

AUTORES

- Vago, Matias
- Vega, Sharim
- Grandinetti, Juan
- Scoflich, Lautaro

```
In [ ]: # Procesamiento y estructura de datos
import pandas as pd
import numpy as np

# Visualización
import matplotlib.pyplot as plt
import seaborn as sns

from pprint import pprint

# Eliminar warnings
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
```

```
In [ ]: def Vertical(x):
        pprint(list(x))
```

DATOS

Carga

```
In [ ]: df = pd.read_csv("BankChurners.csv")
df.head()
```

```
Out[ ]:
```

	CLIENTNUM	Attrition_Flag	Customer_Age	Gender	Dependent_count	Education_Level	Marital_Status	Incc
0	768805383	Existing Customer	45	M	3	High School	Married	
1	818770008	Existing Customer	49	F	5	Graduate	Single	
2	713982108	Existing Customer	51	M	3	Graduate	Married	
3	769911858	Existing Customer	40	F	4	High School	Unknown	
4	709106358	Existing Customer	40	M	3	Uneducated	Married	

5 rows × 23 columns

Nuestro objetivo es intentar clasificar a los clientes como existentes o desertores de la entidad bancaria, a partir de las features dadas.

Pre-Análisis

```
In [ ]: # Cantidad de muestras
len(df)
```

```
Out[ ]: 10127
```

```
In [ ]: # Features
pprint(len(df.columns))
Vertical(df.columns)

23
['CLIENTNUM',
 'Attrition_Flag',
 'Customer_Age',
 'Gender',
 'Dependent_count',
 'Education_Level',
 'Marital_Status',
 'Income_Category',
 'Card_Category',
 'Months_on_book',
 'Total_Relationship_Count',
 'Months_Inactive_12_mon',
 'Contacts_Count_12_mon',
 'Credit_Limit',
 'Total_Revolving_Bal',
 'Avg_Open_To_Buy',
 'Total_Amt_Chng_Q4_Q1',
 'Total_Trans_Amt',
 'Total_Trans_Ct',
 'Total_Ct_Chng_Q4_Q1',
 'Avg_Utilization_Ratio',
 'Naive_Bayes_Classifier_Attrition_Flag_Card_Category_Contacts_Count_12_mon_Dependent_count_Education_Level_Months_Inactive_12_mon_1',
 'Naive_Bayes_Classifier_Attrition_Flag_Card_Category_Contacts_Count_12_mon_Dependent_count_Education_Level_Months_Inactive_12_mon_2']
```

```
In [ ]: df = df.drop( ["CLIENTNUM",
 'Naive_Bayes_Classifier_Attrition_Flag_Card_Category_Contacts_Count_12_mon_Dependent_count_Ed
 'Naive_Bayes_Classifier_Attrition_Flag_Card_Category_Contacts_Count_12_mon_Dependent_count_Ed
 axis=1)
```

```
In [ ]: Inc_dict = {'Less than $40K': '<40K' , '$40K - $60K': '40K - 60K', '$60K - $80K': '60K - 80K', '
df["Income_Category"] = df["Income_Category"].replace(Inc_dict)

df["Income_Category"].unique()
```

```
Out[ ]: array(['60K - 80K', '<40K', '80K - 120K', '40K - 60K', '>120K', 'Unknown'],
             dtype=object)
```

```
In [ ]: # Features
pprint(len(df.columns))
Vertical(df.columns)
```

20

```
['Attrition_Flag',  
'Customer_Age',  
'Gender',  
'Dependent_count',  
'Education_Level',  
'Marital_Status',  
'Income_Category',  
'Card_Category',  
'Months_on_book',  
'Total_Relationship_Count',  
'Months_Inactive_12_mon',  
'Contacts_Count_12_mon',  
'Credit_Limit',  
'Total_Revolving_Bal',  
'Avg_Open_To_Buy',  
'Total_Amt_Chng_Q4_Q1',  
'Total_Trans_Amt',  
'Total_Trans_Ct',  
'Total_Ct_Chng_Q4_Q1',  
'Avg_Utilization_Ratio']
```

```
In [ ]: categoricas = ['Attrition_Flag', 'Gender', 'Education_Level', 'Marital_Status', 'Income_Categ  
  
numericas = ['Customer_Age', 'Dependent_count', 'Months_on_book', 'Total_Relationship_Count', 'Mon  
            'Contacts_Count_12_mon', 'Credit_Limit', 'Total_Revolving_Bal',  
            'Avg_Open_To_Buy', 'Total_Amt_Chng_Q4_Q1', 'Total_Trans_Amt',  
            'Total_Trans_Ct', 'Total_Ct_Chng_Q4_Q1', 'Avg_Utilization_Ratio'  
            ]
```

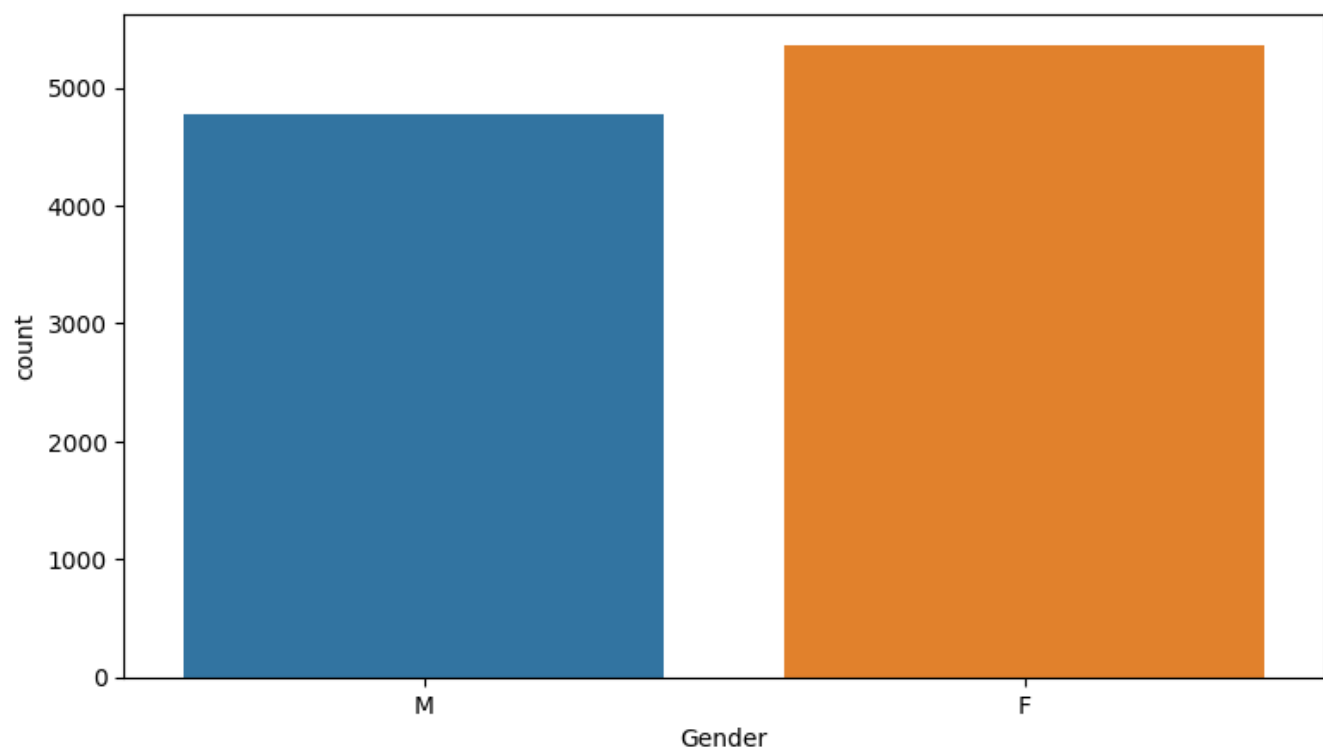
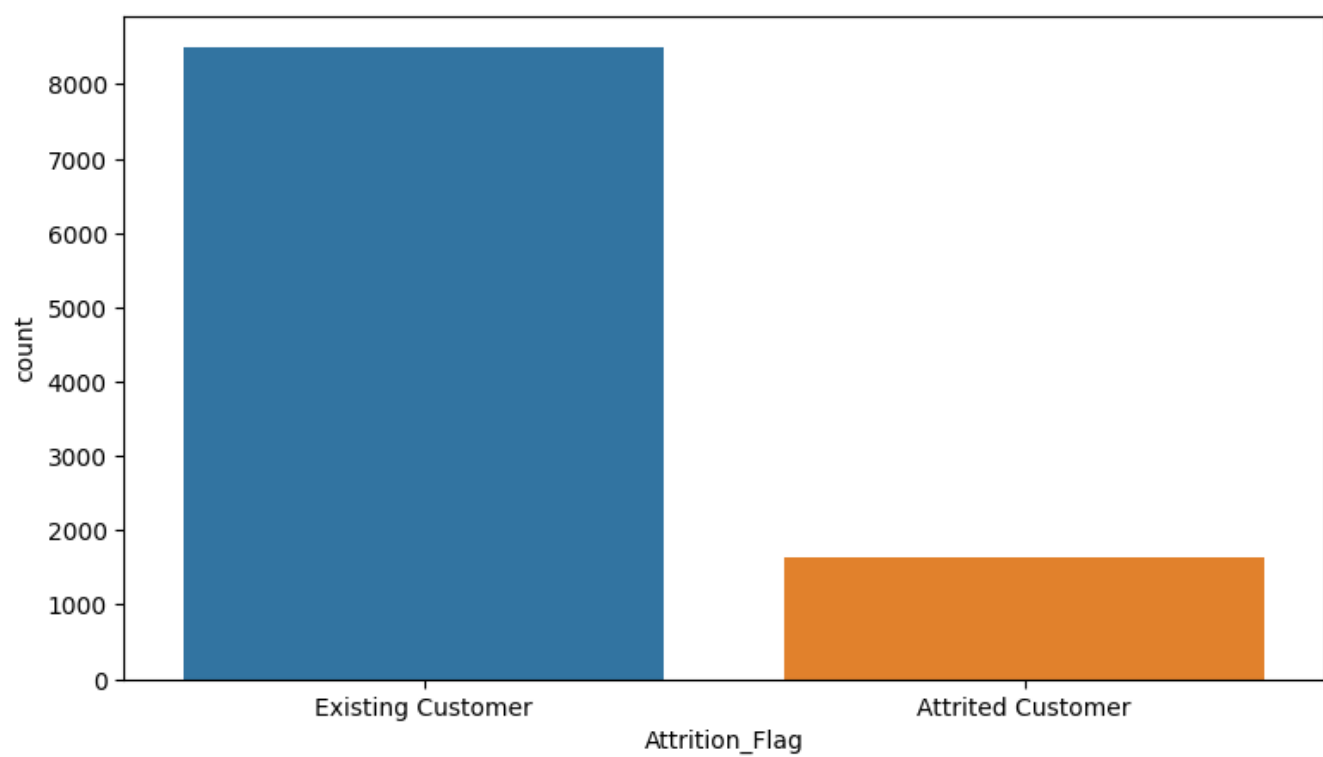
Análisis exploratorio de datos

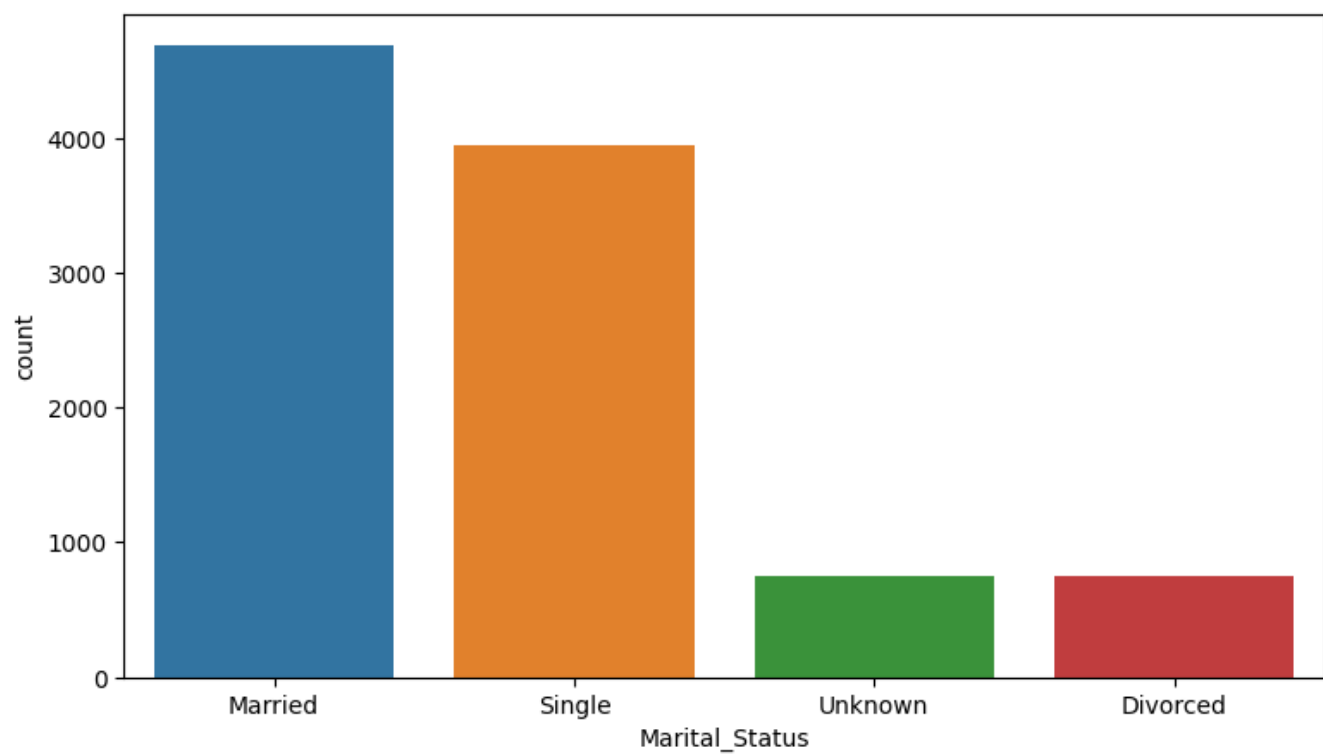
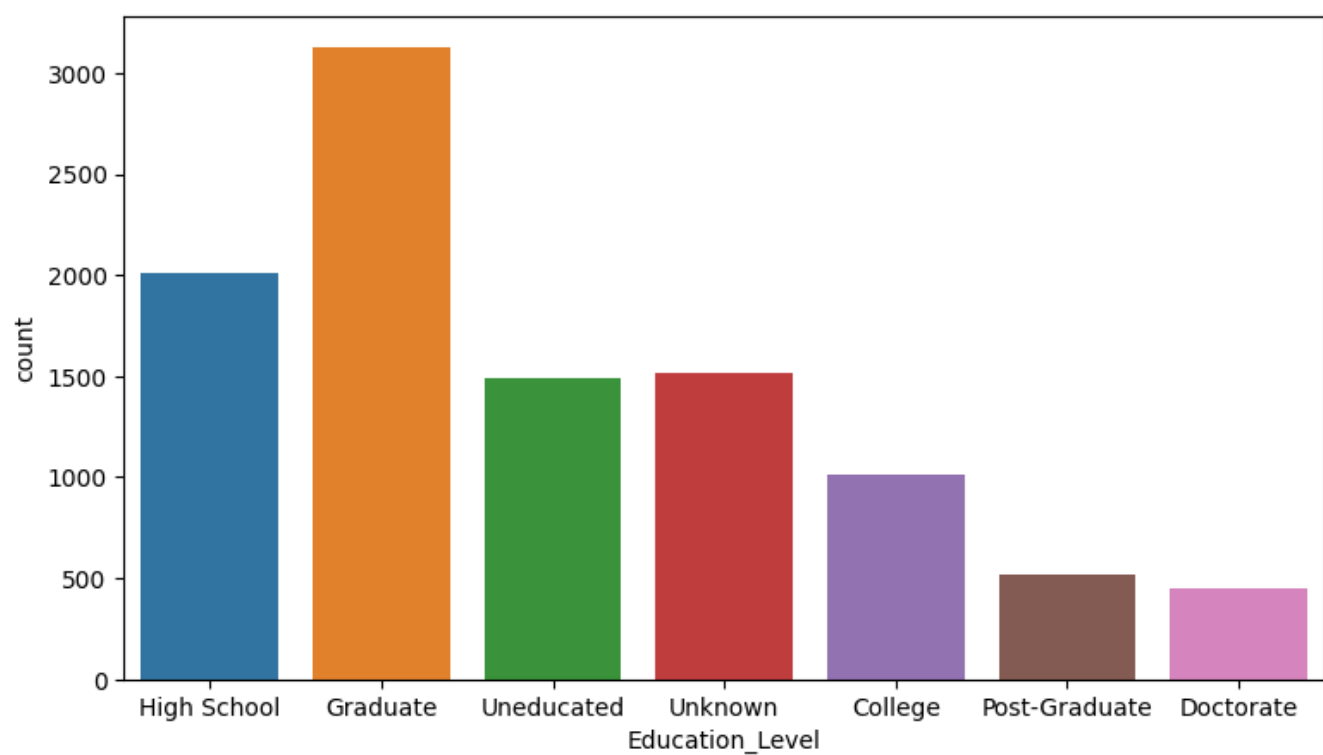
```
In [ ]: df[numericas].describe()
```

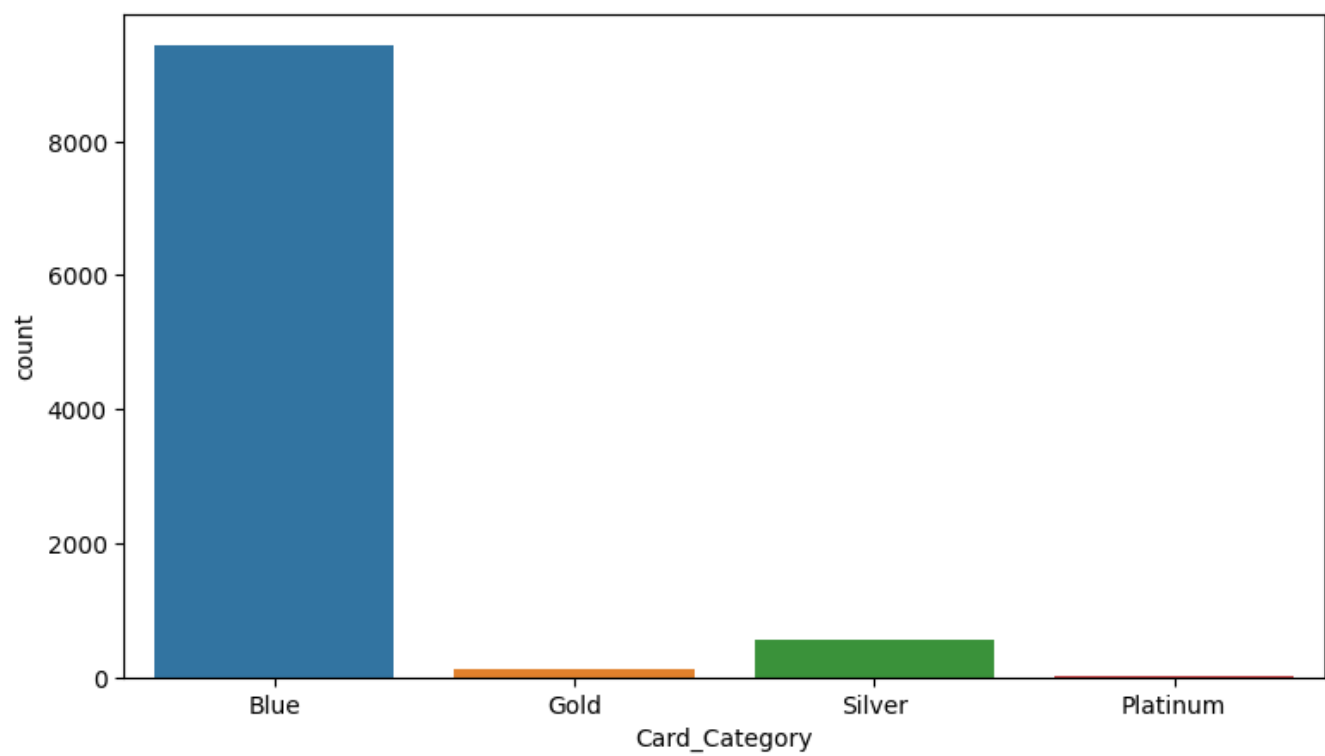
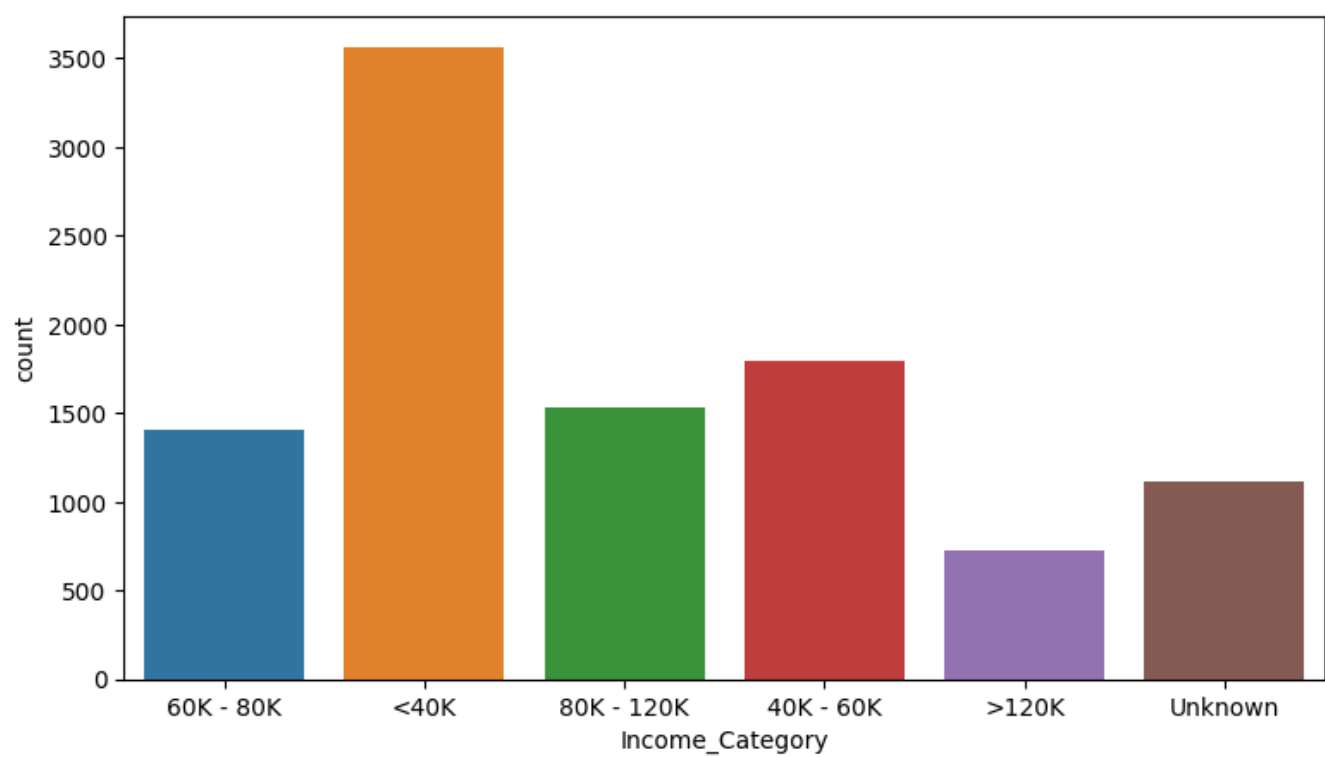
```
Out [ ]:
```

	Customer_Age	Dependent_count	Months_on_book	Total_Relationship_Count	Months_Inactive_12_mon
count	10127.000000	10127.000000	10127.000000	10127.000000	10127.000000
mean	46.325960	2.346203	35.928409	3.812580	2.341167
std	8.016814	1.298908	7.986416	1.554408	1.010622
min	26.000000	0.000000	13.000000	1.000000	0.000000
25%	41.000000	1.000000	31.000000	3.000000	2.000000
50%	46.000000	2.000000	36.000000	4.000000	2.000000
75%	52.000000	3.000000	40.000000	5.000000	3.000000
max	73.000000	5.000000	56.000000	6.000000	6.000000

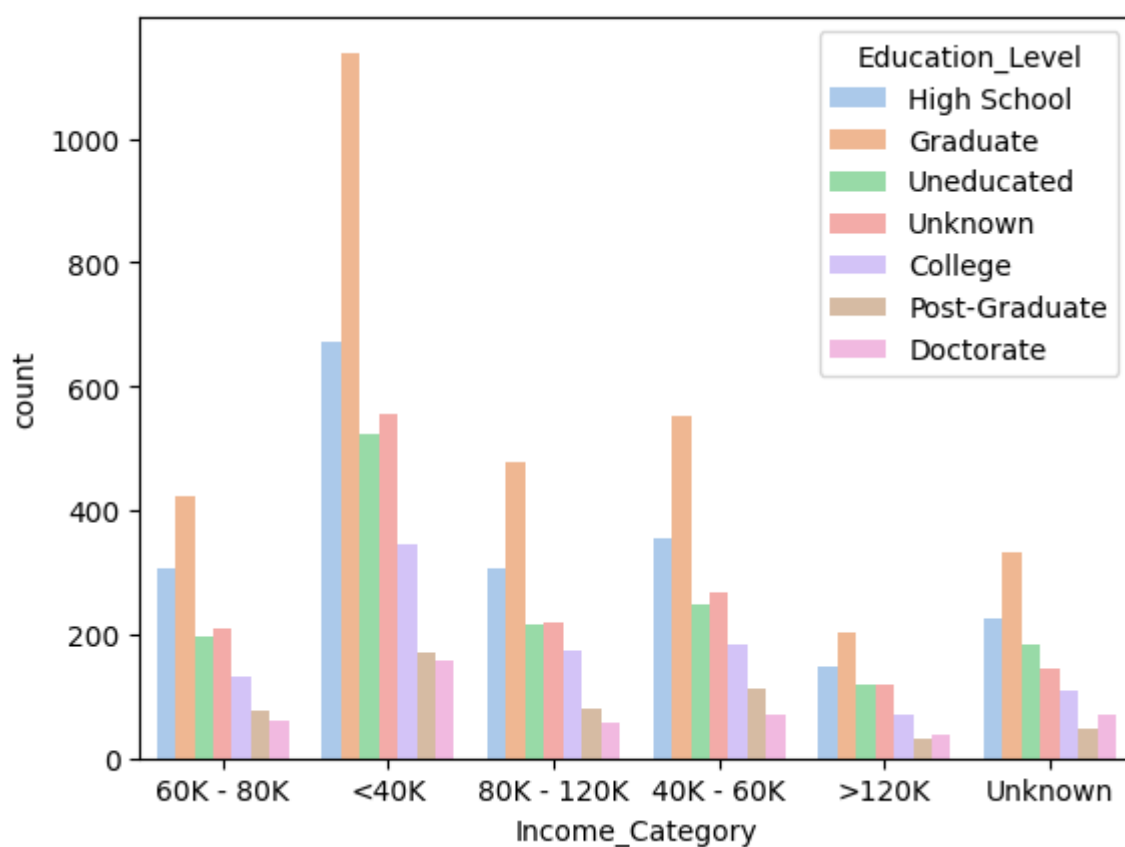
```
In [ ]: for i, col in enumerate(df[categoricas]):  
        plt.figure(i, figsize=(9,5), dpi=100)  
        sns.countplot(x=col, data=df[categoricas])
```







```
In [ ]: _ = sns.countplot(x='Income_Category', hue='Education_Level', palette='pastel', data=df)
```



Gracias a estas visualizaciones, podemos extraer algunas conclusiones:

GENDER

- Podemos tratar a los clientes cuya situación matrimonial es desconocida como solteros.

ATRITION_FLAG

- Estamos frente a un problema desbalanceado, donde el estándar es que un cliente se quede en el banco. Debemos tenerlo en cuenta al momento de entrenar y evaluar al modelo, ya que no queremos que este simplemente se dedique a predecir a todos los clientes como "No Desertores".

Tratamiento de datos desconocidos

```
In [ ]: len(df)
```

```
Out[ ]: 10127
```

```
In [ ]: # Situación sentimental desconocida -> Solteros
```

```
df["Marital_Status"] = df["Marital_Status"].replace( {"Unknown":"Single"})
```

```
In [ ]: # Nivel educativo desconocido -> Eliminamos el registro
```

```
df["Education_Level"] = df["Education_Level"].replace( {"Unknown":pd.NA})
```

```
In [ ]: # Categoría de ingresos desconocida -> Eliminamos el registro
```

```
df["Income_Category"] = df["Income_Category"].replace( {"Unknown":pd.NA})
```

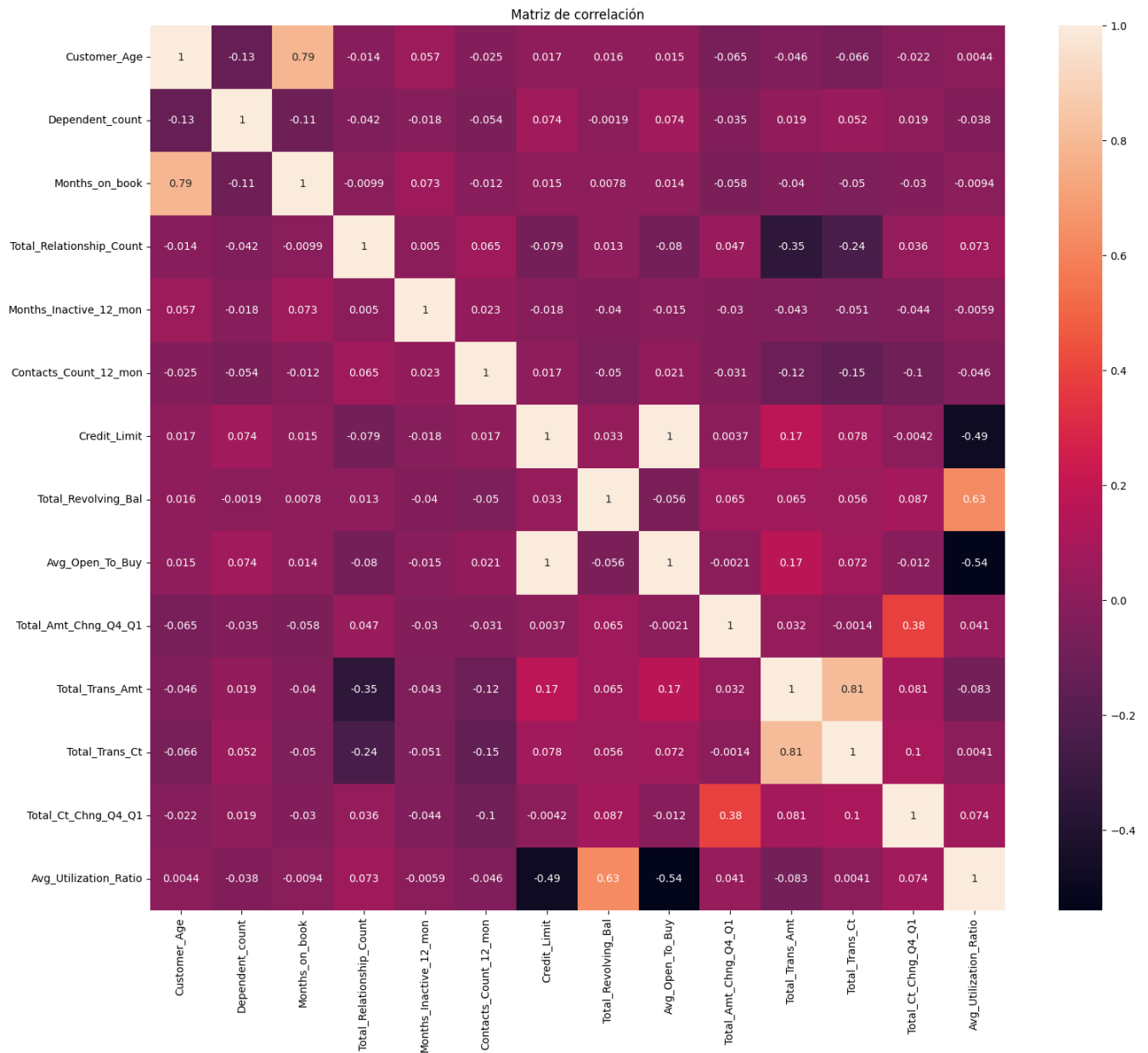
```
In [ ]: df.dropna(inplace=True)
```

```
In [ ]: len(df)
```

```
Out[ ]: 7641
```

```
In [ ]: _ = plt.figure(figsize=(18,15))
_ = sns.heatmap(df.corr(), annot=True)

_ = plt.title("Matriz de correlación")
```

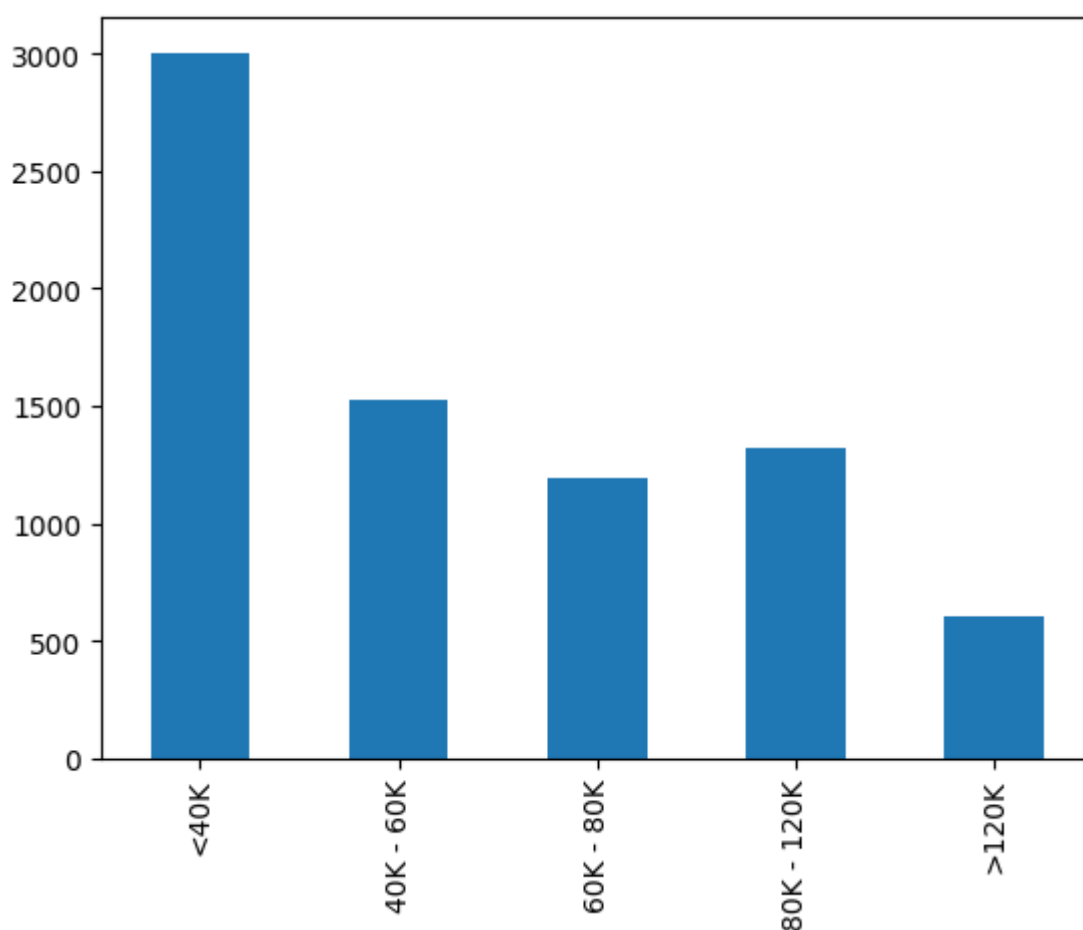


```
In [ ]: Inc_dict = {'Less than $40K': '<40K' , '$40K - $60K': '40K - 60K', '$60K - $80K': '60K - 80K', '
df["Income_Category"] = df["Income_Category"].replace(Inc_dict)

df["Income_Category"].unique()
```

```
Out[ ]: array(['60K - 80K', '<40K', '80K - 120K', '40K - 60K', '>120K'],
dtype=object)
```

```
In [ ]: _ = df["Income_Category"].value_counts().loc[['<40K', '40K - 60K', '60K - 80K', '80K - 120K',
```

PROCESAMIENTO DE DATOS

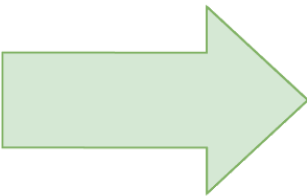
Categoricos - Codificación

Al estar trabajando con datos categóricas, hay que transformarlos en algo que pueda ser procesado por un algoritmo que trabaja con datos numéricos.

En caso de tratarse de variables con dos posibles estados, eso se resuelve con One Hot Encoding, es decir, utilizando 1 para la presencia y 0 para la ausencia de esta categoría.

Sin embargo, para variables categóricas que puedan tomar varios valores, se procede a crear nuevas variables llamadas Dummies o "ficticias", de tal manera que si tenemos por ejemplo una variable categórica que puede tomar 3 posibles valores, se convertiría de la siguiente manera:

Color				
0	Rojo			
1	Azul			
2	Verde			



	COLOR_Rojo	COLOR_Azul	COLOR_Verde
0	1	0	0
1	0	1	0
2	0	0	1

```
In [ ]: # One Hot Encoding para Attrition Flag

df['Attrition_Flag'].replace(['Existing Customer', 'Attrited Customer'], [0, 1], inplace=True)
```

```
In [ ]: categoricas
```

```
Out[ ]: ['Attrition_Flag',  
        'Gender',  
        'Education_Level',  
        'Marital_Status',  
        'Income_Category',  
        'Card_Category']
```

```
In [ ]: # One Hot Encoding + Variables ficticias para Las variables categóricas
```

```
encoded = pd.get_dummies(df[categoricas], drop_first=True)  
  
data = pd.concat([df.drop(categoricas, axis=1), encoded], axis=1)  
data
```

```
Out[ ]:
```

	Customer_Age	Dependent_count	Months_on_book	Total_Relationship_Count	Months_Inactive_12_mon
0	45	3	39	5	1
1	49	5	44	6	1
2	51	3	36	4	1
3	40	4	34	3	4
4	40	3	21	5	1
...
10121	56	1	50	4	1
10122	50	2	40	3	2
10124	44	1	36	5	3
10125	30	2	36	4	3
10126	43	2	25	6	2

7641 rows × 30 columns

Numéricos - Escalamiento

Los datos de entrada numéricos están distribuidos a lo largo de un rango de valores, cuya extensión depende de la naturaleza de los mismos. Algunos pueden ir de 0 a 1, otros de 0 a 10000, otros tomar valores negativos, etc.

Esta diferencia de escalas presente entre las variables de entrada puede distorsionar la percepción del modelo sobre la importancia de cada una, produciendo inestabilidades y haciendo que nuestro modelo se comporte de forma errática en el proceso de aprendizaje, y entregue pobres resultados durante su implementación.

Es por eso que se procede a escalar los datos, para que todos varíen entre los mismos valores pero conserven sus posiciones relativas. De esta manera, no se modifica la información pero se evitan los problemas mencionados para el modelo.

```
In [ ]: X=data.drop(['Attrition_Flag'], axis=1)  
print(f"Entradas del modelo: {len(X.columns)}")  
  
y=data['Attrition_Flag']  
print("Salidas del modelo: 1")
```

Entradas del modelo: 29
Salidas del modelo: 1

```
In [ ]: from sklearn.preprocessing import StandardScaler
        scaler = StandardScaler()

        X = scaler.fit_transform(X)

        X_PRUEBA = X[-20:-1]
        y_PRUEBA = y[-20:-1]
```

Modelo

Selección de variables

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

        X_train, X_test, y_train, y_test = train_test_split(X[0:-20], y[0:-20], test_size=0.2, random
```

Definición de la red neuronal

```
In [ ]: import tensorflow as tf

        capas = [
            tf.keras.layers.BatchNormalization(),
            tf.keras.layers.Dense(29, activation='relu'),
            tf.keras.layers.Dense(45, activation='relu'),
            tf.keras.layers.Dense(7, activation='relu'),
            tf.keras.layers.Dense(1, activation='sigmoid'),
        ]

In [ ]: modelo = tf.keras.Sequential(capas)
```

Para compilar nuestro modelo, debemos indicar qué métricas utilizar, lo que nos lleva a la siguiente explicación.

F1 Score

Fundamentación teórica

Como se planteó anteriormente, estamos trabajando con un conjunto de datos **desbalanceado**, es decir, nuestra variable a predecir posee una distribución desigual, donde una de los posibles valores es mucho menos frecuente que otro. Esto puede traer un gran inconveniente, que es que nuestro modelo 'aprenda' a clasificar a todos los vectores de entrada como pertenecientes a la categoría más frecuente, debido a que si hace esto obtendrá por lo general altos puntajes de exactitud.

Para evitar esto, debemos implementar un sistema de puntaje para nuestro modelo que contemple los falsos positivos y los falsos negativos (error de tipo I y tipo II). Estos se consiguen con las métricas de *Precisión y Recall*

PRECISION

La precisión es la proporción de verdaderos positivos (TP) por la suma de verdaderos positivos (TP) y falsos positivos (FP).

RECALL

El Recall es la proporción de verdaderos positivos (TP) por la suma de verdaderos positivos (TP) y falsos negativos (FN).

F1 Score

Finalmente, si deseamos exigir que nuestro modelo tenga la capacidad de entregar tanto pocos falsos negativos como falsos positivos, debemos aplicar la métrica de F1 Score. Esta métrica se calcula como la media armónica entre la Precisión y el Recall de un modelo, por lo que valores bajos de uno de estos, castigará el resultado final sin importar que tan grande sea el otro. En otras palabras, por mas que tengamos muy pocos falsos positivos, si tenemos una gran cantidad de falsos negativos el puntaje caerá abruptamente, haciendo que las próximas iteraciones de entrenamiento de nuestro modelo intenten corregir esas clasificaciones.

Implementación

Por defecto, Keras nos permite compilar un modelo definiendo su función pérdida y sus métricas. Dentro de las métricas disponibles, no se encuentra el F1 Score. Sin embargo, tambien es posible agregar una función propia para calcular la métrica del modelo y pasarla por parámetro al compilador.

La definición de la función F1 Score fue tomada de [este enlace](#), que dirige al blog de Aakash Goel y a quien se le da el crédito de esta parte del presente trabajo y se le agradece por su útil contribución.

```
In [ ]: from keras.callbacks import Callback, ModelCheckpoint
from keras.models import Sequential, load_model
from keras.layers import Dense, Dropout
from keras.wrappers.scikit_learn import KerasClassifier
import keras.backend as K

def F1_SCORE(y_true, y_pred): #taken from old keras source code
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    possible_positives = K.sum(K.round(K.clip(y_true, 0, 1)))
    predicted_positives = K.sum(K.round(K.clip(y_pred, 0, 1)))
    precision = true_positives / (predicted_positives + K.epsilon())
    recall = true_positives / (possible_positives + K.epsilon())
    f1_val = 2*(precision*recall)/(precision+recall+K.epsilon())
    return f1_val
```

Learning Rate

Fundamentación teórica

También, debemos indicar qué tasa de aprendizaje empleará nuestro optimizador para hacer los ajustes luego de cada época, la cual le indica al optimizador cuánto variar los parámetros del modelo para realizar las correcciones.

Valores grandes de la tasa de aprendizaje resultan en una rápida estabilización del modelo, es decir, se corrigen los parámetros bruscamente y se llega rápido a un valor cercano al óptimo. Sin embargo, debido a que los saltos de valores son muy grandes, los parametros dificilmente lleguen a ser los optimos, sino que oscilaran alrededor de los mismos, produciendo que el modelo no pueda ajustarse finamente para seguir mejorando.

Entonces, optaríamos por una tasa de aprendizaje pequeña, que le daría una mayor sensibilidad al sistema y permite que el mismo se ajuste finamente. Sin embargo, esto provoca que si los parámetros estan lejos del valor deseado (como suele suceder al inicio del proceso de aprendizaje), el optimizador esté varias epocas realizando pequeños ajustes siempre en la misma dirección, para alcanzar un valor

que está muy lejos. Esto se traduce en la necesidad de muchas épocas de aprendizaje, y un enorme consumo de tiempo y recursos.

Para solucionar esto, podemos decirle inteligentemente al optimizador que comience con una tasa de aprendizaje grande para lograr una rápida estabilización del modelo, y que con el pasar de las épocas, a medida que el modelo está más cerca de los valores óptimos, la disminuya para poder realizar los ajustes finos.

Implementación

Para implementar este comportamiento de la tasa de aprendizaje, Keras nos proporciona las Schedules, funciones que devuelven una tasa de aprendizaje distinta, dependiendo del valor que toma la curva de aprendizaje. Podemos utilizar aquellas que vienen por defecto, o definir nosotros el comportamiento. Para este caso, optamos por una incluida en Keras llamada ExponentialDecay que nos dió buenos resultados.

```
In [ ]: lr_schedule = tf.keras.optimizers.schedules.ExponentialDecay(  
        initial_learning_rate=0.01,  
        decay_steps=10000,  
        decay_rate=0.7)
```

```
In [ ]: modelo.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=lr_schedule),  
                      loss=tf.keras.losses.BinaryCrossentropy(),  
                      metrics=["accuracy", F1_SCORE])
```

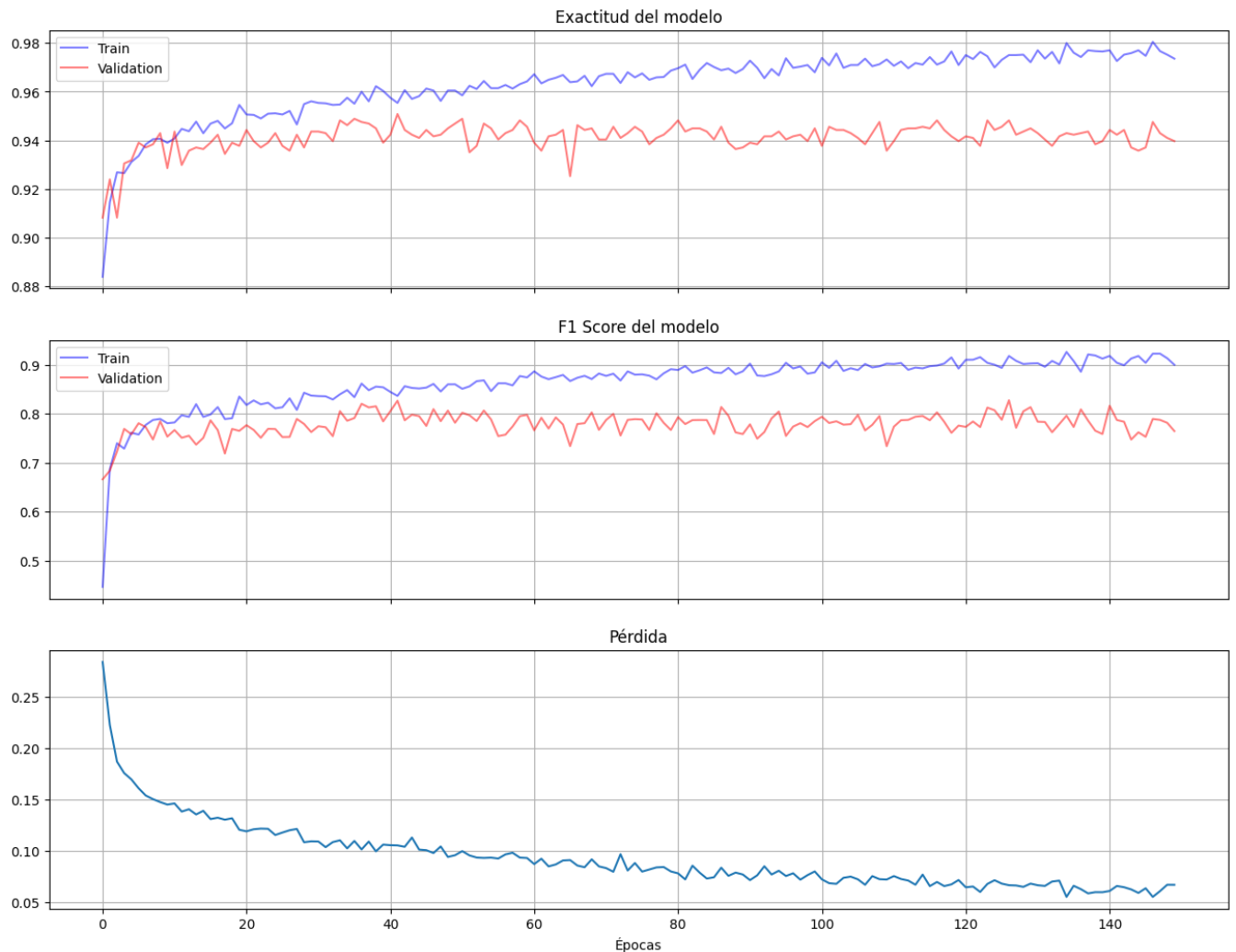
```
In [ ]: historial = modelo.fit(X_train, y_train, epochs=150, validation_data=(X_test,y_test), verbose
```

RESULTADOS

MÉTRICAS

A continuación, se visualizarán las métricas del modelo (Exactitud, F1 Score) y la función pérdida empleada por el optimizador, junto a su evolución a medida que avanzan las épocas de entrenamiento.

```
In [ ]: fig, axs = plt.subplots(3, 1, sharex=True, figsize = (16,12))  
  
axs[0].set_title("Exactitud del modelo")  
  
_ = axs[0].plot(historial.history["accuracy"], color = 'blue', label = "Train", alpha = 0.5)  
_ = axs[0].plot(historial.history["val_accuracy"], color = 'red', label = "Validation", alpha  
_ = axs[0].legend()  
_ = axs[0].grid()  
  
axs[1].set_title("F1 Score del modelo")  
  
_ = axs[1].plot(historial.history["F1_SCORE"], color = 'blue', label = "Train", alpha = 0.5)  
_ = axs[1].plot(historial.history["val_F1_SCORE"], color = 'red', label = "Validation", alpha  
_ = axs[1].legend()  
_ = axs[1].grid()  
  
axs[2].set_title("Pérdida")  
_ = axs[2].plot(historial.history["loss"])  
_ = axs[2].set_xlabel("Épocas")  
_ = axs[2].grid()
```



Estos son los valores finales del modelo:

```
In [ ]: print( f" Exactitud del modelo (Validación): {round(historial.history['val_accuracy'][-1],2)}
print( f" F1 Score del modelo (Validación): {round(historial.history['val_F1_SCORE'][-1],2)}

Exactitud del modelo (Validación): 0.94
F1 Score del modelo (Validación): 0.76
```

PRUEBAS

Finalmente, utilizaremos los valores que hemos dejado a un lado durante todo el proceso de aprendizaje (X_PRUEBA e y_PRUEBA), y por lo tanto nunca han sido vistos por el modelo en ninguna de sus etapas, para una pequeña demostración de su funcionamiento

Para ello, definiremos manualmente un umbral (Threshold) que se comparará con la salida de nuestro modelo. Si la salida supera dicho umbral, se considera que el cliente abandonará el banco, caso contrario permanecerá como cliente.

```
In [ ]: THRESHOLD = 0.6
original = y_PRUEBA.values
sigmoid = modelo.predict(X_PRUEBA)[:,-1]
resultados = pd.DataFrame([original, sigmoid]).T
resultados.columns = ["Original", "Sigmoid"]
resultados.head()
```

1/1 [=====] - 0s 69ms/step

Out[]:

	Original	Sigmoid
0	0.0	5.227188e-11
1	1.0	7.424560e-01
2	0.0	5.812403e-16
3	1.0	2.544011e-02
4	1.0	9.989283e-01

```
In [ ]: resultados["Sigmoid"] = resultados["Sigmoid"].apply(lambda t: round(t,2))
resultados["Predicción"] = resultados["Sigmoid"].apply(lambda t: 1 if t >= THRESHOLD else 0)
resultados
```

Out[]:

	Original	Sigmoid	Predicción
0	0.0	0.00	0
1	1.0	0.74	1
2	0.0	0.00	0
3	1.0	0.03	0
4	1.0	1.00	1
5	0.0	0.00	0
6	0.0	0.00	0
7	0.0	0.00	0
8	1.0	1.00	1
9	1.0	0.10	0
10	0.0	0.00	0
11	0.0	0.00	0
12	0.0	0.00	0
13	0.0	0.00	0
14	0.0	0.00	0
15	0.0	0.00	0
16	0.0	0.00	0
17	1.0	1.00	1
18	1.0	0.99	1

Como se puede ver, acierta en la mayoría de los casos con el entrenamiento y el umbral especificados.

CONCLUSIÓN

Hemos logrado predecir con éxito qué usuarios dejarán la entidad bancaria mediante un proceso de aprendizaje automático.

A lo largo del trabajo, hemos realizado distintos procesos de un trabajo de Machine Learning, tales como obtención de datos, visualización, análisis exploratorio, tratamiento de valores faltantes, entrenamiento, validación y testeo, junto con definición, análisis y visualización de métricas.

Como futuras líneas, este trabajo puede verse beneficiado por la reducción de dimensionalidad, eliminando variables que no aporten mucha información al análisis. También, con un mayor conocimiento técnico específico del tema a tratar, podemos trabajar sobre los datos de entrada para generar nuevas variables de mayor valor y aporte al modelo. Finalmente, se puede profundizar sobre la selección del umbral de decisión, buscando el valor óptimo del mismo y graficando el desempeño del mismo a medida que toma distintos valores.