ms/step - loss: 0.1539 - accuracy:
0.9513 - recall: 0.9081 - val_loss: 0.1788 - val_accuracy: 0.9398 - val_recall:
0.9091
Epoch 74/75
13/13 [==============================] - 0s 6ms/step - loss: 0.1683 - accuracy:
0.9423 - recall: 0.9191 - val_loss: 0.1864 - val_accuracy: 0.9398 - val_recall:
0.8788
Epoch 75/75
13/13 [==============================] - 0s 6ms/step - loss: 0.1946 - accuracy:
0.9319 - recall: 0.8731 - val_loss: 0.1823 - val_accuracy: 0.9398 - val_recall:
0.8788
```

Exploremos los resultados del modelo graficamente

[234]:
```
fig,(ax1,ax2)=plt.subplots(1,2,figsize=(20,5))

plot_metrics(history.history,metrics=["accuracy","val_accuracy"],ax=ax1)
plot_metrics(history.history,metrics=["recall","val_recall"],ax=ax2)
```



[244]:
```
plot_metrics(history.history,metrics=["loss","val_loss"])
```



VALIDACION DEL MODELO: MODELO SUBMUESTREADO

[235]:
```
model_unsampling.evaluate(x_val,y_val)
```

```
5/5 [==============================] - 0s 3ms/step - loss: 0.1823 - accuracy:
0.9398 - recall: 0.8788
```

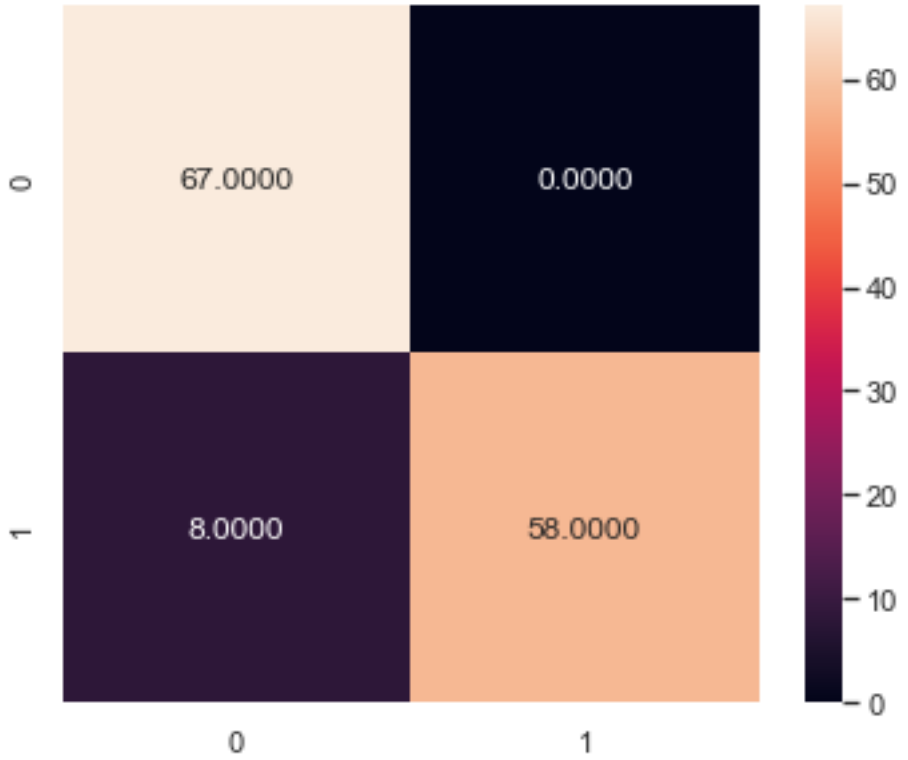
[235]:
```
[0.1822652816772461, 0.9398496150970459, 0.8787878751754761]
```

Se obtuvo una puntuacion **mas alta que del modelo base y modelo base V2**,con un **accuracy 93.98%** que se solventa con un **recall de 87.88%** sobre los datos de validacion

[236]:
```
display_report_and_matrix_confusion(model_unsampling,x_val,y_val)
```

```
REPORTE DE CLASIFICACION
              precision    recall  f1-score   support

         0.0       0.89      1.00      0.94        67
         1.0       1.00      0.88      0.94        66

    accuracy                           0.94       133
   macro avg       0.95      0.94      0.94       133
weighted avg       0.95      0.94      0.94       133
```

Matriz de confusion



Con esta matriz de confusion podemos determinar que el **submuestreo V3** es mucho mejor que los modelos base y modelo baseV2.

- Los falsos negativos se reducen a 8 unidades, y los falsos positivos a 0. Nosotros buscamos anular los falsos negativos lo mas posible

EVALUACION DEL MODELO CON SUBMUESTREO V3

- El conjunto de validacion nos sirve para detener el entrenamiento cuando los resultados son buenos, ahora tendremos que probarlos con el conjunto de pruebas

[238]:
```
x_test,y_test=load_data("./creditcard_undersampling/test.csv")
x_test=scaler_usmp.transform(x_test)
```

[239]:
```
model_unsampling.evaluate(x_test,y_test)
```

```
9/9 [==============================] - 0s 2ms/step - loss: 0.1896 - accuracy:
0.9296 - recall: 0.8732
```

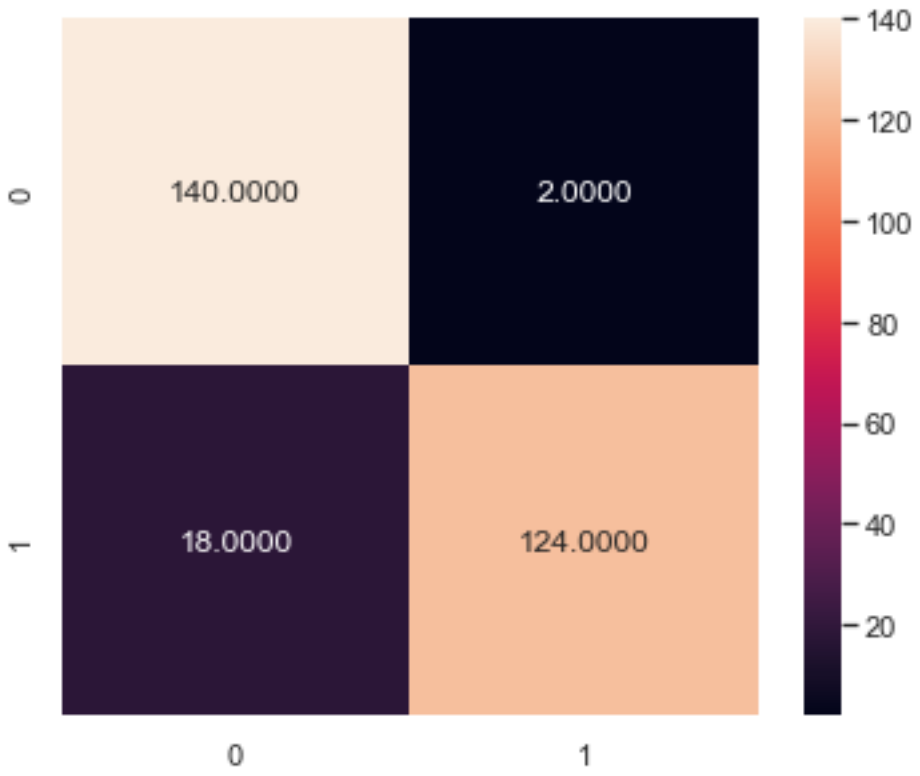
[239]:
```
[0.18961967527866364, 0.9295774698257446, 0.8732394576072693]
```

De hecho estos son resultados muy favorables ya que tenemos un **accuracy 92.9%** y un **recall de 87.3%** wow estos son resultados muy buenos, y mas si lo que buscabamos era un buen rendimiento con la metrica **Recall**

- Ahora veamos la matriz de confusion, nuestro objetivo aqui es ver si los falsos negativos han reducido

[240]:
```
display_report_and_matrix_confusion(model_unsampling,x_test,y_test)
```

```
REPORTE DE CLASIFICACION
              precision    recall  f1-score   support

         0.0       0.89      0.99      0.93       142
         1.0       0.98      0.87      0.93       142

    accuracy                           0.93       284
   macro avg       0.94      0.93      0.93       284
weighted avg       0.94      0.93      0.93       284


 Matriz de confusion
```



Como el modelo tiene un buen rendimiento optare por guardar el modelo con el formato **SavedModel** para futuras integraciones

[241]:
```
tf.saved_model.save(model_unsampling,"./best_endpoints/model_unsampling")
```

```
INFO:tensorflow:Assets written to: ./best_endpoints/model_unsampling\assets
```

[243]:
```
#guardamos el metodo escalador
import joblib
joblib.dump(scaler_usmp,"./best_endpoints/scaler_unsampling.pkl")
```

[243]:
```
['./best_endpoints/scaler_unsampling.pkl']
```

0.2.5 CONCLUSIONES PARCIALES DE LOS RESULTADOS

Tener en cuenta que se tomo como criterio principal aquel que posee un mayor numero de **recall** porque el caso de las transacciones son muy criticas

- El mejor modelo fue el de datos sobremuestreados con un puntaje de **99% de accuracy y 99% de recall** sobre datos nunca vistos.
- 2do mejor modelo fue el de datos submuestreados con **92% de accuracy** y **87% de recall**

PRUEBA FINAL. EVALUACION EN TODOS LOS DATOS -> No Necesaria

[20]:
```
from sklearn.metrics import␣
↪accuracy_score,recall_score,precision_score,f1_score,confusion_matrix
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import joblib
import tensorflow as tf

def test_best_models(X,y,list_components=None):

    score_models={
        "accuracy":[],"recall":[],"precision":[],"f1":[]
    }

    list_matrix=[]

    for path_scaler,path_model in␣
↪zip(list_components["scalers"],list_components["models"]):

        scaler=joblib.load(path_scaler,mmap_mode="r")
        model=tf.keras.models.load_model(path_model)
        x_scaled=scaler.transform(X)
        y_pred=model.predict(x_scaled)
        y_pred=np.where(y_pred>=0.5,1,0)

        score_models["accuracy"].append(accuracy_score(y,y_pred))
        score_models["recall"].append(recall_score(y,y_pred))
        score_models["precision"].append(precision_score(y,y_pred))
        score_models["f1"].append(f1_score(y,y_pred))

        mat=confusion_matrix(y,y_pred)
        list_matrix.append(mat)

    report=pd.DataFrame(score_models,index=list_components["names"])

    return report,list_matrix
```

La funcion que preparamos espera un diccionario que posea las rutas de los modelos y los objetos escalers que se usaron sobre los datos

[16]:
```
list_components={
    "names":["Base Model","Modelo sobremuestreado","Base Model␣
↪Mejorado","Modelo Submuestreado"],
```

```
    "models":["./best_endpoints/base_model","./best_endpoints/
 ↪model_oversampling","./best_endpoints/best_model_imbalanced","./
 ↪best_endpoints/model_unsampling"],
    "scalers":["./best_endpoints/scaler_imbalanced.pkl","./best_endpoints/
 ↪scaler_oversampling.pkl","./best_endpoints/scaler_best_imbalanced.pkl","./
 ↪best_endpoints/scaler_unsampling.pkl"]
}

X,y=load_data("./creditcard_imbalanced.csv") #este es el verdadero archivo␣
 ↪inicial con los datos verdaderos
```

Mostramos las comparativas

[21]:
```
report,list_matrix=test_best_models(X,y,list_components)
```

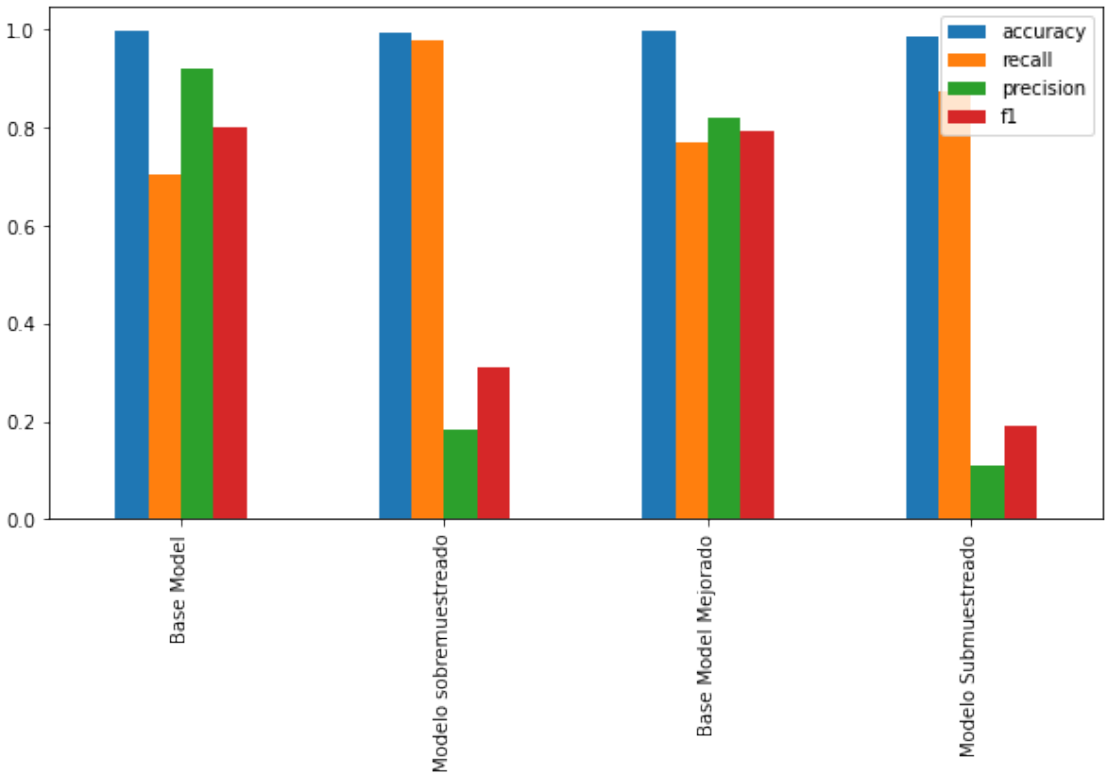
[22]:
```
report
```

[22]:

	accuracy	recall	precision	f1
Base Model	0.999408	0.706131	0.920110	0.799043
Modelo sobremuestreado	0.992708	0.978858	0.183584	0.309182
Base Model Mejorado	0.999334	0.771670	0.818386	0.794342
Modelo Submuestreado	0.987773	0.873150	0.108059	0.192317

[26]:
```
report.plot(kind="bar",figsize=(10,5))
```

[26]:
```
<AxesSubplot:>
```

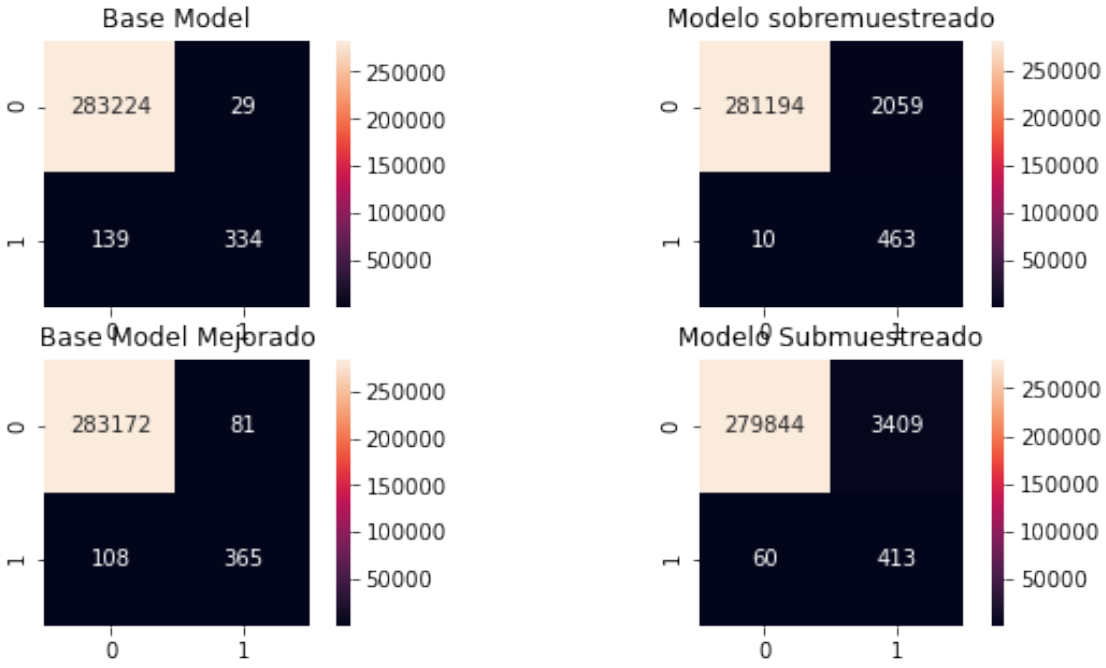


A primera vista es normal pensar que el modelo sobremuestreado no es mejor que el modelo base y otros, pero la verdad es que no es asi, **debemos fijarnos mas en los objetivos del modelo y los datos predecidos**

- **Objetivos iniciales**
- Evitar las transacciones fraudulentas
- Construir un modelo sensible al fraude donde sea preferible clasificar una transaccion normal como sospechosa que dejarla ir y al final sea un fraude que traera costos economicos a la entidad financiera.
- Las transacciones frudulentas deben ser lo mas minimo posible

Ahora veamos los resultados a traves de la matriz de confusion

[33]:
```
fig,axes=plt.subplots(2,2,figsize=(10,5))

for name,mat,ax in zip(list_components["names"],list_matrix,axes.flat):
    sns.heatmap(mat,annot=True,square=True,fmt="0.0f",ax=ax)
    ax.set_title(name)
plt.show()
```



Breve explicacion de las metricas bajas:

- **Precision**: Mide intuitivamente la capacidad de no clasificar como positivo una clase que es negativa [tp / (tp + fp)].Por ello la metrica de precision observada en el modelo sobremuestreado es muy baja, por que ha obtenido muchos falsos positivos siendo el modelo sensible a las transacciones fraudulentas
- **f1**: Combina intuitivamente las metricas Recall y Precision

0.2.6 CONCLUSIONES FINALES

- De hecho se suele pensar que el modelo con mas **accuracy** es el mejor, pero este no siempre es asi. En problemas de ciencia de datos donde es mejor obtener mas falsos positivos que falsos negativos como es el deteccion de fraude bancario, debemos tener en cuenta todas las metricas, y en este caso el **recall** debido a que si nos fijamos muy bien, el **modelo con datos sobremuestreados** ha sido el modelo que mas **recall** ha tenido y dejo pasar menos transacciones fraudulentas.
- El **modelo sobremuestreado solo tiene 10 Falsos negativos** que el resto de modelos. Esto a coste de que sea mas sensible al fraude y tenga mas Falsos positivos.
- En la vida real tener muchos falsos negativos, que son en este caso transacciones fraudulentas que no identificamos bien, equivalen a millones de soles de perdida de una entidad financiera y mas aun cuando los clientes estan asegurados.
- Los falsos positivos son aquellas transacciones normales que a menudo el modelo reconoce como fraudulenta, y es mejor asi, debido a que nos conviene enviar una confirmacion de la actividad bancaria del cliente por si hay sospechas de la transaccion, a dejarla pasar corriendo

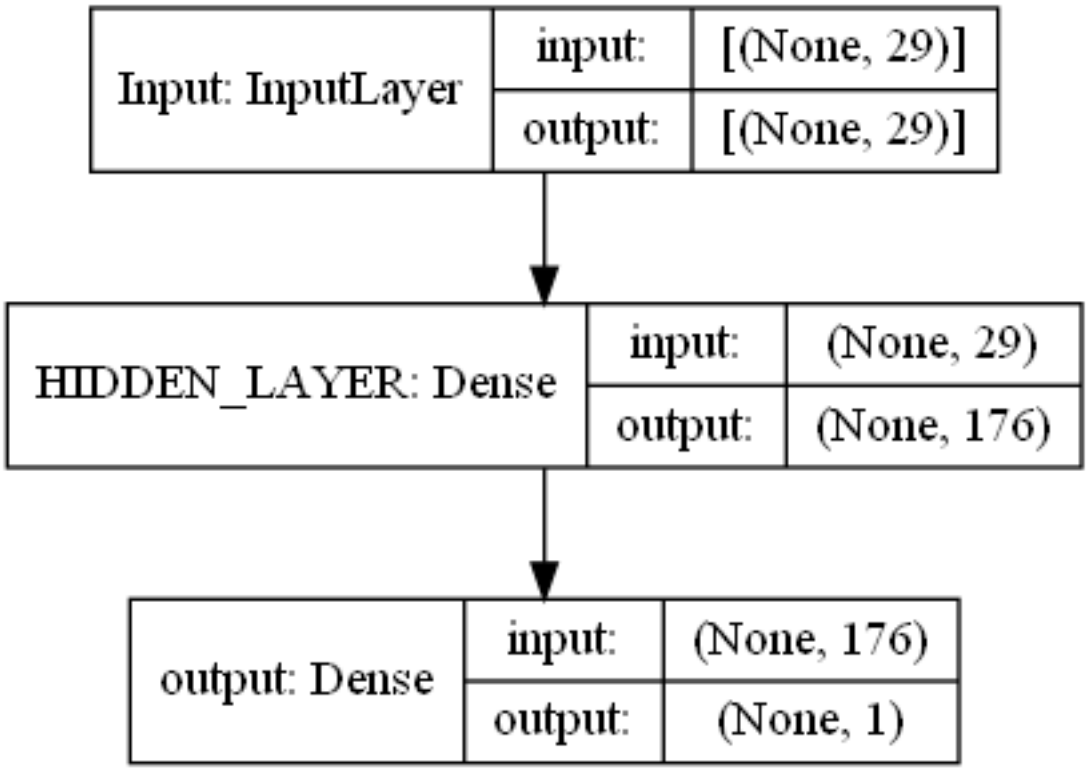
el riesgo de que si sea fraudulenta. Esto se llama **Costo del Falso Positivo por Falsos Negativos**

El mejor modelo es el que uso sobremuestreo para obtener un conjunto de datos mas equilibrado, este es el que debe ir a produccion.

[35]:
```
model_deploy=tf.keras.models.load_model("./best_endpoints/model_oversampling")
```

[36]:
```
tf.keras.utils.plot_model(model_deploy,"./model_final.png",show_shapes=True)
```

[36]:



[37]:
```
model_deploy.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
HIDDEN_LAYER (Dense)         (None, 176)               5280
_________________________________________________________________
output (Dense)               (None, 1)                 177
=================================================================
Total params: 5,457
Trainable params: 5,457
Non-trainable params: 0
_________________________________________________________________
```

El modelo se puede implementar con **Tensorlow Serving o ML Flow** para asegurar su escalamiento, e inclusive se pueden hacer pruebas de test usando **Flask**.

Autor * Johan valerio Mitma Huaccha, 20 años * Estudiante de Ing. sistemas en la UNMSM

Dedicado a mi madre Isabel Huaccha

Gracias

[39]:
```
%load_ext tensorboard
%tensorboard --logdir="./graph/fraud_detection/" #visualizacion interactiva de␣
↪las graficas de los modelos
```

```
Reusing TensorBoard on port 6006 (pid 32204), started 1:44:14 ago. (Use '!kill␣
↪32204' to kill it.)
```

```
<IPython.core.display.HTML object>
```

[ ]: