Actividad Módulo 26: Introducción a Machine Learning Parte 2

- Importación del archivo de uso (bank-additional-full.csv)
- Feature Engineering (https://towardsdatascience.com/how-and-why-to-standardize-your-data-996926c2c832):
  - Variables de alto sesgo
  - Estandarización de variables
- Eliminar variables con alta correlación o variables no necesarias
- Generar test de training y testing (70% - 30%)
- Modelo simple de regresión logística
- Matriz de confusión y conclusiones del modelo
- Cálculo de las primeras medidas de calidad del modelo: Accuracy, Precision, Recall, F1, ROC
- Conclusiones

Standardization vs Normalization of data: https://towardsdatascience.com/standardization-vs-normalization-dc81f23085e3

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

In [ ]:
```
df = pd.read_csv('D:/Documentos/Documentos/DataAnalysis/ebac/Python/Modulo25/bank-a
df.sample(5)
```

Out[ ]:

	age	job	marital	education	default	housing	loan	contact	month	day
3952	27	services	single	high.school	no	yes	no	telephone	may	
12338	34	technician	single	high.school	no	yes	no	telephone	jul	
2648	35	technician	married	professional.course	no	no	no	telephone	may	
3742	36	blue-collar	married	basic.9y	no	no	no	telephone	may	
25478	30	blue-collar	single	basic.9y	no	no	no	cellular	nov	

5 rows × 21 columns

In [ ]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 41188 entries, 0 to 41187
Data columns (total 21 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   age             41188 non-null  int64  
 1   job             41188 non-null  object 
 2   marital         41188 non-null  object 
 3   education       41188 non-null  object 
 4   default         41188 non-null  object 
 5   housing         41188 non-null  object 
 6   loan            41188 non-null  object 
 7   contact         41188 non-null  object 
 8   month           41188 non-null  object 
 9   day_of_week     41188 non-null  object 
 10  duration        41188 non-null  int64  
 11  campaign        41188 non-null  int64  
 12  pdays           41188 non-null  int64  
 13  previous        41188 non-null  int64  
 14  poutcome        41188 non-null  object 
 15  emp.var.rate    41188 non-null  float64
 16  cons.price.idx  41188 non-null  float64
 17  cons.conf.idx   41188 non-null  float64
 18  euribor3m       41188 non-null  float64
 19  nr.employed     41188 non-null  float64
 20  y               41188 non-null  object 
dtypes: float64(5), int64(5), object(11)
memory usage: 6.6+ MB
```

In [ ]:

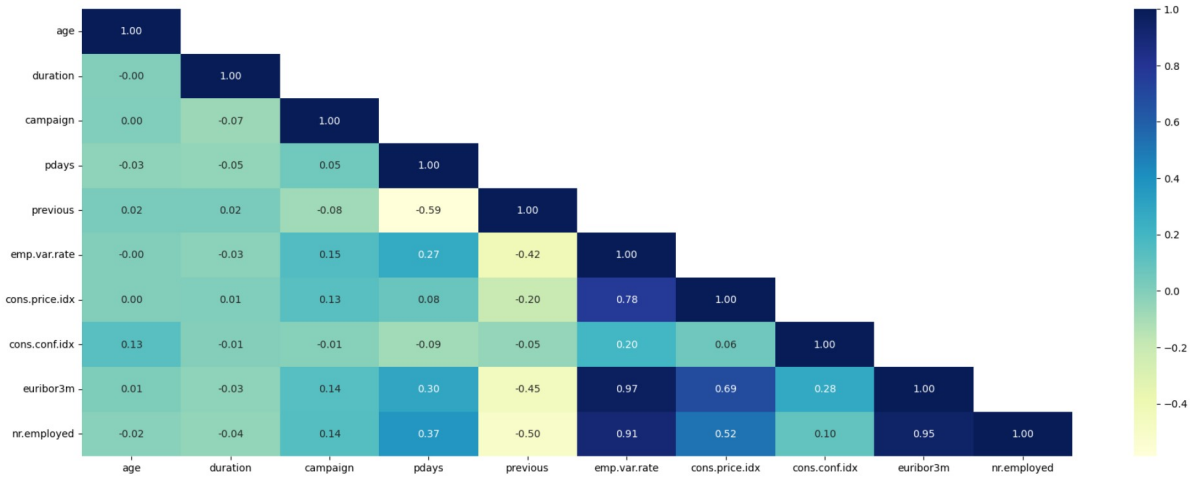
```
# 1. Imputación de datos
# Determinar si es necesaria la imputación de datos en caso de que haya campos vací
print('Cuenta de campos nulos por variable')
df.isnull().sum()
```

```
Cuenta de campos nulos por variable
```

Out[ ]:
```
age               0
job               0
marital           0
education         0
default           0
housing           0
loan              0
contact           0
month             0
day_of_week       0
duration          0
campaign          0
pdays             0
previous          0
poutcome          0
emp.var.rate      0
cons.price.idx    0
cons.conf.idx     0
euribor3m         0
nr.employed       0
y                 0
dtype: int64
```

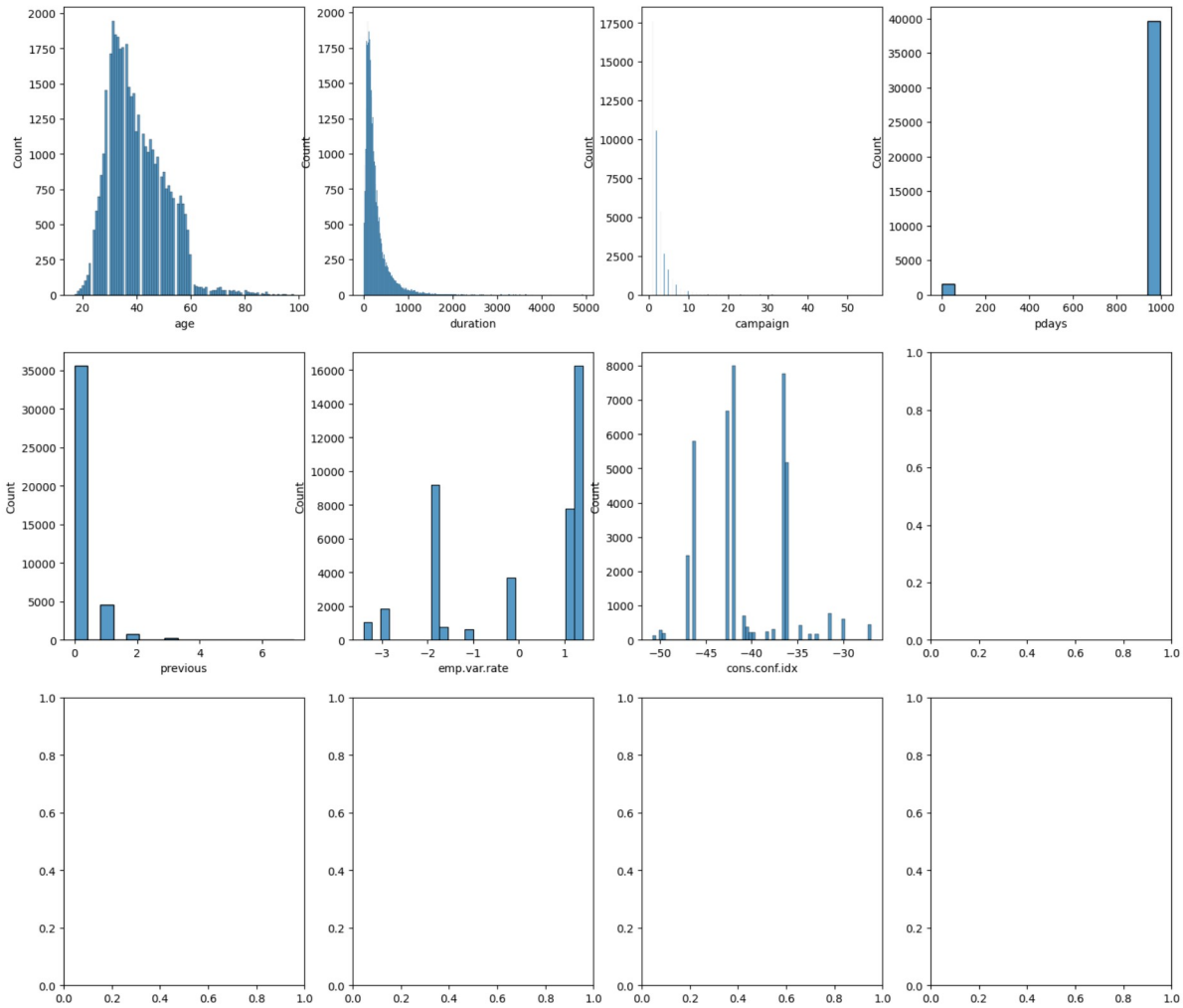
In [ ]:
```
# Eliminar variables con alta correlación y variables innecesarias
plt.figure(figsize=(22,8))
corr = df.corr(method='pearson')
df_lt = corr.where(np.tril(np.ones(corr.shape)).astype(bool))
hmap  = sns.heatmap(data=df_lt, annot=True, cmap='YlGnBu', fmt='.2f')
```

```
C:\Users\USER\AppData\Local\Temp\ipykernel_2112\1786607000.py:3: FutureWarning: The
default value of numeric_only in DataFrame.corr is deprecated. In a future version,
it will default to False. Select only valid columns or specify the value of numeric
_only to silence this warning.
  corr = df.corr(method='pearson')
```

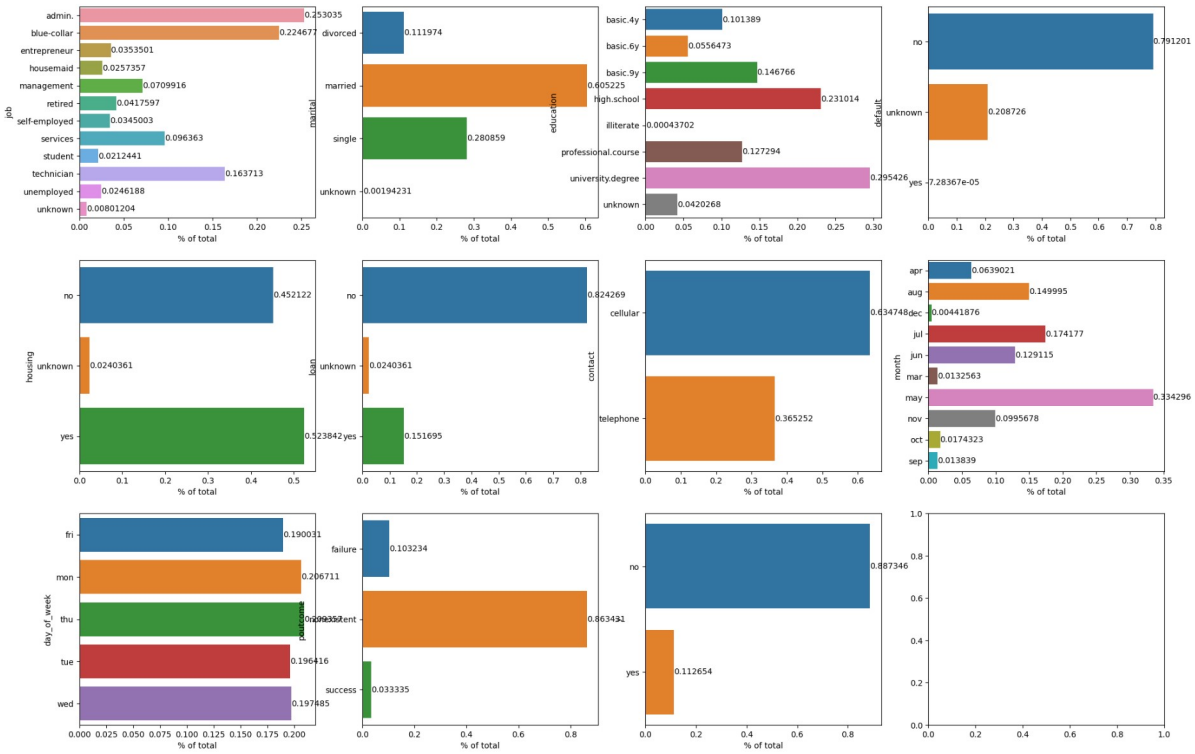


In [ ]:
```
# Eliminar columnas que tienen correlación alta
df.drop(columns=['cons.price.idx', 'euribor3m', 'nr.employed'], inplace=True)

# Nos quedamos con solo una variable que tiene correlación con todas las variables
```

In [ ]:
```
# 2. Análisis univariado - variables numéricas
# Graficar la distribución de las variables numéricas que presenta el dataset para

numeric_columns = []

for column in df.columns:
    if df[column].dtype != 'object':
        numeric_columns.append(column)

fig, ax = plt.subplots(nrows=3, ncols=4, figsize=(18,16))
for i, column in enumerate(numeric_columns):
    sns.histplot(data=df, x=column, ax=ax[i//4, i%4])
```



In [ ]:
```
# 2. Análisis univariado - variables categóricas

categorical_columns = []

for column in df.columns:
    if df[column].dtype == 'object':
        categorical_columns.append(column)

fig, ax = plt.subplots(nrows=3, ncols=4, figsize=(24,16))
for i, column in enumerate(categorical_columns):
    pivot = df.groupby(by=[column]).count()['age'].reset_index()
    pivot['age'] = pivot['age'] / np.sum(pivot['age'])
    pivot.style.format(formatter={'age':'{:.2%}'})
    fig = sns.barplot(data=pivot.rename(columns={'age': '% of total'}), x='% of tot
    for i in fig.containers:
        fig.bar_label(i,)
```

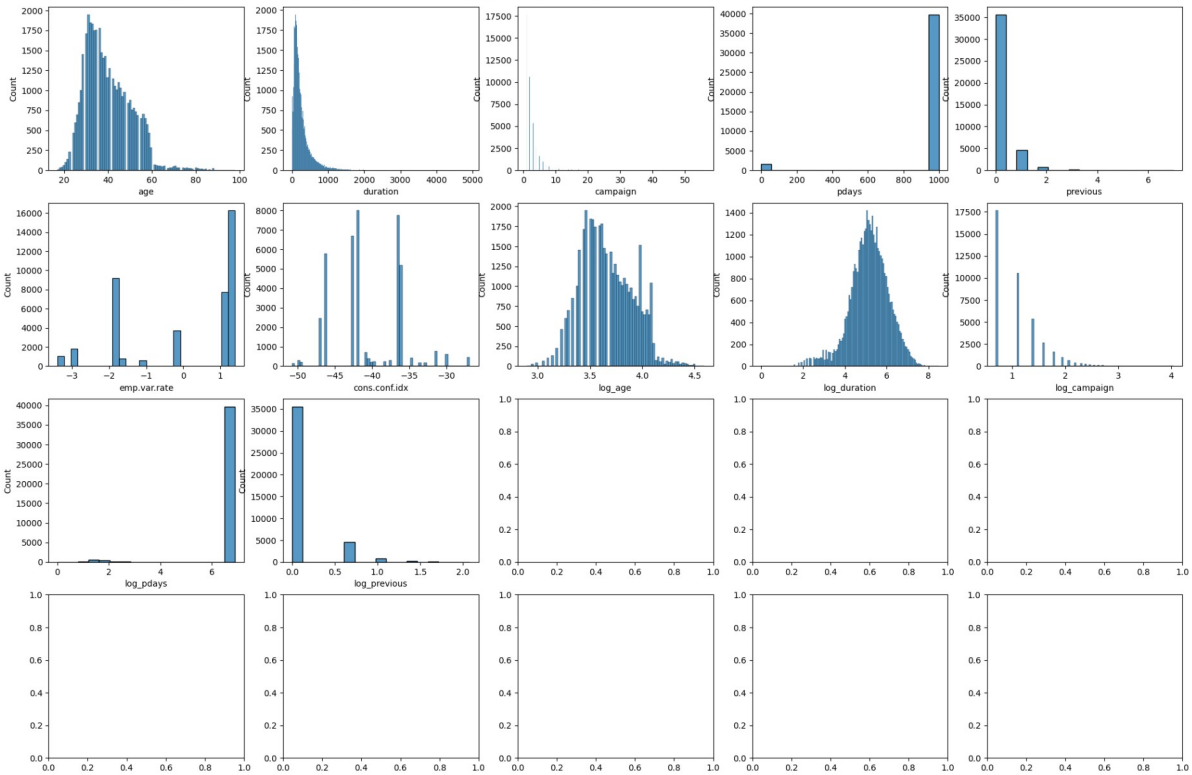


Insights:

- Las variables numéricas tienen, en general, mucho sesgo por lo que se tendrán que normalizar
- Algunas de las varaibles categórcias tienen clases desbalanceadas (representando menos del 5% de los registros)
    - Algunas de estas clases son la categoría unknown en todas las variables que aparece y otros grupos más puntuales
    - Será necesario juntar categorías en más de una variable bajo el concepto "Otros" para poder obtener una

In [ ]:
```
# Normalizar variables con sesgo

for column in numeric_columns:
    df['log_'+column] = np.log(df[column] + 1)

# Drop two that throw errors after being transformed with natural logarithm
df.drop(columns=['log_emp.var.rate','log_cons.conf.idx'], inplace=True)

numeric_columns = []
for column in df.columns:
    if df[column].dtype != 'object':
        numeric_columns.append(column)

fig, ax = plt.subplots(nrows=4, ncols=5, figsize=(24,16))
for i, column in enumerate(numeric_columns):
    sns.histplot(data=df, x=column, kde=False, ax=ax[i//5, i%5])


# La estandarización de variables usando StandardScaler no es necesario puesto que
```

```
d:\Programas\Python311\Lib\site-packages\pandas\core\arraylike.py:402: RuntimeWarni
ng: invalid value encountered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
d:\Programas\Python311\Lib\site-packages\pandas\core\arraylike.py:402: RuntimeWarni
ng: invalid value encountered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
```



In [ ]:
```
df.isna().sum()
```

Out[ ]:
```
age              0
job              0
marital          0
education        0
default          0
housing          0
loan             0
contact          0
month            0
day_of_week      0
duration         0
campaign         0
pdays            0
previous         0
poutcome         0
emp.var.rate     0
cons.conf.idx    0
y                0
log_age          0
log_duration     0
log_campaign     0
log_pdays        0
log_previous     0
dtype: int64
```

In [ ]:
```
# Agrupar clases desbalanceadas por variable categórica
# Variable job
df['job'] = df['job'].apply(lambda x : 'other' if  x in ['entrepreneur', 'self-empl

# Variable education
df['eduation'] = df['education'].apply(lambda x : 'other' if  x in ['unknown', 'ill

# El resto de las variables no tienen más de un grupo subrepresentado que se puedan
```

In [ ]:
```
# Transformación del resto de variables categóricas

# Transformación de la variable de respuesta en 1 y 0
df['y'] = df['y'].apply(lambda x : 1 if x == 'yes' else 0)

# Transformación de los meses en números
month_lookup = {'jan': 1, 'feb': 2,
                'mar': 3, 'apr': 4,
                'may': 5, 'jun': 6,
                'jul': 7, 'aug': 8,
                'sep': 9, 'oct': 10,
                'nov':11, 'dec':12
                }

df['month'] = df['month'].apply(lambda x : month_lookup[x])

df.sample(10)
```

Out[ ]:

	age	job	marital	education	default	housing	loan	contact	month	d
28942	47	blue-collar	married	basic.9y	no	no	no	cellular	4	
4919	42	blue-collar	divorced	basic.9y	no	no	no	telephone	5	
14887	52	admin.	married	high.school	no	no	no	cellular	7	
25678	34	technician	married	high.school	no	yes	no	telephone	11	
15140	49	admin.	married	high.school	unknown	yes	no	cellular	7	
5776	59	other	married	professional.course	unknown	yes	yes	telephone	5	
14069	43	technician	divorced	basic.9y	no	no	no	cellular	7	
39274	62	technician	married	unknown	no	no	no	cellular	3	
15620	27	other	single	high.school	unknown	yes	no	cellular	7	
18334	55	technician	married	professional.course	no	yes	yes	cellular	7	

10 rows × 24 columns

In [ ]:
```
df2 = df.copy()
```

In [ ]:
```
# Train - Test Split
from sklearn.model_selection import train_test_split

x = df2.drop(columns=['y'])
y = df2['y']

# Transformar variables categóricas en dummies
x = pd.get_dummies(x, drop_first=True)

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=.3, random_stat
```

In [ ]:
```
print("Original Personal Loan True Values    : {0} ({1:0.2f}%)".format(len(df2.loc[d
print("Original Personal Loan False Values   : {0} ({1:0.2f}%)\n".format(len(df2.loc

print("Training Personal Loan True Values    : {0} ({1:0.2f}%)".format(len(y_train.l
print("Training Personal Loan False Values   : {0} ({1:0.2f}%)".format(len(y_train.l
print('The Training dataset contains {0:0.2f}% of the registers in the original dat

print("Testing Personal Loan True Values    : {0} ({1:0.2f}%)".format(len(y_test.loc
print("Testing Personal Loan False Values   : {0} ({1:0.2f}%)".format(len(y_test.loc
print('The Testing dataset contains {0:0.2f}% of the registers in the original data
```

```
Original Personal Loan True Values   : 4640 (11.27%)
Original Personal Loan False Values  : 36548 (88.73%)

Training Personal Loan True Values   : 3227 (11.19%)
Training Personal Loan False Values  : 25604 (88.81%)
The Training dataset contains 70.00% of the registers in the original dataset

Testing Personal Loan True Values   : 1413 (11.43%)
Testing Personal Loan False Values  : 10944 (88.57%)
The Testing dataset contains 30.00% of the registers in the original dataset
```

In [ ]:
```
# Modelo de regresión logística simple
from sklearn.linear_model import LogisticRegression

# Crear y entrenar modelo de regresión logística
model = LogisticRegression(class_weight={0:.15, 1:.85}, solver='liblinear')
model.fit(X=x_train, y=y_train)

# Correr modelo con los datos de test
y_predict = model.predict(x_test)
```

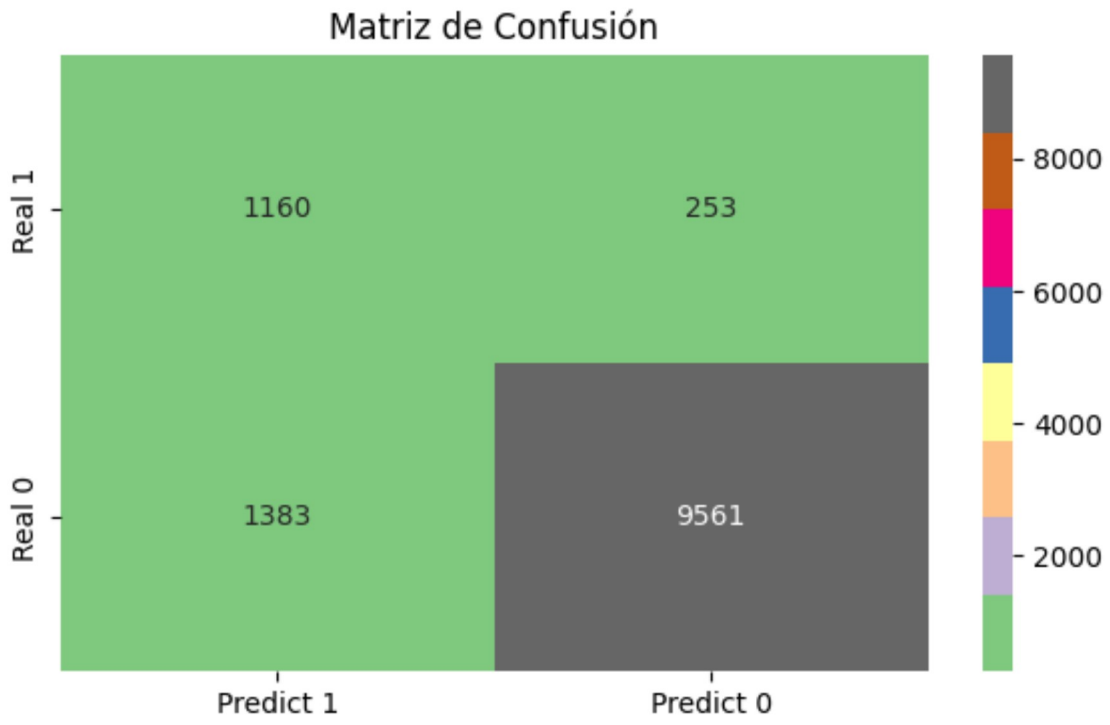
In [ ]:
```
# Hacer df con la variables de respuesta real y predicha
z = x_test.copy()
z['Real Response'] = y_test
z['Predicted Response'] = y_predict
z[['Real Response', 'Predicted Response']].sample(5)
```

Out[ ]:

	Real Response	Predicted Response
3403	0	0
38591	1	1
32289	0	1
13328	0	1
25768	0	0

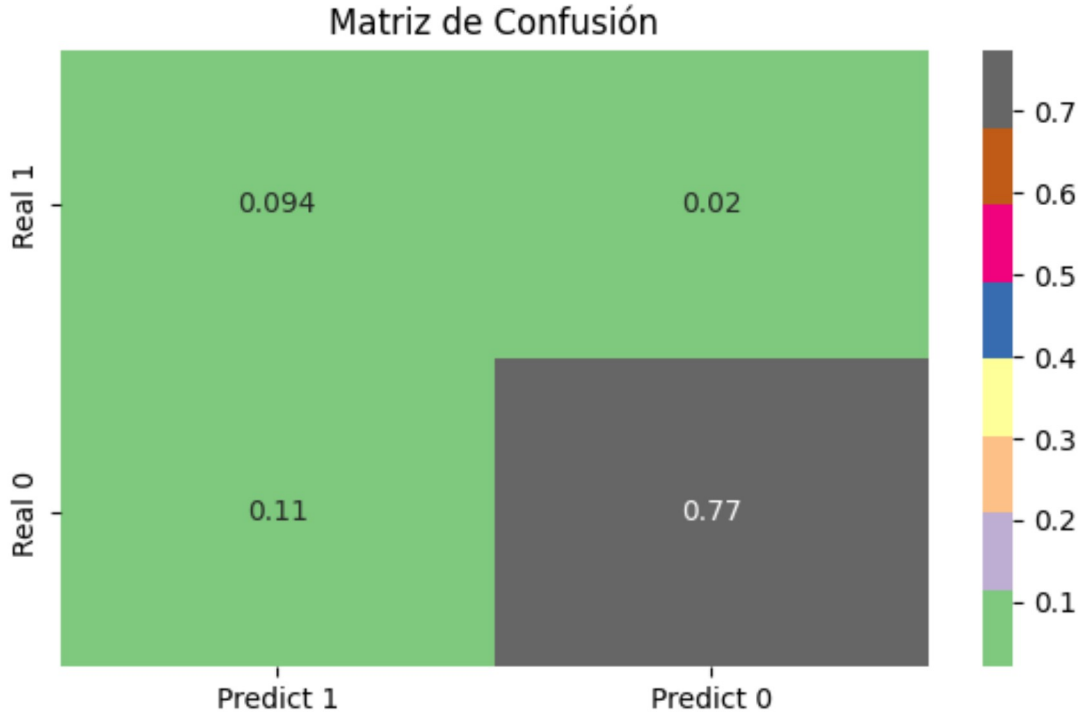
In [ ]:
```
# Matriz de confusión
from sklearn import metrics

cm = metrics.confusion_matrix(y_true=y_test, y_pred=y_predict, labels=[1,0])
df_cm = pd.DataFrame(cm, index=[i for i in ['Real 1', 'Real 0']], columns=['Predict

plt.figure(figsize=(7,4))
sns.heatmap(data=df_cm, annot=True, cmap='Accent', fmt='d').set_title('Matriz de Co
```

Out[ ]:
```
Text(0.5, 1.0, 'Matriz de Confusión')
```



In [ ]:
```
# Matriz de confusión por porcentaje
plt.figure(figsize=(7,4))
sns.heatmap(data=df_cm / df_cm.sum().sum(), annot=True, cmap='Accent', fmt='.2').se
```

Out[ ]: Text(0.5, 1.0, 'Matriz de Confusión')



In [ ]:
```
# Medidas de calidad del modelo
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score

model_score = accuracy_score(y_true=y_test, y_pred=y_predict)
recall_score = recall_score(y_true=y_test, y_pred=y_predict)
precision_score = precision_score(y_true=y_test, y_pred=y_predict)
f1_score = f1_score(y_true=y_test, y_pred=y_predict)

print('Medidas del modelo')
print('----------------------')
print('Accuracy = {0:0.2f}'.format(model_score))
print('Precision = {0:0.2f}'.format(precision_score))
print('Recall = {0:0.2f}'.format(recall_score))
print('F1 Score = {0:0.2f}'.format(f1_score))
print('ROC AUC = {0:0.2f}'.format(roc_auc_score(y_test, y_predict)))
```

```
Medidas del modelo
----------------------
Accuracy = 0.87
Precision = 0.46
Recall = 0.82
F1 Score = 0.59
ROC AUC = 0.85
```

In [ ]:
```
df_check = pd.DataFrame()
df_check['Real'] = y_test
df_check['Predict'] = y_predict
df_check['Different'] = df_check['Real'] != df_check['Predict']
missed_predictions = df_check['Different'].sum()

print(f'Hubo {missed_predictions} registros donde no se predijo correctamente el re
```

```
Hubo 1636 registros donde no se predijo correctamente el resultado (13.24%)
```

Conclusiones

- El modelo es bastante exacto, con un Accuracy de 0.87. Esto podría parecer como algo sumamente bueno, sin embargo, hay que tomar en cuenta que la gran mayoría de los resultados que predijo correctamente son de falsos verdaderos.
    - Esto por un lado puede serle útil a la compañía para predecir con qué personas no gastar recursos para hacer las campañas, pero por el otro, no sirve de nada para detectar las campañas que realmente puede convertir en clientes y en dinero para la empresa
- La precisión del modelo no es tan buena con 0.46. Esto quiere decir que de todos las predicciones positivas, solo el 46% fueron reales,. Esto confirma el punto anterior. Si la empresa quisiera usar este modelo para convertir campañas en clientes, de todos las personas a las que les indique que tienen que hacer campaña, menos de la mitad terminarían conviertiendose en su cliente
- El recall del modelo es de .82. Esto quiere decir que el 82% de los positivos que venían en el dataset de testing fueron predichos correctamente como positivos
- ROC AUC es bueno con .85. Esto quiere decir que en general el modelo es bueno, pero aun asi dependería de la empresa considerar el impacto que pueda tener la baja precisión del modelo
- El modelo puede ser útil para reducir la cantidad de recursos que dedica la empresa a la realización de campañas puesto que el modelo es bueno para predecir campañas de las cuales no se obtendrán clientes para la empresa. Esto subiría la probabilidad de que una campaña termine en la conversión de un cliente a pesar de que el modelo no sea tan preciso dado que se espera que, de los registros que el modelo prediga que serán campañas exitosas, el 46% realmente sean exitosas.