Developed by Thiago Bulgarelli

Contact: bugath36@gmail.com

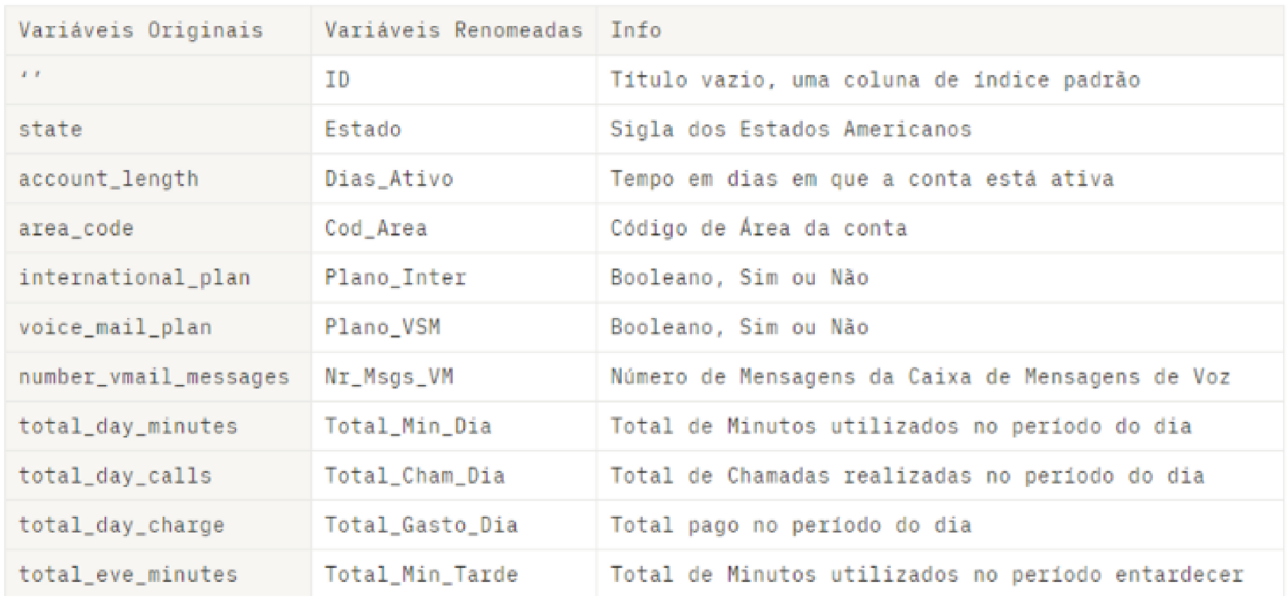
Predicting CHURN in Telecommunications with Machine Learning

Business Problem Definition

Customer Churn refers to a customer's decision to end a business relationship or the loss of customers. It also represents customer attrition. Customer loyalty and churn rate always add up to 100%. If a company has a loyalty rate of 60%, then the churn rate is 40%. According to the 80/20 customer profitability rule, 20% of customers generate 80% of revenue. Therefore, it is crucial to predict users who are likely to churn and identify the factors that affect customer decisions.

In this project, we will predict Customer Churn in a Telecom Operator. Our task is to create a machine learning model that can predict whether a customer is likely to cancel their plan and the probability of it happening. The dataset header provides a description of the information in each column, which will be detailed in our data dictionary.

Data Dictionary



Variáveis Originais	Variáveis Renomeadas	Info
''	ID	Titulo vazio, uma coluna de índice padrão
state	Estado	Sigla dos Estados Americanos
account_length	Dias_Ativo	Tempo em dias em que a conta está ativa
area_code	Cod_Area	Código de Área da conta
international_plan	Plano_Inter	Booleano, Sim ou Não
voice_mail_plan	Plano_VSM	Booleano, Sim ou Não
number_vmail_messages	Nr_Msgs_VM	Número de Mensagens da Caixa de Mensagens de Voz
total_day_minutes	Total_Min_Dia	Total de Minutos utilizados no período do dia
total_day_calls	Total_Cham_Dia	Total de Chamadas realizadas no período do dia
total_day_charge	Total_Gasto_Dia	Total pago no período do dia
total_eve_minutes	Total_Min_Tarde	Total de Minutos utilizados no período entardecer

Developed by Thiago Bulgarelli

Contact: bugath36@gmail.com

total_eve_calls	Total_Cham_Tarde	Total de Chamadas realizadas no período entardecer
total_eve_charge	Total_Gasto_Tarde	Total pago no período entardecer
total_night_minutes	Total_Min_Noite	Total de Minutos utilizados no período noturno
total_night_calls	Total_Cham_Noite	Total de Chamadas realizadas no período noturno
total_night_charge	Total_Gasto_Noite	Total pago no período noturno
total_intl_minutes	Total_Min_Inter	Total de Minutos em ligações Internacionais
total_intl_calls	Total_Cham_Inter	Total de Chamadas em ligações Internacionais
total_intl_charge	Total_Gasto_Inter	Total pago em ligações Internacionais
number_customer_service_calls	Total_Cham_Atend	Número de ligações para o Serviço de Atendimento ao Consumidor
churn	CHURN	Boleano, Sim ou Não. Trocou ou não Operadora

Package and Versions

```
# Registrando Versão Python
from platform import python_version
print('A Versão da Linguagem Python utilizada neste projeto é ', python_version())
```

```
A Versão da Linguagem Python utilizada neste projeto é  3.9.16
```

```
# Importando os Pacotes
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import scipy as sc
import sklearn as sk
import imblearn as imb
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import SGDClassifier
from sklearn.linear_model import RidgeClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler, LabelEncoder
from imblearn.over_sampling import SMOTE
```

Developed by Thiago Bulgarelli

Contact: bugath36@gmail.com

```
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import roc_auc_score
from scipy.stats import shapiro

# Versão dos Pamcotes
%reload_ext watermark
%watermark -a "Thiago Bulgarelli" --iversions
```

```
Author: Thiago Bulgarelli

imblearn  : 0.0
seaborn   : 0.12.2
numpy     : 1.23.5
matplotlib: 3.6.2
pandas    : 1.5.2
scipy     : 1.9.3
sklearn   : 1.0.2
```

Loading the Data

To load the data, we'll start by combining the training and testing datasets into a single dataset. Then, we'll check the variable types identified by the Python interpreter, look for missing data, and analyze the distribution of the response variable, as we're dealing with a binary classification problem.

```
# Carregando os Dados
df = pd.concat([pd.read_csv('Dados/projeto4_telecom_treino.csv', sep=','),
                pd.read_csv('Dados/projeto4_telecom_teste.csv', sep=',')], axis=0)
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 5000 entries, 0 to 1666
Data columns (total 21 columns):
 #   Column                         Non-Null Count  Dtype
---  ------                         --------------  -----
 0   Unnamed: 0                     5000 non-null   int64
 1   state                          5000 non-null   object
 2   account_length                 5000 non-null   int64
 3   area_code                      5000 non-null   object
 4   international_plan             5000 non-null   object
 5   voice_mail_plan                5000 non-null   object
 6   number_vmail_messages          5000 non-null   int64
 7   total_day_minutes              5000 non-null   float64
 8   total_day_calls                5000 non-null   int64
 9   total_day_charge               5000 non-null   float64
 10  total_eve_minutes              5000 non-null   float64
 11  total_eve_calls                5000 non-null   int64
 12  total_eve_charge               5000 non-null   float64
 13  total_night_minutes            5000 non-null   float64
 14  total_night_calls              5000 non-null   int64
 15  total_night_charge             5000 non-null   float64
 16  total_intl_minutes             5000 non-null   float64
 17  total_intl_calls               5000 non-null   int64
 18  total_intl_charge              5000 non-null   float64
 19  number_customer_service_calls  5000 non-null   int64
 20  churn                          5000 non-null   object
dtypes: float64(8), int64(8), object(5)
memory usage: 859.4+ KB
```

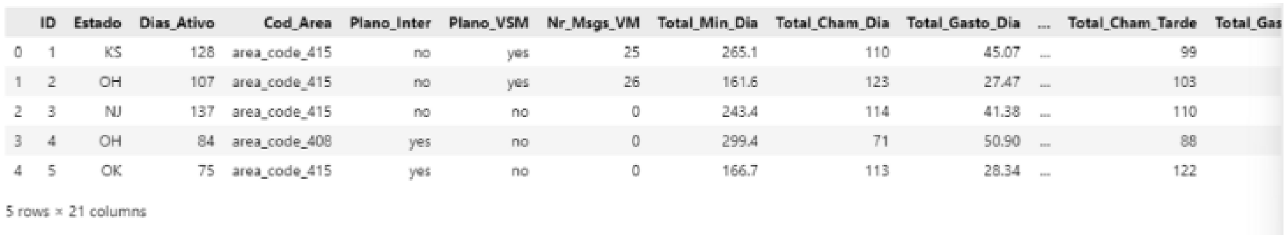
Our interpreter has done a good job identifying the numerical and categorical variables, so we don't need to make any changes at this point. Let's proceed with the organization.

```
# Criando um Dicionário DePara
DePara = {'Unnamed: 0' : 'ID',
          'state' : 'Estado',
          'account_length' : 'Dias_Ativo',
          'area_code' : 'Cod_Area',
          'international_plan' : 'Plano_Inter',
          'voice_mail_plan' : 'Plano_VSM',
          'number_vmail_messages':'Nr_Msgs_VM',
          'total_day_minutes' : 'Total_Min_Dia',
          'total_day_calls' : 'Total_Cham_Dia',
          'total_day_charge' : 'Total_Gasto_Dia',
          'total_eve_minutes' : 'Total_Min_Tarde',
          'total_eve_calls' : 'Total_Cham_Tarde',
          'total_eve_charge' : 'Total_Gasto_Tarde',
          'total_night_minutes' : 'Total_Min_Noite',
          'total_night_calls' : 'Total_Cham_Noite',
          'total_night_charge' : 'Total_Gasto_Noite',
          'total_intl_minutes' : 'Total_Min_Inter',
          'total_intl_calls' : 'Total_Cham_Inter',
          'total_intl_charge' : 'Total_Gasto_Inter',
          'number_customer_service_calls' : 'Total_Cham_Atend',
          'churn' : 'CHURN'}
```

Developed by Thiago Bulgarelli

Contact: bugath36@gmail.com

```
# Renomeando as Colunas
df = df.rename(columns=DePara)
df.head()
```



	ID	Estado	Dias_Ativo	Cod_Area	Plano_Inter	Plano_VSM	Nr_Msgs_VM	Total_Min_Dia	Total_Cham_Dia	Total_Gasto_Dia	...	Total_Cham_Tarde	Total_Gas
0	1	KS	128	area_code_415	no	yes	25	265.1	110	45.07	...	99	
1	2	OH	107	area_code_415	no	yes	26	161.6	123	27.47	...	103	
2	3	NJ	137	area_code_415	no	no	0	243.4	114	41.38	...	110	
3	4	OH	84	area_code_408	yes	no	0	299.4	71	50.90	...	88	
4	5	OK	75	area_code_415	yes	no	0	166.7	113	28.34	...	122	

5 rows × 21 columns

```
# Verificando Ausência de Informação
df.isnull().sum() # Não temos dados faltantes
```

```
ID                  0
Estado              0
Dias_Ativo          0
Cod_Area            0
Plano_Inter         0
Plano_VSM           0
Nr_Msgs_VM          0
Total_Min_Dia       0
Total_Cham_Dia      0
Total_Gasto_Dia     0
Total_Min_Tarde     0
Total_Cham_Tarde    0
Total_Gasto_Tarde   0
Total_Min_Noite     0
Total_Cham_Noite    0
Total_Gasto_Noite   0
Total_Min_Inter     0
Total_Cham_Inter    0
Total_Gasto_Inter   0
Total_Cham_Atend    0
CHURN               0
dtype: int64
```

```
# Verificando a Distribuição da Variável Resposta CHURN
df.CHURN.value_counts()
```

```
no     4293
yes     707
Name: CHURN, dtype: int64
```

We can see that we have an issue related to the distribution of data in the target variable. We will need to address this issue during the preprocessing stage to avoid having poor performance of the predictive model.

Exploratory Data Analysis (EDA)

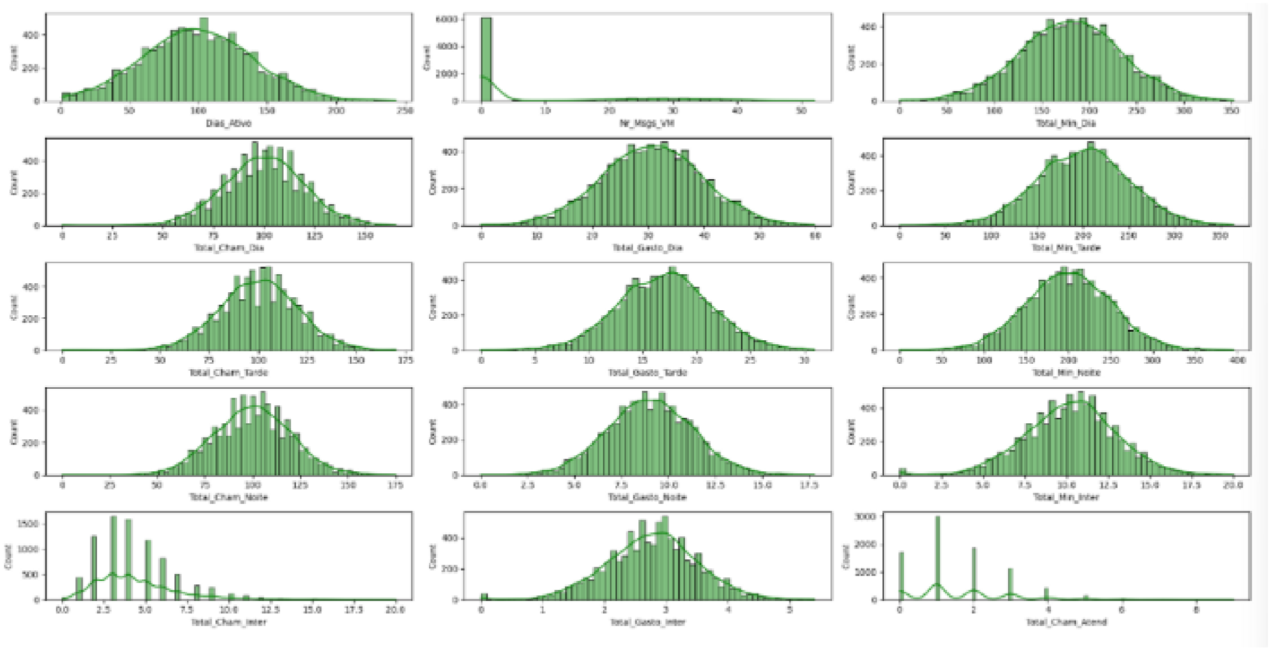
Let's start by studying the variables and identify their distributions, key statistical measures, their distributions, and possible correlations between each other and the target variable.

```
# Dividindo as variáveis em Numéricas e Categóricas
Cat = df.select_dtypes(include='object').columns
Num = df.select_dtypes(exclude='object').columns
```

Numeric Variables

Let's start by taking a big picture view of the variables and then study individually those that catch our attention due to their characteristics. The objective here is to better understand each variable and extract possible interesting insights for our business.

```
# Analisando as Distribuições das Variáveis
features = VarNum.columns[:-1]
plt.figure(figsize = (20, 10))
for i in range(0, len(features)):
    plt.subplot(5, len(features)//5, i+1)
    sns.histplot(x = VarNum[features[i]], kde = True, color = 'green')
    plt.xlabel(features[i])
    plt.tight_layout()
```



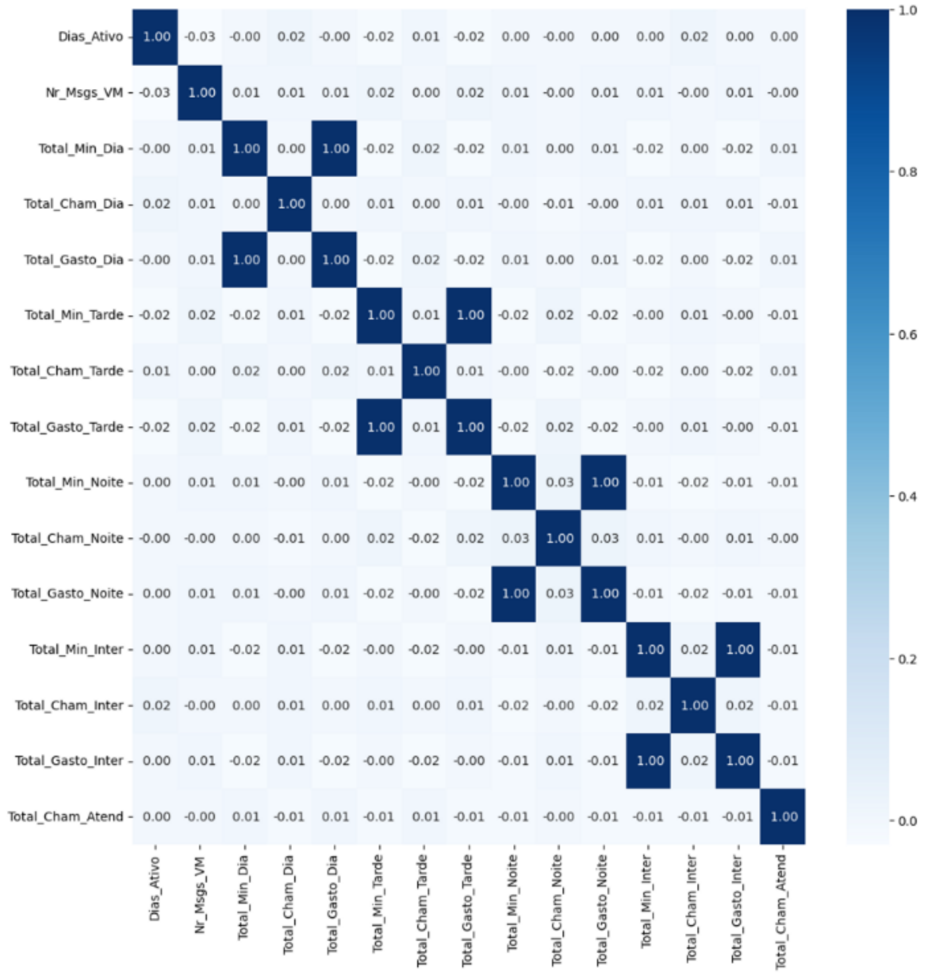
We can observe that the distributions approximate a normal distribution in all variables. We will further investigate this to make statistical assertions.

Insight -> We noticed that the number of Voice Mail Messages is zero for the majority of users, indicating that this service is rarely used. This is likely because people nowadays prefer to send messages through applications rather than traditional telephony services.

Insight -> The number of calls to Customer Service also stands out, with many customers making at least 1 or 2 calls. This could be correlated with customer dissatisfaction.

We will explore these points in more detail when we study the variables individually.

```python
# Analisando a Corelação entre as Variáveis
CorrVarNum = round(VarNum.iloc[:, :-1].corr(), 2)
plt.figure(figsize=(12,12))
sns.heatmap(CorrVarNum, cmap='Blues', annot=True, fmt = ' .2f');
```



Developed by Thiago Bulgarelli

Contact: bugath36@gmail.com

We can see from the heat map that there are variables with multicollinearity (Total_Gasto x Total_Min), meaning they are highly correlated with each other. This occurs because these variables contain similar information and can either be discarded or combined into a single variable. We will address these variables later during the data preprocessing stage. The goal is to retain the information while avoiding duplication, which could bias the learning of our predictive model.

Variable Dias_Ativo

```
# Medidas de Tendência Central para CHURN negativo
VarNum[(VarNum['CHURN']=='no')]['Dias_Ativo'].describe()
```

```
count    7192.00000
mean       99.66574
std        39.87100
min         1.00000
25%        73.00000
50%        99.00000
75%       127.00000
max       243.00000
Name: Dias_Ativo, dtype: float64
```

```
# Calculando a Mediana da Variável em Função do CHURN negativo
VarNum[(VarNum['CHURN']=='no')]['Dias_Ativo'].median()
```

```
99.0
```

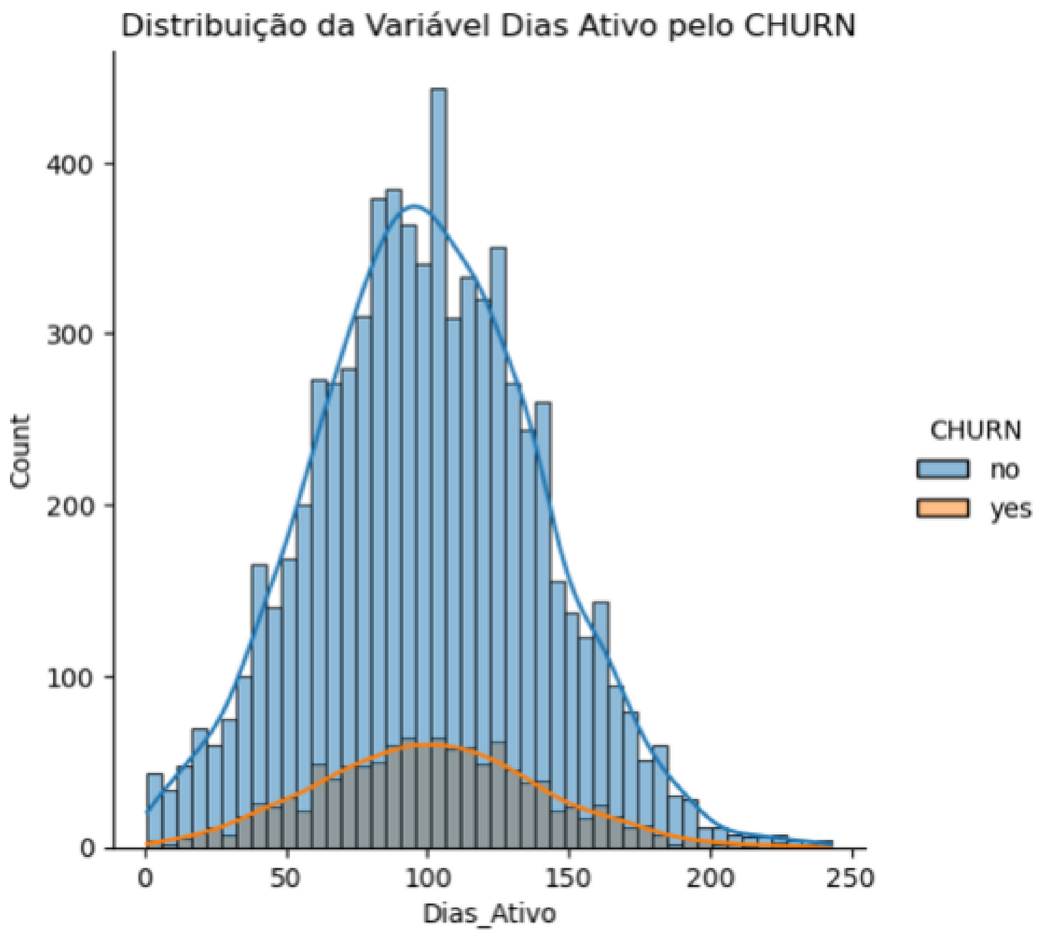
```
# Medidas de Tendência Central para CHURN positivo
VarNum[(VarNum['CHURN']=='yes')]['Dias_Ativo'].describe()
```

```
count    1142.000000
mean      101.101576
std        39.020458
min         1.000000
25%        74.000000
50%       100.000000
75%       127.000000
max       225.000000
Name: Dias_Ativo, dtype: float64
```

```
# Calculando a Mediana da Variável em Função do CHURN positivo
VarNum[(VarNum['CHURN']=='yes')]['Dias_Ativo'].median()
```

```
100.0
```

```
# Comparando as Distribuições Graficamente com base no CHURN
sns.displot(data=VarNum, x = 'Dias_Ativo', kde = True, hue='CHURN' )
plt.title('Distribuição da Variável Dias Ativo pelo CHURN');
```



Insight: We have a similar distribution for both customers who remain with the phone plan (negative CHURN) and those who churned (positive CHURN). On average, customers stay with the operator for 100 days, with a maximum of approximately 240 days and a standard deviation of 39 days. It appears that we have a normal distribution for both CHURN situations, as the median is very close to the mean. However, we will verify this information by conducting a non-parametric Shapiro-Wilk test.

Being:

H0 -> The data follows a normal distribution

H1 -> The data does not follow a normal distribution

```
# Shapiro-Wilk Test para CHURN Positivo
shapiro(VarNum[(VarNum['CHURN']=='yes')]['Dias_Ativo'])
```

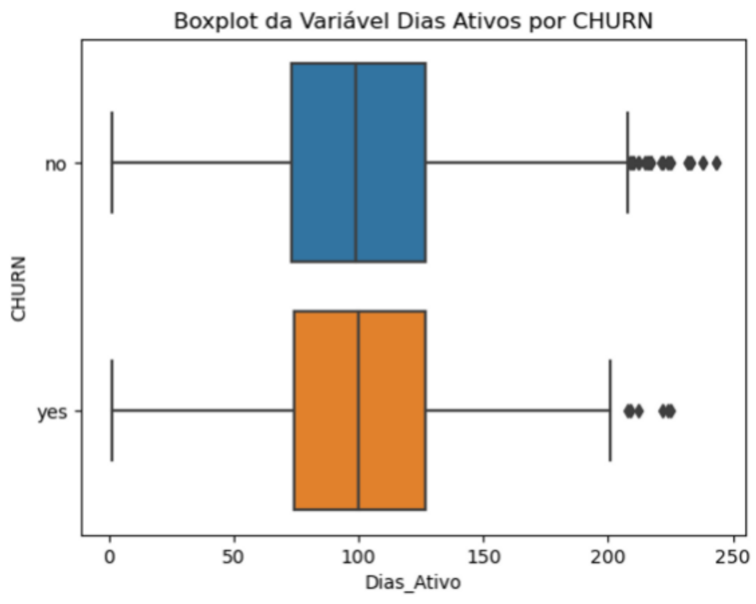
```
ShapiroResult(statistic=0.9974399209022522, pvalue=0.06757985800504684)
```

```
# Shapiro-Wilk Test para CHURN Negativo
shapiro(VarNum[(VarNum['CHURN']=='no')]['Dias_Ativo'].iloc[:4500])
```

```
ShapiroResult(statistic=0.9975619912147522, pvalue=1.3558333193941507e-06)
```

For the positive CHURN ('yes') situation, we can conclude that the data follows a normal distribution (we fail to reject H0, p-value = 0.067). However, for the negative CHURN situation, we reject H0 (p-value close to 0), indicating that it does not follow a normal distribution.

```
# Box plot por Categoria do CHURN
sns.boxplot(data=VarNum, x= 'Dias_Ativo', y='CHURN', orient='h')
plt.title('Boxplot da Variável Dias Ativos por CHURN');
```



We conclude with the boxplot, which confirms that the median is close to the mean in both cases. However, this is not sufficient to prove the normal distribution according to the hypothesis test.

Variable Nr_Msgs_VM

```
# Medidas de Tendência Central para CHURN negativo
VarNum[(VarNum['CHURN']=='no')]['Nr_Msgs_VM'].describe()
```

```
count    7192.000000
mean        8.096357
std        13.729524
min         0.000000
25%         0.000000
50%         0.000000
75%        19.000000
max        52.000000
Name: Nr_Msgs_VM, dtype: float64
```

```
# Calculando a Mediana da Variável em Função do CHURN negativo
VarNum[(VarNum['CHURN']=='no')]['Nr_Msgs_VM'].median()
```

```
0.0
```

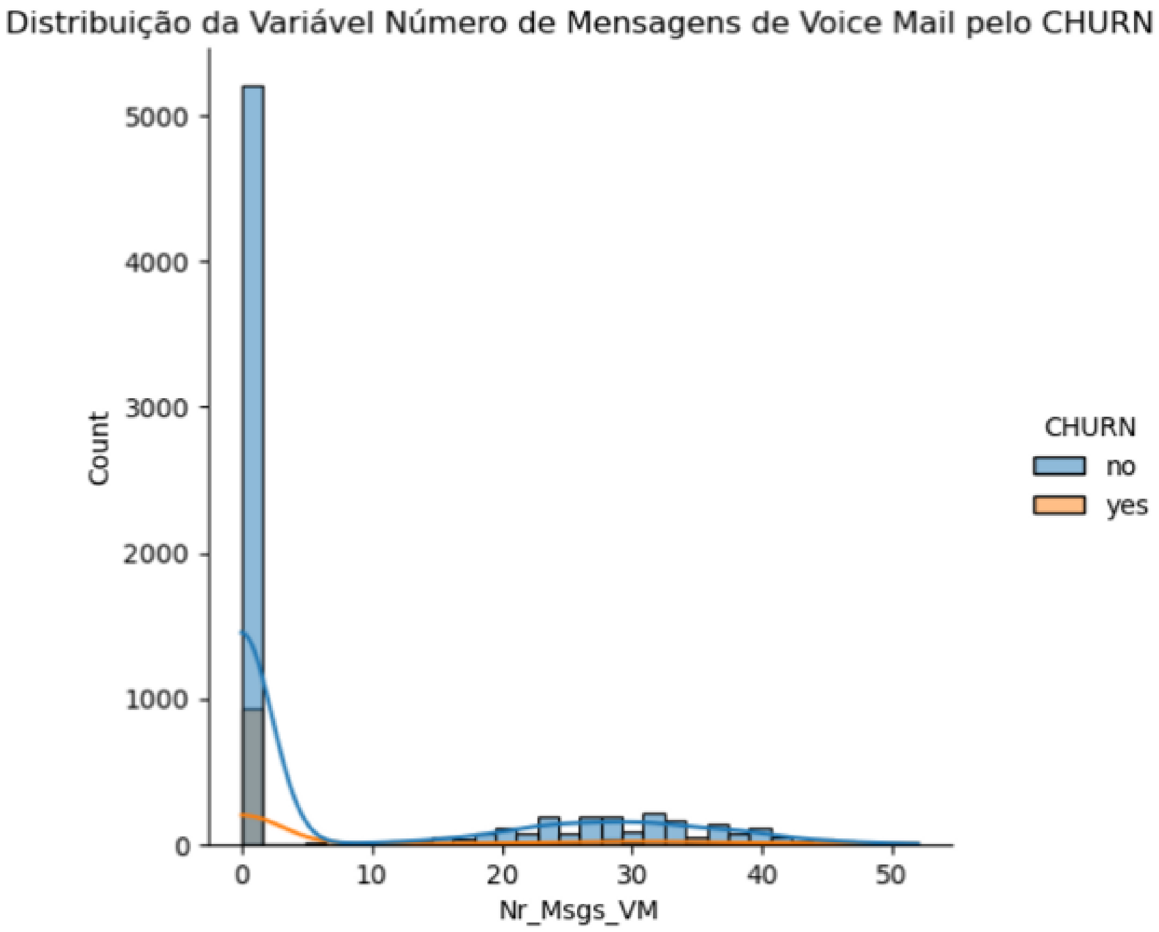
```
# Medidas de Tendência Central para CHURN positivo
VarNum[(VarNum['CHURN']=='yes')]['Nr_Msgs_VM'].describe()
```

```
count    1142.000000
mean        5.705779
std        12.380441
min         0.000000
25%         0.000000
50%         0.000000
75%         0.000000
max        49.000000
Name: Nr_Msgs_VM, dtype: float64
```

```
# Calculando a Mediana da Variável em Função do CHURN positivo
VarNum[(VarNum['CHURN']=='yes')]['Nr_Msgs_VM'].median()
```

```
0.0
```

```
# Comparando as Distribuições Graficamente com base no CHURN
sns.displot(data=VarNum, x = 'Nr_Msgs_VM', kde = True, hue='CHURN' )
plt.title('Distribuição da Variável Número de Mensagens de Voice Mail pelo CHURN');
```



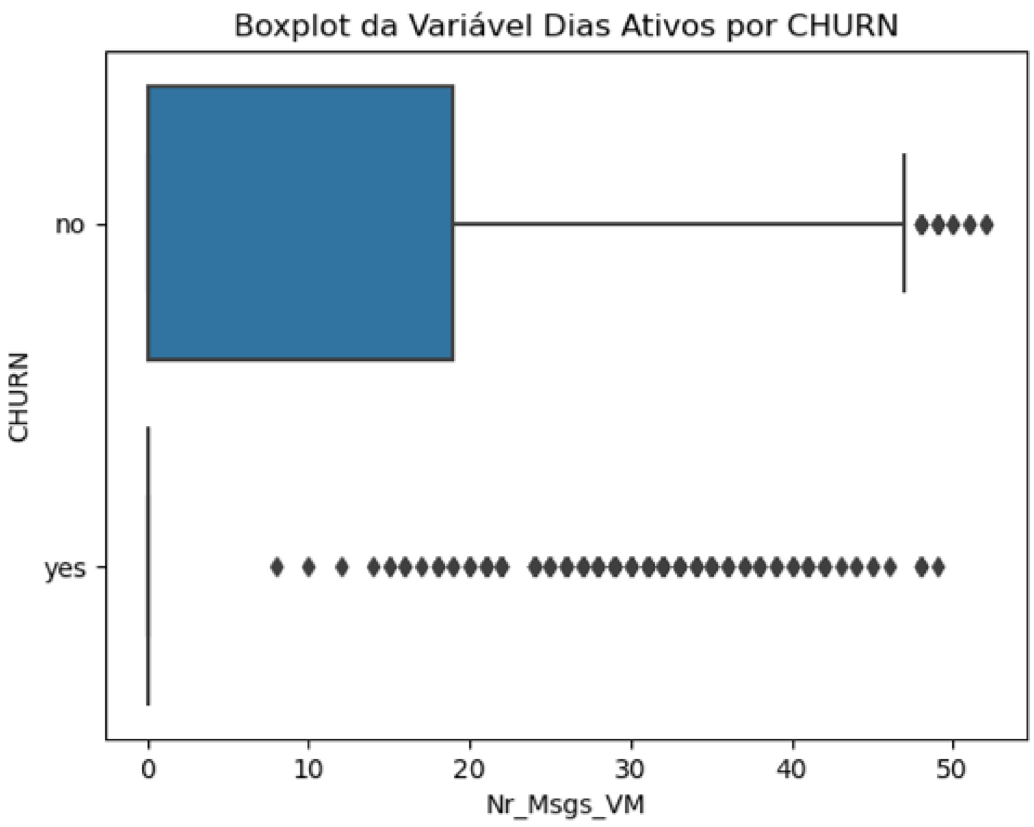
Insight -> It is important to note that in the majority of customers, this service is not utilized. In the case of customers who churned (CHURN = 'yes'), we see that almost 90% of them do not use the service.

Perhaps we can consider reducing the cost of this service and instead offer another service that is more useful to customers.

Developed by Thiago Bulgarelli

Contact: bugath36@gmail.com

```
# Box plot por Categoria do CHURN
sns.boxplot(data=VarNum, x= 'Nr_Msgs_VM', y='CHURN', orient='h')
plt.title('Boxplot da Variável Dias Ativos por CHURN');
```



Clearly, we can observe that the median for both categories of CHURN is 0.0, except for a few customers who still use this service.

Variable Total_Dia

Let's analyze the variables related to the time of day, as we have already identified that they are correlated. Let's understand how they behave.

```
# Medidas de Tendência Central para CHURN negativo
VarNum[(VarNum['CHURN']=='no')].iloc[:, 2:5].describe()
```

	Total_Min_Dia	Total_Cham_Dia	Total_Gasto_Dia
count	7192.000000	7192.000000	7192.000000
mean	177.971732	99.849277	30.255755
std	51.304135	19.687266	8.721635
min	0.000000	0.000000	0.000000
25%	143.700000	87.000000	24.430000
50%	179.200000	100.000000	30.460000
75%	213.400000	113.000000	36.280000
max	350.800000	165.000000	59.640000

```
# Calculando a Mediana da Variável em Função do CHURN negativo
VarNum[(VarNum['CHURN']=='no')].iloc[:, 2:5].median()
```

```
Total_Min_Dia      179.20
Total_Cham_Dia     100.00
Total_Gasto_Dia     30.46
dtype: float64
```

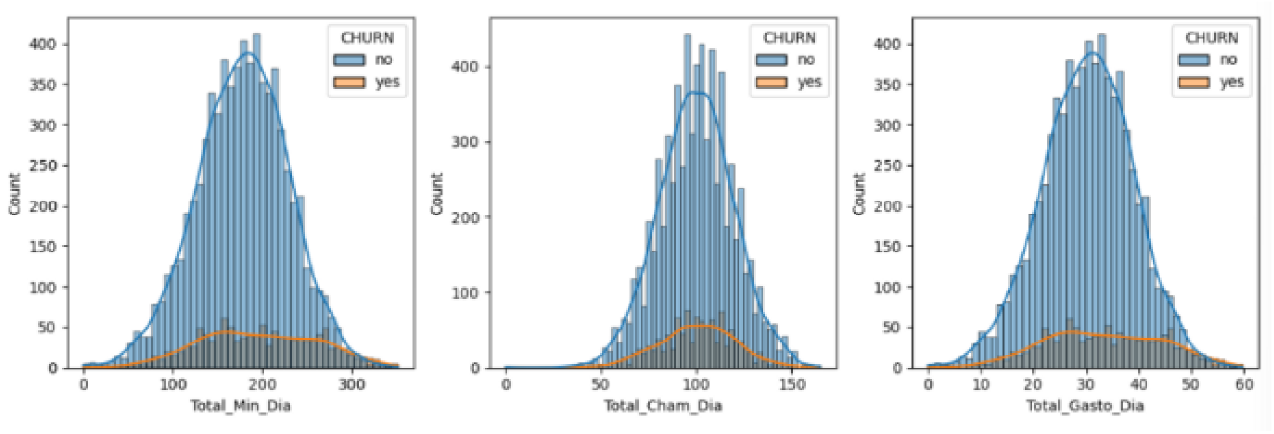
```
# Medidas de Tendência Central para CHURN positivo
VarNum[(VarNum['CHURN']=='yes')].iloc[:, 2:5].describe()
```

	Total_Min_Dia	Total_Cham_Dia	Total_Gasto_Dia
count	1142.000000	1142.000000	1142.000000
mean	196.210858	100.337128	33.356349
std	64.883646	20.369875	11.030359
min	0.000000	0.000000	0.000000
25%	148.500000	88.000000	25.250000
50%	195.350000	101.000000	33.210000
75%	247.650000	114.000000	42.102500
max	351.500000	165.000000	59.760000

```
# Calculando a Mediana da Variável em Função do CHURN positivo
VarNum[(VarNum['CHURN']=='yes')].iloc[:, 2:5].median()
```

```
Total_Min_Dia      195.35
Total_Cham_Dia     101.00
Total_Gasto_Dia     33.21
dtype: float64
```

```
# Comparando as Distribuições Graficamente com base no CHURN
features = VarNum.columns[2:5]
plt.figure(figsize = (12, 4))
for i in range(0, 3):
    plt.subplot(1, 3, i+1)
    sns.histplot(x = VarNum[features[i]], kde = True, hue=VarNum['CHURN'])
    plt.xlabel(features[i])
    plt.tight_layout()
```



We can observe that the 3 variables have similar behaviors, including when we look at measures of central tendency. When we analyze only the observations with positive CHURN, we can see a negative Kurtosis, indicating a flatter curve with the same amplitude as the observations with negative CHURN.

Let's verify this by calculating the coefficients of Kurtosis and Skewness.

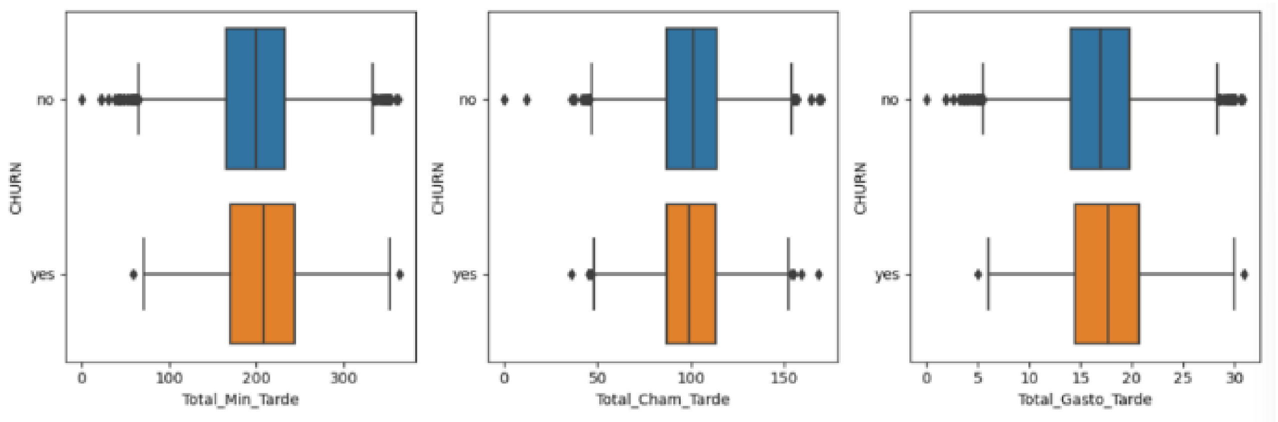
```
# Calculando os Coeficiente de Kurtosis e Skenewss das Variáveis Total_Dia com CHURN Negativo
dic = {}
dic['Kurtosis para CHURN Negativo'] = VarNum[(VarNum['CHURN']=='no')].iloc[:, 2:5].kurtosis()
dic['Kurtosis para CHURN Positivo'] = VarNum[(VarNum['CHURN']=='yes')].iloc[:, 2:5].kurtosis()
dic['Skewness para CHURN Negativo'] = VarNum[(VarNum['CHURN']=='no')].iloc[:, 2:5].skew()
dic['Skewness para CHURN Positivo'] = VarNum[(VarNum['CHURN']=='yes')].iloc[:, 2:5].skew()
Res = pd.DataFrame(data=dic)
Res
```

	Kurtosis para CHURN Negativo	Kurtosis para CHURN Positivo	Skewness para CHURN Negativo	Skewness para CHURN Positivo
Total_Min_Tarde	0.117819	-0.196702	-0.027925	0.016287
Total_Cham_Tarde	0.100113	0.017120	-0.036369	0.002269
Total_Gasto_Tarde	0.117755	-0.196843	-0.027877	0.016399

Analyzing the Skewness, we have distributions very close to zero, indicating symmetry. As for the Kurtosis, we observe similar characteristics to the Periodo_do_Dia variable, with distributions close to normal for negative CHURN and slightly flatter for positive CHURN.

Insight -> Finally, similarly to the Total_Day variable, what stands out in the behavior of these variables is that the means and medians are close for both negative and positive CHURN, suggesting that they may have little influence on the customer's decision to stay or leave the company's services.

```
# Boxplot para Confirmar as proximidades da Média e Mediana
features = VarNum.columns[5:8]
plt.figure(figsize = (12, 4))
for i in range(0, 3):
    plt.subplot(1, 3, i+1)
    sns.boxplot(x = VarNum[features[i]], y=VarNum['CHURN'], orient='h')
    plt.xlabel(features[i])
    plt.tight_layout()
```



Variable Total_Noite

Let's proceed in the same way as we did with the Day and Evening period variables.

```
# Medidas de Tendência Central para CHURN negativo
VarNum[(VarNum['CHURN']=='no')].iloc[:, 8:11].describe()
```

	Total_Min_Noite	Total_Cham_Noite	Total_Gasto_Noite
count	7192.000000	7192.000000	7192.000000
mean	199.645634	99.927976	8.984173
std	50.685747	19.943435	2.280855
min	0.000000	0.000000	0.000000
25%	166.200000	87.000000	7.480000
50%	199.500000	100.000000	8.980000
75%	233.850000	113.000000	10.522500
max	395.000000	175.000000	17.770000

```
# Calculando a Mediana da Variável em Função do CHURN negativo
VarNum[(VarNum['CHURN']=='no')].iloc[:, 8:11].median()
```

```
Total_Min_Noite      199.50
Total_Cham_Noite     100.00
Total_Gasto_Noite      8.98
dtype: float64
```

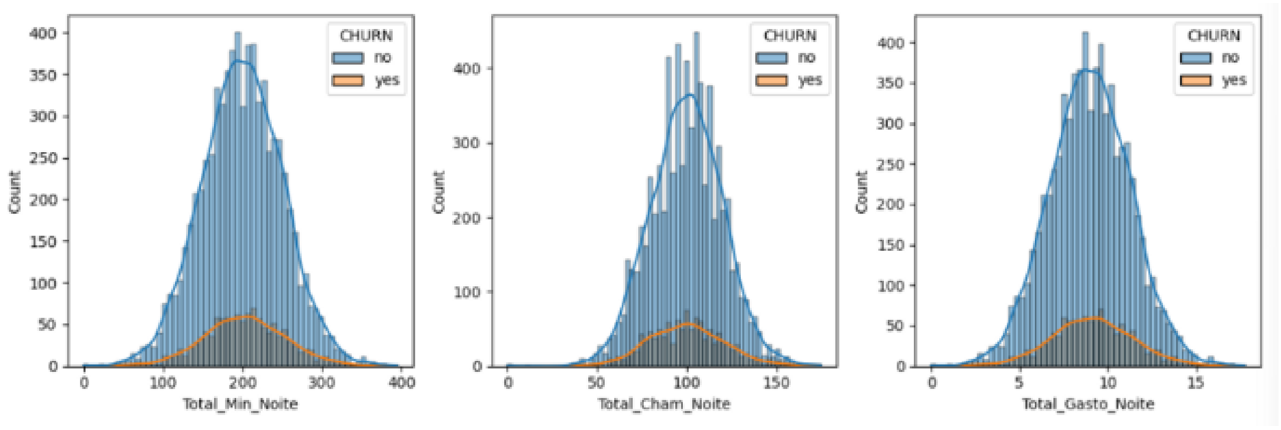
```
# Medidas de Tendência Central para CHURN positivo
VarNum[(VarNum['CHURN']=='yes')].iloc[:, 8:11].describe()
```

	Total_Min_Noite	Total_Cham_Noite	Total_Gasto_Noite
count	1142.000000	1142.000000	1142.000000
mean	203.162522	99.724168	9.142373
std	50.517638	20.379272	2.273330
min	47.400000	12.000000	2.130000
25%	169.800000	85.250000	7.640000
50%	203.450000	100.000000	9.155000
75%	238.225000	113.000000	10.722500
max	381.900000	160.000000	17.190000

```
# Calculando a Mediana da Variável em Função do CHURN positivo
VarNum[(VarNum['CHURN']=='yes')].iloc[:, 8:11].median()
```

```
Total_Min_Noite      203.450
Total_Cham_Noite     100.000
Total_Gasto_Noite      9.155
dtype: float64
```

```
# Comparando as Distribuições Graficamente com base no CHURN
features = VarNum.columns[8:11]
plt.figure(figsize = (12, 4))
for i in range(0, 3):
    plt.subplot(1, 3, i+1)
    sns.histplot(x = VarNum[features[i]], kde = True, hue=VarNum['CHURN'])
    plt.xlabel(features[i])
    plt.tight_layout()
```



Similar to the variables for the Day and Evening periods, we can observe that the 3 variables exhibit similar behaviors, including measures of central tendency. When we analyze only the observations for positive CHURN, we can see a negative Kurtosis, indicating a flatter curve with the same amplitude as the observations for negative CHURN. Let's verify this by calculating the Kurtosis and Skewness coefficients.

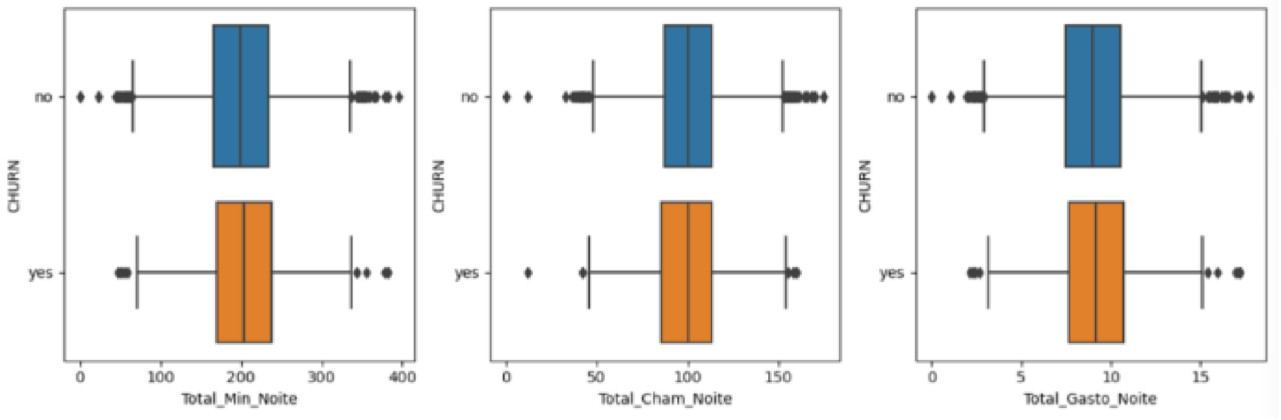
```
# Calculando os Coeficiente de Kurtosis e Skenewss das Variáveis Total_Dia com CHURN Negativo
dic = {}
dic['Kurtosis para CHURN Negativo'] = VarNum[(VarNum['CHURN']=='no')].iloc[:, 5:8].kurtosis()
dic['Kurtosis para CHURN Positivo'] = VarNum[(VarNum['CHURN']=='yes')].iloc[:, 5:8].kurtosis()
dic['Skewness para CHURN Negativo'] = VarNum[(VarNum['CHURN']=='no')].iloc[:, 5:8].skew()
dic['Skewness para CHURN Positivo'] = VarNum[(VarNum['CHURN']=='yes')].iloc[:, 5:8].skew()
Res = pd.DataFrame(data=dic)
Res
```

	Kurtosis para CHURN Negativo	Kurtosis para CHURN Positivo	Skewness para CHURN Negativo	Skewness para CHURN Positivo
Total_Min_Noite	0.063903	0.213053	0.002524	0.090927
Total_Cham_Noite	0.196319	0.093716	-0.019809	0.051902
Total_Gasto_Noite	0.063925	0.213872	0.002536	0.090784

Analyzing the Skewness, we can see distributions that are very close to zero, indicating symmetry. As for the Kurtosis, we observe that all distributions are close to normal, both for negative CHURN and positive CHURN.

Insight -> Finally, similar to the Total_Day and Total_Afternoon variables, what stands out in the behavior of these variables is that the means and medians are close for both negative and positive CHURN. This suggests that these variables may have little influence on the customer's decision to stay or leave the company's services.

```
# Boxplot para Confirmar as proximidades da Média e Mediana
features = VarNum.columns[8:11]
plt.figure(figsize = (12, 4))
for i in range(0, 3):
    plt.subplot(1, 3, i+1)
    sns.boxplot(x = VarNum[features[i]], y=VarNum['CHURN'], orient='h')
    plt.xlabel(features[i])
    plt.tight_layout()
```



Variable Total_Inter

Let's apply the same analyses to the Day, Afternoon, and Evening periods and examine how International Calls behave.

```
# Medidas de Tendência Central para CHURN negativo
VarNum[(VarNum['CHURN']=='no')].iloc[:, 11:14].describe()
```

	Total_Min_Inter	Total_Cham_Inter	Total_Gasto_Inter
count	7192.000000	7192.000000	7192.000000
mean	10.222720	4.462736	2.760642
std	2.750893	2.460197	0.742680
min	0.000000	0.000000	0.000000
25%	8.500000	3.000000	2.300000
50%	10.300000	4.000000	2.780000
75%	12.000000	6.000000	3.240000
max	20.000000	19.000000	5.400000

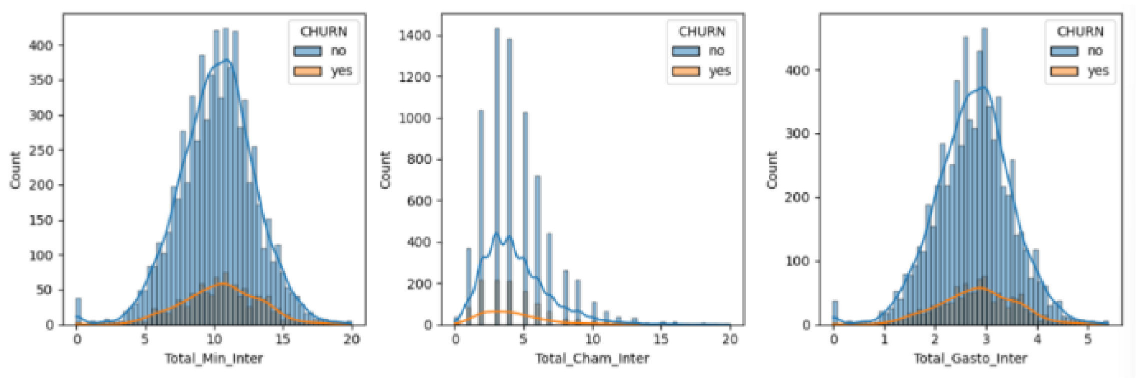
```
# Calculando a Mediana da Variável em Função do CHURN negativo
VarNum[(VarNum['CHURN']=='no')].iloc[:, 11:14].median()
```

	Total_Min_Inter	Total_Cham_Inter	Total_Gasto_Inter
count	1142.000000	1142.000000	1142.000000
mean	10.478109	4.234676	2.829641
std	2.794340	2.501026	0.754428
min	0.000000	0.000000	0.000000
25%	8.700000	2.000000	2.350000
50%	10.550000	4.000000	2.850000
75%	12.400000	5.000000	3.350000
max	20.000000	20.000000	5.400000

```
# Calculando a Mediana da Variável em Função do CHURN positivo
VarNum[(VarNum['CHURN']=='yes')].iloc[:, 11:14].median()
```

```
Total_Min_Inter      10.55
Total_Cham_Inter      4.00
Total_Gasto_Inter     2.85
dtype: float64
```

```python
# Comparando as Distribuições Graficamente com base no CHURN
features = VarNum.columns[11:14]
plt.figure(figsize = (12, 4))
for i in range(0, 3):
    plt.subplot(1, 3, i+1)
    sns.histplot(x = VarNum[features[i]], kde = True, hue=VarNum['CHURN'])
    plt.xlabel(features[i])
    plt.tight_layout()
```



We can observe that the variables Total_Min and Total_Gasto have similar behavior for both Positive and Negative CHURN. As for the variable Total_Cham, we see a discrepancy in the behavior of the graphs, but we will confirm it by calculating the Kurtosis and Skewness coefficients.

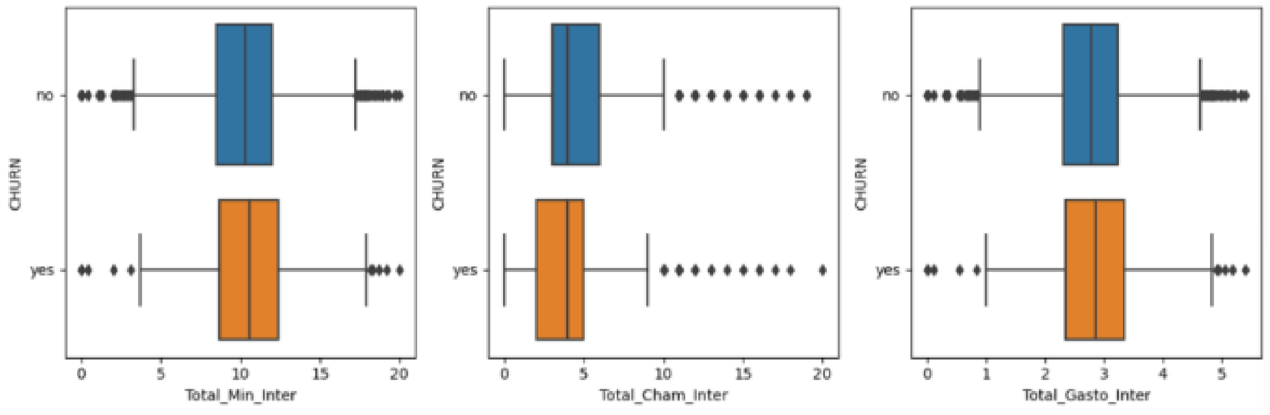
```python
# Calculando os Coeficiente de Kurtosis e Skenewss das Variáveis Total_Dia com CHURN Negativo
dic = {}
dic['Kurtosis para CHURN Negativo'] = VarNum[(VarNum['CHURN']=='no')].iloc[:, 8:11].kurtosis()
dic['Kurtosis para CHURN Positivo'] = VarNum[(VarNum['CHURN']=='yes')].iloc[:, 8:11].kurtosis()
dic['Skewness para CHURN Negativo'] = VarNum[(VarNum['CHURN']=='no')].iloc[:, 8:11].skew()
dic['Skewness para CHURN Positivo'] = VarNum[(VarNum['CHURN']=='yes')].iloc[:, 8:11].skew()
Res = pd.DataFrame(data=dic)
Res
```

	Kurtosis para CHURN Negativo	Kurtosis para CHURN Positivo	Skewness para CHURN Negativo	Skewness para CHURN Positivo
Total_Min_Inter	0.721851	0.345652	-0.201176	-0.123875
Total_Cham_Inter	3.148646	4.410804	1.346080	1.595644
Total_Gasto_Inter	0.722868	0.344596	-0.201484	-0.124144

Despite our suspicions regarding the distribution of the Total_Cham variable, the coefficients confirm that they have similar behaviors in both Kurtosis and Skewness, with Kurtosis being slightly higher.

Insight -> Finally, similarly to the variables related to the time periods, what stands out in the behavior of these variables is that the means and medians are close for both CHURN Negative and CHURN Positive. However, one point of attention is the Total_Cham_Inter variable, as the 25th and 75th quartiles for CHURN Positive are shifted downwards compared to CHURN Negative, indicating that it is a less utilized service for those observations, despite the similar median and mean values.

```
# Boxplot para Confirmar as proximidades da Média e Mediana
features = VarNum.columns[11:14]
plt.figure(figsize = (12, 4))
for i in range(0, 3):
    plt.subplot(1, 3, i+1)
    sns.boxplot(x = VarNum[features[i]], y=VarNum['CHURN'], orient='h')
    plt.xlabel(features[i])
    plt.tight_layout()
```



Variable Total_Cham_Atend

This variable may have a strong relationship with the customer's decision-making process. Let's analyze it further!

```
# Medidas de Tendência Central para CHURN negativo
VarNum[(VarNum['CHURN']=='no')]['Total_Cham_Atend'].describe()
```

```
0    1
Name: Total_Cham_Atend, dtype: int64
```

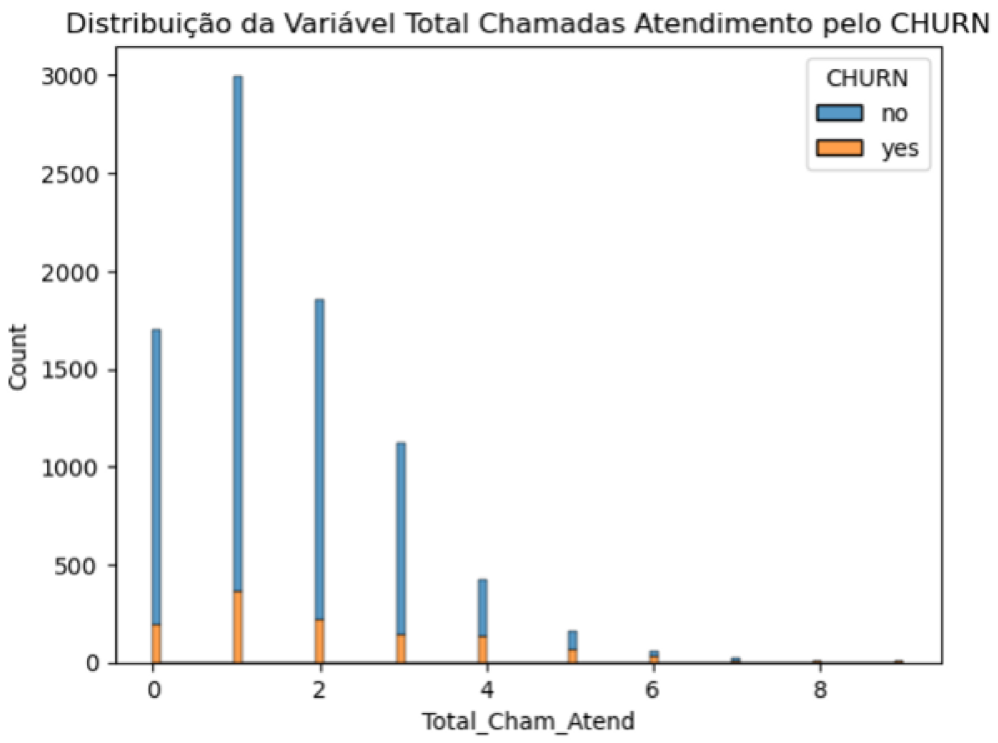
```python
# Medidas de Tendência Central para CHURN Positivo
VarNum[(VarNum['CHURN']=='yes')]['Total_Cham_Atend'].describe()
```

```
count    1142.000000
mean        2.014886
std         1.665957
min         0.000000
25%         1.000000
50%         2.000000
75%         3.000000
max         9.000000
Name: Total_Cham_Atend, dtype: float64
```

```python
# Medidas de Tendência Central para CHURN Positivo
VarNum[(VarNum['CHURN']=='yes')]['Total_Cham_Atend'].mode()
```

```
0    1
Name: Total_Cham_Atend, dtype: int64
```

```python
# Comparando as Distribuições Graficamente com base no CHURN - REFAZER
sns.histplot(data=VarNum, x = 'Total_Cham_Atend', hue='CHURN', fill=True, stat='count', cumulative=False, element='bars',
multiple='stack')
plt.title('Distribuição da Variável Total Chamadas Atendimento pelo CHURN');
```

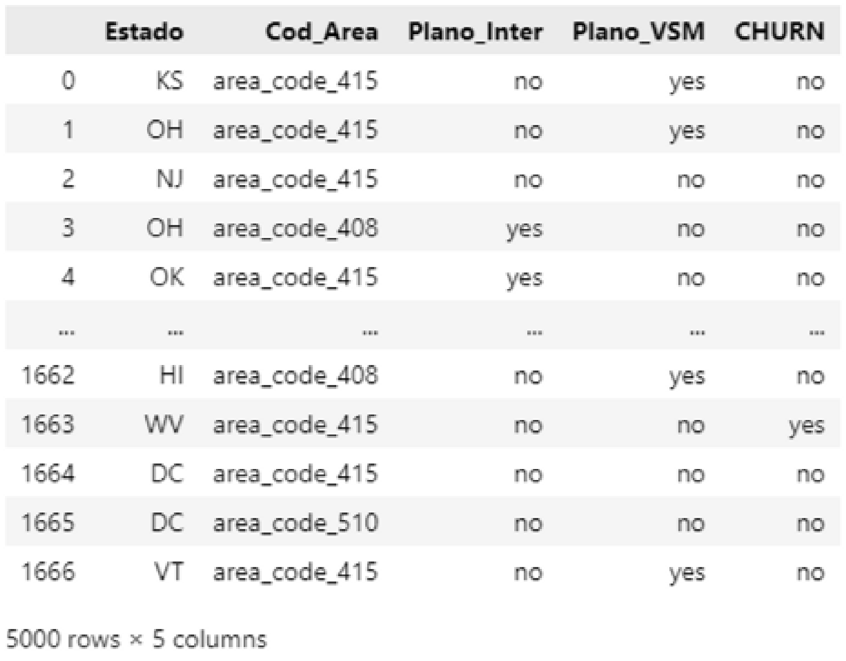


Insight -> Based on the data, we can see that the average number of calls for customers with positive CHURN is 2. For negative CHURN, the average is also around 2, considering that we don't have 1.5 calls. However, the key point here is the Mode, which represents the most frequent value in the variable. It is evident that both positive and negative CHURN cases have a Mode of 1 call. This indicates that this variable alone does not determine a change in the telephony plan. It is quite surprising since it was a business doubt: the more people call customer support (SAC), the more dissatisfaction and, consequently, the higher the positive CHURN.

We have finished analyzing the numerical variables. Now, let's move on to analyzing the categorical variables.

Categorical Variables

```
# Dataset somente com as Variáveis Numéricas + Variável Resposta
VarCat = df[Cat]
VarCat
```



	Estado	Cod_Area	Plano_Inter	Plano_VSM	CHURN
0	KS	area_code_415	no	yes	no
1	OH	area_code_415	no	yes	no
2	NJ	area_code_415	no	no	no
3	OH	area_code_408	yes	no	no
4	OK	area_code_415	yes	no	no
...	...	...	...	...	...
1662	HI	area_code_408	no	yes	no
1663	WV	area_code_415	no	no	yes
1664	DC	area_code_415	no	no	no
1665	DC	area_code_510	no	no	no
1666	VT	area_code_415	no	yes	no

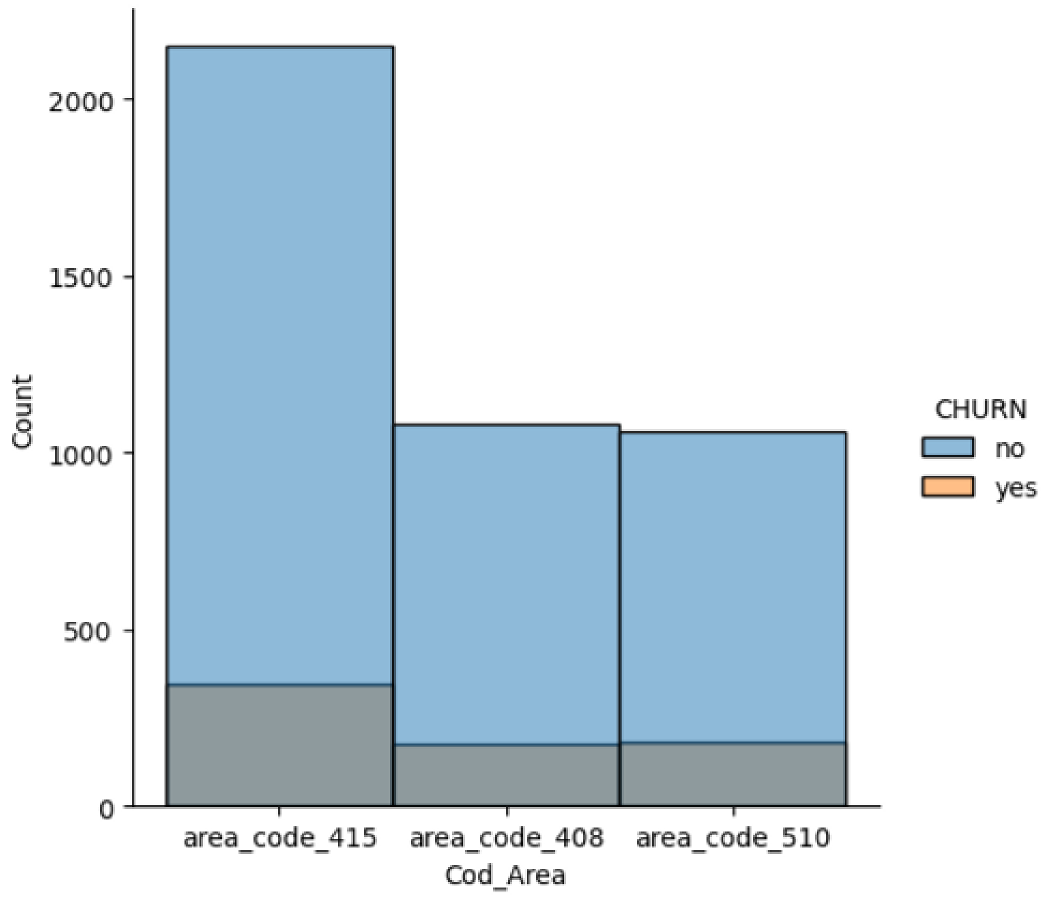
5000 rows × 5 columns

Variable Cod_Area

```
# Quantidade de Observações por Código de Área
VarCat.groupby(['Cod_Area', 'CHURN']).count()
```

		Estado	Plano_Inter	Plano_VSM
Cod_Area	**CHURN**			
area_code_408	no	1082	1082	1082
	yes	177	177	177
area_code_415	no	2149	2149	2149
	yes	346	346	346
area_code_510	no	1062	1062	1062
	yes	184	184	184

```
# Gráfico de Barras com a Contagem das Observações por Cod_Area e Categoria de CHURN
sns.displot(data=VarCat, x='Cod_Area', hue='CHURN');
```



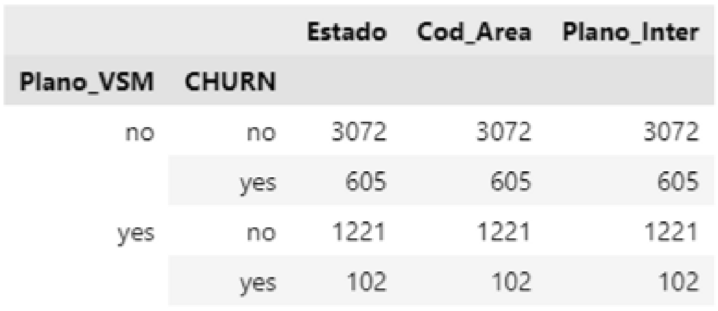
```
# Calculando as proporções de cada estado
PropCA = pd.crosstab(index=VarCat['Cod_Area'], columns=VarCat['CHURN'])
PropCA['%CHURN']=round(PropCA['yes']/(PropCA['no']+PropCA['yes']), 2).sort_values(ascending=False)
PropCA['%CHURN'].sort_values(ascending=False).head(10)
```

```
Plano_Inter
yes     0.42
no      0.11
Name: %CHURN, dtype: float64
```

Insights -> For the International Plan variable, we observe that 42% of customers with an International Plan churned, indicating a positive churn. This is an interesting piece of information as it may be one of the reasons why customers become dissatisfied.

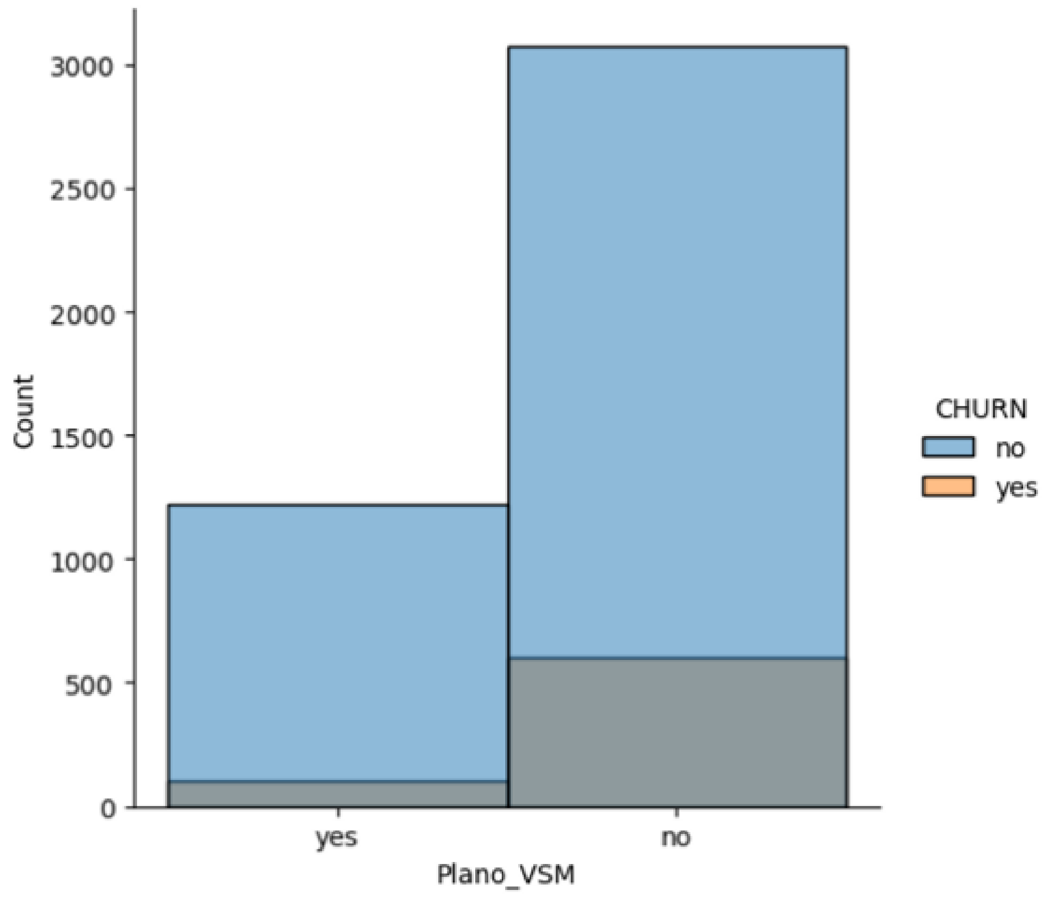
Variable Plano_VSM

```
# Quantidade de Observações por Categoria do Plano de Voice Service Messenger
VarCat.groupby(['Plano_VSM', 'CHURN']).count()
```



		Estado	Cod_Area	Plano_Inter
Plano_VSM	**CHURN**			
no	no	3072	3072	3072
	yes	605	605	605
yes	no	1221	1221	1221
	yes	102	102	102

```
# Gráfico de Barras com a Contagem das Observações por Categoria do Plano de VSM e Categoria de CHURN
sns.displot(data=VarCat, x='Plano_VSM', hue='CHURN', kind='hist');
```



```
# Calculando as proporções de cada Categoria do Plano VSM
PropPV = pd.crosstab(index=VarCat['Plano_VSM'], columns=VarCat['CHURN'])
PropPV['%CHURN']=round(PropPV['yes']/(PropPV['no']+PropPV['yes']), 2).sort_values(ascending=False)
PropPV['%CHURN'].sort_values(ascending=False).head(10)
```

```
Plano_VSM
no      0.16
yes     0.08
Name: %CHURN, dtype: float64
```

Insight -> For the variable related to the use of Voice Service Messenger, we notice that positive churn does not seem to be related to the Voice Messenger service. We have a 8% positive churn for those who use VSM and 16% for those who do not use the plan.

Data Preprocessing

At this stage, we will start preparing our data for predictive modeling.

The first step is to perform Label Encoding on the Categorical Variables, then apply a technique to balance the number of observations in the target variable. After that, we will split the data into training and testing sets, and finally, we will standardize the data to equalize the scales of each predictor variable.

Feature Engineering

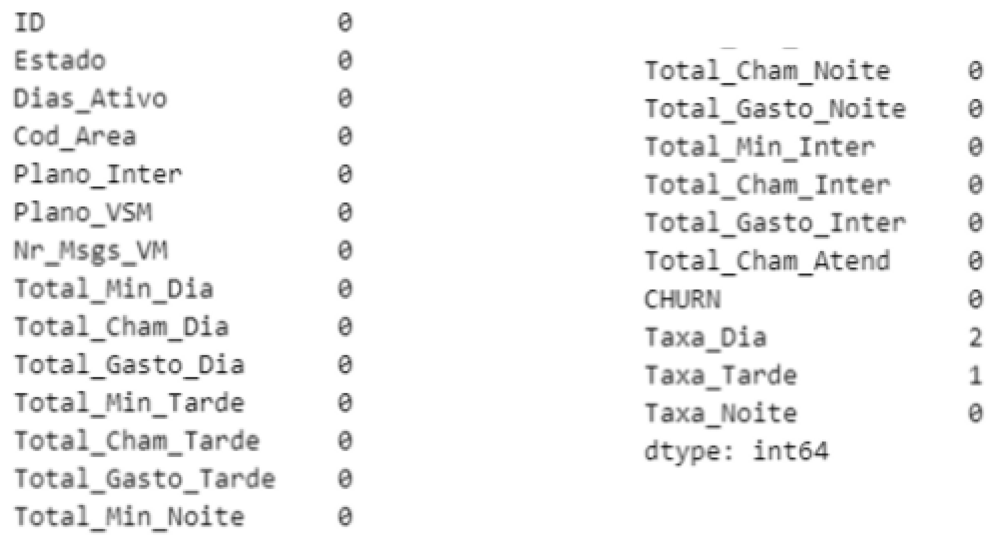
As we identified during the correlation analysis in the exploratory data phase, there is multicollinearity between the variables Total_Min and Total_Gasto. Therefore, we will create a new variable, R$/Min rate, for each period, and then remove the Total_Min and Total_Gasto variables from the modeling dataset.

For the variables related to Plano_Internacional (International Plan), we have a different situation. There is the same multicollinearity between Total_Min and Total_Gasto, but calculating a new variable for R$/min rate will result in NaN values due to division by zero. This is because we have a categorical variable indicating the usage of this type of plan as either "Yes" or "No." Since all these variables are correlated, we will exclude Total_Min and the categorical variable Plano_Inter, and only keep Total_Cham_Inter and Total_Gasto_Inter.

By doing this, we aim to reduce the model's bias, simplify the dimensions, and increase generalization.

```
# Criando um Novo Dataset com as Variáveis Taxa_Dia, Taxa_Tarde, Taxa_Noite
df1 = df.copy()
df1['Taxa_Dia'] = df1['Total_Gasto_Dia']/df1['Total_Min_Dia']
df1['Taxa_Tarde'] = df1['Total_Gasto_Tarde']/df1['Total_Min_Tarde']
df1['Taxa_Noite'] = df1['Total_Gasto_Tarde']/df1['Total_Min_Noite']

# Verificando se temos Dados NaN
df1.isnull().sum()
```



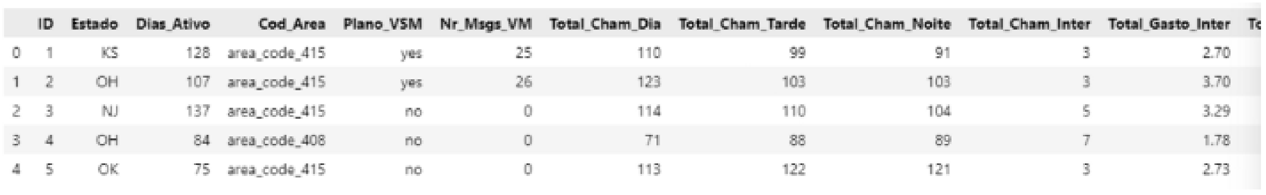
```
ID                    0
Estado                0
Dias_Ativo            0
Cod_Area              0
Plano_Inter           0
Plano_VSM             0
Nr_Msgs_VM            0
Total_Min_Dia         0
Total_Cham_Dia        0
Total_Gasto_Dia       0
Total_Min_Tarde       0
Total_Cham_Tarde      0
Total_Gasto_Tarde     0
Total_Min_Noite       0

Total_Cham_Noite      0
Total_Gasto_Noite     0
Total_Min_Inter       0
Total_Cham_Inter      0
Total_Gasto_Inter     0
Total_Cham_Atend      0
CHURN                 0
Taxa_Dia              2
Taxa_Tarde            1
Taxa_Noite            0
dtype: int64
```

```
# Excluindo os dados NaN gerados
df1.dropna(inplace=True)
df1.drop(index=1374, inplace=True)

# Excluindo as Variáveis que não usaremos
df1.drop(['Total_Min_Dia', 'Total_Gasto_Dia', 'Total_Min_Tarde', 'Total_Gasto_Tarde', 'Total_Min_Noite',
          'Total_Gasto_Noite', 'Plano_Inter', 'Total_Min_Inter'],axis=1, inplace=True)
```

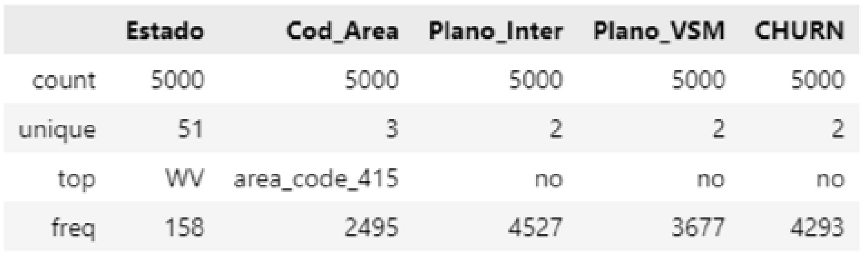
```
# Visualizando o Dataset Resultante e organizando a Variável Resposta no Final do Dataset
VR = df1['CHURN'].copy()
df1['CHURN1'] = VR
df1.drop(['CHURN'], axis=1, inplace=True)
df1.rename(columns={'CHURN1': 'CHURN'}, inplace=True)
df1.head()
```



	ID	Estado	Dias_Ativo	Cod_Area	Plano_VSM	Nr_Msgs_VM	Total_Cham_Dia	Total_Cham_Tarde	Total_Cham_Noite	Total_Cham_Inter	Total_Gasto_Inter	Tc
0	1	KS	128	area_code_415	yes	25	110	99	91	3	2.70	
1	2	OH	107	area_code_415	yes	26	123	103	103	3	3.70	
2	3	NJ	137	area_code_415	no	0	114	110	104	5	3.29	
3	4	OH	84	area_code_408	no	0	71	88	89	7	1.78	
4	5	OK	75	area_code_415	no	0	113	122	121	3	2.73	

Applying Label Encoding Technique

```
# Listando as Variáveis Categóricas
VarCat.describe()
```



	Estado	Cod_Area	Plano_Inter	Plano_VSM	CHURN
count	5000	5000	5000	5000	5000
unique	51	3	2	2	2
top	WV	area_code_415	no	no	no
freq	158	2495	4527	3677	4293

What we can observe is that the State variable has 51 unique values. The Label Encoding technique is not suitable for this case due to the large number of factors that would be created.

Applying One-Hot Encoding would also not be ideal as it would introduce 51 additional variables to our project, increasing the complexity of the model.

Another consideration is the geographical representation provided by this variable, which can be represented by the Area Code variable, thus retaining at least one form of geographical information in our modeling. Taking these details into account, we will exclude the use of the State variable and keep the Cod_Area and Plano_VSM variables.

The response variable, CHURN, also needs to undergo Label Encoding, and we will address it accordingly.

```
# Verificando os valores únicos da Variável Cod_Area
df['Cod_Area'].value_counts()
```

```
area_code_415    2492
area_code_408    1258
area_code_510    1245
Name: Cod_Area, dtype: int64
```

```
# Criando modelo de Label Encoding Cod Area
Label = LabelEncoder().fit(df1['Cod_Area'])
Label.classes_
```

```
array(['area_code_408', 'area_code_415', 'area_code_510'], dtype=object)
```

```
# Aplicando Modelo na Variável Cod Area
df1['Cod_Area'] = Label.transform(df1['Cod_Area'])
df1['Cod_Area'].value_counts()
```

```
1    2492
0    1258
2    1245
Name: Cod_Area, dtype: int64
```

For the Cod_Area variable, we will use Label Encoding as follows:

- area_code_408 -> 0
- area_code_415 -> 1
- area_code_510 -> 2

```
# Verificando os valores únicos da Variável Plano_VSM
df1['Plano_VSM'].value_counts()
```

```
no      3673
yes     1322
Name: Plano_VSM, dtype: int64
```

```
# Criando modelo de Label Encoding Plano_VSM
Label = LabelEncoder().fit(df1['Plano_VSM'])
Label.classes_
```

```
array(['no', 'yes'], dtype=object)
```

```
# Aplicando Modelo na Variável Plano_VSM
df1['Plano_VSM'] = Label.transform(df1['Plano_VSM'])
df1['Plano_VSM'].value_counts()
```

```
0    3673
1    1322
Name: Plano_VSM, dtype: int64
```

For the Plano_VSM variable, we will use Label Encoding as follows:

- no -> 0
- yes -> 1

```
# Verificando os valores únicos da Variável CHURN
df['CHURN'].value_counts()
```

```
no      4289
yes      706
Name: CHURN, dtype: int64
```

```
# Criando modelo de Label Encoding "CHURN"
Label = LabelEncoder().fit(df1['CHURN'])
Label.classes_
```

```
array(['no', 'yes'], dtype=object)
```

```
# Aplicando Modelo na Variável CHURN
df1['CHURN'] = Label.transform(df1['CHURN'])
df1['CHURN'].value_counts()
```

```
0    4289
1     706
Name: CHURN, dtype: int64
```

For the CHURN variable, we will use Label Encoding as follows:

- no -> 0
- yes -> 1

Applying Oversampling to the response variable

We will use the oversampling technique to balance our dataset in terms of the number of observations for positive and negative CHURN.

```
# Criando nosso Simulador SMOTE
sm = SMOTE(random_state=0)
df2, Resp2 = sm.fit_resample(df1.iloc[:, 0:13], df1['CHURN'])
```

```
# Visualizando o Resultado do Oversampling
Resp2.value_counts()
```

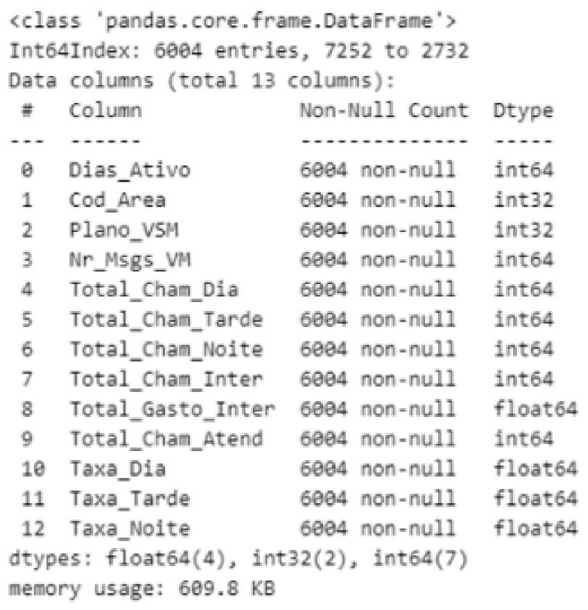
```
0    4289
1    4289
Name: CHURN, dtype: int64
```

Splitting the data into training and testing sets.

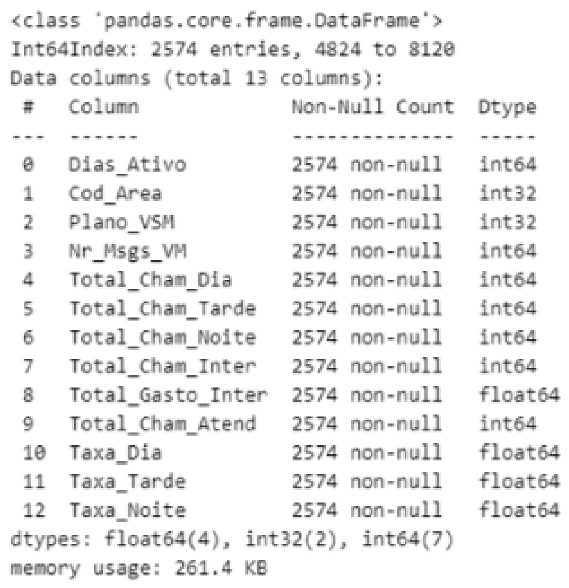
We will split the data into training and testing sets, and then scale each dataset to finalize our preprocessing.

```
# Dividindo os dados em Treino e Teste com Train_Test_Split
X_train, X_test, Y_train, Y_test = train_test_split(df2, Resp2, test_size=0.3, random_state=0)
```

```
X_train.info()
X_test.info()
```

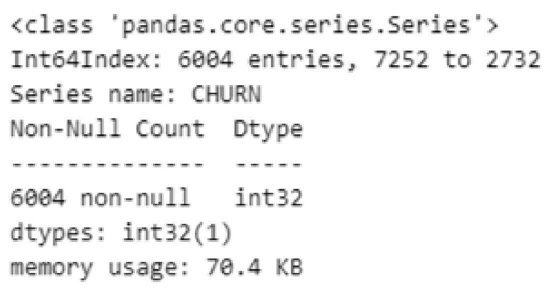


```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 6004 entries, 7252 to 2732
Data columns (total 13 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   Dias_Ativo        6004 non-null   int64  
 1   Cod_Area          6004 non-null   int32  
 2   Plano_VSM         6004 non-null   int32  
 3   Nr_Msgs_VM        6004 non-null   int64  
 4   Total_Cham_Dia    6004 non-null   int64  
 5   Total_Cham_Tarde  6004 non-null   int64  
 6   Total_Cham_Noite  6004 non-null   int64  
 7   Total_Cham_Inter  6004 non-null   int64  
 8   Total_Gasto_Inter 6004 non-null   float64
 9   Total_Cham_Atend  6004 non-null   int64  
 10  Taxa_Dia          6004 non-null   float64
 11  Taxa_Tarde        6004 non-null   float64
 12  Taxa_Noite        6004 non-null   float64
dtypes: float64(4), int32(2), int64(7)
memory usage: 609.8 KB
```

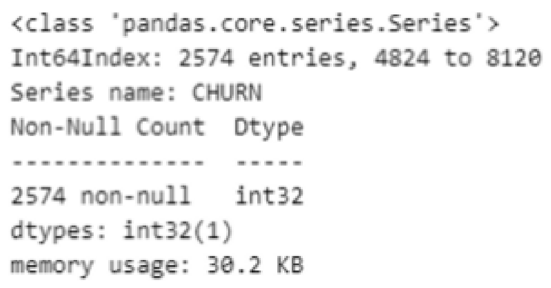


```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2574 entries, 4824 to 8120
Data columns (total 13 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Dias_Ativo         2574 non-null   int64  
 1   Cod_Area           2574 non-null   int32  
 2   Plano_VSM          2574 non-null   int32  
 3   Nr_Msgs_VM         2574 non-null   int64  
 4   Total_Cham_Dia     2574 non-null   int64  
 5   Total_Cham_Tarde   2574 non-null   int64  
 6   Total_Cham_Noite   2574 non-null   int64  
 7   Total_Cham_Inter   2574 non-null   int64  
 8   Total_Gasto_Inter  2574 non-null   float64
 9   Total_Cham_Atend   2574 non-null   int64  
 10  Taxa_Dia           2574 non-null   float64
 11  Taxa_Tarde         2574 non-null   float64
 12  Taxa_Noite         2574 non-null   float64
dtypes: float64(4), int32(2), int64(7)
memory usage: 261.4 KB
```

```
Y_train.info()
Y_test.info()
```



```
<class 'pandas.core.series.Series'>
Int64Index: 6004 entries, 7252 to 2732
Series name: CHURN
Non-Null Count  Dtype
--------------  -----
6004 non-null   int32
dtypes: int32(1)
memory usage: 70.4 KB
```



```
<class 'pandas.core.series.Series'>
Int64Index: 2574 entries, 4824 to 8120
Series name: CHURN
Non-Null Count  Dtype
--------------  -----
2574 non-null   int32
dtypes: int32(1)
memory usage: 30.2 KB
```

```
# Criando nosso Simulador SMOTE
sm = SMOTE(random_state=0)
df2, Resp2 = sm.fit_resample(df1.iloc[:, 0:13], df1['CHURN'])
```

Applying Standardization to the training and testing datasets.

As we observed, the predictor variables have a behavior very close to a Normal Distribution, so using the Z-Score method for Standardization is more appropriate than Normalization in this case.

```
# Criando um modelo de StandarScaler()
scaler = StandardScaler().fit(X_train)
```

```
# Aplicando Padronizador aos dados de Treino e Test
X_trainS = scaler.transform(X_train)
X_testS = scaler.transform(X_test)
```

Preprocessing Completed! Let's proceed with Predictive Modeling...

Machine Learning

In this stage, we will create several models, calculate their respective metrics, and determine which model to use for the final delivery to the client.

Model 00 - Logistic Regression

```
# Criando um Modelo com Regressão Logística
Modelo00 = LogisticRegression(penalty='l2', max_iter=1000)
```

```
# Treinando o Modelo
Modelo00 = Modelo00.fit(X_trainS, Y_train)
```

```
# Aplicando Modelo aos Dados de Teste
Prev00 = Modelo00.predict(X_testS)
```

```
# Calculando as Métricas do Modelo
Acc00 = accuracy_score(Y_test, Prev00)
Prec00 = precision_score(Y_test, Prev00)
rAUC00 = roc_auc_score(Y_test, Prev00)

# Visualizando as Métricas do Modelo
print(' Acurácia = ', Acc00, '\n',
      'Precisão = ', Prec00, '\n',
      'rAUC = ', rAUC00)
```

```
Acurácia =  0.6585081585081585
Precisão =  0.6480529022777369
rAUC =  0.6586222427225262
```

```
# Prevendo as probabilidades das Categorias de CHURN
PrevP00 = Modelo00.predict_proba(X_testS)
PrevP00 = round(pd.DataFrame(PrevP00, columns=['CHURN No', 'CHURN Yes']), 2)*100
PrevP00.head()
```

	CHURN No	CHURN Yes
0	75.0	25.0
1	47.0	53.0
2	37.0	63.0
3	29.0	71.0
4	56.0	44.0

Model 01 - Stochastic Gradient Descent Classifier

```
# Criando um Modelo
Modelo01 = SGDClassifier(loss='log', max_iter=10000)

# Treinando o Modelo
Modelo01 = Modelo01.fit(X_trainS, Y_train)

# Aplicando Modelo aos Dados de Teste
Prev01 = Modelo01.predict(X_testS)
```

```
# Calculando as Métricas do Modelo
Acc01 = accuracy_score(Y_test, Prev01)
Prec01 = precision_score(Y_test, Prev01)
rAUC01 = roc_auc_score(Y_test, Prev01)

# Visualizando as Métricas do Modelo
print(' Acurácia = ', Acc01, '\n',
      'Precisão = ', Prec01, '\n',
      'rAUC = ', rAUC01)
```

```
Acurácia =  0.6177156177156177
Precisão =  0.6446601941747573
rAUC =  0.6173294919412876
```

```
# Prevendo as probabilidades das Categorias de CHURN
PrevP01 = Modelo01.predict_proba(X_testS)
PrevP01 = round(pd.DataFrame(PrevP01, columns=['CHURN No', 'CHURN Yes']), 2)*100
PrevP01.head()
```

	CHURN No	CHURN Yes
0	69.0	31.0
1	66.0	34.0
2	36.0	64.0
3	29.0	71.0
4	52.0	48.0

Model 02 - Ridge Classifier

```
# Criando um Modelo
Modelo02 = RidgeClassifier()

# Treinando o Modelo
Modelo02 = Modelo02.fit(X_trainS, Y_train)

# Aplicando Modelo aos Dados de Teste
Prev02 = Modelo02.predict(X_testS)
```

```
# Calculando as Métricas do Modelo
Acc02 = accuracy_score(Y_test, Prev02)
Prec02 = precision_score(Y_test, Prev02)
rAUC02 = roc_auc_score(Y_test, Prev02)

# Visualizando as Métricas do Modelo
print(' Acurácia = ', Acc02, '\n',
      'Precisão = ', Prec02, '\n',
      'rAUC = ', rAUC02)
```

```
 Acurácia =  0.6596736596736597
 Precisão =  0.6471014492753623
 rAUC =  0.6598164390971923
```

```
# Visualizando as Previsões
PrevP02 = Modelo02.predict(X_testS)
PrevP02 = pd.DataFrame(PrevP02, columns=['CHURN'])
PrevP02.head()
```

	CHURN
0	0
1	1
2	1
3	1
4	0

Model 03 - Decision Tree Classifier

```
# Criando um Modelo
Modelo03 = DecisionTreeClassifier(max_depth=100, criterion='gini')

# Treinando o Modelo
Modelo03 = Modelo03.fit(X_trainS, Y_train)

# Aplicando Modelo aos Dados de Teste
Prev03 = Modelo03.predict(X_testS)

# Calculando as Métricas do Modelo
Acc03 = accuracy_score(Y_test, Prev03)
Prec03 = precision_score(Y_test, Prev03)
rAUC03 = roc_auc_score(Y_test, Prev03)
```

```
# Visualizando as Métricas do Modelo
print(' Acurácia = ', Acc03, '\n',
      'Precisão = ', Prec03, '\n',
      'rAUC = ', rAUC03)
```

```
Acurácia =  0.7331002331002331
Precisão =  0.7231807951987997
rAUC =  0.7331731814164207
```

```
# Visualizando as Previsões
PrevP03 = Modelo03.predict(X_testS)
PrevP03 = pd.DataFrame(PrevP03, columns=['CHURN'])
PrevP03.head()
```

	CHURN
0	0
1	1
2	0
3	1
4	1

Model 04 - Support Vector Machine

```
# Criando um Modelo
Modelo04 = SVC(probability=True)

# Treinando o Modelo
Modelo04 = Modelo04.fit(X_trainS, Y_train)

# Aplicando Modelo aos Dados de Teste
Prev04 = Modelo04.predict(X_testS)

# Calculando as Métricas do Modelo
Acc04 = accuracy_score(Y_test, Prev04)
Prec04 = precision_score(Y_test, Prev04)
rAUC04 = roc_auc_score(Y_test, Prev04)

# Visualizando as Métricas do Modelo
print(' Acurácia = ', Acc0
      'Precisão = ', Prec0
      'rAUC = ', rAUC04)
```

```
Acurácia =  0.7226107226107226
Precisão =  0.6997187060478199
rAUC =  0.722817844602329
```

```
# Prevendo as probabilidades das Categorias de CHURN
PrevP04 = Modelo04.predict_proba(X_testS)
PrevP04 = round(pd.DataFrame(PrevP04, columns=['CHURN No', 'CHURN Yes']), 2)*100
PrevP04.head()
```

	CHURN No	CHURN Yes
0	87.0	13.0
1	18.0	82.0
2	36.0	64.0
3	15.0	85.0
4	30.0	70.0

Model 05 - Support Vector Machine with Grid Search (Hyper parameter Optimization)

```
# Otimização de Hiperparâmetros
param = {'kernel':('linear', 'rbf'), 'C': [0.1, 1, 10]}

# Criando um Modelo
Modelo05 = SVC(probability=True)

# Aplicando GridSearch
clf = GridSearchCV(Modelo05, param).fit(X_trainS, Y_train)
set = clf.best_params_
set
```

```
{'C': 10, 'kernel': 'rbf'}
```

```
# Criando Modelo com os Parâmetros Otimizados
Modelo05ot = SVC(kernel='rbf', C=10, probability=True)

# Treinando o Modelo
Modelo05ot = Modelo05ot.fit(X_trainS, Y_train)
```

```
# Aplicando Modelo aos Dados de Teste
Prev05 = Modelo05ot.predict(X_testS)

# Calculando as Métricas do Modelo
Acc05 = accuracy_score(Y_test, Prev05)
Prec05 = precision_score(Y_test, Prev05)
rAUC05 = roc_auc_score(Y_test, Prev05)

# Visualizando as Métricas do Modelo
print(' Acurácia = ', Acc05, '\n',
      'Precisão = ', Prec05, '\n',
      'rAUC = ', rAUC05)
```

```
 Acurácia =  0.7731157731157731 
 Precisão =  0.7354925775978407 
 rAUC =  0.7734142183024784
```

```
# Prevendo as probabilidades das Categorias de CHURN
PrevP05 = Modelo05ot.predict_proba(X_testS)
PrevP05 = round(pd.DataFrame(PrevP05, columns=['CHURN No', 'CHURN Yes']), 2)*100
PrevP05.head()
```

	CHURN No	CHURN Yes
0	95.0	5.0
1	17.0	83.0
2	48.0	52.0
3	13.0	87.0
4	25.0	75.0

Conclusion

We have finally arrived at several models with different performances. Let's compare them and decide which model to deliver to the decision maker.

```
# Criando um Dataset com os resultados de Acurácia, Precisão e rAUC
Resultado = pd.DataFrame({'Modelo00': [Acc00, Prec00, rAUC00],
                          'Modelo01': [Acc01, Prec01, rAUC01],
                          'Modelo02': [Acc02, Prec02, rAUC02],
                          'Modelo03': [Acc03, Prec03, rAUC03],
                          'Modelo04': [Acc04, Prec04, rAUC04],
                          'Modelo05ot': [Acc05, Prec05, rAUC05]}, index=['Acurácia', 'Precisão', 'rAUC'])
```

	Modelo00	Modelo01	Modelo02	Modelo03	Modelo04	Modelo05ot
Acurácia	0.658508	0.617716	0.659674	0.733100	0.722611	0.773116
Precisão	0.648053	0.644660	0.647101	0.723181	0.699719	0.735493
rAUC	0.658622	0.617329	0.659816	0.733173	0.722818	0.773414

As we can see, Model 05 with the Support Vector Machine algorithm and Hyper parameter Optimization using Grid Search produced the best metrics, as shown in the table below. This will be the model we deliver for predictions on new data as desired by the decision maker.

```
# Métricas do Modelo Final
Resultado['Modelo05ot']
```

```
Acurácia    0.773116
Precisão    0.735493
rAUC        0.773414
Name: Modelo05ot, dtype: float64
```

Summary of Insights:

- Low usage of Voice E-mail service among the majority of customers. In the case of customers who switched their phone plans (CHURN "yes"), nearly 90% of them do not use the service. This suggests the possibility of reducing the cost of this service and offering a more useful alternative to customers.
- The time period in which calls are made does not have a direct influence on customer decisions to stay or leave the phone plan. We did not identify any behavioral differences between positive and negative churn.
- Regarding International Calls, what stands out is that the means and medians are similar for both positive and negative churn. However, when looking at the quartiles of the Total International Calls variable, we notice a difference between churn categories, indicating that it is a less utilized service among customers who have left the phone plan, despite the similar median and mean values.
- When it comes to the Total Calls for Customer Service, the key point to note is the mode, which indicates that both positive and negative churn customers have made one call. This suggests that this variable alone does not characterize a phone plan change. It is surprising

because there was a business concern that more calls to customer service would lead to higher customer dissatisfaction and, consequently, higher positive churn.

- Although area code 415 has a higher absolute number of positive churn, when considering the number of customers in each area code, we observe that the churn rate is similar across area codes, with a slight superiority of area code 510.
- Certain states stand out with a high churn rate compared to the total number of customers. These states include California, New Jersey, Washington, Texas, Montana, and Maryland, with churn rates above 20%.
- For the International Plan variable, we notice that 42% of customers with an international plan switched their phone plan, indicating positive churn. This is an interesting finding as it may be one of the reasons why customers become dissatisfied, either due to call quality or tariff rates.
- Regarding the variable related to Voice Service Messenger (VSM) usage, positive churn does not seem to be related to VSM. We have an 8% positive churn rate for users of VSM and 16% for non-users of the plan.