PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS
NÚCLEO DE EDUCAÇÃO A DISTÂNCIA
Pós-graduação *Lato Sensu* em Ciência de Dados e Big Data

JOÃO MARCELO FONSECA CUNHA

APLICAÇÃO DE CHURN PREDICTION EM EMPRESA DE TELECOMUNICAÇÕES COM MACHINE LEARNING

Belo Horizonte
2023

João Marcelo Fonseca Cunha

APLICAÇÃO DE CHURN PREDICTION EM EMPRESA DE TELECOMUNICAÇÕES COM MACHINE LEARNING

Trabalho de Conclusão de Curso apresentado ao Curso de Especialização em Ciência de Dados e Big Data como requisito parcial à obtenção do título de especialista.

Belo Horizonte
2023

SUMÁRIO

1. Introdução....4
1.1. Contextualização....4
1.2. O problema proposto....5
2. Coleta de Dados....9
3. Processamento/Tratamento de Dados....18
4. Análise e Exploração dos Dados....23
5. Criação de Modelos de Machine Learning....37
5.1. Avaliando modelos com Cross Validation (Dados Desbalanceados)....41
5.2. Avaliando modelos com Cross Validation (Dados Balanceados)....44
5.3. Otimização do modelo e ajuste dos hiper parâmetros....47
6. Apresentação e Interpretação dos Resultados....51
6.1. Apresentação dos Resultados pelo Modelo Canvas....55
7. Links....56
REFERÊNCIAS....57
APÊNDICE....58

1. Introdução

1.1. Contextualização

Nos últimos anos, a área de telecomunicações tem enfrentado desafios significativos devido à crescente competição e à constante evolução tecnológica. Nesse contexto, manter e fidelizar os clientes tornou-se uma prioridade estratégica para as empresas do setor. O fenômeno conhecido como *churn*, que se refere à perda de clientes, é uma preocupação recorrente, pois impacta diretamente a rentabilidade e a sustentabilidade dessas empresas.

O *churn* do consumidor é definido pela propensão do cliente/consumidor de cessar a realização de negócios com uma empresa em um determinado período de tempo; tornou-se um dos principais desafios para as empresas em todo o mundo (CHANDAR et al. 2006, apud MA; TAN; SHU, 2015). Ele está diretamente relacionado à retenção de clientes e funciona como um indicador da eficiência na retenção de uma empresa. Reduzir o *churn* de clientes, orientando campanhas de marketing especificamente para clientes com maior probabilidade de cancelamento, provou ser rentável às empresas. Objetivando aumentar a eficiência dessas campanhas, um modelo de predição é necessário, para que a identificação desses clientes seja possível (VERBRAKEN; VERBEKE; BAESENS, 2014).

Com a entrada de novas companhias em vários nichos de mercado, é inevitável que ocorra o acirramento da concorrência entre elas. Desse modo, variados tipos de mercado, passam a ficar cada vez mais saturados e pressionados pelo aumento da competitividade (PIMENTEL, 2019). Como resultado, as empresas vêm notando que suas estratégias comerciais devem priorizar a manutenção dos clientes atuais, ao invés de atraírem novos (COUSSEMENT; POEL, 2009). Nessa perspectiva, existe um aumento da relevância dada às iniciativas de gerenciamento com o consumidor (em inglês, *CRM – Client Relationship Managemenent*) dentro das organizações. O CRM é uma abordagem de gerenciamento que visa desenvolver, aprimorar e criar os relacionamentos com clientes criteriosamente segmentados para maximizar a rentabilidade corporativa e o valor do cliente (A. PAYNE, 2005). Um dos maiores desafios enfrentados pelos CRM é a identificação de clientes propensos ao *churn* (i.e., cancelamento) de serviços e/ou produtos (HADDEN et al., 2007).

Para lidar com o desafio do *churn* e identificar antecipadamente os clientes propensos a cancelar seus serviços, a aplicação de técnicas de *Churn Prediction* tem se mostrado uma abordagem promissora. Essa abordagem, aliada ao uso de *machine learning*, permite às

empresas de telecomunicações prever com maior precisão quais clientes estão mais propensos a abandonar seus serviços, permitindo ações proativas para reter esses clientes e reduzir a taxa de *churn*.

1.2. O problema proposto

O presente trabalho tem como objetivo explorar a aplicação de *Churn Prediction* em uma empresa de telecomunicações, utilizando técnicas de *machine learning*. Os dados utilizados neste projeto foram originalmente disponibilizados na plataforma de ensino da IBM *Developer Business Analytics*, mas foram acessados através do *Kaggle*. (Disponível em: https://www.kaggle.com/datasets/yeanzc/telco-customer-churn-ibm-dataset. Acesso em 10 de maio de 2023). Apesar de não haver informações explícitas disponíveis, os nomes das colunas permitem um entendimento a respeito do problema, conforme observado mais adiante na etapa de análise exploratória.

Serão utilizados dados históricos de clientes, combinados com variáveis relevantes do contexto do setor, a fim de construir modelos preditivos para identificar padrões e tendências nos dados dos clientes, utilizando técnicas de validação cruzada, seleção de características e otimização de hiper parâmetros.

Para nosso projeto iremos utilizar os seguintes modelos:

a) Árvore de Decisão
b) Random Forest
c) Regressão Logística
d) SGD Classifier
e) XGBoost Classifier

A ideia é comparar as métricas de cada um dos modelos na etapa de validação afim de verificamos o que apresenta os melhores resultados aplicáveis ao nosso problema. A Tabela 1, documentada abaixo, sintetiza bem cada um dos modelos utilizados neste trabalho, elencando os pontos positivos e negativos de cada um deles.

Tabela 1 - Síntese dos Modelos De Machine Learning

Modelo	Definição	Ponto Positivo	Ponto Negativo
Árvore de decisão	Uma árvore de decisão é um algoritmo de aprendizado de máquina supervisionado que é utilizado para classificação e para regressão. Assim como um fluxograma, a árvore de decisão estabelece nós (*decision nodes*) que se relacionam entre si por uma hierarquia. A partir dela, pode-se classificar a amostra desconhecida sem necessariamente testar todos os valores dos seus atributos.	Rápido uso computacional Possuem fácil interpretabilidade	Sensível a pequenas variações nos dados Possuem tendência de criar árvores muito complexas que se ajustam em excesso aos dados de treinamento
Random Forest	Combinação de preditores de árvores de decisão de modo que cada árvore depende dos valores de um vetor aleatório amostrado independentemente e com a mesma distribuição para todas as árvores na floresta (BREIMAN, 2001).	Tendem a possui alta precisão nas classificações Lida bem com dados desbalanceados	É mais complexa do que uma única árvore de decisão, tornando-a menos interpretável Pode ter um custo computacional mais elevado
Regressão Logística	Relaciona um conjunto de variáveis independentes com uma variável dependente categórica (GUANGLI et al., 2011).	Apresenta boa performance com recursos limitados Método padrão para análise de variáveis dicotômicas.	Não se estende facilmente para problemas de classificação multiclasse sem adaptações adicionais, sendo projetada para resolver problemas de classificação binária. A multicolinearidade, que é a alta correlação entre as variáveis de entrada, pode afetar negativamente os resultados da regressão logística.

SGD Classifier	É um algoritmo de aprendizado de máquina usado para problemas de classificação. Ele pertence à família de algoritmos baseados em gradiente descendente estocástico. Utiliza o método de otimização de gradiente descendente estocástico para ajustar os parâmetros do modelo aos dados de treinamento. O gradiente descendente estocástico é uma abordagem iterativa que ajusta os parâmetros do modelo para minimizar uma função de perda.	O *SGD Classifier* é altamente eficiente e escalável, especialmente em grandes conjuntos de dados. O *SGD Classifier* é flexível e pode ser aplicado em diferentes tipos de problemas de classificação, incluindo classificação binária e multiclasse.	O desempenho do *SGD Classifier* é sensível à escolha adequada dos hiper parâmetros, como a taxa de aprendizado (learning rate) e os parâmetros de regularização. O *SGD Classifier* pode ser influenciado por problemas de desbalanceamento de classe, onde uma classe é significativamente mais frequente que a outra.
XGBoost Classifier	O modelo *XGBoost Classifier* (*Extreme Gradient Boosting Classifier*) é um algoritmo de aprendizado de máquina baseado em *gradient boosting* e é uma implementação do algoritmo de *gradient boosting* para classificação. Ele pertence à família de algoritmos de *boosting*, que combinam vários modelos de aprendizado fracos para criar um modelo forte. O *XGBoost Classifier* utiliza a técnica de *boosting*, onde várias árvores de decisão simples são construídas de forma sequencial. Cada árvore é treinada para corrigir os erros cometidos pelas árvores anteriores, enfatizando os exemplos que foram classificados incorretamente.	Tende a possuir alta performance e eficácia em várias tarefas de classificação. Ele é frequentemente usado em competições de ciência de dados e é considerado um dos algoritmos mais poderosos para classificação. Oferece uma ampla gama de hiper parâmetros que podem ser ajustados para controlar o desempenho e a complexidade do modelo.	Assim como muitos outros algoritmos de machine learning, o desempenho do *XGBost Classifier* pode depender fortemente da seleção adequada de hiper parâmetros. O *XGBoost Classifier* é um algoritmo mais avançado e pode exigir um conhecimento mais aprofundado para sua configuração e otimização.

Fonte: (Autor)

Em síntese, pelo método do 5W's, a Tabela 2 documentada abaixo apresenta a estratificação do problema da pesquisa.

Tabela 2 - Estratificação do problema da pesquisa com o método 5W's

W	Resposta
(*Why*?) Por que esse problema é importante?	Para identificação de possíveis clientes que podem cancelar suas assinaturas de serviços de telecomunicação, mitigando o risco de perda desses clientes e eliminando custo de aquisição de novos clientes, que em geral é mais alto que o custo para os manter.
(*Who*?) De quem são os dados analisados? De um governo? Um ministério ou secretaria? Dados de clientes?	Dados disponibilizados na plataforma de ensino da *IBM Developer*, e tratam de um problema típico de uma companhia de telecomunicações.
(*What*?) Quais os objetivos com essa análise? O que iremos analisar?	Analisar dados de *churn* de uma empresa de telecomunicações e criar modelos de machine learning para classificar possíveis clientes que possam cancelar seus planos, comparando as métricas e buscando as melhores soluções
(*Where*?) Trata dos aspectos geográficos e logísticos de sua análise.	Informações sobre uma empresa de telecomunicações fictícia que fornecia serviços de telefone residencial e Internet para 7.043 clientes na Califórnia
(*When*?) Qual o período está sendo analisado? A última semana? Os últimos 6 meses? O ano passado?	O *dataset* não apresenta informações de data das ocorrências.

Fonte: (Autor)

Neste trabalho buscamos abordar a aplicação da metodologia *CRISP-DM (Cross-Industry Standard Process for Data Mining*) de forma adaptada e estruturada para desenvolver um modelo de *churn prediction* em uma empresa de telecomunicações, utilizando técnicas de *machine learning*. Nesse contexto, é fundamental seguir uma abordagem metodológica adequada. A metodologia *CRISP-DM* surge como um guia confiável e amplamente utilizado para projetos de mineração de dados, incluindo aqueles relacionados a *machine learning.*

Segundo MARTÍNEZ-PLUMED et al. (2019), em muitas pesquisas feitas com empresas e usuários, o *CRISP-DM* ainda é o padrão para o desenvolvimento de projetos de mineração

de dados e descoberta de conhecimento. O campo avançou bastante em vinte anos com a ciência de dados, sendo agora o método preferido na mineração de dados.

As etapas e procedimentos metodológicos adotados neste estudo são apresentados na Tabela 3.

Tabela 3 - Procedimentos metodológicos utilizados no trabalho

ENTENDIMENTO DO PROBLEMA	
Etapa 1	A primeira etapa consiste na identificação das necessidades do nosso projeto. De forma resumida, é nessa etapa que fazemos a identificação do tema abordado e entendimento do processo. Em nosso projeto, vamos abordar a aplicação de *churn prediction* em empresa de telecomunicações com *machine learning*.
ENTENDIMENTO DOS DADOS	
Etapa 2	Esta etapa consiste em coletar, organizar e documentar todos os dados que se encontram disponíveis. Precisamos identificar os dados importantes para a resolução do problema, analisar a qualidade dos dados, realizar análise descritiva e identificar padrões. É nesta etapa que realizamos a análise exploratória em que visamos compreender a estrutura e a natureza de um conjunto de dados, identificando padrões, relações, outliers, entre outros aspectos.
PREPARAÇÃO DOS DADOS	
Etapa 3	Nesta etapa precisamos tratar os dados para certificarmos que as informações estão de acordo com o que se espera. Consistência de erros e valores ausentes, dados desbalanceados devem ser resolvidos para que possamos selecionar amostras aleatórias e utilizá-las para treino, validação e teste.
MODELOS DE MACHINE LEARNING	
Etapa 4	Nesta etapa definimos e avaliamos os modelos de *machine learning* que vamos testar. Documentamos as técnicas selecionadas, motivo da escolha, avaliação e análise do comportamento das métricas para cada modelo de *machine learning* escolhido.
AVALIAÇÃO FINAL DO MODELO	
Etapa 5	Nesta etapa, serão avaliados os modelos desenvolvidos com base nas métricas definidas anteriormente. Será realizada uma análise aprofundada do desempenho com base em métricas como acurácia, precisão, *recall* e *F1-score*. A validação cruzada e a avaliação em conjunto com dados de teste serão utilizadas para garantir a robustez do modelo. As definições e cálculos das métricas de avaliação estão detalhadas mais adiante em nosso projeto, na descrição das etapas de avaliação dos modelos de *machine learning*.

Fonte: (Autor)

2. Coleta de Dados

Para realizarmos nosso projeto utilizamos o Python versão 3.10.2 (Figura 1), uma linguagem de programação interpretada, que se destaca pela sua sintaxe clara e legível. Uma das principais características do Python é a sua ampla biblioteca padrão, que fornece uma vasta gama de módulos e funções para realizar várias tarefas, como manipulação de arquivos, acesso a bancos de dados, criação de interfaces gráficas.

Utilizamos também Jupyter Notebook, versão 6.4.8 (Figura 1), uma aplicação web de código aberto que permite criar e compartilhar documentos interativos contendo código, texto explicativo, visualizações e outros elementos. Ele é amplamente utilizado na ciência de dados, pesquisa acadêmica e em outros campos relacionados. O passo a passo para instalação do Jupyter Notebook pode ser acessado no link https://learnpython.com/blog/jupyter-notebook-python-ide-installation-tips/.

Figura 1 - Versão do Python e Jupyter Notebook

Fonte: (Autor)

Importamos algumas bibliotecas Python (Figura 2), e utilizamos também bibliotecas nativas da linguagem, para realizarmos as etapas de análise, processamento, tratamento dos dados e criação dos nossos modelos de *machine learning.* Antes de importar nossas bibliotecas, executamos código *"! pip install scikit-plot -q*", que é uma instrução para instalar o pacote "*scikit-plot*" usando o gerenciador de pacotes "*pip*" no Jupyter Notebook. O "*pip*" é uma

ferramenta usada para instalar pacotes e bibliotecas em Python. O pacote "*scikit-plot*" é uma biblioteca de visualização de gráficos para modelos de aprendizado de máquina em Python. O parâmetro "*-q*" passado para o comando "*pip install*" significa "*quiet*" (silencioso), o que faz com que a instalação seja executada sem exibir mensagens detalhadas no output. Também executamos o código *"! pip install imbalanced-learn -q"* que é uma biblioteca em Python projetada para lidar com problemas de desequilíbrio de classes em conjuntos de dados de aprendizado de máquina, e será aplicado aos balanceamentos do nosso dataset antes de aplicarmos ao nosso modelo. Por fim, executamos o código para importar a função "*simplefilter*" do módulo "*warnings*" que configura o filtro de avisos para ignorar os "FutureWarning" durante a execução do código no Jupyter Notebook, evitando a exibição desses avisos no output.

Utilizamos as seguintes bibliotecas em nosso trabalho: "Pandas" e "Numpy" para manipulação, análise de dados tabulares e para realizar operações matemáticas e numéricas eficientes em *arrays* e matrizes; "Matplotlib", "Seaborn" e "Scikitplot", para criar gráficos e visualizações, incluindo visualização de métricas de desempenho de modelos de aprendizado de máquina; da biblioteca "imblearn", importamos alguns métodos de balanceamento para lidar com problemas de desequilíbrio de classes em conjuntos de dados de aprendizado de máquina; a biblioteca "sklearn" (Scikit-learn), que é uma biblioteca de aprendizado de máquina de código aberto que suporta aprendizado supervisionado e não supervisionado. Ele também fornece várias ferramentas para ajuste de modelo, pré-processamento de dados, seleção de modelo e avaliação de modelo (Disponível em https://scikit-learn.org/stable/getting_started.html. Acesso em 10 de maio de 2023); "XGBoost" é uma biblioteca otimizada de aumento de gradiente distribuída projetada para ser altamente eficiente, flexível e portátil. Ele implementa algoritmos de aprendizado de máquina sob a estrutura *Gradient Boosting*. O XGBoost fornece um aumento de árvore paralela (também conhecido como GBDT, GBM) que resolve muitos problemas de ciência de dados de maneira rápida e precisa. (Disponível https://xgboost.readthedocs.io/en/stable/. Acesso em 10 de maio de 2023).

Figura 2 - Importando as Bibliotecas Python

```
# importando os pacotes necessários
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import scikitplot as skplt

# importando balanceamento RUS
from imblearn.under_sampling import RandomUnderSampler

# importando balanceamento SMOTE
from imblearn.over_sampling import SMOTE

# importando balanceamento ADASYN
from imblearn.over_sampling import ADASYN

# importando métricas
from sklearn.metrics import recall_score, roc_auc_score, accuracy_score, f1_score, confusion_matrix, classification_report
from scikitplot.metrics import plot_confusion_matrix, plot_roc


# importando pacotes de padronização e tratamento de variaveis categóricas
from sklearn.preprocessing import StandardScaler, LabelEncoder, RobustScaler

# importando pipeline
from sklearn.pipeline import make_pipeline

# importando model_selection
from sklearn.model_selection import train_test_split, cross_val_score, StratifiedKFold, GridSearchCV

# importando modelos
from sklearn.linear_model import LogisticRegression, SGDClassifier
from sklearn.tree import DecisionTreeClassifier
from xgboost import XGBClassifier
from sklearn.ensemble import RandomForestClassifier

# random seed
np.random.seed(42)
```

Fonte: (Autor)

Ao longo do presente trabalho será possível notar que os códigos do notebook elaborado possuem comentários como o trecho acima, de modo a explicar o uso de todas as classes, objetos, funções e métodos.

A base de dados utilizada está em formato csv (*Comma Separated Values*) que é o formato de importação e exportação mais comum para planilhas e bancos de dados. Basicamente é um tipo de arquivo de texto simples que armazena dados tabulares, em que cada linha representa um registro e os valores são separados por vírgulas (ou outro caractere delimitador).

Os dados utilizados neste projeto foram originalmente disponibilizados na plataforma de ensino da IBM *Developer Business Analytics*, mas foram acessados através do *Kaggle*. (Disponível em: https://www.kaggle.com/datasets/yeanzc/telco-customer-churn-ibm-dataset. Acesso em 10 de maio de 2023). Apesar de não haver informações explícitas disponíveis, os nomes das colunas permitem um entendimento a respeito do problema, conforme observado mais adiante na etapa de análise exploratória. Conforme consta na Figura 3, utilizamos o código

"pd.read_csv" para ler e carregar um arquivo no *format* CSV (*Comma-Separated Values*) através da biblioteca “Pandas”. O código "df.head( )" em Python é usado para exibir as primeiras cinco linhas de um *DataFrame*, por padrão, mostrando uma amostra dos dados contidos no *DataFrame*. É bastante útil para visualizar rapidamente a estrutura e o conteúdo dos dados carregados. Outro detalhe é que utilizamos também o código "pd.set_option('display.max_columns', None)" para definir uma opção de exibição no pandas. Nesse caso específico, está configurando a opção "display.max_columns" para que todas as colunas de um *DataFrame* sejam exibidas quando ele for mostrado no output. O valor "None" indica que não há limite para o número de colunas exibidas. Isso é útil quando se lida com *DataFrames* que possuem muitas colunas e se deseja visualizar todas elas em vez de uma versão truncada.

Figura 3 - Realizando a leitura do DataFrame importado

```
In [3]: # importando os dados
data_path = "https://raw.githubusercontent.com/carlosfab/dsnp2/master/datasets/WA_Fn-UseC_-Telco-Customer-Churn.csv"

df = pd.read_csv(data_path)

# utilizando o pd.set_option para mostrar todas as colunas
pd.set_option('display.max_columns', None)
```

```
In [4]: # verificando os primeiros registros
df.head()
```

Out[4]:

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	OnlineBackup	DeviceProtectior
0	7590-VHVEG	Female	0	Yes	No	1	No	No phone service	DSL	No	Yes	No
1	5575-GNVDE	Male	0	No	No	34	Yes	No	DSL	Yes	No	Yes
2	3668-QPYBK	Male	0	No	No	2	Yes	No	DSL	Yes	Yes	No
3	7795-CFOCW	Male	0	No	No	45	No	No phone service	DSL	Yes	No	Yes
4	9237-HQITU	Female	0	No	No	2	Yes	No	Fiber optic	No	No	No

Fonte: (Autor)

Após importarmos o nosso dataset, conseguimos ter uma breve visualização dos dados contidos em nosso conjunto. No entanto nossa visualização mostra apenas os primeiros registros do *dataframe*. Iremos utilizar o código “print(f'Nome das variáveis do dataset:{df.columns.values}') para mostrar e armazenar em uma lista o nome de cada uma das variáveis, que são as colunas do nosso dataframe. Para explorarmos um pouco mais os dados contidos no *dataframe* utilizamos um código que itera por meio de um loop e exibe as informações únicas presentes em cada coluna utilizando a função "print". Essa abordagem é utilizada como uma técnica exploratória para entender a natureza dos dados e identificar os diferentes valores únicos em cada coluna. Ao exibir os valores únicos, é possível obter insights sobre a

distribuição dos dados, identificar possíveis erros ou discrepâncias nos dados, e ter uma visão geral das características presentes em cada coluna. Utilizamos também o código “df.info( )”, para verificamos o tipo de cada variável. Os códigos utilizados estão na Figura 4, Figura 5 e Figura 6.

Figura 4 - Verificando nome das variáveis do dataset

```
# verificando o nome das variáveis do dataset
print(f'Nome das variáveis do dataset:{df.columns.values}')
```

```
Nome das variáveis do dataset:['customerID' 'gender' 'SeniorCitizen' 'Partner' 'Dependents' 'tenure'
 'PhoneService' 'MultipleLines' 'InternetService' 'OnlineSecurity'
 'OnlineBackup' 'DeviceProtection' 'TechSupport' 'StreamingTV'
 'StreamingMovies' 'Contract' 'PaperlessBilling' 'PaymentMethod'
 'MonthlyCharges' 'TotalCharges' 'Churn']
```

Fonte: (Autor)

Figura 5 - Verificando os dados únicos por coluna do dataset

```
# verificando os dados por coluna para checar distribuição
for column in df.columns:
  print(f'Coluna {column}: {df[column].unique()}')
  print('------------'*10)
```

```
Coluna customerID: ['7590-VHVEG' '5575-GNVDE' '3668-QPYBK' ... '4801-JZAZL' '8361-LTMKD'
 '3186-AJIEK']
------------------------------------------------------------------------------------------------------------------------
Coluna gender: ['Female' 'Male']
------------------------------------------------------------------------------------------------------------------------
Coluna SeniorCitizen: [0 1]
------------------------------------------------------------------------------------------------------------------------
Coluna Partner: ['Yes' 'No']
------------------------------------------------------------------------------------------------------------------------
Coluna Dependents: ['No' 'Yes']
------------------------------------------------------------------------------------------------------------------------
Coluna tenure: [ 1 34  2 45  8 22 10 28 62 13 16 58 49 25 69 52 71 21 12 30 47 72 17 27
  5 46 11 70 63 43 15 60 18 66  9  3 31 50 64 56  7 42 35 48 29 65 38 68
 32 55 37 36 41  6  4 33 67 23 57 61 14 20 53 40 59 24 44 19 54 51 26  0
 39]
------------------------------------------------------------------------------------------------------------------------
Coluna PhoneService: ['No' 'Yes']
------------------------------------------------------------------------------------------------------------------------
Coluna MultipleLines: ['No phone service' 'No' 'Yes']
------------------------------------------------------------------------------------------------------------------------
Coluna InternetService: ['DSL' 'Fiber optic' 'No']
------------------------------------------------------------------------------------------------------------------------
```

```
Coluna OnlineSecurity: ['No' 'Yes' 'No internet service']
--------------------------------------------------------------------------------------------------------------------------
Coluna OnlineBackup: ['Yes' 'No' 'No internet service']
--------------------------------------------------------------------------------------------------------------------------
Coluna DeviceProtection: ['No' 'Yes' 'No internet service']
--------------------------------------------------------------------------------------------------------------------------
Coluna TechSupport: ['No' 'Yes' 'No internet service']
--------------------------------------------------------------------------------------------------------------------------
Coluna StreamingTV: ['No' 'Yes' 'No internet service']
--------------------------------------------------------------------------------------------------------------------------
Coluna StreamingMovies: ['No' 'Yes' 'No internet service']
--------------------------------------------------------------------------------------------------------------------------
Coluna Contract: ['Month-to-month' 'One year' 'Two year']
--------------------------------------------------------------------------------------------------------------------------
Coluna PaperlessBilling: ['Yes' 'No']
--------------------------------------------------------------------------------------------------------------------------
Coluna PaymentMethod: ['Electronic check' 'Mailed check' 'Bank transfer (automatic)'
 'Credit card (automatic)']
--------------------------------------------------------------------------------------------------------------------------
Coluna MonthlyCharges: [29.85 56.95 53.85 ... 63.1  44.2  78.7 ]
--------------------------------------------------------------------------------------------------------------------------
Coluna TotalCharges: ['29.85' '1889.5' '108.15' ... '346.45' '306.6' '6844.5']
--------------------------------------------------------------------------------------------------------------------------
Coluna Churn: ['No' 'Yes']
--------------------------------------------------------------------------------------------------------------------------
```

Fonte: (Autor)

Figura 6 - Verificando os tipos de cada variável do dataset

```
# verificando os tipos de cada variável do dataset
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7043 entries, 0 to 7042
Data columns (total 21 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   customerID        7043 non-null   object 
 1   gender            7043 non-null   object 
 2   SeniorCitizen     7043 non-null   int64  
 3   Partner           7043 non-null   object 
 4   Dependents        7043 non-null   object 
 5   tenure            7043 non-null   int64  
 6   PhoneService      7043 non-null   object 
 7   MultipleLines     7043 non-null   object 
 8   InternetService   7043 non-null   object 
 9   OnlineSecurity    7043 non-null   object 
 10  OnlineBackup      7043 non-null   object 
 11  DeviceProtection  7043 non-null   object 
 12  TechSupport       7043 non-null   object 
 13  StreamingTV       7043 non-null   object 
 14  StreamingMovies   7043 non-null   object 
 15  Contract          7043 non-null   object 
 16  PaperlessBilling  7043 non-null   object 
 17  PaymentMethod     7043 non-null   object 
 18  MonthlyCharges    7043 non-null   float64
 19  TotalCharges      7043 non-null   object 
 20  Churn             7043 non-null   object 
dtypes: float64(1), int64(2), object(18)
memory usage: 1.1+ MB
```

Fonte: (Autor)

Após estas etapas, já é possível uma análise mais detalhada das variáveis do nosso dataset. Documentamos na Tabela 4 o nome, descrição de cada campo/coluna e o tipo de cada dado que estamos utilizando. Sobre os tipos, identificamos em nosso dataset três deles, que são:

- *Object:* O tipo "object" em Pandas é usado para representar colunas de dados que são heterogêneas, ou seja, podem conter diferentes tipos de dados, como *strings*, números e objetos Python. Em nosso dataset as colunas do tipo object apresentam dados que são *strings*, uma sequência de caracteres usada para representar texto ou palavras dentro de um programa.

- *Int64:* O tipo "int64", em Pandas é usado para armazenar colunas de dados numéricos inteiros de 64 bits com sinal. É útil para representar valores inteiros grandes e permite realizar operações matemáticas e manipulações de dados com eficiência e precisão.

- *Float64:* o tipo "float64" em Pandas é usado para representar colunas de dados numéricos de ponto flutuante de 64 bits. É adequado para armazenar valores decimais e permite cálculos matemáticos precisos, embora seja necessário ter cuidado com possíveis erros de arredondamento.

Tabela 4 - Descrição das colunas e tipos de dados do dataset

Nome da coluna/campo	**Descrição**	**Tipo**
customerID	Código de identificação do cliente	*object*
Gender	Identidade de gênero do cliente	*object*
SeniorCitizen	Identificação de cliente idoso (*0 No/1 Yes*)	*int64*
Partner	O cliente possui parceiro(a)? (*Yes/No*)	*object*
Dependents	O cliente possui dependentes? (*Yes/No*)	*object*
tenure	Número de meses que o cliente permaneceu na empresa	*int64*
PhoneService	O cliente possui serviço de telefonia contratado? (*Yes/No*)	*object*
MultipleLines	O cliente possui múltiplas linhas telefônicas? (*Yes/No/No phone services*)	*object*
InternetService	Tipo de serviço de internet contratado (*DSL, Fiber optic, No*)	*object*
OnlineSecurity	O cliente possui serviço de segurança online contratado? (*Yes, No, No internet service*)	*object*

OnlineBackup	O cliente possui serviço de backup online contratado? (*Yes, No, No internet service*)	*object*
DeviceProtection	O cliente possui plano de proteção de dispositivo? (*Yes, No, No internet service*)	*object*
TechSupport	O cliente possui serviço de suporte tecnológico contratado? (*Yes, No, No internet* service)	*object*
StreamingTV	O cliente possui serviço de streaming de TV contratado? (*Yes, No, No internet service*)	*object*
StreamingMovies	O cliente possui serviço de streaming de filmes contratado? (*Yes, No, No internet service*)	*object*
Contract	Tipo de contrato do cliente (*Month to month, One Year, Two year*)	*object*
PaperlessBilling	O cliente recebe suas contas apenas eletronicamente, sem envio de correspondência física? (*Yes, No*)	*object*
PaymentMethod	Forma de pagamento (*Electronic check, Mailed check, Bank transfer (automatic), Credit card (automatic)*)	*object*
MonthlyCharges	Valor cobrado mensalmente do cliente	*float64*
TotalCharges	Valor total cobrado do cliente	*object*
Churn	Cliente identificado com métrica de Churn? (*Yes/No*)	*object*

Fonte: (Autor)

Com os dados importados para dentro de uma estrutura Dataframe, e após conhecermos melhor as variáveis do nosso dataset podemos realizar a etapa posterior, de processamento e tratamento dos dados, a fim de obter algum *insight* ou informação relevante que possa influenciar em nossos modelos de Machine Learning.

3. Processamento/Tratamento de Dados

Após importarmos e termos um primeiro entendimento dos nossos dados, vamos para a fase de processamento e tratamento desses dados. Esta etapa consiste em coletar, organizar e documentar todos os dados que se encontram disponíveis. Precisamos identificar os dados importantes para a resolução do problema, analisar a qualidade dos dados, realizar análise descritiva e buscar identificar padrões em nossos procedimentos.

É nesta etapa que visamos compreender a estrutura e a natureza de um conjunto de dados, identificando padrões, relações, outliers, entre outros aspectos. Dando início as nossas análises, primeiro verificamos o total de linhas e total de colunas que nosso dataset possui. Também vamos verificar a quantidade de dados únicos por coluna que ele apresenta, através do método “df.nunique( )”. A importância desse código reside na sua capacidade de fornecer informações sobre a diversidade dos dados em cada coluna. Ao aplicarmos o “df.nunique( )”, nós conseguimos obter a contagem dos valores únicos em cada coluna individualmente. Isso pode ser útil para entender a variabilidade dos dados, identificar a presença de valores duplicados ou inconsistentes, além de destacar possíveis problemas de qualidade dos dados. Também vamos verificar novamente os tipos de cada dado do nosso dataframe. (Figura 7 e Figura 8).

Figura 7 - Verificando total de linhas e colunas e dados únicos do dataframe

```
In [ ]: # verificando as dimensões do dataset
        print(f'Total de linhas:\t{df.shape[0]}')
        print(f'Total de colunas:\t{df.shape[1]}')

        Total de linhas:        7043
        Total de colunas:       21

In [ ]: # verificando quantidade de dados únicos por feature
        df.nunique()

Out[7]: customerID          7043
        gender                 2
        SeniorCitizen          2
        Partner                2
        Dependents             2
        tenure                73
        PhoneService           2
        MultipleLines          3
        InternetService        3
        OnlineSecurity         3
        OnlineBackup           3
        DeviceProtection       3
        TechSupport            3
        StreamingTV            3
        StreamingMovies        3
        Contract               3
        PaperlessBilling       2
        PaymentMethod          4
        MonthlyCharges      1585
        TotalCharges        6531
        Churn                  2
        dtype: int64
```

Fonte: (Autor)

Figura 8 - Verificando os tipos de cada variável

```
In [ ]: # verificando os tipos de cada variável do dataset
        df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7043 entries, 0 to 7042
Data columns (total 21 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   customerID        7043 non-null   object
 1   gender            7043 non-null   object
 2   SeniorCitizen     7043 non-null   int64
 3   Partner           7043 non-null   object
 4   Dependents        7043 non-null   object
 5   tenure            7043 non-null   int64
 6   PhoneService      7043 non-null   object
 7   MultipleLines     7043 non-null   object
 8   InternetService   7043 non-null   object
 9   OnlineSecurity    7043 non-null   object
 10  OnlineBackup      7043 non-null   object
 11  DeviceProtection  7043 non-null   object
 12  TechSupport       7043 non-null   object
 13  StreamingTV       7043 non-null   object
 14  StreamingMovies   7043 non-null   object
 15  Contract          7043 non-null   object
 16  PaperlessBilling  7043 non-null   object
 17  PaymentMethod     7043 non-null   object
 18  MonthlyCharges    7043 non-null   float64
 19  TotalCharges      7043 non-null   object
 20  Churn             7043 non-null   object
dtypes: float64(1), int64(2), object(18)
memory usage: 1.1+ MB
```

Fonte: (Autor)

Conforme análises prévias é possível observar que: nosso *dataset* é composto por um total de 7.043 linhas e 21 colunas (*features*); A maioria dos nossos dados são textuais, do tipo *object*, possuindo de 2 a 4 dados únicos por *feature*; observamos que a *feature* Total Charges, embora seja relacionada a valores numéricos, também está representada como tipo *object*, o que nos demandará uma conversão para o tipo numérico adequado; a feature customerID se refere ao código de identificação de cada cliente, o que não é uma informação relevante em nossas análises, podendo, portanto, ser descartada para o nosso modelo.

Na Figura 9 mostramos os códigos utilizados para realizar as primeiras adequações em nosso dataset. Basicamente, criamos uma nova cópia do dataframe, onde convertemos a coluna Total Charges para o tipo numérico e removemos a feature relacionada ao código do cliente. Toda essa adequação é feita em uma cópia do dataframe, de modo a preservar os dados do dataset original. Importante destacar que utilizamos o parâmetro “inplace=True” que indica que a modificação deve ser feita diretamente na cópia do nosso dataframe criado. Se “inplace=False” (valor padrão), o método “drop( )” retornaria um novo DataFrame com a coluna removida, mantendo o DataFrame original inalterado. Já dentro do código que utilizamos para converter os valores da coluna “TotalCharges” do *DataFrame* df_clean para o tipo numérico, destacamos o parâmetro “erros=’coerce’”, que é um parâmetro opcional que

indica como tratar erros durante a conversão. No caso de 'coerce', qualquer valor que não possa ser convertido em numérico será definido como NaN (*Not a Number*).

Figura 9 - Primeiras adequações no dataset

```
In [ ]: # criando uma cópia do dataframe
        df_clean = df.copy()

        # drop da coluna customer ID
        df_clean.drop('customerID', axis=1, inplace=True)

        # convertendo string TotalCharges para float
        df_clean['TotalCharges'] = pd.to_numeric(df_clean['TotalCharges'], errors='coerce')
```

Fonte: (Autor)

Após realizadas as adequações iniciais identificadas, vamos verificar se existem dados nulos/vazios em alguma dessas features, vide Figura 10. Esta é uma etapa muito importante que nos permite verificar rapidamente quais colunas contêm valores ausentes, o que é crucial para a limpeza e preparação dos dados. Além disso, a contagem de valores nulos pode ser usada para avaliar a integridade dos dados. Se uma coluna tiver um grande número de valores nulos, isso pode indicar problemas de captura ou perda de informações. Por fim, também pode nos auxiliar a decidir como lidar com esses dados nulos, seja removendo as linhas ou colunas com valores nulos, preenchendo-os com valores adequados ou aplicando outras técnicas de tratamento de dados.

Figura 10 - Verificando dados nulos e vazios no dataset

```
In [ ]: # verificando se o dataset possui dados nulos/vazios
        df_clean.isnull().sum()

Out[10]: gender             0
         SeniorCitizen      0
         Partner            0
         Dependents         0
         tenure             0
         PhoneService       0
         MultipleLines      0
         InternetService    0
         OnlineSecurity     0
         OnlineBackup       0
         DeviceProtection   0
         TechSupport        0
         StreamingTV        0
         StreamingMovies    0
         Contract           0
         PaperlessBilling   0
         PaymentMethod      0
         MonthlyCharges     0
         TotalCharges      11
         Churn              0
         dtype: int64

In [ ]: # verificando o percentual dos dados nulos/vazios em relação ao total do dataset
        df_clean.isnull().sum() * 100 / len(df_clean)

Out[11]: gender            0.000000
         SeniorCitizen     0.000000
         Partner           0.000000
         Dependents        0.000000
         tenure            0.000000
         PhoneService      0.000000
         MultipleLines     0.000000
         InternetService   0.000000
         OnlineSecurity    0.000000
         OnlineBackup      0.000000
         DeviceProtection  0.000000
         TechSupport       0.000000
         StreamingTV       0.000000
         StreamingMovies   0.000000
         Contract          0.000000
         PaperlessBilling  0.000000
         PaymentMethod     0.000000
         MonthlyCharges    0.000000
         TotalCharges      0.156183
         Churn             0.000000
         dtype: float64
```

Fonte: (Autor)

Ao realizarmos a análise detalhada do conjunto de dados, constatamos que há a presença de valores nulos ou vazios exclusivamente na variável denominada TotalCharges. Essa inconsistência é observada em um total de 11 linhas, correspondendo a uma porcentagem mínima de apenas 0,15% em relação ao total de registros do dataset. Com o intuito de oferecer maior transparência, apresentamos na Figura 11 os registros afetados por essa particularidade.

Figura 11 - Localizando os dados nulos do dataset

```
In [ ]: df_clean.loc[df.TotalCharges == " "]
```

Out[12]:

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	OnlineBackup	DeviceProtection	TechSu
488	Female	0	Yes	Yes	0	No	No phone service	DSL	Yes	No	Yes	
753	Male	0	No	Yes	0	Yes	No	No	No internet service	No internet service	No internet service	No in
936	Female	0	Yes	Yes	0	Yes	No	DSL	Yes	Yes	Yes	
1082	Male	0	Yes	Yes	0	Yes	Yes	No	No internet service	No internet service	No internet service	No in
1340	Female	0	Yes	Yes	0	No	No phone service	DSL	Yes	Yes	Yes	
3331	Male	0	Yes	Yes	0	Yes	No	No	No internet service	No internet service	No internet service	No in
3826	Male	0	Yes	Yes	0	Yes	Yes	No	No internet service	No internet service	No internet service	No in
4380	Female	0	Yes	Yes	0	Yes	No	No	No internet service	No internet service	No internet service	No in
5218	Male	0	Yes	Yes	0	Yes	No	No	No internet service	No internet service	No internet service	No in
6670	Female	0	Yes	Yes	0	Yes	Yes	DSL	No	Yes	Yes	
6754	Male	0	No	Yes	0	Yes	Yes	DSL	Yes	Yes	No	

```
In [ ]: df_clean.loc[df.TotalCharges == " "]
```

Out[12]:

OnlineBackup	DeviceProtection	TechSupport	StreamingTV	StreamingMovies	Contract	PaperlessBilling	PaymentMethod	MonthlyCharges	TotalCharges	Churn
No	Yes	Yes	Yes	No	Two year	Yes	Bank transfer (automatic)	52.55	NaN	No
No internet service	No internet service	No internet service	No internet service	No internet service	Two year	No	Mailed check	20.25	NaN	No
Yes	Yes	No	Yes	Yes	Two year	No	Mailed check	80.85	NaN	No
No internet service	No internet service	No internet service	No internet service	No internet service	Two year	No	Mailed check	25.75	NaN	No
Yes	Yes	Yes	Yes	No	Two year	No	Credit card (automatic)	56.05	NaN	No
No internet service	No internet service	No internet service	No internet service	No internet service	Two year	No	Mailed check	19.85	NaN	No
No internet service	No internet service	No internet service	No internet service	No internet service	Two year	No	Mailed check	25.35	NaN	No
No internet service	No internet service	No internet service	No internet service	No internet service	Two year	No	Mailed check	20.00	NaN	No
No internet service	No internet service	No internet service	No internet service	No internet service	One year	Yes	Mailed check	19.70	NaN	No
Yes	Yes	Yes	Yes	No	Two year	No	Mailed check	73.35	NaN	No
Yes	No	Yes	No	No	Two year	Yes	Bank transfer (automatic)	61.90	NaN	No

Fonte: (Autor)

Ao analisarmos os dados apresentados na Figura 11, pudemos constatar que nenhum dos registros pertencem a clientes identificados como potenciais cancelamentos, pois todos os valores da coluna Churn estão categorizados como "No". Levando em consideração essa constatação e também levando em conta a insignificante representatividade desses dados em nosso conjunto de dados, tomamos a decisão de excluí-los de nossa base de dados. Conforme Figura 12 realizamos a exclusão dos dados ausentes e fizemos uma nova checagem de valores nulos.

Figura 12 - Eliminando os dados ausentes do dataset

```
In [ ]: # eliminando as linhas com dados ausentes
        df_clean.drop(index=df.query('TotalCharges == " "').index, axis=0, inplace=True)

        # checando novamente os dados
        df_clean.isnull().sum()

Out[13]: gender            0
         SeniorCitizen     0
         Partner           0
         Dependents        0
         tenure            0
         PhoneService      0
         MultipleLines     0
         InternetService   0
         OnlineSecurity    0
         OnlineBackup      0
         DeviceProtection  0
         TechSupport       0
         StreamingTV       0
         StreamingMovies   0
         Contract          0
         PaperlessBilling  0
         PaymentMethod     0
         MonthlyCharges    0
         TotalCharges      0
         Churn             0
         dtype: int64
```

Fonte: (Autor)

Após realizarmos a exclusão dos dados ausentes de nossa base, e outras etapas de preparação e adequação dos nossos dados é chegada a hora de avançarmos para uma etapa de análise mais aprofundada das características presentes e sua relação com nosso objetivo do nosso trabalho.

4. Análise e Exploração dos Dados

Com o intuito de facilitar nossa análise exploratória dos dados e garantir uma compreensão mais abrangente das informações contidas, procederemos a uma nova verificação dos valores únicos em cada coluna. Essa verificação permitirá uma melhor compreensão da distribuição dos dados.

Ao realizar esse procedimento, poderemos explorar de forma mais precisa a diversidade e a natureza das features, identificando possíveis padrões, tendências ou peculiaridades que possam contribuir significativamente para o entendimento e a análise subsequente.

A análise exploratória em Machine Learning é uma etapa fundamental no processo de exploração e compreensão inicial dos dados. Ela envolve a aplicação de técnicas estatísticas e visualizações para identificar padrões, tendências, anomalias e relacionamentos nos dados, a fim de extrair insights valiosos.

Uma definição de análise exploratória em Machine Learning:

> *"Exploratory data analysis is an approach to analyzing data sets to summarize their main characteristics, often with visual methods. It aims to uncover patterns, relationships, and hidden insights in the data, facilitating the understanding and identification of important variables for subsequent modeling and analysis"* (Adaptado de: Tukey, J. W. (1977). Exploratory Data Analysis. Addison-Wesley).

Nesta definição, é enfatizado o uso de métodos visuais para resumir as principais características dos conjuntos de dados, com o objetivo de descobrir padrões, relacionamentos e insights ocultos. Isso auxilia na compreensão dos dados e na identificação das variáveis importantes para análises e modelagem subsequentes.

Dando início a nossa etapa de análise exploratória vamos verificar novamente os dados únicos para coluna, para melhor consciência situacional (Figura 13).

Figura 13 - Verificando os dados únicos por coluna para checar distribuição

```
In [ ]: # verificando os dados por coluna para checar distribuição
        for column in df_clean.columns:
          print(f'Coluna {column}: {df_clean[column].unique()}')
          print('------------'*10)
```

```
Coluna gender: ['Female' 'Male']
--------------------------------------------------------------------------------------------------------------------------------------------
Coluna SeniorCitizen: [0 1]
--------------------------------------------------------------------------------------------------------------------------------------------
Coluna Partner: ['Yes' 'No']
--------------------------------------------------------------------------------------------------------------------------------------------
Coluna Dependents: ['No' 'Yes']
--------------------------------------------------------------------------------------------------------------------------------------------
Coluna tenure: [ 1 34  2 45  8 22 10 28 62 13 16 58 49 25 69 52 71 21 12 30 47 72 17 27
  5 46 11 70 63 43 15 60 18 66  9  3 31 50 64 56  7 42 35 48 29 65 38 68
 32 55 37 36 41  6  4 33 67 23 57 61 14 20 53 40 59 24 44 19 54 51 26 39]
--------------------------------------------------------------------------------------------------------------------------------------------
Coluna PhoneService: ['No' 'Yes']
--------------------------------------------------------------------------------------------------------------------------------------------
Coluna MultipleLines: ['No phone service' 'No' 'Yes']
--------------------------------------------------------------------------------------------------------------------------------------------
Coluna InternetService: ['DSL' 'Fiber optic' 'No']
--------------------------------------------------------------------------------------------------------------------------------------------
Coluna OnlineSecurity: ['No' 'Yes' 'No internet service']
--------------------------------------------------------------------------------------------------------------------------------------------
Coluna OnlineBackup: ['Yes' 'No' 'No internet service']
--------------------------------------------------------------------------------------------------------------------------------------------
Coluna DeviceProtection: ['No' 'Yes' 'No internet service']
--------------------------------------------------------------------------------------------------------------------------------------------
Coluna TechSupport: ['No' 'Yes' 'No internet service']
--------------------------------------------------------------------------------------------------------------------------------------------
Coluna StreamingTV: ['No' 'Yes' 'No internet service']
--------------------------------------------------------------------------------------------------------------------------------------------
Coluna StreamingMovies: ['No' 'Yes' 'No internet service']
--------------------------------------------------------------------------------------------------------------------------------------------
Coluna Contract: ['Month-to-month' 'One year' 'Two year']
--------------------------------------------------------------------------------------------------------------------------------------------
Coluna PaperlessBilling: ['Yes' 'No']
--------------------------------------------------------------------------------------------------------------------------------------------
Coluna PaymentMethod: ['Electronic check' 'Mailed check' 'Bank transfer (automatic)'
 'Credit card (automatic)']
--------------------------------------------------------------------------------------------------------------------------------------------
Coluna MonthlyCharges: [29.85 56.95 53.85 ... 63.1  44.2  78.7 ]
--------------------------------------------------------------------------------------------------------------------------------------------
Coluna TotalCharges: [  29.85 1889.5   108.15 ...  346.45  306.6  6844.5 ]
--------------------------------------------------------------------------------------------------------------------------------------------
Coluna Churn: ['No' 'Yes']
--------------------------------------------------------------------------------------------------------------------------------------------
```

Fonte: (Autor)

Ao verificarmos as distribuições de dados por cada coluna, juntamente com as análises de tipos de cada um desses dados, é possível estruturarmos uma melhor forma de analisar cada uma das features e sua relação na identificação de potenciais cancelamentos dos serviços por parte dos clientes. Nossas descobertas revelaram que o conjunto de dados em questão apresenta informações que podem ser categorizadas, proporcionando uma orientação mais direcionada para nossas análises. Essas categorias são as seguintes:

Vetor alvo - Representado pela coluna "Churn", que identifica o potencial de cancelamento dos serviços pelos clientes.

Informações de valores: Representadas pelas colunas "MonthlyCharges", "TotalCharges" e "tenure". Essas colunas refletem os valores mensais pagos pelo cliente, os valores totais pagos até o momento e a quantidade de meses em que o cliente permaneceu na empresa.

Informações dos clientes: Englobam características específicas dos clientes, como gênero ("*gender*"), se são idosos ("*SeniorCitizen*"), se possuem parceira ("*Partner*") e se têm dependentes ("*Dependents*").

Informações dos serviços contratados: Referem-se aos serviços contratados por cada cliente e estão contidas nas colunas "*PhoneService", "MultipleLines", "InternetService", "OnlineSecurity", "OnlineBackup", "DeviceProtection", "TechSupport", "StreamingTV" e "StreamingMovies".*

Informações dos pagamentos: Dizem respeito ao tipo de contrato, opção de faturamento eletrônico *("PaperlessBilling"*) e método de pagamento *("PaymentMethod")* escolhidos por cada cliente.

Ao categorizarmos nossas características dessa maneira, torna-se mais viável realizar uma análise mais aprofundada de cada uma delas. Essa abordagem nos permite realizar uma análise exploratória com maior consciência situacional, resultando em um entendimento mais claro do problema em questão. Agora, direcionaremos nossa atenção para uma análise detalhada de cada uma dessas características, avaliando sua relação com nosso vetor alvo, que consiste na identificação da previsão de *churn*.

Para as análises da categoria **Vetor alvo,** inicialmente, verificamos a divisão de nosso *dataset* e balanceamento das classes considerando a coluna "*Churn*" (Figura 14). Destacamos em nosso código a função criada para marcar a porcentagem no *plot* de nossos gráficos.

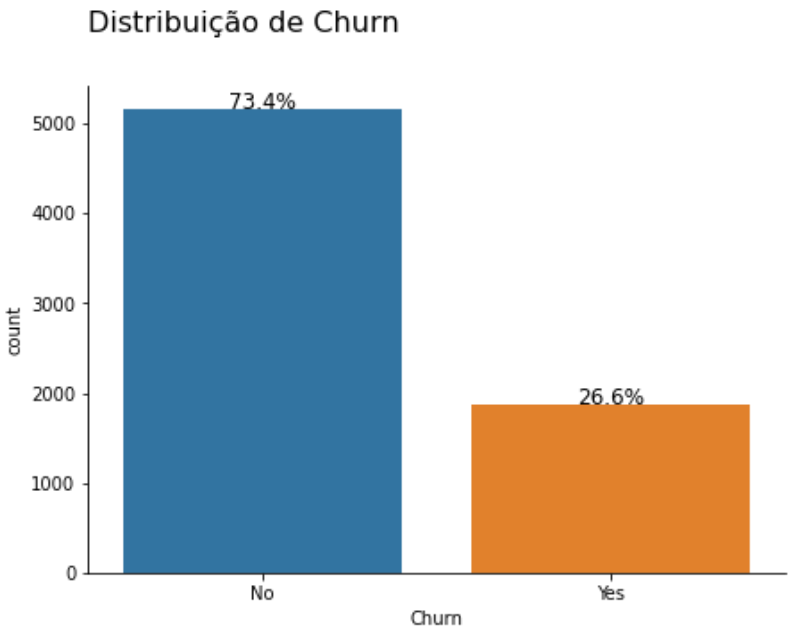
Por ser uma função ela pode ser aplicada a qualquer outro gráfico, desde que passada respeitando o parâmetro correto. O gráfico da Figura 15 nos ilustra o balanceamento das classes do nosso vetor alvo.

Figura 14 - Código para verificar distribuição das classes

```
In [ ]: df_clean.Churn.value_counts() * 100 / len(df_clean)

Out[15]: No     73.421502
         Yes    26.578498
         Name: Churn, dtype: float64

In [ ]: # Função auxiliar para marcar porcentagem nos plots
        def porcentagem(ax, dados):
          total = float(len(dados))
          for p in ax.patches:
              percentage = f'{(p.get_height()/total)*100:.1f}%'
              x = p.get_x() + p.get_width()/2
              y = p.get_height()
              ax.annotate(percentage, (x, y), fontsize=12, horizontalalignment='center')

In [ ]: # gráfico de distribuição de churn
        fig, ax = plt.subplots(figsize=(7,5))

        sns.countplot(x='Churn', data=df_clean, ax=ax)
        ax.set_title('Distribuição de Churn', loc='left', fontsize=16, pad=30)
        ax.spines['top'].set_visible(False)
        ax.spines['right'].set_visible(False)
        ax.grid(False)

        porcentagem(ax, df_clean)
        plt.show();
```

Fonte: (Autor)

Figura 15 - Gráfico de distribuição de classes



Fonte: (Autor)

Na análise da variável *Churn*, observa-se uma discrepância significativa entre as classes *No e Yes*, evidenciando um desbalanceamento acentuado no conjunto de dados. Essa disparidade será corrigida durante a etapa de construção do modelo de aprendizado de máquina, a

fim de garantir uma representação mais equilibrada das classes e evitar possíveis vieses e distorções nos resultados.

Ao examinarmos as variáveis classificadas como **informações de valores**, uma análise inicialmente relevante a ser realizada para as características que possuem informações numéricas é a utilização do método "describe", a fim de obter as principais medidas estatísticas desses números. O método describe é uma função fornecida pela biblioteca Pandas em Python. Ele é aplicado a objetos DataFrame ou Series e fornece um resumo estatístico das variáveis numéricas presentes nos dados. As estatísticas descritivas incluem aquelas que resumem a tendência central, a dispersão e a forma da distribuição de um conjunto de dados. Na Figura 16 documentamos a utilização do método “describe” em nossas variáveis.

Figura 16 - Aplicação do método describe

```
In [ ]: df_clean[['MonthlyCharges', 'TotalCharges', 'tenure']].describe()
```

Out[18]:

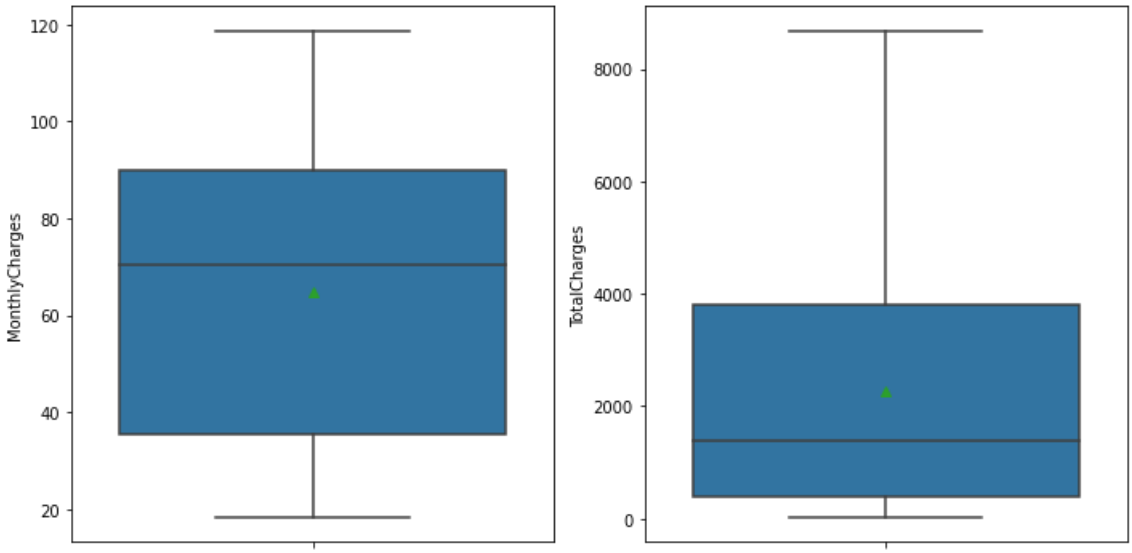
	MonthlyCharges	TotalCharges	tenure
count	7032.000000	7032.000000	7032.000000
mean	64.798208	2283.300441	32.421786
std	30.085974	2266.771362	24.545260
min	18.250000	18.800000	1.000000
25%	35.587500	401.450000	9.000000
50%	70.350000	1397.475000	29.000000
75%	89.862500	3794.737500	55.000000
max	118.750000	8684.800000	72.000000

Fonte: (Autor)

Ao analisar a tabela apresentada, observamos que não há evidências de valores discrepantes nas características em questão. O desvio padrão de cada uma delas, representado na linha std, não é significativamente elevado, o que é consistente com a média. Além disso, o valor máximo não difere consideravelmente do terceiro quartil. Essa constatação é reforçada pela representação gráfica das features “MonthlyCharges” e TotalCharges por meio de um *boxplot*, conforme consta na Figura 17. O *boxplot* ilustra a distribuição desses valores, evidenciando que não há pontos além dos limites superior e inferior estabelecidos. Essa visualização reforça a ausência de outliers nessas variáveis.

Dessa forma, os dados sugerem que não há valores atípicos nas features analisadas, o que indica uma consistência e coerência nos dados estatísticos. Essa constatação é relevante para o entendimento da distribuição e comportamento dessas variáveis em nosso estudo.

Figura 17 - Boxplot das variáveis "Monthly Charges" e "Total Charges"



Fonte: (Autor)

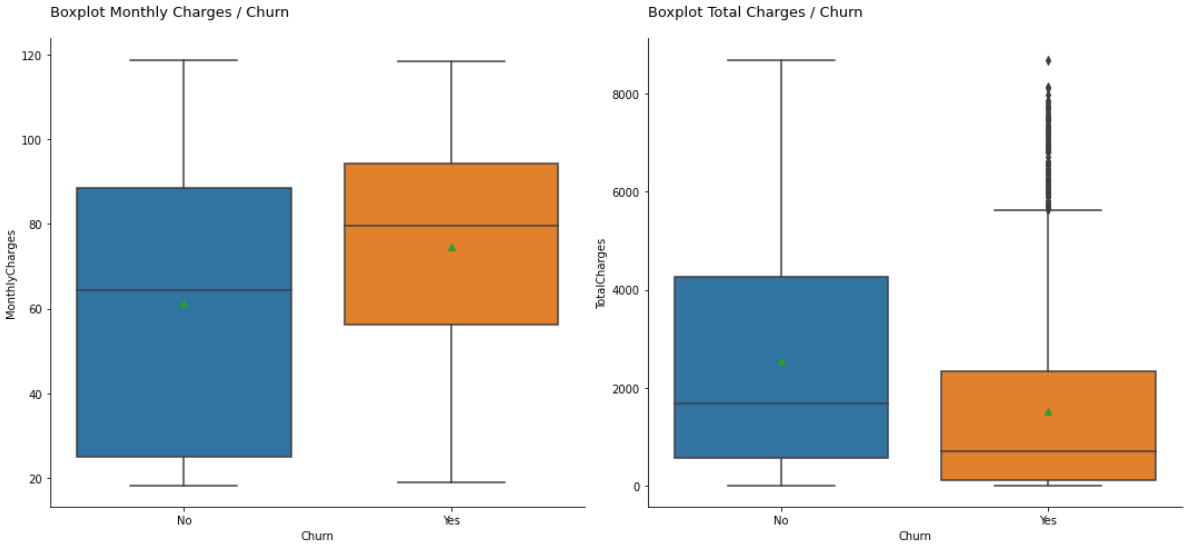
A seguir, procedemos à análise desses valores, considerando-os em relação à coluna de *churn*, com o objetivo de investigar possíveis relações ou correlações entre essas variáveis e a ocorrência de churn no contexto do nosso estudo. (Figura 18 e Figura 19).

Figura 18 - Código para plot de boxplot

```
In [ ]: fig, ax = plt.subplots(1,2, figsize=(15,7))

sns.boxplot(x='Churn', y='MonthlyCharges', data=df_clean, ax=ax[0], showmeans=True)
ax[0].set_title('Boxplot Monthly Charges / Churn', loc='left', fontsize=13, pad=20)
ax[0].spines['right'].set_visible(False)
ax[0].spines['top'].set_visible(False)
ax[0].grid(False)

sns.boxplot(x='Churn', y='TotalCharges', data=df_clean, ax=ax[1], showmeans=True)
ax[1].set_title('Boxplot Total Charges / Churn', loc='left', fontsize=13, pad=20)
ax[1].spines['right'].set_visible(False)
ax[1].spines['top'].set_visible(False)
ax[1].grid(False)

plt.tight_layout();
```

Fonte: (Autor)

Figura 19 - Boxplot considerando a variável *Churn*



Fonte: (Autor)

Ao analisar o gráfico da Figura 19, podemos observar inicialmente que a distribuição dos dados varia significativamente entre as categorias identificadas como Churn e não Churn. É evidente que isso ocorre devido à representatividade dos clientes com churn positivo, que compreendem apenas 26,5% do conjunto de dados. Essa diferença na distribuição reflete uma disparidade clara entre as duas categorias.

Outro aspecto relevante é a diferença na média dos valores mensais cobrados entre os casos identificados como *churn prediction*. Observamos que esses clientes apresentam uma média mais alta de cobranças mensais em comparação com aqueles que não foram identificados como *churn*. Essa diferença sugere que o preço elevado das cobranças pode ser um dos principais fatores que contribuem para o *churn*.

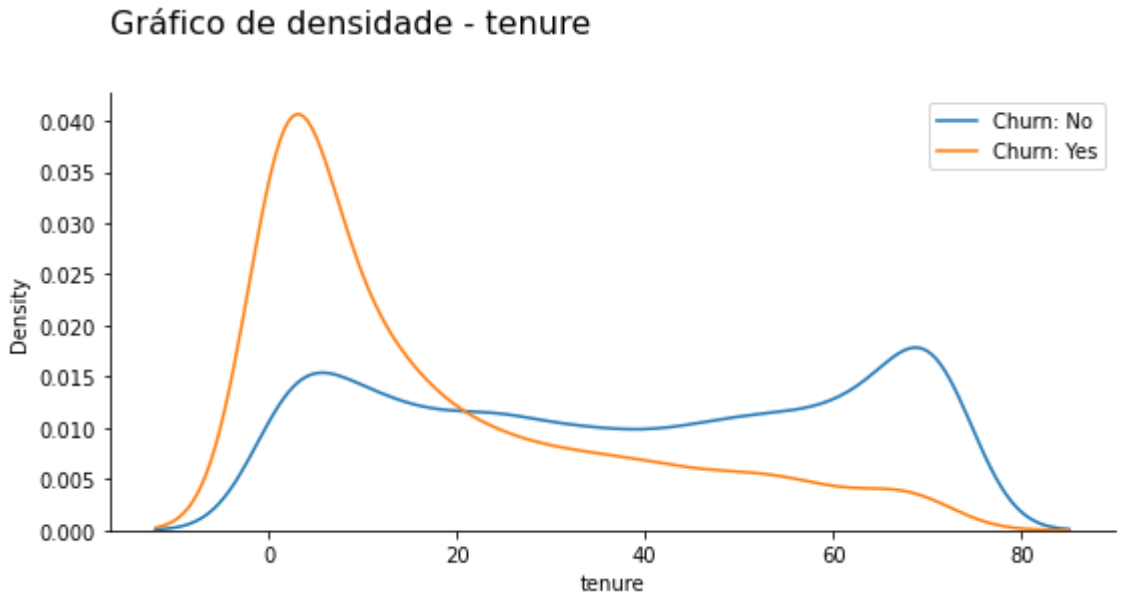
Além disso, realizamos uma análise de densidade das variáveis “TotalCharges”, “MonthlyCharges” e “tenure”, a fim de examinar a distribuição de cada uma delas em relação à previsão de *churn* (vide Figura 21, Figura 22 e Figura 23). Essa análise permite verificar como a densidade dessas variáveis está relacionada à previsão de *churn*, fornecendo insights adicionais sobre os padrões de comportamento dos clientes que apresentam maior propensão ao churn. O código da Figura 20 define uma função chamada *kdeplot* que gera um gráfico de densidade para uma determinada variável feature. O gráfico de densidade mostra a distribuição dos valores dessa variável ao longo de um eixo horizontal, indicando a probabilidade de encontrar determinados valores.

Figura 20 - Código para plot de gráfico de densidade

```
In [ ]: def kdeplot(feature, hist, kde):
            plt.figure(figsize=(9, 4))
            plt.title("Gráfico de densidade - {}".format(feature), loc='left', fontsize=16, pad=30)
            ax0 = sns.distplot(df_clean[df_clean['Churn'] == 'No'][feature].dropna(), hist=hist, kde=kde,
                     label= 'Churn: No')
            ax0.spines['right'].set_visible(False)
            ax0.spines['top'].set_visible(False)

            ax1 = sns.distplot(df_clean[df_clean['Churn'] == 'Yes'][feature].dropna(), hist=hist, kde=kde,
                     label= 'Churn: Yes')
            ax1.spines['right'].set_visible(False)
            ax1.spines['top'].set_visible(False)

            plt.legend()
```

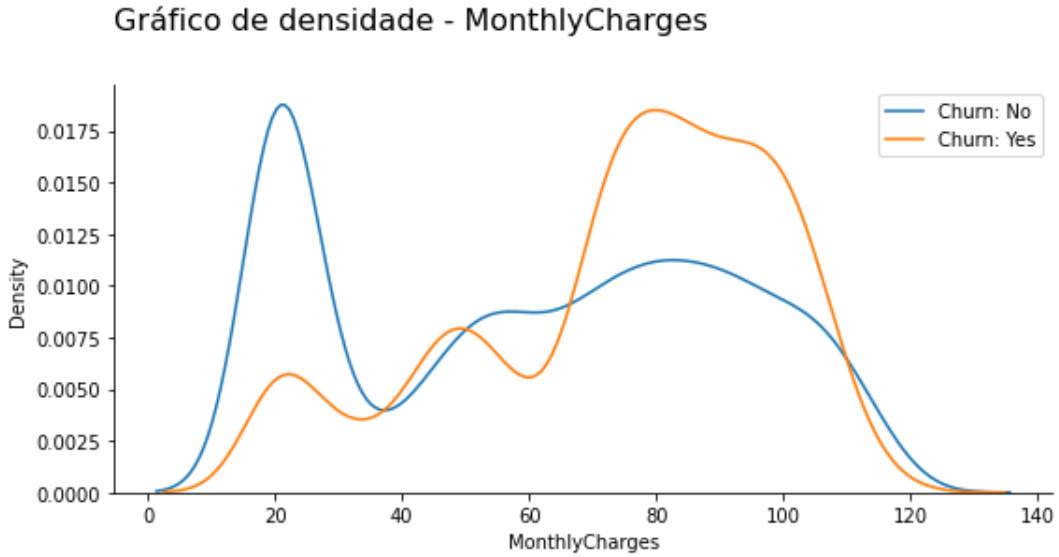
Fonte: (Autor)

Figura 21 - Gráfico de densidade - variável "tenure"



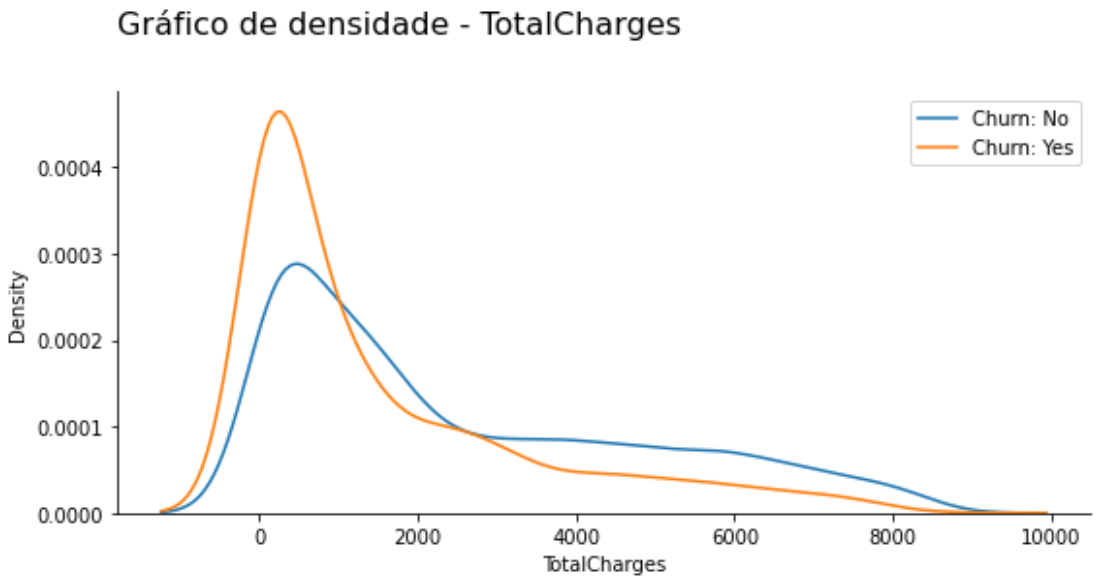
Fonte: (Autor)

Figura 22 - Gráfico de densidade - variável "MonthlyCharges"



Fonte: (Autor)

Figura 23 - Gráfico de densidade - variável "TotalCharges"

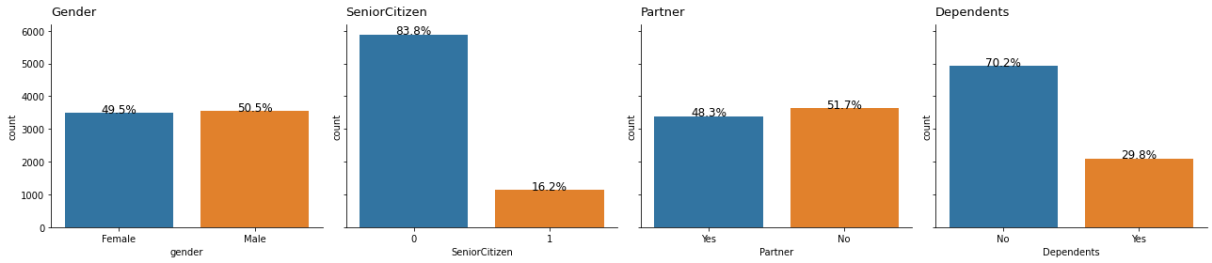


Fonte: (Autor)

A partir das análises dos gráficos acima, pode-se observar que o *churn* ocorre com maior frequência entre os clientes que possuem um tempo de contrato relativamente curto, geralmente entre 10 e 20 meses, enquanto é menos comum em períodos mais avançados do contrato. Além disso, constata-se que os clientes que possuem cobranças mensais mais elevadas apresentam uma probabilidade maior de *churn*. Quanto ao valor total das cobranças, verifica-se que ambos os casos exibem comportamentos semelhantes.

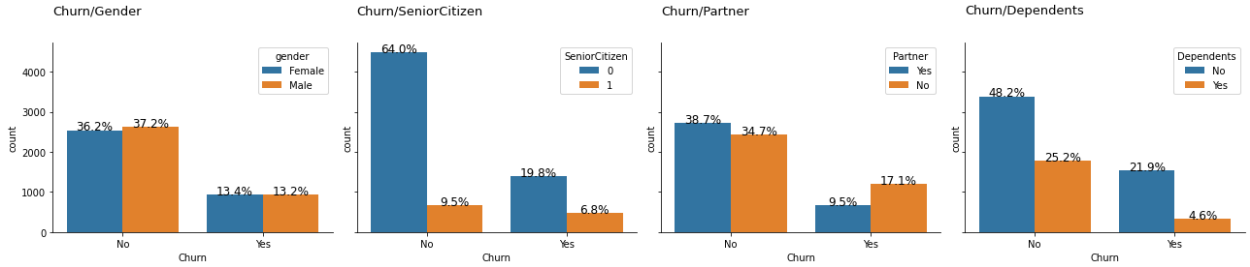
Ao examinarmos as variáveis classificadas como **informações dos clientes,** que são tipo *object*, também plotamos gráficos para verificar distribuição de cada uma delas e como elas se relacionam com o churn prediction. Nos primeiros gráficos, conforme consta na Figura 24 buscamos visualizar a distribuição dessas variáveis enquanto que na Figura 25, buscamos verificar a relação de cada uma delas com nosso vetor alvo de predição de churn. Com base nos gráficos apresentados, podemos observar que não há diferenças significativas em relação ao sexo dos clientes para determinar se eles cancelarão ou não o serviço. Além disso, constatamos que os clientes que não possuem parceiros ou dependentes têm uma taxa de churn mais elevada.

Figura 24 - Gráfico de distribuição de variáveis contendo informações dos clientes



Fonte: (Autor)

Figura 25 - Gráfico de distribuição contendo informações dos clientes e relação com o churn prediction

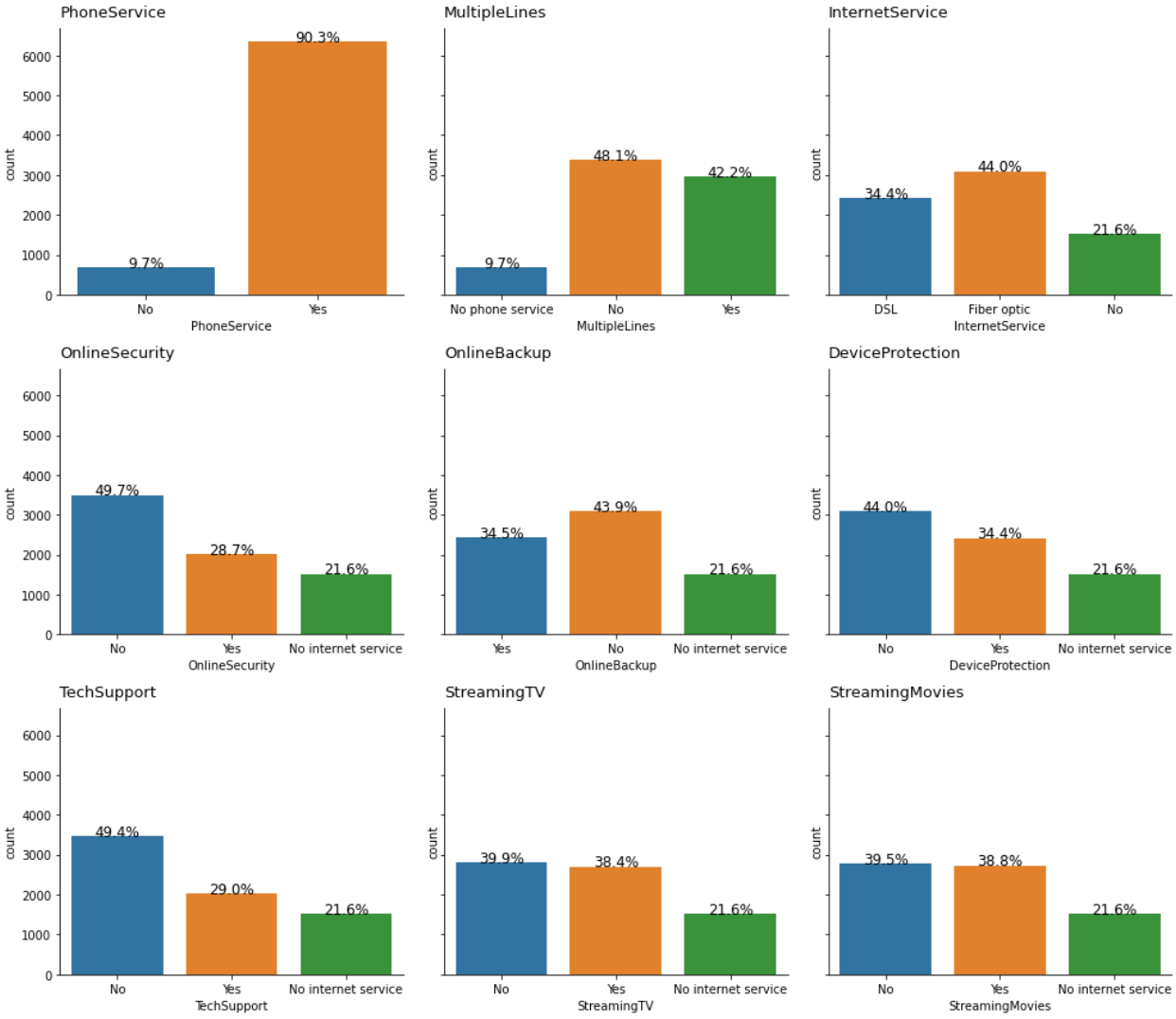


Fonte: (Autor)

Na Figura 26 e Figura 27 , procedemos com as análises das variáveis correlacionadas às **informações dos serviços contratados pelo cliente**, examinando a amplitude de

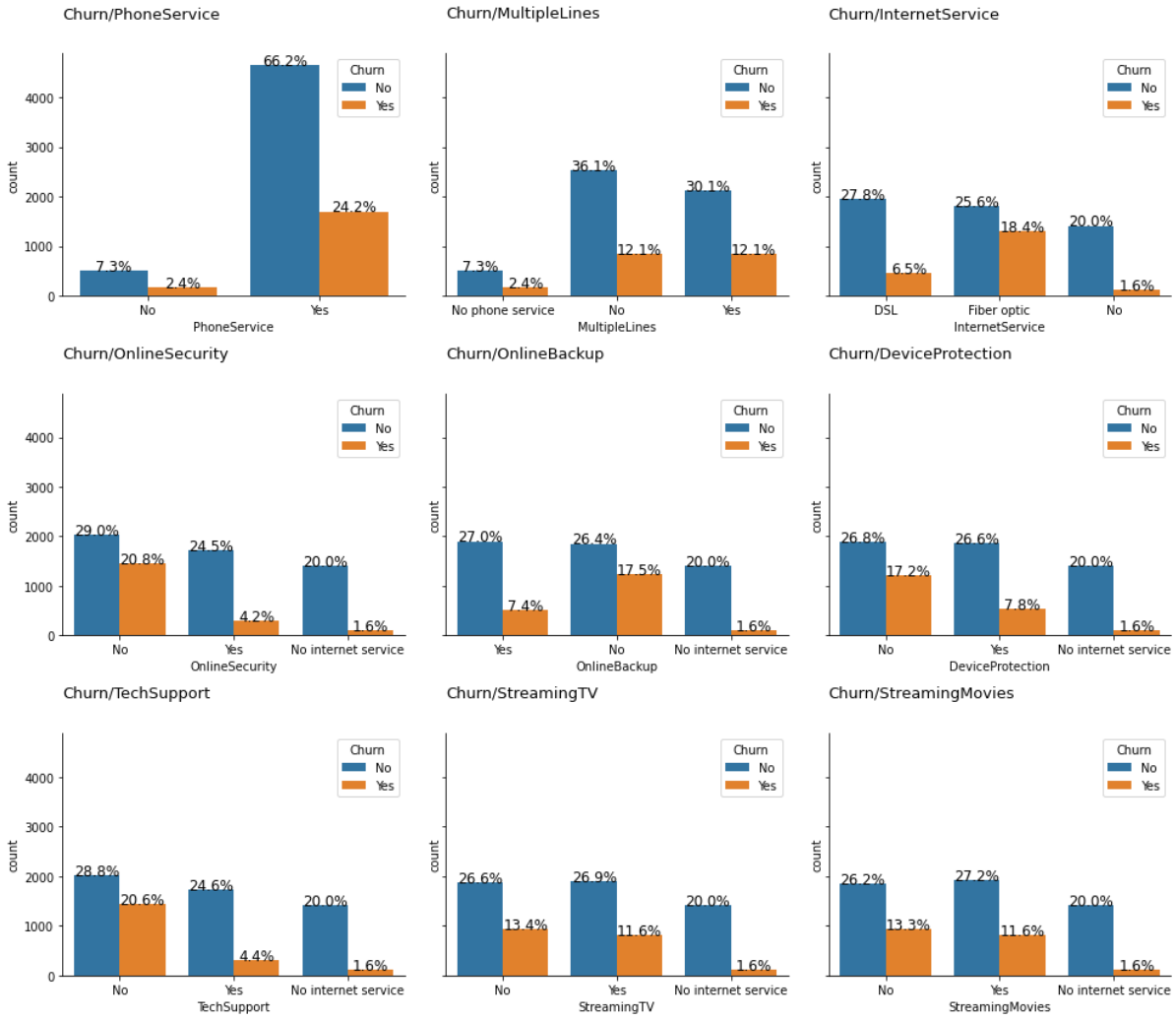
distribuição de cada uma dessas características e sua influência também nas previsões de evasão de nossos clientes, com base em nosso modelo de *churn prediction*.

Figura 26 - Gráfico de distribuição de variáveis contendo informações dos serviços contratados



Fonte: (Autor)

Figura 27 - Gráfico de distribuição contendo informações dos serviços contratados e relação com churn

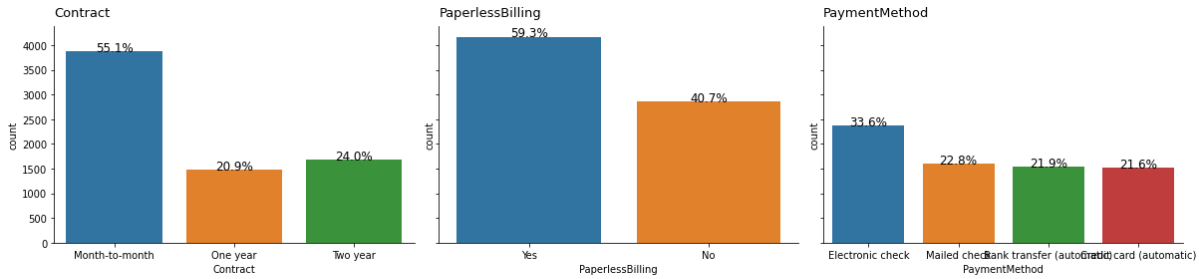


Fonte: (Autor)

Com base nas análises realizadas, observa-se que há uma redução significativa na taxa de churn entre os clientes que não possuem serviço telefônico, múltiplas linhas ou serviço de internet. Também se verifica que os clientes que utilizam o serviço de fibra óptica apresentam uma propensão maior a cancelar o plano com a operadora. A maioria dos clientes não faz uso de serviços adicionais de segurança e suporte, sendo que aqueles que usufruem desses serviços demonstram uma taxa de *churn* inferior. Observamos também que a taxa de churn é mais elevada entre os clientes que não possuem acesso a *streaming* de TV e filmes.

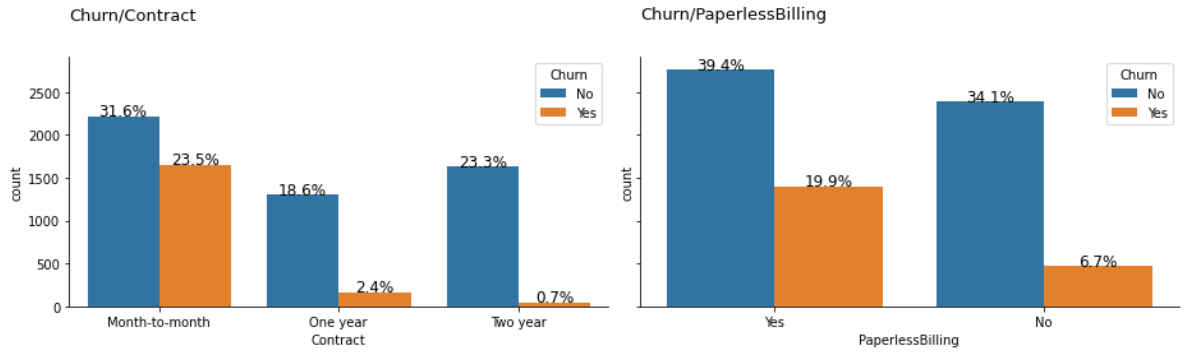
Por fim realizamos as análises das features relacionadas ao **tipo de contrato e formas de pagamento de cada cliente**, verificando a distribuição de cada uma delas e como elas se relacionam com o *churn prediction* (vide Figura 28 e Figura 29).

Figura 28 - Gráfico distribuição de tipo de contrato e forma de pagamento



Fonte: (Autor)

Figura 29 - Gráfico tipo de contrato e forma de pagamento

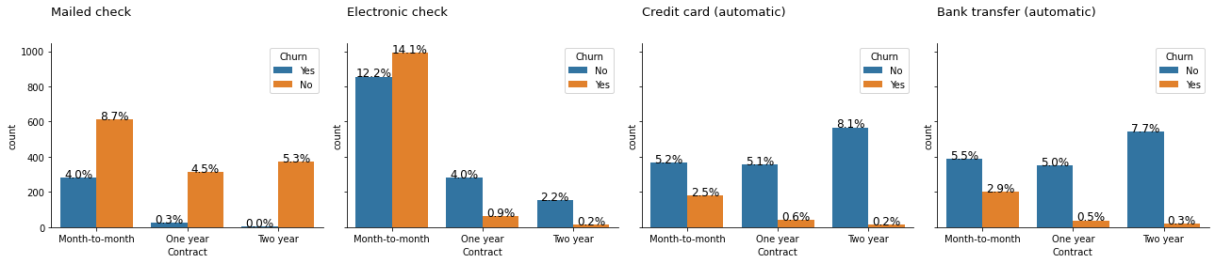


Fonte: (Autor)

No que se refere aos tipos de contrato, é possível constatar que a maioria dos clientes adota o contrato mensal, sendo esse grupo aquele que registra a maior taxa de *churn*. Por outro lado, a menor taxa de *churn* é observada nos contratos com duração de dois anos. Ademais, é importante ressaltar que a maioria dos clientes opta por receber a fatura de forma eletrônica, caracterizando essa categoria como aquela com maior probabilidade de churn.

Também podemos verificar na Figura 30 que o tipo de pagamento com maior taxa de *churn* é dos clientes que pagam por meio de cheque eletrônico. Observamos também que, dos tipos de contrato de modalidade mensal, o menor *churn* é para os clientes que realizam pagamento através de cartão de crédito.

Figura 30 - Gráfico de Churn por tipo de pagamento



Fonte: (Autor)

4.1. Pré-processamento das features binárias e dos dados categóricos

Nesta etapa de preparação dos dados, faremos um pré-processamento inicial, visando construir um modelo base. Para isso, aplicaremos técnicas de pré-processamento básicas. No caso das features binárias, que apresentam apenas dois valores únicos, utilizaremos o “LabelEncoder”, uma técnica utilizada em *machine* learning para converter dados categóricos em números, permitindo que algoritmos de aprendizado de máquina processem e trabalhem com esses dados. É importante ressaltar que a variável alvo "Churn" também será incluída nesse processo, garantindo uma análise abrangente.

Na sequência, procederemos com a separação das variáveis em duas categorias distintas: categóricas e numéricas (vide Figura 31). Essa distinção nos permitirá adotar abordagens específicas para cada tipo de variável. No caso das variáveis categóricas, empregaremos o método “getDummies”. Basicamente este método cria uma nova coluna binária para cada categoria. Cada coluna binária representa se uma determinada categoria está presente ou não para uma determinada instância de dados.

Figura 31 - Aplicando Label Encoder para colunas binárias e categóricas

In [ ]:
```
# separando as colunas por tipo de variável
binary_var = df_clean.nunique()[df_clean.nunique() == 2].keys().tolist()
num_var = [col for col in df_clean.select_dtypes(['int', 'float']).columns.tolist() if col not in binary_var]
cat_var = [col for col in df_clean.columns.tolist() if col not in binary_var + num_var]

# criando uma copia do dataframe
df_proc = df_clean.copy()


# Label Encoding para as variáveis binárias
le = LabelEncoder()
for coluna in binary_var:
  df_proc[coluna] = le.fit_transform(df_proc[coluna])

# Encoding com get_dummies para as colunas categóricas com múltiplas classes
df_proc = pd.get_dummies(df_proc, columns=cat_var)

# verificando as 5 primeiras entradas do novo dataframe
df_proc.head()
```

Out[32]:

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	PaperlessBilling	MonthlyCharges	TotalCharges	Churn	...	StreamingMovies_No	Streamin int
0	0	0	1	0	1	0	1	29.85	29.85	0	...	1	
1	1	0	0	0	34	1	0	56.95	1889.50	0	...	1	
2	1	0	0	0	2	1	1	53.85	108.15	1	...	1	
3	1	0	0	0	45	0	0	42.30	1840.75	0	...	1	
4	0	0	0	0	2	1	1	70.70	151.65	1	...	1	

5 rows × 41 columns

Fonte: (Autor)

5. Criação de Modelos de Machine Learning

O campo de Aprendizado de Máquina (AM), também conhecido como Machine Learning, abrange métodos computacionais que se dedicam à automática aquisição de conhecimento, desenvolvimento de novas habilidades e reorganização de informações existentes (Mitchell, 1997). Muller e Guido (2017) descrevem AM como um campo de pesquisa que se cruza com a Estatística, Inteligência Artificial e Ciência da Computação, sendo igualmente referido como Análise Preditiva ou Aprendizagem Estatística.

O aprendizado de máquina (machine learning) pode ser classificado em três tipos: a) aprendizado supervisionado, que busca encontrar uma função a partir de dados rotulados de entrada e saída, com o objetivo de prever uma variável desejada; b) aprendizado não supervisionado, que visa explorar ou descrever um conjunto de dados sem o uso de atributos de saída, com o intuito de identificar padrões desconhecidos; e c) aprendizado por reforço, onde a interação e o feedback desempenham um papel crucial no desenvolvimento do processo. O presente trabalho está focado no aprendizado supervisionado, mais especificamente na classificação de churn em um dataset. Nesse contexto, os dados estão rotulados e a variável-alvo é a determinação do churn. Portanto, o objetivo do modelo proposto é encontrar uma função que possa prever de forma precisa o churn com base nos dados de entrada, buscando identificar padrões relevantes previamente desconhecidos.

Nesta etapa do trabalho, estabelecemos e avaliamos os modelos de Aprendizado de Máquina que serão testados. Registramos as técnicas selecionadas, a razão por trás da escolha, bem como a avaliação e análise do desempenho das métricas para cada modelo de Aprendizado de Máquina selecionado. Após o processamento realizado na etapa anterior, a maioria das nossas características (features) já foi preparada para os modelos. Agora, é necessário padronizar as características numéricas e determinar os modelos que serão utilizados.

A fim de comparar o desempenho e as melhorias do modelo, estabeleceremos bases de referência (baselines) para cada um dos modelos a serem avaliados. Nenhum ajuste nos hiper parâmetros será feito nesta etapa, e também não será considerado o balanceamento de dados, engenharia de características ou seleção de características (entre outros). Além disso, procederemos com a separação dos dados que serão usados para treinamento e teste. Os dados de teste serão utilizados apenas na última fase do projeto, para obtermos uma avaliação mais realista.

Em nosso trabalho serão utilizados 5 modelos de Machine de classificação, são eles: Random Forest, Árvore de Decisão, *SGD Classifier*, Regressão Logística *e XGBoost Classifier*. A ideia é comparar as métricas de cada um dos modelos a fim de verificarmos o que apresenta os melhores resultados aplicáveis ao nosso problema.

Antes de aplicarmos os algoritmos de classificação, é essencial determinar qual métrica de avaliação será analisada para verificar a eficácia do modelo. Com esse propósito, iremos utilizar duas funções da biblioteca Scikit-learn: "accuracy_score" e "classification_report". Essas funções fornecem as métricas de acurácia, precisão, recall e pontuação F1.

A acurácia é a quantidade de acertos do modelo dividido pelo total da amostra e indica uma performance geral do modelo. De certa forma é a métrica mais intuitiva e fácil para se entender. A acurácia mostra diretamente a porcentagem de acertos do nosso modelo.

$$Acurácia = \frac{Verdadeiros\ Positivos + Verdadeiros\ Negativos}{Total\ de\ Amostras}$$

A precisão define os chamados positivos verdadeiros, ou seja, dentre os exemplos classificados como verdadeiros, quantos eram realmente verdadeiros. A precisão diz respeito à quantidade (proporcional) de identificações positivas feita corretamente, e nos dá a informação de qual a proporção de identificações positivas que estava correta. Em nosso problema

esse risco seria o nosso modelo classificar como churn um cliente que não pretenda realizar nenhum cancelamento.

$$Precisão = \frac{Verdadeiros\ Positivos}{Verdadeiros\ Positivos + Falsos\ Positivos}$$

O *recall* indica a proporção de dados corretamente classificados como verdadeiros em relação ao total de resultados verdadeiros presentes na amostra. Em nosso problema esse risco seria o nosso modelo não classificar como churn prediction um cliente que pretenda realizar o cancelamento.

$$Recall = \frac{Verdadeiros\ Positivos}{Verdadeiros\ Positivos + Falsos\ Negativos}$$

A pontuação F1 traz a média ponderada da precisão e do *recall*, fornecendo um número único que indica a qualidade geral do modelo. É uma média harmônica entre a precisão e o recall. O valor ideal para a pontuação F1 é 1, enquanto o pior valor é 0.

$$F1\ score = \frac{2\ x\ Precisão\ x\ Recall}{Precisão + Recall}$$

Vamos analisar em nosso modelo também a o índice AUC. O AUC, ou Área sob a Curva ROC, que é uma métrica usada em machine learning para avaliar o desempenho de modelos de classificação binária. A curva ROC representa graficamente a taxa de verdadeiros positivos em relação à taxa de falsos positivos em diferentes pontos de corte do modelo. O AUC é um valor numérico entre 0 e 1, onde um valor mais próximo de 1 indica um melhor desempenho do modelo em classificar corretamente as amostras positivas. O AUC fornece uma medida agregada do desempenho do modelo em todos os possíveis pontos de corte, sendo menos sensível a desequilíbrios de classes. Quanto maior o valor do AUC, melhor é a capacidade do modelo de discriminar entre as classes positiva e negativa.

Ao avaliar as métricas em nosso projeto de *Churn Prediction*, é essencial compreender o objetivo do nosso modelo e os riscos envolvidos. Identificar corretamente um possível cliente cancelador do serviço, evitando o envio de e-mails e pesquisas de satisfação desnecessárias, é crucial para mitigar o risco de perda de clientes. Por outro lado, falhar em detectar um cliente que realmente deseja cancelar o plano pode resultar na perda desse cliente, impactando negativamente a receita e aumentando os custos da empresa.

Considerando o impacto financeiro que um *churn* não identificado e não monitorado pode causar, vamos priorizar a métrica de *recall* em nossas análises. Reconhecemos que a rotatividade de clientes afeta diretamente a receita e que é mais custoso adquirir novos clientes do que manter os existentes. Portanto, nosso foco será em maximizar o r*ecall* para evitar perdas financeiras decorrentes do churn não detectado. Evidentemente isso não vai descartar as outras métricas, que também serão utilizadas na nossa avaliação.

Dando continuidade ao nosso trabalho, procederemos com a divisão do conjunto de dados em conjuntos de treinamento e teste, visando testar o modelo com uma base de dados à qual ele não teve contato prévio. A divisão será realizada de maneira estratificada, levando em consideração a classe, e seguirá a seguinte proporção: 70% para o conjunto de treinamento e 30% para o conjunto de teste (vide Figura 32).

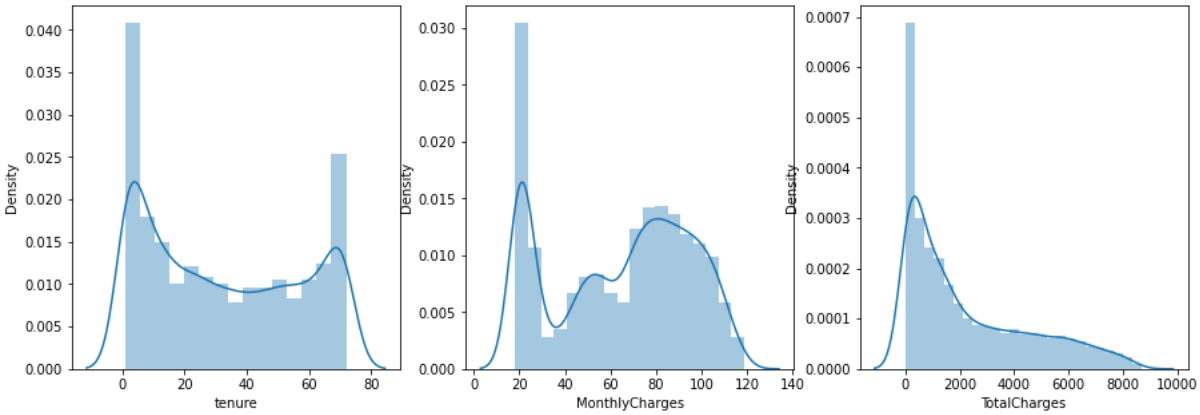
Figura 32 - Separando a base de dados em treino e teste

```
In [ ]: # serparar os dados entre feature matrix e target vector
        X = df_proc.drop('Churn', axis=1)
        y = df_proc.Churn

        # dividindo os dados entre treino e teste
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
```

Fonte: (Autor)

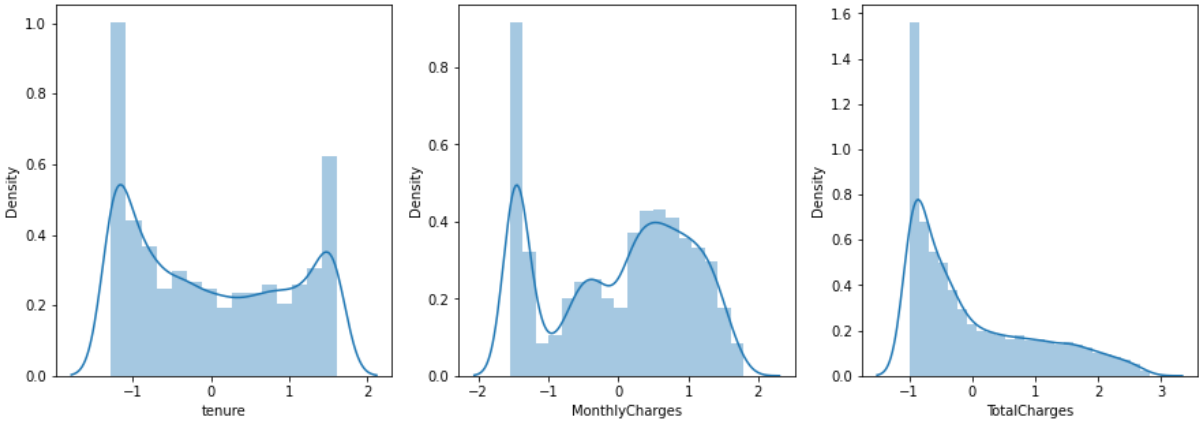
Além disso, realizaremos a padronização dos dados, com o intuito de colocá-los em uma escala comum. Avaliaremos diversos modelos por meio da técnica de Validação Cruzada (*Cross Validation*), considerando que essa validação será realizada inicialmente sem otimização de hiper parâmetros. Para a padronização, utilizaremos dois métodos distintos, o “StandardScaler” e o “RobustScaler” permitindo uma comparação mais precisa do desempenho dos modelos em relação a esses dados, considerando cada tipo de padronização. É importante ressaltar que analisaremos os modelos primeiramente utilizando os dados desbalanceados e, posteriormente, realizaremos a avaliação com os dados balanceados. A Figura 33 e Figura 34 ilustram os efeitos da padronização nos dados. Para efeito ilustrativo utilizamos apenas o método “StandardScaler”.

Figura 33 - Plot variáveis numéricas antes da padronização



Fonte: (Autor)

Figura 34 - Plot variáveis numéricas após a padronização



Fonte: (Autor)

Na Figura 33 verificamos que as variáveis numéricas estão com escalas de valores diferentes. Observando os valores de TotalCharges com MonthlyCharges fica evidente essa discrepância. Na Figura 34 verificamos que, após o processo de padronização as variáveis passam a ter o mesmo padrão de escala. Novamente, ressaltamos que o plot foi realizado para ilustrar o efeito dessa tratativa nos dados. Variáveis numéricas com escalar discrepantes influenciam negativamente na performance do nosso modelo, por isso precisam ser e padronizadas.

5.1. Avaliando modelos com Cross Validation (Dados Desbalanceados)

A validação cruzada (*Cross Validation*) é uma técnica essencial no contexto de modelagem de aprendizado de máquina (*machine learning*), utilizada para avaliar o desempenho e a capacidade de generalização de um modelo preditivo. Essa abordagem consiste em dividir

o conjunto de dados disponível em diferentes partes, normalmente chamadas de *folds*, e realizar múltiplos experimentos de treinamento e teste, de forma que cada *fold* seja utilizado tanto como conjunto de treinamento quanto como conjunto de teste em diferentes iterações.

A importância da validação cruzada reside no fato de que ela nos fornece uma estimativa mais confiável e robusta do desempenho do modelo. Ao realizar múltiplos testes com diferentes partições dos dados, é possível obter uma avaliação mais precisa da capacidade de generalização do modelo para novos dados, minimizando o impacto de variações na distribuição dos dados ou na seleção de um único conjunto de teste. Dessa forma, a validação cruzada nos permite ter uma ideia mais realista do desempenho esperado do modelo em condições de uso real.

É igualmente relevante contar com um modelo de referência, conhecido como modelo baseline, antes de avançar para a construção do modelo efetivo com hiper parâmetros configurados. O modelo baseline é um ponto de partida simples e relativamente ingênuo, que pode ser utilizado como base de comparação para os modelos subsequentes. Ele serve como uma linha de base para avaliar se os modelos mais complexos e sofisticados conseguem melhorar significativamente o desempenho em relação ao modelo mais simples. A utilização de um modelo baseline é importante por diversos motivos. Primeiramente, ele nos permite estabelecer um ponto de referência objetivo, possibilitando uma avaliação mais precisa do ganho de desempenho alcançado com as técnicas e configurações mais avançadas. Além disso, o modelo baseline ajuda a identificar se os resultados obtidos são de fato significativos, pois se os modelos mais complexos não superarem o desempenho do modelo baseline, podemos questionar a eficácia dessas abordagens mais elaboradas.

Conforme consta na Figura 35 criamos uma função denominada “val_model” que realiza *cross validation* com os dados de treino para determinado modelo. Utilizaremos essa função para estimar o erro da baseline e dos modelos iniciais. Inicialmente, iremos aplicar o modelos aos dados desbalanceados.

A função retorna a média dos scores da validação cruzada, que é uma medida de desempenho do modelo. O código da função inicia convertendo X e y para *arrays* do NumPy, pois muitas vezes o Scikit-learn requer esse tipo de formato para a entrada dos dados. Em seguida, são criados dois *pipelines* utilizando a função “make_pipeline” do Scikit-learn. Cada pipeline consiste em uma etapa de padronização dos dados (utilizando o StandardScaler em um pipeline e o RobustScaler em outro) seguida pelo modelo classificador clf. Essa abordagem

permite aplicar a padronização aos dados durante a validação cruzada. A função “cross_val_score” é utilizada para executar a validação cruzada, computando o recall como a métrica de avaliação. Ela recebe como parâmetros o pipeline, os arrays X e y, e o argumento “scoring='recall'”. Após a execução da validação cruzada, os resultados são impressos caso o parâmetro quite seja False. São exibidos o recall médio e o intervalo de confiança (2 desvios padrão) para cada um dos pipelines. Por fim, a função retorna a média dos scores da validação cruzada, correspondendo ao recall médio do modelo.

Figura 35 - Criação de função de validação de modelo

```
In [37]: # função de validação de modelo
def val_model(X, y,clf, quite=False):
  """
    Realiza cross-validation com os dados de treino para determinado modelo.

  # Arguments
      X: DataFrame, contém as variáveis independetes.
      y: Series, vetor contendo a variável alvo.
      clf: modelo classificador do Scikit-learn.
      quite: bool, indicando se a função deve imprimir os resultados ou não.

  # Returns
        float, média dos scores da cross-validation.
  """

  X = np.array(X)
  y = np.array(y)

  pipeline1 = make_pipeline(StandardScaler(), clf)
  pipeline2 = make_pipeline(RobustScaler(), clf)
  scores1 = cross_val_score(pipeline1, X, y, scoring='recall')
  scores2 = cross_val_score(pipeline2, X, y, scoring='recall')

  if quite == False:
    print('Recall StandardScaler: {:.4f} (+/- {:.4f})'.format(scores1.mean(), scores1.std() * 2))
    print('Recall RobustScaler: {:.4f} (+/- {:.4f})'.format(scores2.mean(), scores2.std() * 2))

  return scores1.mean()
```

Fonte: (Autor)

Conforme Figura 36 estamos instanciando os modelos e aplicando a função de validação a cada um deles. Logo em seguida exibimos o *score* de cada um, para checarmos o desempenho.

Figura 36 - Instanciando os modelos antes do balanceamento

```
In [38]: # Instanciando modelos a serem avaliados
rf = RandomForestClassifier()
dt = DecisionTreeClassifier()
sgd = SGDClassifier()
lr = LogisticRegression()
xgb = XGBClassifier()
```

```
In [39]: # printar o desempenho dos modelos com os dados padronizados
print('Cross-validation RF:')
score_teste1 = val_model(X_train, y_train, rf)
print('\nCross-validation DT:')
score_teste2 = val_model(X_train, y_train, dt)
print('\nCross-validation SGD:')
score_teste3 = val_model(X_train, y_train, sgd)
print('\nCross-validation LR:')
score_teste4 = val_model(X_train, y_train, lr)
print('\nCross-validation XGB:')
score_teste5 = val_model(X_train, y_train, xgb)
```

Fonte: (Autor)

Conforme mostrado na Figura 37, é possível observar que a melhor métrica de recall encontrada corresponde a 0,55 para o modelo de Regressão Logística. Contudo, é importante salientar que essa métrica é consideravelmente baixa e ocorre em função do desbalanceamento dos nossos dados utilizados.

Figura 37 - Métricas antes do balanceamento dos dados

```
Cross-validation RF:
Recall StandardScaler: 0.4954 (+/- 0.0375)
Recall RobustScaler: 0.4947 (+/- 0.0223)

Cross-validation DT:
Recall StandardScaler: 0.5023 (+/- 0.0459)
Recall RobustScaler: 0.5015 (+/- 0.0421)

Cross-validation SGD:
Recall StandardScaler: 0.4857 (+/- 0.3456)
Recall RobustScaler: 0.4069 (+/- 0.4069)

Cross-validation LR:

C:\Users\Usuário\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\linear_model\_logistic.py:458: ConvergenceWa
rning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(

Recall StandardScaler: 0.5558 (+/- 0.0441)
Recall RobustScaler: 0.5520 (+/- 0.0541)

Cross-validation XGB:
Recall StandardScaler: 0.5283 (+/- 0.0467)
Recall RobustScaler: 0.5267 (+/- 0.0446)
```

Fonte: (Autor)

5.2. Avaliando modelos com Cross Validation (Dados Balanceados)

Para tratarmos o problema de desequilíbrio das classes da variável "*churn*" em nosso conjunto de dados iremos utilizar técnicas de balanceamento. O balanceamento de dados em machine learning refere-se ao processo de ajuste da distribuição das classes em um conjunto de dados de treinamento. Em problemas de classificação, é comum ter classes desproporcionalmente representadas, ou seja, algumas classes possuem um número significativamente maior de exemplos em comparação com outras. O objetivo do balanceamento de dados é equalizar a distribuição das classes, a fim de garantir que cada classe tenha uma representação adequada no conjunto de treinamento.

Os nossos dados serão combinados com 3 métodos de balanceamento de dados. Sem esse balanceamento nossas métricas ficam muito abaixo do desejável, como mostramos na Figura 37. Serão utilizados os seguintes métodos de balanceamento:

- ***RandomUnderSampling (RUS)*** - Este método descarta um subconjunto aleatório da classe majoritária, preservando as características da classe minoritária.
- ***ADASYN*** - A ideia principal do algoritmo ADASYN é usar a distribuição de densidade como um critério para decidir automaticamente o número de dados sintéticos que precisam ser gerados para cada exemplo da classe minoritária.
- ***SMOTE*** - É um dos métodos de sobreamostragem mais comumente usados para resolver o problema de desequilíbrio. O algoritmo Smote cria dados artificiais com base no recurso da semelhança entre os exemplos da classe minoritária existentes.

Na Figura 38 demonstramos o código utilizado para realizar os balanceamentos.

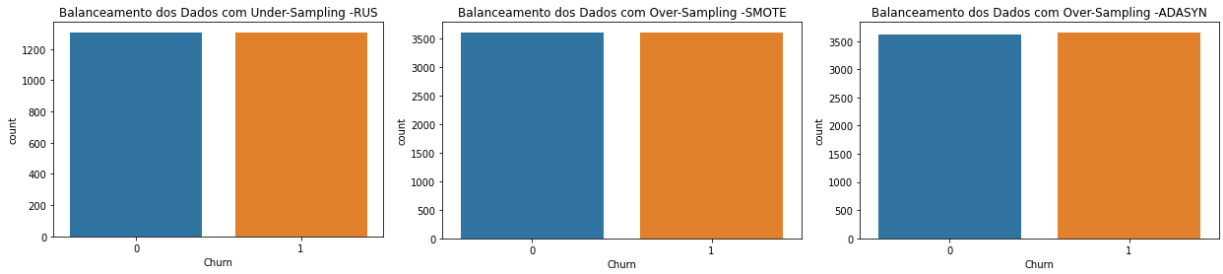
Figura 38 - Código para balanceamento dos dados

```
In [ ]: # importar metodos para realizacao do feature scaling com fit nos dados de treino
scaler = StandardScaler()
Rob_scaler = RobustScaler()

# padronizando os dados de treino
X_train_scaled = scaler.fit_transform(X_train)
X_train_Rscaled = Rob_scaler.fit_transform(X_train)

# balanceamento RUS
rus = RandomUnderSampler()
X_train_rus_scaled, y_train_rus_scaled = rus.fit_resample(X_train_scaled, y_train)
X_train_rus_Rscaled, y_train_rus_Rscaled = rus.fit_resample(X_train_Rscaled, y_train)

# balanceamento SMOTE
smo = SMOTE()
X_train_smo_scaled, y_train_smo_scaled = smo.fit_resample(X_train_scaled, y_train)
X_train_smo_Rscaled, y_train_smo_Rscaled = smo.fit_resample(X_train_Rscaled, y_train)


# balanceamento ADASYN
ada = ADASYN()
X_train_ada_scaled, y_train_ada_scaled = ada.fit_resample(X_train_scaled, y_train)
X_train_ada_Rscaled, y_train_ada_Rscaled = ada.fit_resample(X_train_Rscaled, y_train)
```

Fonte: (Autor)

A Figura 39 nos mostra um gráfico com os dados balanceados. É possível observar que o balanceamento iguala a distribuição de classes da variável “churn”. O método de *Random Under Sampling*, diferente dos métodos de *Over Sampling* ADASYN e SMOTE, reduz a distribuição para cerca de 1.300 dados de cada classe, o que inevitavelmente acarreta em perda de um subconjunto aleatório da classe maioritária. Os demais métodos apresentam cerca de 3.500 dados para cada classe, justamente por serem métodos em que são criados dados artificiais para igualar a classe minoritária.

Figura 39 - Balanceamento dos Dados



Fonte: (Autor)

Na Figura 40, mostramos a comparação das métricas geradas pela função de validação dos modelos balanceados. Nota-se que comparamos as métricas para cada modelo, em cada tipo de balanceamento, comparando inclusive os métodos de padronização “StandardScaler” representado na tabela como “Scaled” e o RobustScaler, representado na tabela como “Rscaled”. A primeira tabela mostra as métricas do balanceamento *Random Under Sampling*, a segunda mostra as métricas do balanceamento *Over Sampling* SMOTE e a última nos mostra as métricas do balanceamento *Over Sampling* ADASYN.

Figura 40 - **Comparativo de métricas entre modelos balanceados**

	Recall
RF RusScaled	0.752283
RF RusRscaled	0.779808
DT RusScaled	0.683507
DT RusRscaled	0.684241
SGD RusScaled	0.759054
SGD RusRscaled	0.875262
LR RusScaled	0.797374
LR RusRscaled	0.805785
XGBoost RusScaled	0.797400
XGBoost RusRscaled	0.822605

	Recall
RF SMOScaled	0.872746
RF SMORscaled	0.869979
DT SMOScaled	0.793890
DT SMORscaled	0.798864
SGD SMOScaled	0.783888
SGD SMORscaled	0.804645
LR SMOScaled	0.804933
LR SMORscaled	0.806862
XGBoost SMOScaled	0.857256
XGBoost SMORscaled	0.846742

	Recall
RF ADAScaled	0.868273
RF ADARscaled	0.867272
DT ADAScaled	0.770376
DT ADARscaled	0.752068
SGD ADAScaled	0.782570
SGD ADARscaled	0.788235
LR ADAScaled	0.795982
LR ADARscaled	0.791004
XGBoost ADAScaled	0.846147
XGBoost ADARscaled	0.846952

Fonte: (Autor)

Ao analisar a Figura 40, verificamos que os modelos SGD, LR e XGBoost obtiveram as melhores métricas. A maior métrica de *recall* foi do modelo SGD, no balanceamento pelo Random Under Sampling, apresentando um índice de 0.8752. No entanto, para o balanceamento ADA essa métrica diminuiu para 0.788235. O mesmo ocorre com o modelo de Random Forest

(RF) que apresenta métrica de *recall* de 0.872746 no balanceamento SMOTE, mas, pelo *Random Under Sampling* mostra um índice de 0.752283. Dos modelos apresentados o que se destaca é o XGBoost, que apresenta bons resultados em todos os balanceamentos. Portanto, constatando as melhores métricas de recall apresentadas e considerando que o XGBoost oferece uma ampla gama de hiper parâmetros que podem ser ajustados para controlar o desempenho e a complexidade do modelo, este será o modelo escolhido a ser aplicado em nosso problema. Na Figura 41 mostramos as melhores métricas deste modelo nos balanceamentos realizados.

Figura 41 - Métricas Modelo XGBoost

	Recall
XGBoost RusRscaled	0.822605
XGBoost SMOScaled	0.857256
XGBoost ADARScaled	0.846952

Fonte: (Autor)

Agora vamos prosseguir para a etapa de otimização e ajuste refinado dos hiper parâmetros do modelo, a fim de comparar os resultados para cada balanceamento. Nesta etapa, ocorrerá a otimização dos hiper parâmetros do modelo *XGBoost* para cada tipo de balanceamento, com os ajustes necessários. O modelo XGBoost possui diversos parâmetros, sendo que alguns exercem maior impacto na qualidade do modelo do que outros. Uma prática recomendada consiste em definir uma taxa de aprendizado e número de estimadores, realizar a afinação de outros parâmetros e, por fim, verificar diferentes taxas de aprendizado.

5.3. Otimização do modelo e ajuste dos hiper parâmetros

Para otimização do modelo iremos utilizar o “Gridsearch” que é uma técnica de aprendizado de máquina usada para encontrar a melhor combinação de hiper parâmetros de um modelo. Envolve a criação de uma grade de valores possíveis para os hiper parâmetros, que são parâmetros pré-definidos do modelo. Cada combinação de valores é testada e avaliada usando métricas de desempenho, como recall, acurácia ou precisão, visando encontrar a configuração que resulta no melhor desempenho do modelo. Essa abordagem sistemática testa

todas as combinações possíveis, tornando-a computacionalmente custosa, mas amplamente usada para otimizar modelos de aprendizado de máquina. Para validação vamos utilizar o *cross validation* também. A validação cruzada é uma técnica usada para avaliar o desempenho e a capacidade de generalização de modelos de aprendizado de máquina. Conforme mencionado anteriormente, ela envolve a divisão do conjunto de dados em partes menores, chamadas de *folds*, onde o modelo é treinado em uma parte e avaliado nas outras. O método mais comum é o *k-fold cross validation*, onde o conjunto de dados é dividido em *k-folds* e o modelo é treinado e testado várias vezes. Isso fornece uma estimativa robusta do desempenho do modelo, evitando o sobreajuste e permitindo uma avaliação mais precisa em dados não vistos. A validação cruzada é amplamente utilizada na seleção de modelos, ajuste de hiper parâmetros e avaliação de desempenho em diferentes problemas de aprendizado de máquina.

Na Figura 42, Figura 43, Figura 44 e Figura 45 mostramos os códigos de otimização e ajuste dos hiper parâmetros do modelo *XGBoost Classifier*, com o balanceamento Random Under Sampling. As mesmas etapas foram seguidas para o modelo no balanceamento *Over Samping* SMOTE e *Over Sampling* ADASYN e estão documentadas no Apêndice do nosso trabalho, contendo o código completo.

A primeira linha declara uma variável chamada "xgb_rus_Rscaled" que está sendo atribuída a uma instância do classificador XGBClassifier, com um parâmetro "learning_rate" definido como 0.1. A segunda linha mostra a otimização pelo “gridsearch” através da variável chamada "param_grid" que é um dicionário contendo um único parâmetro "n_estimators" com uma lista de valores a serem otimizados. Logo após, criamos uma instância de "StratifiedKFold" chamada "kfold" com o número de splits definido como 10. Assim nosso código vai dividir o conjunto de dados em 10 partes iguais (chamadas de "folds") e, em seguida, realiza 10 iterações de treinamento e teste. Em seguida cria uma instância de "GridSearchCV" chamada "grid_search", que recebe o modelo "xgb_rus_Rscaled", o dicionário de parâmetros "param_grid", uma métrica de pontuação definida como 'recall', o número de jobs definido como -1 (usando todos os processadores disponíveis) e o objeto "kfold" definido anteriormente como o cross-validation. Logo após realizamos a otimização dos parâmetros usando o conjunto de treinamento "X_train_rus_Rscaled" e "y_train_rus_Rscaled" através do método "fit( )" do objeto "grid_search", armazenando os resultados na variável "grid_result". Por fim imprimimos o melhor resultado de pontuação encontrado pelo “gridsearch”, juntamente com

os melhores parâmetros encontrados. Em nosso código o melhor resultado veio do parâmetro “n_estimator” configurado como 10.

Figura 42 - Otimização "gridsearch" parâmetro “param_grid”

```
In [ ]: # modelo a ser otimizado
xgb_rus_Rscaled = XGBClassifier(learning_rate=0.1)

# parâmetros a serem otimizados
param_grid = {
      'n_estimators':[0,10,50,100],
}

# identificar os melhores parâmetros
o
grid_search = GridSearchCV(xgb_rus_Rscaled, param_grid, scoring='recall', n_jobs=-1, cv=kfold)
grid_result = grid_search.fit(X_train_rus_Rscaled, y_train_rus_Rscaled)

# ver resultados
print(f'Melhor resultado: {grid_result.best_score_} para {grid_result.best_params_}')

Melhor resultado: 0.8249559600704639 para {'n_estimators': 10}
```

Fonte: (Autor)

Uma vez com o número de estimadores definido para 10, vamos realizar a busca para os parâmetros “max_depth” e “min_child_weight” (vide Figura 43). Nota-se que no código já incluímos o parâmetro de “n_estimators=10”, que foi o melhor resultado encontrado na validação anterior.

Figura 43 - Otimização "gridsearch" parâmetros "max_depth" e "min_child_weight"

```
In [ ]: xgb_rus_Rscaled = XGBClassifier(learning_rate=0.1, n_estimators=10)

param_grid = {
 'max_depth': range(1,4,1),
 'min_child_weight': range(1,4,1)
}

# identificar melhor parâmetro
kfold = StratifiedKFold(n_splits=10, shuffle=True)
grid_search = GridSearchCV(xgb_rus_Rscaled, param_grid, scoring="recall", n_jobs=-1, cv=kfold)
grid_result = grid_search.fit(X_train_rus_Rscaled, y_train_rus_Rscaled)

# ver resultados
print("Melhor: {} para {}".format(grid_result.best_score_, grid_result.best_params_))

Melhor: 0.8914445096887846 para {'max_depth': 1, 'min_child_weight': 1}
```

Fonte: (Autor)

Obtidos os valores de “max_depth” = 1 e “min_child_weight” = 1, iremos otimizar o parâmetro “gamma” (vide Figura 44), utilizando os mesmos procedimentos de gridsearch documentados anteriormente e acrescentando os parâmetros com melhores resultados encontrados nas validações anteriores.

Figura 44 - Otimização "gridsearch" parâmetros "gamma"

```
In [ ]: # Modelo a ser otimizado
xgb_rus_Rscaled = XGBClassifier(learning_rate=0.1, n_estimators=10, max_depth=1, min_child_weight=1)

# Parâmetros a serem otimizados
param_grid = {
    'gamma':[0, 0.5, 1, 3]
}

# Identificar melhores parâmetros
kfold = StratifiedKFold(n_splits=10, shuffle=True)
grid_search = GridSearchCV(xgb_rus_Rscaled, param_grid, scoring='recall', n_jobs=-1, cv=kfold)
grid_result = grid_search.fit(X_train_rus_Rscaled, y_train_rus_Rscaled)

# Ver resultados
print(f'Melhor: {grid_result.best_score_} para {grid_result.best_params_}')
```

```
Melhor: 0.8914679976512037 para {'gamma': 0}
```

Fonte: (Autor)

Com “gamma” = 0, vamos testar outros valores para “learning_rate”. Dessa forma, podemos otimizar ainda mais o nosso modelo e saber qual valor nos entregará o melhor resultado.

Figura 45 - Otimização "gridsearch" parâmetros "learning_rate"

```
In [ ]: xgb_rus_Rscaled = XGBClassifier(n_estimators=50, max_depth=1, min_child_weight=1, gamma=0)

param_grid = {
 'learning_rate':[0.001, 0.01, 0.015, 0.1]
}

# identificar melhor parâmetro
kfold = StratifiedKFold(n_splits=10, shuffle=True)
grid_search = GridSearchCV(xgb_rus_Rscaled, param_grid, scoring="recall", n_jobs=-1, cv=kfold)
grid_result = grid_search.fit(X_train_rus_Rscaled, y_train_rus_Rscaled)

# ver resultados
print("Melhor: {} para {}".format(grid_result.best_score_, grid_result.best_params_))
```

```
Melhor: 0.8914034057545509 para {'learning_rate': 0.001}
```

Fonte: (Autor)

Após as etapas de otimização e ajuste dos hiper parâmetros do modelo, podemos observar uma evolução da métrica de recall, passando de 0.8249 antes da aplicação do “gridsearch” para 0.8914 após as modificações.

Abaixo nas Figuras 46, 47 e 48 mostramos os códigos com os hiper parâmetros ajustados para cada modelo, após todo processo de ajuste e de validação cruzada dos modelos. Lembrando que utilizamos a base de treino para validação e avaliação de desempenho do modelo e aplicamos o modelo final na nossa base de testes, conforme já havíamos segregado anteriormente. Com isso nosso modelo, após ajustado, será testado em uma base com dados não vistos anteriormente, o que é essencial para verificar se o modelo está generalizando bem e não apenas "decorando" os dados de treinamento.

Figura 46 - Modelo XGBClassifier ajustado pelo balanceamento RUS (Random Under Sampling)

```
In [ ]: # modelo final
xgb_rus_Rscaled = XGBClassifier(learning_rate=0.001 , n_estimators=50, max_depth=1, min_child_weight=1, gamma=0)

# treinando o modelo
xgb_rus_Rscaled.fit(X_train_rus_Rscaled, y_train_rus_Rscaled)

# fazer a previsão
X_test_Rscaled = scaler.transform(X_test)
X_test_scaled = scaler.transform(X_test)
y_pred_xgb_rus_Rscaled = xgb_rus_Rscaled.predict(X_test_Rscaled)

# Classification Report
print(classification_report(y_test, y_pred_xgb_rus_Rscaled))

# imprimir a área sob a curva e Recall score
print("AUC: {:.4f}\n".format(roc_auc_score(y_test, y_pred_xgb_rus_Rscaled)))
print(f'Recall:\t\t{recall_score(y_test, y_pred_xgb_rus_Rscaled, pos_label=1):0.4f}')

# plotar matriz de confusão
plot_confusion_matrix(y_test, y_pred_xgb_rus_Rscaled, normalize=True)
plt.show();
```

Fonte: (Autor)

Figura 47 - Modelo XGBClassifier ajustado pelo balanceamento Over Sampling - SMOTE

```
In [ ]: # modelo final
xgb_smo_scaled = XGBClassifier(learning_rate=0.1 , n_estimators=50, max_depth=3, min_child_weight=3, gamma=1)

# treinando o modelo
xgb_smo_scaled.fit(X_train_smo_scaled, y_train_smo_scaled)

# fazer a previsão
# X_test_Rscaled = scaler.transform(X_test)
# X_test_scaled = scaler.transform(X_test)
y_pred_xgb_smo_scaled = xgb_smo_scaled.predict(X_test_scaled)

# Classification Report
print(classification_report(y_test, y_pred_xgb_smo_scaled))

# imprimir a área sob a curva e Recall score
print("AUC: {:.4f}\n".format(roc_auc_score(y_test, y_pred_xgb_smo_scaled)))
print(f'Recall:\t\t{recall_score(y_test, y_pred_xgb_smo_scaled, pos_label=1):0.4f}')

# plotar matriz de confusão
plot_confusion_matrix(y_test, y_pred_xgb_smo_scaled, normalize=True)
plt.show();
```

Fonte: (Autor)

Figura 48 - Modelo XGBClassifier ajustado pelo balanceamento Over Sampling - ADASYN

```
In [ ]: # modelo final
xgb_ada_Rscaled = XGBClassifier(learning_rate=0.1 , n_estimators=50, max_depth=3, min_child_weight=3, gamma=3)

# treinando o modelo
xgb_ada_Rscaled.fit(X_train_ada_Rscaled, y_train_ada_Rscaled)

# fazer a previsão
# X_test_Rscaled = scaler.transform(X_test)
# X_test_scaled = scaler.transform(X_test)
y_pred_xgb_ada_Rscaled = xgb_ada_Rscaled.predict(X_test_Rscaled)

# Classification Report
print(classification_report(y_test, y_pred_xgb_ada_Rscaled))

# imprimir a área sob a curva e Recall score
print("AUC: {:.4f}\n".format(roc_auc_score(y_test, y_pred_xgb_ada_Rscaled)))
print(f'Recall:\t\t{recall_score(y_test, y_pred_xgb_ada_Rscaled, pos_label=1):0.4f}')

# plotar matriz de confusão
plot_confusion_matrix(y_test, y_pred_xgb_ada_Rscaled, normalize=True)
plt.show();
```

Fonte: (Autor)

6. Apresentação e Interpretação dos Resultados

Agora vamos apresentar os resultados obtidos com base na aplicação do modelo de *machine learning* desenvolvidos para a previsão de *churn*, através do XGBClassifier, já ajustado

com os dados balanceados e com os hiper parâmetros que apresentaram as melhores métricas. Os principais indicadores de desempenho utilizados serão o recall, F1-score e o índice AUC. Além disso, também analisaremos as matrizes de confusão para avaliar o desempenho dos modelos nos cenários dos três tipos de balanceamentos que foram realizados. Nesta etapa final nosso modelo para cada balanceamento já é aplicado diretamente em nossa base de teste.

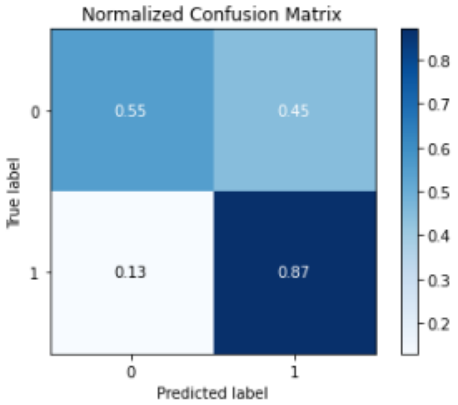
Na Figura 49, Figura 50, e Figura 51 apresentamos a matriz de confusão de cada um dos modelos, juntamente com as métricas apresentadas. A matriz de confusão é uma tabela que mostra a performance de um modelo de classificação comparando as previsões feitas pelo modelo com as classes reais dos dados. Ela é usada para avaliar a precisão do modelo e identificar erros de classificação. Utilizamos o parâmetro de normalize na matriz definido como True, para exibir as proporções de cada classe em relação ao total de previsões. Isso ajuda a obter uma visão mais clara das taxas de erro relativas entre as classes.

Figura 49 - Resultado Modelo XGBClassifier balanceado pelo RUS

	precision	recall	f1-score	support
0	0.92	0.55	0.69	1549
1	0.41	0.87	0.56	561
accuracy			0.64	2110
macro avg	0.67	0.71	0.62	2110
weighted avg	0.79	0.64	0.65	2110

AUC: 0.7108

Recall: 0.8717



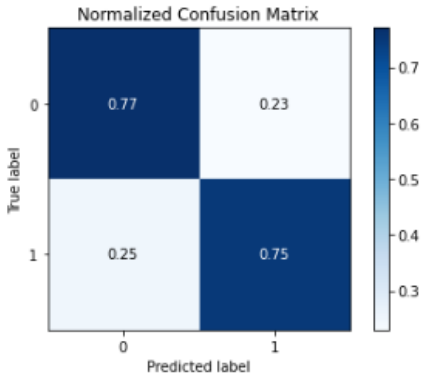
Fonte: (Autor)

Figura 50 - Resultado Modelo XGBClassifier balanceado pelo SMOTE

	precision	recall	f1-score	support
0	0.89	0.77	0.83	1549
1	0.54	0.75	0.63	561
accuracy			0.77	2110
macro avg	0.72	0.76	0.73	2110
weighted avg	0.80	0.77	0.78	2110

AUC: 0.7599

Recall: 0.7451



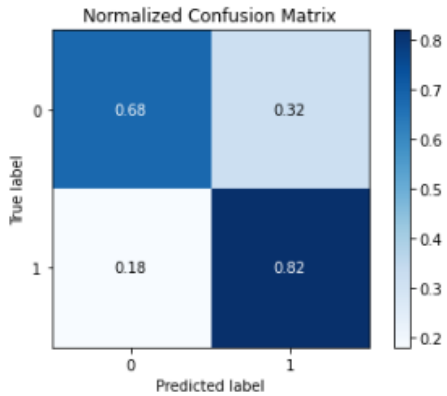
Fonte: (Autor)

Figura 51 - Resultado Modelo XGBClassifier balanceado pelo ADASYN

	precision	recall	f1-score	support
0	0.91	0.68	0.78	1549
1	0.48	0.82	0.61	561
accuracy			0.72	2110
macro avg	0.70	0.75	0.69	2110
weighted avg	0.80	0.72	0.74	2110

AUC: 0.7504

Recall: 0.8164



Fonte: (Autor)

Para melhor comparabilidade dos modelos, apresentamos na Figura 52 as métricas de cada um dos modelos e na Figura 53 as matrizes de confusão juntas, para facilitar a

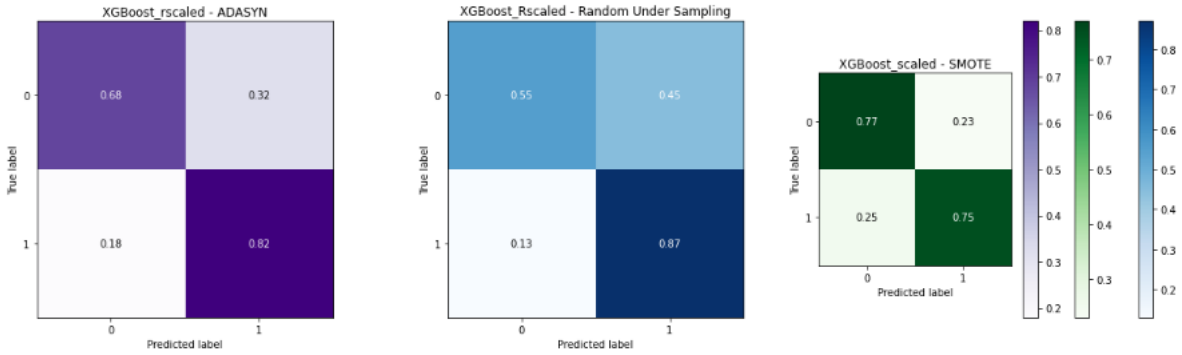
visualização. Nas métricas mostramos o *recall*, que já foi definido como métrica principal no início do nosso trabalho e também o AUC, que é uma métrica que avalia a habilidade do modelo de classificação em distinguir entre as classes positiva e negativa, onde um valor mais próximo de 1 nos indica um melhor desempenho.

Figura 52 - Métricas dos modelos de XGBClassifier

```
In [ ]: print('Métricas - XGBoost_Rscaled - Random Under Sampling')
        print(f'Recall:\t\t{recall_score(y_test, y_pred_xgb_rus_Rscaled, pos_label=1):0.4f}')
        print(f'AUC:\t\t{roc_auc_score(y_test, y_pred_xgb_rus_Rscaled):0.4f}')

        print('\nMétricas - XGBoost_scaled - SMOTE')
        print(f'Recall:\t\t{recall_score(y_test, y_pred_xgb_smo_scaled, pos_label=1):0.4f}')
        print(f'AUC:\t\t{roc_auc_score(y_test, y_pred_xgb_smo_scaled):0.4f}')

        print('\nMétricas - XGBoost_rscaled - ADASYN')
        print(f'Recall:\t\t{recall_score(y_test, y_pred_xgb_ada_Rscaled, pos_label=1):0.4f}')
        print(f'AUC:\t\t{roc_auc_score(y_test, y_pred_xgb_ada_Rscaled):0.4f}')

Métricas - XGBoost_Rscaled - Random Under Sampling
Recall:         0.8717
AUC:            0.7108

Métricas - XGBoost_scaled - SMOTE
Recall:         0.7451
AUC:            0.7599

Métricas - XGBoost_rscaled - ADASYN
Recall:         0.8164
AUC:            0.7504
```

Fonte: (Autor)

Figura 53 - Comparativo de matriz de confusão para cada balanceamento



Fonte: (Autor)

Após a execução dos procedimentos de limpeza de dados, padronização, balanceamento, comparação dos modelos baselines e ajuste de hiperparâmetros, constatamos que nosso trabalho obteve um resultado satisfatório.

Entre os modelos avaliados, o XGBoost foi selecionado para ser aplicado em cada técnica de balanceamento, devido à sua flexibilidade na configuração de hiperparâmetros. Embora tenhamos realizado apenas alguns ajustes, seguindo um padrão para essa etapa de ajuste fino, reconhecemos que seria possível configurar individualmente cada modelo, para

cada método de balanceamento, considerando outros parâmetros não abordados. No entanto, essa abordagem demandaria mais tempo e aumentaria o custo do projeto. Portanto, é fundamental e sensato avaliar o resultado desejado, o tempo disponível, considerando as metas do projeto e ponderar todos esses aspectos.

Em conclusão, ao analisar os modelos acima mencionados, observamos que o modelo que obteve o melhor desempenho na detecção de Churn na base de testes foi o XGBoost, utilizando dados padronizados pelo Robust Scaler e balanceados pelo Random Under Sampling, alcançando uma taxa de acerto de 87% dos casos de churn.

Além disso, observamos as melhorias proporcionadas pelos ajustes de hiperparâmetros no modelo XGBoost balanceado pelo Random Under Sampling. No modelo baseline, sem ajustes, a métrica de recall apresentou um valor de 0.82. Após a aplicação dos ajustes finos, obtivemos um resultado de 0.87, representando uma melhora de aproximadamente 6% no recall.

O modelo que apresentou o melhor desempenho na detecção dos casos que não são Churn foi o XGBoost, utilizando dados padronizados pelo Standard Scaler e balanceados pelo Over Sampling SMOTE, obtendo uma taxa de acerto de 77%.

O modelo XGBoost balanceado pelo Over Sampling ADASYN também demonstrou bons resultados, inclusive superando o modelo Random Under Sampling na detecção dos casos que não são Churn. No entanto, a taxa de acerto para a detecção de Churn foi mais elevada no Random Under Sampling, o que é crucial para o nosso problema.

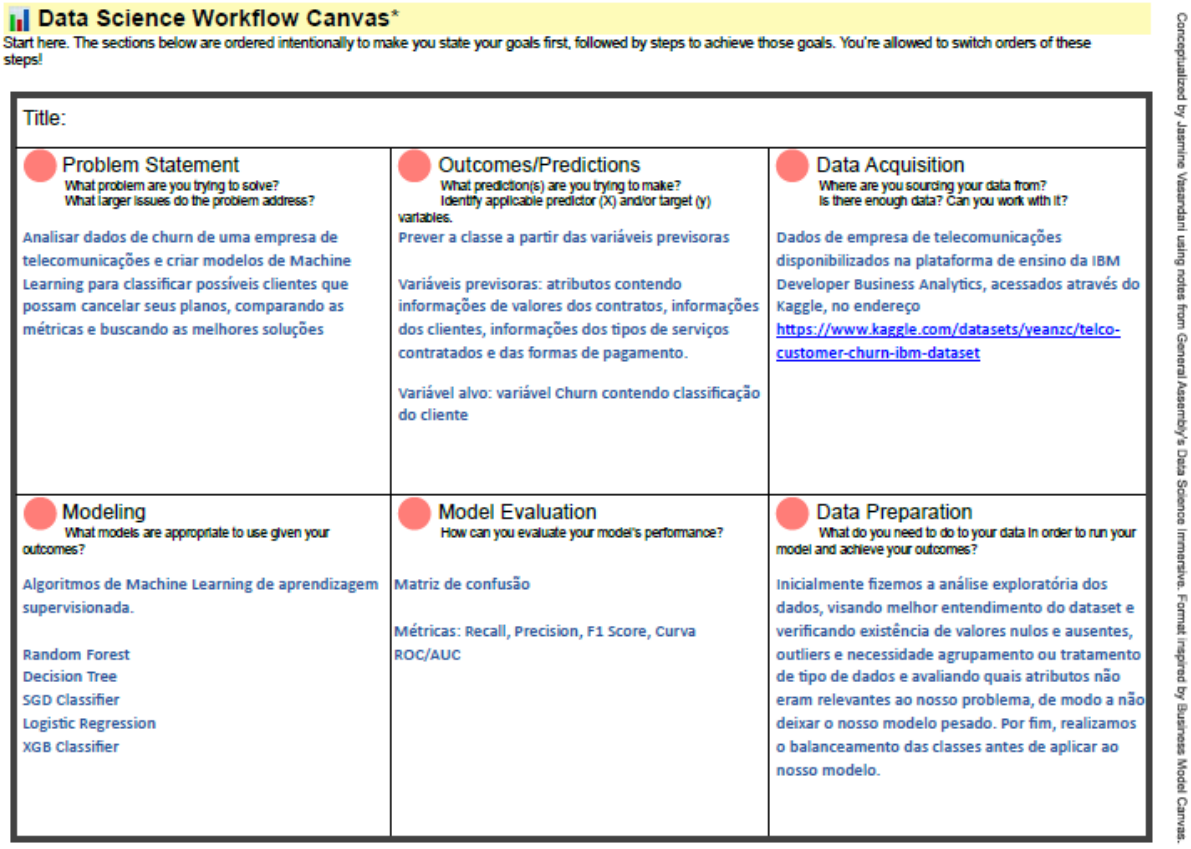
Portanto, considerando que priorizamos a métrica de Recall para o nosso problema e levando em consideração as possíveis perdas financeiras resultantes de um churn não identificado e monitorado, concluímos que o modelo balanceado pelo Random Under Sampling apresentou os melhores resultados.

6.1. Apresentação dos Resultados pelo Modelo Canvas

Os resultados do projeto foram obtidos a partir do estabelecimento das metas mapeadas no modelo Canvas de Vasandani conforme apresentado na Figura 54. Ao trabalharmos em um projeto de ciência de dados, geralmente não temos um conjunto de instruções para alcançar um resultado predeterminado. Em vez disso, é necessário determinar os resultados

e as etapas para alcançá-los. É um processo iterativo. Esse fluxo de trabalho de ciência de dados foi projetado com esse processo em mente.

Figura 54 - Apresentação dos Resultados e Etapas do Projeto pelo Canvas



Data Science Workflow Canvas*

Start here. The sections below are ordered intentionally to make you state your goals first, followed by steps to achieve those goals. You're allowed to switch orders of these steps!

Title:

Problem Statement
What problem are you trying to solve?
What larger issues do the problem address?

Analisar dados de churn de uma empresa de telecomunicações e criar modelos de Machine Learning para classificar possíveis clientes que possam cancelar seus planos, comparando as métricas e buscando as melhores soluções

Outcomes/Predictions
What prediction(s) are you trying to make?
Identify applicable predictor (X) and/or target (y) variables.

Prever a classe a partir das variáveis previsoras

Variáveis previsoras: atributos contendo informações de valores dos contratos, informações dos clientes, informações dos tipos de serviços contratados e das formas de pagamento.

Variável alvo: variável Churn contendo classificação do cliente

Data Acquisition
Where are you sourcing your data from?
Is there enough data? Can you work with it?

Dados de empresa de telecomunicações disponibilizados na plataforma de ensino da IBM Developer Business Analytics, acessados através do Kaggle, no endereço https://www.kaggle.com/datasets/yeanzc/telco-customer-churn-ibm-dataset

Modeling
What models are appropriate to use given your outcomes?

Algoritmos de Machine Learning de aprendizagem supervisionada.

Random Forest
Decision Tree
SGD Classifier
Logistic Regression
XGB Classifier

Model Evaluation
How can you evaluate your model's performance?

Matriz de confusão

Métricas: Recall, Precision, F1 Score, Curva ROC/AUC

Data Preparation
What do you need to do to your data in order to run your model and achieve your outcomes?

Inicialmente fizemos a análise exploratória dos dados, visando melhor entendimento do dataset e verificando existência de valores nulos e ausentes, outliers e necessidade agrupamento ou tratamento de tipo de dados e avaliando quais atributos não eram relevantes ao nosso problema, de modo a não deixar o nosso modelo pesado. Por fim, realizamos o balanceamento das classes antes de aplicar ao nosso modelo.

Conceptualized by Jasmine Vasandani using notes from General Assembly's Data Science Immersive. Format inspired by Business Model Canvas.

Fonte: (Autor)

7. Links

Link para o vídeo: youtube.com/...

Link para o repositório: github.com/...

REFERÊNCIAS

A. PAYNE, P. F. A strategic framework for customer relationship management. Journal of Marketing Research, v. 69, p. 167—176, 2005.

BREIMAN, L. Random Forests, Machine Learning, Vol. 45, pp. 5 – 32, 2001.

BUREZ, J.; VAN DEN POEL, D. CRM at a pay-TV company: Using analytical models to reduce customer attrition by targeted marketing for subscription services. Expert Systems with Applications, v. 32, p. 277–288, 2007.

COUSSEMENT, K.; POEL, D. V. D. Improving customer attrition prediction by integrating emotions from client/company interaction emails and evaluating multiple classifiers. Expert Systems with Applications, v. 36, p. 6127–6134, 2009.

GUANGLI, Nie et al. Credit card churn forecasting by logistic regression and decision tree. Expert Systems with Applications, v. 38, n. 12, p. 15273-15285, 2011. Disponível em: <https://doi.org/10.1016/j.eswa.2011.06.028>. Acesso em: 10 mai. 2023.

HADDEN, J.; TIWARI, A.; ROY, R.; RUTA, D. Computer assisted customer churn management: State-of-the-art and future trends. Computers and Operations Research, v. 34, p. 2902–2917, 2007.

MA, Shaohui, TAN, Hui; SHU, Fang. When is the best time to reactivate your inactive customers? Marketing Letters, v. 26, n. 1, 81-98, 2015. Disponível em: <https://link.springer.com/article/10.1007%2Fs11002-013-9269-7>. Acesso em: 10 mai. 2023

MARTINS, Lucas Gomes. Aplicação de um modelo de aprendizagem de máquina para predição de churn: um estudo de caso. Dissertação (Bacharelado em Engenharia Metalúrgica) - Universidade Federal do Ceará, Fortaleza, 2021.

MARTÍNEZ, Estela. Machine learning algorithms for the prediction of non-metallic inclusions in steel wires for tire reinforcement. 2019.

MITCHELL, T. Aprendizado de máquina. Nova York: McGrawhill, 1997.

MÜLLER, A. C.;GUIDO, S. Introduction to Machine Learning with Python. O'Reilly Media, Inc. 2017. https://doi.org/10.1007/978-3-030-36826-5_10. Acesso em: 10 mai. 2023.

PIMENTEL, Thiago Paiva. Predição de churn baseada em detecção de padrões sequenciais e análise de sentimentos sobre as interações de clientes no CRM. Dissertação (Mestrado em Ciências em Sistemas e Computação) - Instituto Militar de Engenharia, Rio de Janeiro, 2019.

Tukey, J. W. (1962). The future of data analysis. The Annals of Mathematical Statistics, 33, 1–67.

Tukey, J. W. (1977). Exploratory Data Analysis. Reading: Addison-Wesley.

VERBRAKEN, Thomas; VERBEKE, Wouter; BAESENS, Bart. Profit optimizing customer churn prediction with Bayesian network classifiers. Intelligent Data Analysis, v. 18, n. 1, p. 3-24, 2014. Disponível em: <https://doi.org/10.3233/IDA-130625>. Acesso em: 10 mai. 2023.

APÊNDICE

```
# instalando algumas bibliotecas no Colab
!pip install scikit-plot -q
!pip install imbalanced-learn -q
!pip install xgboost

# desconsiderar os warnings
from warnings import simplefilter
simplefilter(action='ignore', category=FutureWarning)
```

```
# importando os pacotes necessários
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import scikitplot as skplt

# importando balanceamento RUS
from imblearn.under_sampling import RandomUnderSampler

# importando balanceamento SMOTE
from imblearn.over_sampling import SMOTE

# importando balanceamento ADASYN
from imblearn.over_sampling import ADASYN

# importando métricas
from sklearn.metrics import recall_score, roc_auc_score, accu-
racy_score, f1_score, confusion_matrix, classification_report
from scikitplot.metrics import plot_confusion_matrix, plot_roc


# importando pacotes de padronização e tratamento de variaveis categó-
ricas
from sklearn.preprocessing import StandardScaler, LabelEncoder, Robus-
tScaler

# importando pipeline
from sklearn.pipeline import make_pipeline

# importando model_selection
from sklearn.model_selection import train_test_split, cross_val_score,
StratifiedKFold, GridSearchCV

# importando modelos
from sklearn.linear_model import LogisticRegression, SGDClassifier
from sklearn.tree import DecisionTreeClassifier
```

```
from xgboost import XGBClassifier
from sklearn.ensemble import RandomForestClassifier

# random seed
np.random.seed(42)
```

```
# importando os dados
data_path = "https://raw.githubusercontent.com/carlosfab/dsnp2/mas-
ter/datasets/WA_Fn-UseC_-Telco-Customer-Churn.csv"

df = pd.read_csv(data_path)

# utilizando o pd.set_option para mostrar todas as colunas
pd.set_option('display.max_columns', None)
```

```
# verificando os primeiros registros
df.head()
```

```
# verificando o nome das variáveis do dataset
print(f'Nome das variáveis do dataset:{df.columns.values}')
```

```
Nome das variáveis do dataset:['customerID' 'gender' 'SeniorCitizen'
'Partner' 'Dependents' 'tenure'
 'PhoneService' 'MultipleLines' 'InternetService' 'OnlineSecurity'
 'OnlineBackup' 'DeviceProtection' 'TechSupport' 'StreamingTV'
 'StreamingMovies' 'Contract' 'PaperlessBilling' 'PaymentMethod'
 'MonthlyCharges' 'TotalCharges' 'Churn']
```

```
# verificando os dados por coluna para checar distribuição
for column in df.columns:
  print(f'Coluna {column}: {df[column].unique()}')
  print('------------'*10)
```

```
Coluna customerID: ['7590-VHVEG' '5575-GNVDE' '3668-QPYBK' ... '4801-
JZAZL' '8361-LTMKD'
 '3186-AJIEK']
--------------------------------------------------------------------------
--------------------------------------------------
Coluna gender: ['Female' 'Male']
--------------------------------------------------------------------------
--------------------------------------------------
Coluna SeniorCitizen: [0 1]
--------------------------------------------------------------------------
--------------------------------------------------
Coluna Partner: ['Yes' 'No']
--------------------------------------------------------------------------
--------------------------------------------------
Coluna Dependents: ['No' 'Yes']
--------------------------------------------------------------------------
--------------------------------------------------
Coluna tenure: [ 1 34  2 45  8 22 10 28 62 13 16 58 49 25 69 52 71 21
12 30 47 72 17 27
```

```
  5 46 11 70 63 43 15 60 18 66  9  3 31 50 64 56  7 42 35 48 29 65 38
68
 32 55 37 36 41  6  4 33 67 23 57 61 14 20 53 40 59 24 44 19 54 51 26
0
 39]
--------------------------------------------------------------------------
--------------------------------------------------
Coluna PhoneService: ['No' 'Yes']
--------------------------------------------------------------------------
--------------------------------------------------
Coluna MultipleLines: ['No phone service' 'No' 'Yes']
--------------------------------------------------------------------------
--------------------------------------------------
Coluna InternetService: ['DSL' 'Fiber optic' 'No']
--------------------------------------------------------------------------
--------------------------------------------------
Coluna OnlineSecurity: ['No' 'Yes' 'No internet service']
--------------------------------------------------------------------------
--------------------------------------------------
Coluna OnlineBackup: ['Yes' 'No' 'No internet service']
--------------------------------------------------------------------------
--------------------------------------------------
Coluna DeviceProtection: ['No' 'Yes' 'No internet service']
--------------------------------------------------------------------------
--------------------------------------------------
Coluna TechSupport: ['No' 'Yes' 'No internet service']
--------------------------------------------------------------------------
--------------------------------------------------
Coluna StreamingTV: ['No' 'Yes' 'No internet service']
--------------------------------------------------------------------------
--------------------------------------------------
Coluna StreamingMovies: ['No' 'Yes' 'No internet service']
--------------------------------------------------------------------------
--------------------------------------------------
Coluna Contract: ['Month-to-month' 'One year' 'Two year']
--------------------------------------------------------------------------
--------------------------------------------------
Coluna PaperlessBilling: ['Yes' 'No']
--------------------------------------------------------------------------
--------------------------------------------------
Coluna PaymentMethod: ['Electronic check' 'Mailed check' 'Bank transfer
(automatic)'
 'Credit card (automatic)']
--------------------------------------------------------------------------
--------------------------------------------------
Coluna MonthlyCharges: [29.85 56.95 53.85 ... 63.1  44.2  78.7 ]
--------------------------------------------------------------------------
--------------------------------------------------
Coluna TotalCharges: ['29.85' '1889.5' '108.15' ... '346.45' '306.6'
'6844.5']
--------------------------------------------------------------------------
--------------------------------------------------
Coluna Churn: ['No' 'Yes']
```

```python
# verificando as dimensões do dataset
print(f'Total de linhas:\t{df.shape[0]}')
print(f'Total de colunas:\t{df.shape[1]}')
```

```
Total de linhas: 7043
Total de colunas:      21
```

```
# verificando quantidade de dados únicos por feature
df.nunique()
```

```
customerID 7043 gender 2 SeniorCitizen 2 Partner 2 Dependents 2 tenure 73
PhoneService 2 MultipleLines 3 InternetService 3 OnlineSecurity 3
OnlineBackup 3 DeviceProtection 3 TechSupport 3 StreamingTV 3
StreamingMovies 3 Contract 3 PaperlessBilling 2 PaymentMethod 4
MonthlyCharges 1585 TotalCharges 6531 Churn 2 dtype: int64
```

```
# verificando os tipos de cada variável do dataset
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7043 entries, 0 to 7042
Data columns (total 21 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   customerID        7043 non-null   object
 1   gender            7043 non-null   object
 2   SeniorCitizen     7043 non-null   int64
 3   Partner           7043 non-null   object
 4   Dependents        7043 non-null   object
 5   tenure            7043 non-null   int64
 6   PhoneService      7043 non-null   object
 7   MultipleLines     7043 non-null   object
 8   InternetService   7043 non-null   object
 9   OnlineSecurity    7043 non-null   object
 10  OnlineBackup      7043 non-null   object
 11  DeviceProtection  7043 non-null   object
 12  TechSupport       7043 non-null   object
 13  StreamingTV       7043 non-null   object
 14  StreamingMovies   7043 non-null   object
 15  Contract          7043 non-null   object
 16  PaperlessBilling  7043 non-null   object
 17  PaymentMethod     7043 non-null   object
 18  MonthlyCharges    7043 non-null   float64
 19  TotalCharges      7043 non-null   object
 20  Churn             7043 non-null   object
dtypes: float64(1), int64(2), object(18)
memory usage: 1.1+ MB
```

```
# criando uma cópia do dataframe
df_clean = df.copy()

# drop da coluna customer ID
df_clean.drop('customerID', axis=1, inplace=True)

# convertendo string TotalCharges para float
df_clean['TotalCharges'] = pd.to_numeric(df_clean['TotalCharges'], er-
rors='coerce')
```

```
# verificando se o dataset possui dados nulos/vazios
df_clean.isnull().sum()
```

```
gender 0 SeniorCitizen 0 Partner 0 Dependents 0 tenure 0 PhoneService 0
MultipleLines 0 InternetService 0 OnlineSecurity 0 OnlineBackup 0
DeviceProtection 0 TechSupport 0 StreamingTV 0 StreamingMovies 0 Contract
0 PaperlessBilling 0 PaymentMethod 0 MonthlyCharges 0 TotalCharges 11
Churn 0 dtype: int64
```

```
# verificando o percentual dos dados nulos/vazios em relação ao total
do dataset
df_clean.isnull().sum() * 100 / len(df_clean)
```

```
gender 0.000000 SeniorCitizen 0.000000 Partner 0.000000 Dependents
0.000000 tenure 0.000000 PhoneService 0.000000 MultipleLines 0.000000
InternetService 0.000000 OnlineSecurity 0.000000 OnlineBackup 0.000000
DeviceProtection 0.000000 TechSupport 0.000000 StreamingTV 0.000000
StreamingMovies 0.000000 Contract 0.000000 PaperlessBilling 0.000000
PaymentMethod 0.000000 MonthlyCharges 0.000000 TotalCharges 0.156183
Churn 0.000000 dtype: float64
```

```
df_clean.loc[df.TotalCharges == " "]
```

```
# eliminando as linhas com dados ausentes
df_clean.drop(index=df.query('TotalCharges == " "').index, axis=0, in-
place=True)

# checando novamente os dados
df_clean.isnull().sum()
```

```
gender 0 SeniorCitizen 0 Partner 0 Dependents 0 tenure 0 PhoneService 0
MultipleLines 0 InternetService 0 OnlineSecurity 0 OnlineBackup 0
DeviceProtection 0 TechSupport 0 StreamingTV 0 StreamingMovies 0 Contract
0 PaperlessBilling 0 PaymentMethod 0 MonthlyCharges 0 TotalCharges 0 Churn
0 dtype: int64
```

```
# verificando os dados por coluna para checar distribuição
for column in df_clean.columns:
  print(f'Coluna {column}: {df_clean[column].unique()}')
  print('------------'*10)
```

```
Coluna gender: ['Female' 'Male']
------------------------------------------------------------------------
------------------------------------------------
Coluna SeniorCitizen: [0 1]
------------------------------------------------------------------------
------------------------------------------------
Coluna Partner: ['Yes' 'No']
------------------------------------------------------------------------
------------------------------------------------
Coluna Dependents: ['No' 'Yes']
------------------------------------------------------------------------
------------------------------------------------
Coluna tenure: [ 1 34  2 45  8 22 10 28 62 13 16 58 49 25 69 52 71 21
12 30 47 72 17 27
```

```
  5 46 11 70 63 43 15 60 18 66  9  3 31 50 64 56  7 42 35 48 29 65 38
68
 32 55 37 36 41  6  4 33 67 23 57 61 14 20 53 40 59 24 44 19 54 51 26
39]
--------------------------------------------------------------------------
--------------------------------------------------
Coluna PhoneService: ['No' 'Yes']
--------------------------------------------------------------------------
--------------------------------------------------
Coluna MultipleLines: ['No phone service' 'No' 'Yes']
--------------------------------------------------------------------------
--------------------------------------------------
Coluna InternetService: ['DSL' 'Fiber optic' 'No']
--------------------------------------------------------------------------
--------------------------------------------------
Coluna OnlineSecurity: ['No' 'Yes' 'No internet service']
--------------------------------------------------------------------------
--------------------------------------------------
Coluna OnlineBackup: ['Yes' 'No' 'No internet service']
--------------------------------------------------------------------------
--------------------------------------------------
Coluna DeviceProtection: ['No' 'Yes' 'No internet service']
--------------------------------------------------------------------------
--------------------------------------------------
Coluna TechSupport: ['No' 'Yes' 'No internet service']
--------------------------------------------------------------------------
--------------------------------------------------
Coluna StreamingTV: ['No' 'Yes' 'No internet service']
--------------------------------------------------------------------------
--------------------------------------------------
Coluna StreamingMovies: ['No' 'Yes' 'No internet service']
--------------------------------------------------------------------------
--------------------------------------------------
Coluna Contract: ['Month-to-month' 'One year' 'Two year']
--------------------------------------------------------------------------
--------------------------------------------------
Coluna PaperlessBilling: ['Yes' 'No']
--------------------------------------------------------------------------
--------------------------------------------------
Coluna PaymentMethod: ['Electronic check' 'Mailed check' 'Bank transfer
(automatic)'
 'Credit card (automatic)']
--------------------------------------------------------------------------
--------------------------------------------------
Coluna MonthlyCharges: [29.85 56.95 53.85 ... 63.1  44.2  78.7 ]
--------------------------------------------------------------------------
--------------------------------------------------
Coluna TotalCharges: [  29.85 1889.5   108.15 ...  346.45  306.6
6844.5 ]
--------------------------------------------------------------------------
--------------------------------------------------
Coluna Churn: ['No' 'Yes']
```

```
df_clean.Churn.value_counts()
No 5163 Yes 1869 Name: Churn, dtype: int64
```

```
# Função auxiliar para marcar porcentagem nos plots
def porcentagem(ax, dados):
  total = float(len(dados))
  for p in ax.patches:
      percentage = f'{(p.get_height()/total)*100:.1f}%'
      x = p.get_x() + p.get_width()/2
      y = p.get_height()
      ax.annotate(percentage, (x, y), fontsize=12, horizontalalign-
ment='center')
```

```
# gráfico de distribuição de churn
fig, ax = plt.subplots(figsize=(7,5))

sns.countplot(x='Churn', data=df_clean, ax=ax)
ax.set_title('Distribuição de Churn', loc='left', fontsize=16, pad=30)
ax.spines['top'].set_visible(False)
ax.spines['right'].set_visible(False)
ax.grid(False)

porcentagem(ax, df_clean)
plt.show();
```

```
df_clean[['MonthlyCharges', 'TotalCharges', 'tenure']].describe()
```

```
fig, ax = plt.subplots(1,2, figsize=(10,5))

sns.boxplot(y=df_clean['MonthlyCharges'], ax=ax[0], showmeans=True)
sns.boxplot(y=df_clean['TotalCharges'], ax=ax[1], showmeans=True)

plt.tight_layout();
```

```
fig, ax = plt.subplots(1,2, figsize=(15,7))

sns.boxplot(x='Churn', y='MonthlyCharges', data=df_clean, ax=ax[0],
showmeans=True)
ax[0].set_title('Boxplot Monthly Charges / Churn', loc='left', font-
size=13, pad=20)
ax[0].spines['right'].set_visible(False)
ax[0].spines['top'].set_visible(False)
ax[0].grid(False)

sns.boxplot(x='Churn', y='TotalCharges', data=df_clean, ax=ax[1], show-
means=True)
ax[1].set_title('Boxplot Total Charges / Churn', loc='left', font-
size=13, pad=20)
ax[1].spines['right'].set_visible(False)
```

```
ax[1].spines['top'].set_visible(False)
ax[1].grid(False)

plt.tight_layout();
```

```
def kdeplot(feature, hist, kde):
    plt.figure(figsize=(9, 4))
    plt.title("Gráfico de densidade - {}".format(feature), loc='left',
fontsize=16, pad=30)
    ax0 = sns.distplot(df_clean[df_clean['Churn'] == 'No'][fea-
ture].dropna(), hist=hist, kde=kde,
             label= 'Churn: No')
    ax0.spines['right'].set_visible(False)
    ax0.spines['top'].set_visible(False)

    ax1 = sns.distplot(df_clean[df_clean['Churn'] == 'Yes'][fea-
ture].dropna(), hist=hist, kde=kde,
             label= 'Churn: Yes')
    ax1.spines['right'].set_visible(False)
    ax1.spines['top'].set_visible(False)

    plt.legend()
```

```
kdeplot('tenure', hist=False, kde=True)
plt.show()
```

```
kdeplot('MonthlyCharges', hist=False, kde=True)
```

```
kdeplot('TotalCharges', hist=False, kde=True)
```

```
# plot de informações dos clientes
fig, ax = plt.subplots(1,4, figsize=(18,4), sharey=True)

sns.countplot(x='gender', data=df_clean, ax=ax[0])
ax[0].set_title('Gender', loc='left', fontsize=13, pad=10)
ax[0].spines['top'].set_visible(False)
ax[0].spines['right'].set_visible(False)
ax[0].grid(False)
porcentagem(ax[0], df_clean)

sns.countplot(x='SeniorCitizen', data=df_clean, ax=ax[1])
ax[1].set_title('SeniorCitizen', loc='left', fontsize=13, pad=10)
ax[1].spines['top'].set_visible(False)
ax[1].spines['right'].set_visible(False)
ax[1].grid(False)
porcentagem(ax[1], df_clean)
```

```
sns.countplot(x='Partner', data=df_clean, ax=ax[2])
ax[2].set_title('Partner', loc='left', fontsize=13, pad=10)
ax[2].spines['top'].set_visible(False)
ax[2].spines['right'].set_visible(False)
ax[2].grid(False)
porcentagem(ax[2], df_clean)

sns.countplot(x='Dependents', data=df_clean, ax=ax[3])
ax[3].set_title('Dependents', loc='left', fontsize=13, pad=10)
ax[3].spines['top'].set_visible(False)
ax[3].spines['right'].set_visible(False)
ax[3].grid(False)
porcentagem(ax[3], df_clean)

plt.tight_layout();
```

```
fig, ax = plt.subplots(1,4, figsize=(18,4), sharey=True)
sns.countplot(x='Churn', data=df_clean, ax=ax[0], hue='gender')
ax[0].set_title('Churn/Gender', loc='left', fontsize=13, pad=30)
ax[0].spines['top'].set_visible(False)
ax[0].spines['right'].set_visible(False)
ax[0].grid(False)
porcentagem(ax[0], df_clean)

sns.countplot(x='Churn', data=df_clean, ax=ax[1], hue='SeniorCitizen')
ax[1].set_title('Churn/SeniorCitizen', loc='left', fontsize=13, pad=30)
ax[1].spines['top'].set_visible(False)
ax[1].spines['right'].set_visible(False)
ax[1].grid(False)
porcentagem(ax[1], df_clean)

sns.countplot(x='Churn', data=df_clean, ax=ax[2], hue='Partner')
ax[2].set_title('Churn/Partner', loc='left', fontsize=13, pad=30)
ax[2].spines['top'].set_visible(False)
ax[2].spines['right'].set_visible(False)
ax[2].grid(False)
porcentagem(ax[2], df_clean)

sns.countplot(x='Churn', data=df_clean, ax=ax[3], hue='Dependents')
ax[3].set_title('Churn/Dependents', loc='left', fontsize=13, pad=30)
ax[3].spines['top'].set_visible(False)
ax[3].spines['right'].set_visible(False)
ax[3].grid(False)
porcentagem(ax[3], df_clean)


plt.tight_layout();
```

```
# informações dos serviços contratados
fig, ax = plt.subplots(3,3, figsize=(14,12), sharey=True)

sns.countplot(x='PhoneService', data=df_clean, ax=ax[0,0])
ax[0,0].set_title('PhoneService', loc='left', fontsize=13, pad=10)
ax[0,0].spines['top'].set_visible(False)
ax[0,0].spines['right'].set_visible(False)
ax[0,0].grid(False)
porcentagem(ax[0,0], df_clean)

sns.countplot(x='MultipleLines', data=df_clean, ax=ax[0,1])
ax[0,1].set_title('MultipleLines', loc='left', fontsize=13, pad=10)
ax[0,1].spines['top'].set_visible(False)
ax[0,1].spines['right'].set_visible(False)
ax[0,1].grid(False)
porcentagem(ax[0,1], df_clean)

sns.countplot(x='InternetService', data=df_clean, ax=ax[0,2])
ax[0,2].set_title('InternetService', loc='left', fontsize=13, pad=10)
ax[0,2].spines['top'].set_visible(False)
ax[0,2].spines['right'].set_visible(False)
ax[0,2].grid(False)
porcentagem(ax[0,2], df_clean)

sns.countplot(x='OnlineSecurity', data=df_clean, ax=ax[1,0])
ax[1,0].set_title('OnlineSecurity', loc='left', fontsize=13, pad=10)
ax[1,0].spines['top'].set_visible(False)
ax[1,0].spines['right'].set_visible(False)
ax[1,0].grid(False)
porcentagem(ax[1,0], df_clean)

sns.countplot(x='OnlineBackup', data=df_clean, ax=ax[1,1])
ax[1,1].set_title('OnlineBackup', loc='left', fontsize=13, pad=10)
ax[1,1].spines['top'].set_visible(False)
ax[1,1].spines['right'].set_visible(False)
ax[1,1].grid(False)
porcentagem(ax[1,1], df_clean)

sns.countplot(x='DeviceProtection', data=df_clean, ax=ax[1,2])
ax[1,2].set_title('DeviceProtection', loc='left', fontsize=13, pad=10)
ax[1,2].spines['top'].set_visible(False)
ax[1,2].spines['right'].set_visible(False)
ax[1,2].grid(False)
porcentagem(ax[1,2], df_clean)

sns.countplot(x='TechSupport', data=df_clean, ax=ax[2,0])
ax[2,0].set_title('TechSupport', loc='left', fontsize=13, pad=10)
```

```
ax[2,0].spines['top'].set_visible(False)
ax[2,0].spines['right'].set_visible(False)
ax[2,0].grid(False)
porcentagem(ax[2,0], df_clean)

sns.countplot(x='StreamingTV', data=df_clean, ax=ax[2,1])
ax[2,1].set_title('StreamingTV', loc='left', fontsize=13, pad=10)
ax[2,1].spines['top'].set_visible(False)
ax[2,1].spines['right'].set_visible(False)
ax[2,1].grid(False)
porcentagem(ax[2,1], df_clean)

sns.countplot(x='StreamingMovies', data=df_clean, ax=ax[2,2])
ax[2,2].set_title('StreamingMovies', loc='left', fontsize=13, pad=10)
ax[2,2].spines['top'].set_visible(False)
ax[2,2].spines['right'].set_visible(False)
ax[2,2].grid(False)
porcentagem(ax[2,2], df_clean)

plt.tight_layout();
```

```
# informações do cliente
fig, ax = plt.subplots(3,3, figsize=(14,12), sharey=True)
sns.countplot(x='PhoneService', data=df_clean, ax=ax[0,0], hue='Churn')
ax[0,0].set_title('Churn/PhoneService', loc='left', fontsize=13,
pad=30)
ax[0,0].spines['top'].set_visible(False)
ax[0,0].spines['right'].set_visible(False)
ax[0,0].grid(False)
porcentagem(ax[0,0], df_clean)

sns.countplot( x='MultipleLines', data=df_clean,ax=ax[0,1],
hue='Churn')
ax[0,1].set_title('Churn/MultipleLines', loc='left', fontsize=13,
pad=30)
ax[0,1].spines['top'].set_visible(False)
ax[0,1].spines['right'].set_visible(False)
ax[0,1].grid(False)
porcentagem(ax[0,1], df_clean)

sns.countplot( x='InternetService', data=df_clean,ax=ax[0,2],
hue='Churn')
ax[0,2].set_title('Churn/InternetService', loc='left', fontsize=13,
pad=30)
ax[0,2].spines['top'].set_visible(False)
ax[0,2].spines['right'].set_visible(False)
ax[0,2].grid(False)
porcentagem(ax[0,2], df_clean)
```

```
sns.countplot( x='OnlineSecurity', data=df_clean,ax=ax[1,0],
hue='Churn')
ax[1,0].set_title('Churn/OnlineSecurity', loc='left', fontsize=13,
pad=30)
ax[1,0].spines['top'].set_visible(False)
ax[1,0].spines['right'].set_visible(False)
ax[1,0].grid(False)
porcentagem(ax[1,0], df_clean)

sns.countplot( x='OnlineBackup', data=df_clean,ax=ax[1,1], hue='Churn')
ax[1,1].set_title('Churn/OnlineBackup', loc='left', fontsize=13,
pad=30)
ax[1,1].spines['top'].set_visible(False)
ax[1,1].spines['right'].set_visible(False)
ax[1,1].grid(False)
porcentagem(ax[1,1], df_clean)

sns.countplot( x='DeviceProtection', data=df_clean,ax=ax[1,2],
hue='Churn')
ax[1,2].set_title('Churn/DeviceProtection', loc='left', fontsize=13,
pad=30)
ax[1,2].spines['top'].set_visible(False)
ax[1,2].spines['right'].set_visible(False)
ax[1,2].grid(False)
porcentagem(ax[1,2], df_clean)

sns.countplot( x='TechSupport', data=df_clean,ax=ax[2,0], hue='Churn')
ax[2,0].set_title('Churn/TechSupport', loc='left', fontsize=13, pad=30)
ax[2,0].spines['top'].set_visible(False)
ax[2,0].spines['right'].set_visible(False)
ax[2,0].grid(False)
porcentagem(ax[2,0], df_clean)

sns.countplot( x='StreamingTV', data=df_clean,ax=ax[2,1], hue='Churn')
ax[2,1].set_title('Churn/StreamingTV', loc='left', fontsize=13, pad=30)
ax[2,1].spines['top'].set_visible(False)
ax[2,1].spines['right'].set_visible(False)
ax[2,1].grid(False)
porcentagem(ax[2,1], df_clean)

sns.countplot( x='StreamingMovies', data=df_clean,ax=ax[2,2],
hue='Churn')
ax[2,2].set_title('Churn/StreamingMovies', loc='left', fontsize=13,
pad=30)
ax[2,2].spines['top'].set_visible(False)
ax[2,2].spines['right'].set_visible(False)
ax[2,2].grid(False)
porcentagem(ax[2,2], df_clean)
```

```
plt.tight_layout();
```

```
fig, ax = plt.subplots(1,3, figsize=(17,4), sharey=True)

sns.countplot(x='Contract', data=df_clean, ax=ax[0])
ax[0].set_title('Contract', loc='left', fontsize=13, pad=10)
ax[0].spines['top'].set_visible(False)
ax[0].spines['right'].set_visible(False)
ax[0].grid(False)
porcentagem(ax[0], df_clean)

sns.countplot(x='PaperlessBilling', data=df_clean, ax=ax[1])
ax[1].set_title('PaperlessBilling', loc='left', fontsize=13, pad=10)
ax[1].spines['top'].set_visible(False)
ax[1].spines['right'].set_visible(False)
ax[1].grid(False)
porcentagem(ax[1], df_clean)

sns.countplot(x='PaymentMethod', data=df_clean, ax=ax[2])
ax[2].set_title('PaymentMethod', loc='left', fontsize=13, pad=10)
ax[2].spines['top'].set_visible(False)
ax[2].spines['right'].set_visible(False)
ax[2].grid(False)
porcentagem(ax[2], df_clean)

plt.tight_layout();
```

```
# plot churn por contrato e tipo de conta
fig, ax = plt.subplots(1,2, figsize=(13,4), sharey=True)

sns.countplot(x='Contract', data=df_clean, ax=ax[0], hue='Churn')
ax[0].set_title('Churn/Contract', loc='left', fontsize=13, pad=30)
ax[0].spines['top'].set_visible(False)
ax[0].spines['right'].set_visible(False)
ax[0].grid(False)
porcentagem(ax[0], df_clean)

sns.countplot(x='PaperlessBilling', data=df_clean, ax=ax[1],
hue='Churn')
ax[1].set_title('Churn/PaperlessBilling', loc='left', fontsize=13,
pad=30)
ax[1].spines['top'].set_visible(False)
ax[1].spines['right'].set_visible(False)
ax[1].grid(False)
porcentagem(ax[1], df_clean)
```

```
plt.tight_layout();
```

```
# plots por tipo de pagamento
fig, ax = plt.subplots(1,4, figsize=(18,4), sharey=True)

sns.countplot(data=df_clean[df_clean.PaymentMethod == 'Mailed che-
ck'].sort_values(by='Contract'), x='Contract', hue='Churn', ax=ax[0])
ax[0].set_title('Mailed check', loc='left', fontsize=13, pad=30)
ax[0].spines['top'].set_visible(False)
ax[0].spines['right'].set_visible(False)
ax[0].grid(False)
porcentagem(ax[0], df_clean)

sns.countplot(data=df_clean[df_clean.PaymentMethod == 'Electronic che-
ck'].sort_values(by='Contract'), x='Contract', hue='Churn', ax=ax[1])
ax[1].set_title('Electronic check', loc='left', fontsize=13, pad=30)
ax[1].spines['top'].set_visible(False)
ax[1].spines['right'].set_visible(False)
ax[1].grid(False)
porcentagem(ax[1], df_clean)

sns.countplot(data=df_clean[df_clean.PaymentMethod == 'Credit card (au-
tomatic)'].sort_values(by='Contract'), x='Contract', hue='Churn',
ax=ax[2])
ax[2].set_title('Credit card (automatic)', loc='left', fontsize=13,
pad=30)
ax[2].spines['top'].set_visible(False)
ax[2].spines['right'].set_visible(False)
ax[2].grid(False)
porcentagem(ax[2], df_clean)

sns.countplot(data=df_clean[df_clean.PaymentMethod == 'Bank transfer
(automatic)'].sort_values(by='Contract'), x='Contract', hue='Churn',
ax=ax[3])
ax[3].set_title('Bank transfer (automatic)', loc='left', fontsize=13,
pad=30)
ax[3].spines['top'].set_visible(False)
ax[3].spines['right'].set_visible(False)
ax[3].grid(False)
porcentagem(ax[3], df_clean)

plt.tight_layout();
```

```
# separando as colunas por tipo de variável
binary_var = df_clean.nunique()[df_clean.nunique() == 2].keys().to-
list()
```

```
num_var = [col for col in df_clean.select_dtypes(['int', 'float']).co-
lumns.tolist() if col not in binary_var]
cat_var = [col for col in df_clean.columns.tolist() if col not in bi-
nary_var + num_var]

# criando uma copia do dataframe
df_proc = df_clean.copy()


# Label Encoding para as variáveis binárias
le = LabelEncoder()
for coluna in binary_var:
  df_proc[coluna] = le.fit_transform(df_proc[coluna])

# Encoding com get_dummies para as colunas categóricas com múltiplas
classes
df_proc = pd.get_dummies(df_proc, columns=cat_var)

# verificando as 5 primeiras entradas do novo dataframe
df_proc.head()
```

```
# plotando a matriz de correlação
fig, ax = plt.subplots(figsize=(14,10))
sns.heatmap(df_proc.corr())

plt.show();
```

```
# serparar os dados entre feature matrix e target vector
X = df_proc.drop('Churn', axis=1)
y = df_proc.Churn

# dividindo os dados entre treino e teste
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3)
```

```
# plot variáveis numéricas antes da padronização
fig, ax = plt.subplots(1,3, figsize=(15,5))

sns.distplot(df_proc['tenure'], ax=ax[0])
sns.distplot(df_proc['MonthlyCharges'], ax=ax[1])
sns.distplot(df_proc['TotalCharges'], ax=ax[2])

plt.show();
```

```
# plot variáveis numéricas depois da padronização
# padronizando
df_scaler = df_proc.copy()
```

```
scaler = StandardScaler()

# fit transform
df_scaler['tenure'] = scaler.fit_transform(df_proc['tenure'].va-
lues.reshape(-1,1))
df_scaler['MonthlyCharges'] = scaler.fit_transform(df_proc['Mon-
thlyCharges'].values.reshape(-1,1))
df_scaler['TotalCharges'] = scaler.fit_transform(df_proc['TotalChar-
ges'].values.reshape(-1,1))

#plot
fig, ax = plt.subplots(1,3, figsize=(15,5))

sns.distplot(df_scaler['tenure'], ax=ax[0])
sns.distplot(df_scaler['MonthlyCharges'], ax=ax[1])
sns.distplot(df_scaler['TotalCharges'], ax=ax[2])

plt.show();
```

```
# função de validação de modelo
def val_model(X, y,clf, quite=False):
  """
    Realiza cross-validation com os dados de treino para determinado
modelo.

  # Arguments
      X: DataFrame, contém as variáveis independetes.
      y: Series, vetor contendo a variável alvo.
      clf: modelo classificador do Scikit-learn.
      quite: bool, indicando se a função deve imprimir os resultados ou
não.

  # Returns
        float, média dos scores da cross-validation.
  """

  X = np.array(X)
  y = np.array(y)

  pipeline1 = make_pipeline(StandardScaler(), clf)
  pipeline2 = make_pipeline(RobustScaler(), clf)
  scores1 = cross_val_score(pipeline1, X, y, scoring='recall')
  scores2 = cross_val_score(pipeline2, X, y, scoring='recall')

  if quite == False:
    print('Recall StandardScaler: {:.4f} (+/- {:.4f})'.format(sco-
res1.mean(), scores1.std() * 2))
```

```
    print('Recall RobustScaler: {:.4f} (+/- {:.4f})'.format(sco-
res2.mean(), scores2.std() * 2))

  return scores1.mean()
```

```
# Instanciando modelos a serem avaliados
rf = RandomForestClassifier()
dt = DecisionTreeClassifier()
sgd = SGDClassifier()
lr = LogisticRegression()
xgb = XGBClassifier()
```

```
# printar o desempenho dos modelos com os dados padronizados
print('Cross-validation RF:')
score_teste1 = val_model(X_train, y_train, rf)
print('\nCross-validation DT:')
score_teste2 = val_model(X_train, y_train, dt)
print('\nCross-validation SGD:')
score_teste3 = val_model(X_train, y_train, sgd)
print('\nCross-validation LR:')
score_teste4 = val_model(X_train, y_train, lr)
print('\nCross-validation XGB:')
score_teste5 = val_model(X_train, y_train, xgb)
```

```
Cross-validation RF:
Recall StandardScaler: 0.4954 (+/- 0.0375)
Recall RobustScaler: 0.4947 (+/- 0.0223)

Cross-validation DT:
Recall StandardScaler: 0.5023 (+/- 0.0459)
Recall RobustScaler: 0.5015 (+/- 0.0421)

Cross-validation SGD:
Recall StandardScaler: 0.4857 (+/- 0.3456)
Recall RobustScaler: 0.4069 (+/- 0.4069)

Cross-validation LR:
C:\Users\Usuário\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\linear_model\_logistic.py:458: ConvergenceWarning:
lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown
in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
Recall StandardScaler: 0.5558 (+/- 0.0441)
Recall RobustScaler: 0.5520 (+/- 0.0541)

Cross-validation XGB:
```

```
Recall StandardScaler: 0.5283 (+/- 0.0467)
Recall RobustScaler: 0.5267 (+/- 0.0446)
```

```
# importar metodos para realizacao do feature scaling com fit nos dados
de treino
scaler = StandardScaler()
Rob_scaler = RobustScaler()

# padronizando os dados de treino
X_train_scaled = scaler.fit_transform(X_train)
X_train_Rscaled = Rob_scaler.fit_transform(X_train)

# balanceamento RUS
rus = RandomUnderSampler()
X_train_rus_scaled, y_train_rus_scaled = rus.fit_resample(X_train_sca-
led, y_train)
X_train_rus_Rscaled, y_train_rus_Rscaled = rus.fit_resam-
ple(X_train_Rscaled, y_train)

# balanceamento SMOTE
smo = SMOTE()
X_train_smo_scaled, y_train_smo_scaled = smo.fit_resample(X_train_sca-
led, y_train)
X_train_smo_Rscaled, y_train_smo_Rscaled = smo.fit_resam-
ple(X_train_Rscaled, y_train)


# balanceamento ADASYN
ada = ADASYN()
X_train_ada_scaled, y_train_ada_scaled = ada.fit_resample(X_train_sca-
led, y_train)
X_train_ada_Rscaled, y_train_ada_Rscaled = ada.fit_resam-
ple(X_train_Rscaled, y_train)
```

```
# Checando o balanceamento das classes
print(pd.Series(y_train_rus_scaled).value_counts())

fig, ax = plt.subplots()
sns.countplot(y_train_rus_scaled)
plt.title('Balanceamento dos Dados com Under-Sampling -RUS')

plt.show();
```

```
# Checando o balanceamento das classes
print(pd.Series(y_train_smo_scaled).value_counts())

fig, ax = plt.subplots()
```

```
sns.countplot(y_train_smo_scaled)
plt.title('Balanceamento dos Dados com Over-Sampling -SMOTE')

plt.tight_layout()
```

```
# Checando o balanceamento das classes
print(pd.Series(y_train_ada_scaled).value_counts())

fig, ax = plt.subplots()

sns.countplot(y_train_ada_scaled)
plt.title('Balanceamento dos Dados com Over-Sampling -ADASYN')


plt.tight_layout();
```

```
# # Definindo funçao de validação com dados balanceados
def val_model_balanced(X, y,clf, quite=False):

  X = np.array(X)
  y = np.array(y)

  scores = cross_val_score(clf, X, y, scoring='recall')

  if quite == False:
    print('Recall: {:.4f} (+/- {:.4f})'.format(scores.mean(), sco-
res.std() * 2))
  return scores.mean()
```

```
# Cross-Validation com Balanceamento RUS e Standard Scaler
print('Cross-validation RF:')
score_teste_rf_rus_scaled = val_model_balanced(X_train_rus_scaled,
y_train_rus_scaled, rf)
print('\nCross-validation DT:')
score_teste_dt_rus_scaled = val_model_balanced(X_train_rus_scaled,
y_train_rus_scaled, dt)
print('\nCross-validation SGD:')
score_teste_sgd_rus_scaled = val_model_balanced(X_train_rus_scaled,
y_train_rus_scaled, sgd)
print('\nCross-validation LR:')
score_teste_lr_rus_scaled = val_model_balanced(X_train_rus_scaled,
y_train_rus_scaled, lr)
print('\nCross-validation XGB:')
score_teste_xgb_rus_scaled = val_model_balanced(X_train_rus_scaled,
y_train_rus_scaled, xgb)
```

```
Cross-validation RF:
Recall: 0.7523 (+/- 0.0551)
```

```
Cross-validation DT:
Recall: 0.6858 (+/- 0.0426)

Cross-validation SGD:
Recall: 0.7844 (+/- 0.1514)

Cross-validation LR:
Recall: 0.7974 (+/- 0.0483)

Cross-validation XGB:
Recall: 0.7470 (+/- 0.0529)
```

```
# Cross-Validation com Balanceamento RUS e RobustScaler
print('Cross-validation RF:')
score_teste_rf_rus_Rscaled = val_model_balanced(X_train_rus_Rscaled,
y_train_rus_Rscaled, rf)
print('\nCross-validation DT:')
score_teste_dt_rus_Rscaled = val_model_balanced(X_train_rus_Rscaled,
y_train_rus_Rscaled, dt)
print('\nCross-validation SGD:')
score_teste_sgd_rus_Rscaled = val_model_balanced(X_train_rus_Rscaled,
y_train_rus_Rscaled, sgd)
print('\nCross-validation LR:')
score_teste_lr_rus_Rscaled = val_model_balanced(X_train_rus_Rscaled,
y_train_rus_Rscaled, lr)
print('\nCross-validation XGB:')
score_teste_xgb_rus_Rscaled = val_model_balanced(X_train_rus_Rscaled,
y_train_rus_Rscaled, xgb)
```

```
Cross-validation RF:
Recall: 0.7798 (+/- 0.0399)

Cross-validation DT:
Recall: 0.6842 (+/- 0.0496)

Cross-validation SGD:
Recall: 0.8753 (+/- 0.1593)

Cross-validation LR:
Recall: 0.8058 (+/- 0.0454)

Cross-validation XGB:
Recall: 0.8226 (+/- 0.0511)
```

```
# Imprimir DataSet com os Dados de Recall dos Modelos Escolhidos
model = []
recall= []

model.append('RF RusScaled')
recall.append(score_teste_rf_rus_scaled)
model.append('RF RusRscaled')
recall.append(score_teste_rf_rus_Rscaled)
model.append('DT RusScaled')
recall.append(score_teste_dt_rus_scaled)
model.append('DT RusRscaled')
```

```
recall.append(score_teste_dt_rus_Rscaled)
model.append('SGD RusScaled')
recall.append(score_teste_sgd_rus_scaled)
model.append('SGD RusRscaled')
recall.append(score_teste_sgd_rus_Rscaled)
model.append('LR RusScaled')
recall.append(score_teste_lr_rus_scaled)
model.append('LR RusRscaled')
recall.append(score_teste_lr_rus_Rscaled)
model.append('XGBoost RusScaled')
recall.append(score_teste_xgb_rus_scaled)
model.append('XGBoost RusRscaled')
recall.append(score_teste_xgb_rus_Rscaled)

pd.DataFrame(data=recall, index=model, columns=['Recall'])
```

```
# Cross-Validation com Balanceamento SMOTE e Standard Scaler
print('Cross-validation RF:')
score_teste_rf_smo_scaled = val_model_balanced(X_train_smo_scaled,
y_train_smo_scaled, rf)
print('\nCross-validation DT:')
score_teste_dt_smo_scaled = val_model_balanced(X_train_smo_scaled,
y_train_smo_scaled, dt)
print('\nCross-validation SGD:')
score_teste_sgd_smo_scaled = val_model_balanced(X_train_smo_scaled,
y_train_smo_scaled, sgd)
print('\nCross-validation LR:')
score_teste_lr_smo_scaled = val_model_balanced(X_train_smo_scaled,
y_train_smo_scaled, lr)
print('\nCross-validation XGB:')
score_teste_xgb_smo_scaled = val_model_balanced(X_train_smo_scaled,
y_train_smo_scaled, xgb)
```

```
Cross-validation RF:
Recall: 0.8727 (+/- 0.2505)

Cross-validation DT:
Recall: 0.7939 (+/- 0.2400)

Cross-validation SGD:
Recall: 0.7839 (+/- 0.0618)

Cross-validation LR:
Recall: 0.8049 (+/- 0.0552)

Cross-validation XGB:
Recall: 0.8573 (+/- 0.3011)
```

```
# Cross-Validation com Balanceamento SMOTE e Robust Scaler
print('Cross-validation RF:')
score_teste_rf_smo_Rscaled = val_model_balanced(X_train_smo_Rscaled,
y_train_smo_Rscaled, rf)
```

```
print('\nCross-validation DT:')
score_teste_dt_smo_Rscaled = val_model_balanced(X_train_smo_Rscaled,
y_train_smo_Rscaled, dt)
print('\nCross-validation SGD:')
score_teste_sgd_smo_Rscaled = val_model_balanced(X_train_smo_Rscaled,
y_train_smo_Rscaled, sgd)
print('\nCross-validation LR:')
score_teste_lr_smo_Rscaled = val_model_balanced(X_train_smo_Rscaled,
y_train_smo_Rscaled, lr)
print('\nCross-validation XGB:')
score_teste_xgb_smo_Rscaled = val_model_balanced(X_train_smo_Rscaled,
y_train_smo_Rscaled, xgb)
```

```
Cross-validation RF:
Recall: 0.8700 (+/- 0.2555)

Cross-validation DT:
Recall: 0.7989 (+/- 0.2130)

Cross-validation SGD:
Recall: 0.8046 (+/- 0.1695)

Cross-validation LR:
/usr/local/lib/python3.8/dist-packages/sklearn/linear_model/_logis-
tic.py:814: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown
in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
Recall: 0.8069 (+/- 0.0599)

Cross-validation XGB:
Recall: 0.8467 (+/- 0.3062)
```

```
# Imprimir DataSet com os Dados de Recall dos Modelos Escolhidos
model = []
recall= []

model.append('RF SMOScaled')
recall.append(score_teste_rf_smo_scaled)
model.append('RF SMORscaled')
recall.append(score_teste_rf_smo_Rscaled)
model.append('DT SMOScaled')
recall.append(score_teste_dt_smo_scaled)
model.append('DT SMORscaled')
recall.append(score_teste_dt_smo_Rscaled)
model.append('SGD SMOScaled')
recall.append(score_teste_sgd_smo_scaled)
model.append('SGD SMORscaled')
```

```
recall.append(score_teste_sgd_smo_Rscaled)
model.append('LR SMOScaled')
recall.append(score_teste_lr_smo_scaled)
model.append('LR SMORscaled')
recall.append(score_teste_lr_smo_Rscaled)
model.append('XGBoost SMOScaled')
recall.append(score_teste_xgb_smo_scaled)
model.append('XGBoost SMORscaled')
recall.append(score_teste_xgb_smo_Rscaled)

pd.DataFrame(data=recall, index=model, columns=['Recall'])
```

```
# Cross-Validation com Balanceamento ADASYN e Standard Scaler
print('Cross-validation RF:')
score_teste_rf_ada_scaled = val_model_balanced(X_train_ada_scaled,
y_train_ada_scaled, rf)
print('\nCross-validation DT:')
score_teste_dt_ada_scaled = val_model_balanced(X_train_ada_scaled,
y_train_ada_scaled, dt)
print('\nCross-validation SGD:')
score_teste_sgd_ada_scaled = val_model_balanced(X_train_ada_scaled,
y_train_ada_scaled, sgd)
print('\nCross-validation LR:')
score_teste_lr_ada_scaled = val_model_balanced(X_train_ada_scaled,
y_train_ada_scaled, lr)
print('\nCross-validation XGB:')
score_teste_xgb_ada_scaled = val_model_balanced(X_train_ada_scaled,
y_train_ada_scaled, xgb)
```

```
Cross-validation RF:
Recall: 0.8683 (+/- 0.2295)

Cross-validation DT:
Recall: 0.7704 (+/- 0.2198)

Cross-validation SGD:
Recall: 0.7826 (+/- 0.1232)

Cross-validation LR:
Recall: 0.7960 (+/- 0.0875)

Cross-validation XGB:
Recall: 0.8461 (+/- 0.3104)
```

```
# Cross-Validation com Balanceamento ADASYN e Robust Scaler
print('Cross-validation RF:')
score_teste_rf_ada_Rscaled = val_model_balanced(X_train_ada_Rscaled,
y_train_ada_Rscaled, rf)
print('\nCross-validation DT:')
score_teste_dt_ada_Rscaled = val_model_balanced(X_train_ada_Rscaled,
y_train_ada_Rscaled, dt)
print('\nCross-validation SGD:')
```

```
score_teste_sgd_ada_Rscaled = val_model_balanced(X_train_ada_Rscaled,
y_train_ada_Rscaled, sgd)
print('\nCross-validation LR:')
score_teste_lr_ada_Rscaled = val_model_balanced(X_train_ada_Rscaled,
y_train_ada_Rscaled, lr)
print('\nCross-validation XGB:')
score_teste_xgb_ada_Rscaled = val_model_balanced(X_train_ada_Rscaled,
y_train_ada_Rscaled, xgb)
```

```
Cross-validation RF:
Recall: 0.8673 (+/- 0.2263)

Cross-validation DT:
Recall: 0.7521 (+/- 0.2254)

Cross-validation SGD:
Recall: 0.7882 (+/- 0.2321)

Cross-validation LR:
/usr/local/lib/python3.8/dist-packages/sklearn/linear_model/_logis-
tic.py:814: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown
in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
Recall: 0.7910 (+/- 0.1121)

Cross-validation XGB:
Recall: 0.8470 (+/- 0.3122)
```

```
# Imprimir DataSet com os Dados de Recall dos Modelos Escolhidos
model = []
recall= []

model.append('RF ADAScaled')
recall.append(score_teste_rf_ada_scaled)
model.append('RF ADARscaled')
recall.append(score_teste_rf_ada_Rscaled)
model.append('DT ADAScaled')
recall.append(score_teste_dt_ada_scaled)
model.append('DT ADARscaled')
recall.append(score_teste_dt_ada_Rscaled)
model.append('SGD ADAScaled')
recall.append(score_teste_sgd_ada_scaled)
model.append('SGD ADARscaled')
recall.append(score_teste_sgd_ada_Rscaled)
model.append('LR ADAScaled')
recall.append(score_teste_lr_ada_scaled)
model.append('LR ADARscaled')
```

```
recall.append(score_teste_lr_ada_Rscaled)
model.append('XGBoost ADAScaled')
recall.append(score_teste_xgb_ada_scaled)
model.append('XGBoost ADARscaled')
recall.append(score_teste_xgb_ada_Rscaled)

pd.DataFrame(data=recall, index=model, columns=['Recall'])
```

```
# Imprimir DataSet com os Dados de Recall dos Modelos Escolhidos
model = []
recall= []

model.append('XGBoost RusRscaled')
recall.append(score_teste_xgb_rus_Rscaled)
model.append('XGBoost SMOScaled')
recall.append(score_teste_xgb_smo_scaled)
model.append('XGBoost ADARScaled')
recall.append(score_teste_xgb_ada_Rscaled)

pd.DataFrame(data=recall, index=model, columns=['Recall'])
```

```
# modelo a ser otimizado
xgb_rus_Rscaled = XGBClassifier(learning_rate=0.1)

# parâmetros a serem otimizados
param_grid = {
      'n_estimators':[0,10,50,100],
}

# identificar os melhores parâmetros
o
grid_search = GridSearchCV(xgb_rus_Rscaled, param_grid, scoring='re-
call', n_jobs=-1, cv=kfold)
grid_result = grid_search.fit(X_train_rus_Rscaled, y_train_rus_Rscaled)

# ver resultados
print(f'Melhor resultado: {grid_result.best_score_} para {grid_re-
sult.best_params_}')
```

```
Melhor resultado: 0.8249559600704639 para {'n_estimators': 10}
```

```
xgb_rus_Rscaled = XGBClassifier(learning_rate=0.1, n_estimators=10)

param_grid = {
 'max_depth': range(1,4,1),
 'min_child_weight': range(1,4,1)
}

# identificar melhor parâmetro
```

```
kfold = StratifiedKFold(n_splits=10, shuffle=True)
grid_search = GridSearchCV(xgb_rus_Rscaled, param_grid, scoring="re-
call", n_jobs=-1, cv=kfold)
grid_result = grid_search.fit(X_train_rus_Rscaled, y_train_rus_Rscaled)

# ver resultados
print("Melhor: {} para {}".format(grid_result.best_score_, grid_re-
sult.best_params_))
```

```
Melhor: 0.8914445096887846 para {'max_depth': 1, 'min_child_weight': 1}
```

```
# Modelo a ser otimizado
xgb_rus_Rscaled = XGBClassifier(learning_rate=0.1, n_estimators=10,
max_depth=1, min_child_weight=1)

# Parâmetros a serem otimizados
param_grid = {
    'gamma':[0, 0.5, 1, 3]
}

# Identificar melhores parâmetros
kfold = StratifiedKFold(n_splits=10, shuffle=True)
grid_search = GridSearchCV(xgb_rus_Rscaled, param_grid, scoring='re-
call', n_jobs=-1, cv=kfold)
grid_result = grid_search.fit(X_train_rus_Rscaled, y_train_rus_Rscaled)

# Ver resultados
print(f'Melhor: {grid_result.best_score_} para {grid_result.best_pa-
rams_}')
```

```
Melhor: 0.8914679976512037 para {'gamma': 0}
```

```
xgb_rus_Rscaled = XGBClassifier(n_estimators=50, max_depth=1,
min_child_weight=1, gamma=0)

param_grid = {
 'learning_rate':[0.001, 0.01, 0.015, 0.1]
}

# identificar melhor parâmetro
kfold = StratifiedKFold(n_splits=10, shuffle=True)
grid_search = GridSearchCV(xgb_rus_Rscaled, param_grid, scoring="re-
call", n_jobs=-1, cv=kfold)
grid_result = grid_search.fit(X_train_rus_Rscaled, y_train_rus_Rscaled)

# ver resultados
print("Melhor: {} para {}".format(grid_result.best_score_, grid_re-
sult.best_params_))
```

```
Melhor: 0.8914034057545509 para {'learning_rate': 0.001}
```

```
# modelo final
xgb_rus_Rscaled = XGBClassifier(learning_rate=0.001 , n_estimators=50,
max_depth=1, min_child_weight=1, gamma=0)
```

```
# treinando o modelo
xgb_rus_Rscaled.fit(X_train_rus_Rscaled, y_train_rus_Rscaled)

# fazer a previsão
X_test_Rscaled = scaler.transform(X_test)
X_test_scaled = scaler.transform(X_test)
y_pred_xgb_rus_Rscaled = xgb_rus_Rscaled.predict(X_test_Rscaled)

# Classification Report
print(classification_report(y_test, y_pred_xgb_rus_Rscaled))

# imprimir a área sob a curva e Recall score
print("AUC: {:.4f}\n".format(roc_auc_score(y_test, y_pred_xgb_rus_Rsca-
led)))
print(f'Recall:\t\t{recall_score(y_test, y_pred_xgb_rus_Rscaled,
pos_label=1):0.4f}')

# plotar matriz de confusão
plot_confusion_matrix(y_test, y_pred_xgb_rus_Rscaled, normalize=True)
plt.show();
```

```
precision    recall  f1-score   support

           0       0.92      0.55      0.69      1549
           1       0.41      0.87      0.56       561

    accuracy                           0.64      2110
   macro avg       0.67      0.71      0.62      2110
weighted avg       0.79      0.64      0.65      2110

AUC: 0.7108
```

```
# modelo a ser otimizado
xgb_smo_scaled = XGBClassifier(learning_rate=0.1)

# parâmetros a serem otimizados
param_grid = {
      'n_estimators':[0,10,50,100],
}

# identificar os melhores parâmetros
kfold = StratifiedKFold(n_splits=10, shuffle=True)
grid_search = GridSearchCV(xgb_smo_scaled, param_grid, scoring='re-
call', n_jobs=-1, cv=kfold)
grid_result = grid_search.fit(X_train_smo_scaled, y_train_smo_scaled)

# ver resultados
print(f'Melhor resultado: {grid_result.best_score_} para {grid_re-
sult.best_params_}')
```

```
Melhor resultado: 0.8774299444452947 para {'n_estimators': 50}
```

```
xgb_smo_scaled = XGBClassifier(learning_rate=0.1, n_estimators=50)
```

```
param_grid = {
 'max_depth': range(1,4,1),
 'min_child_weight': range(1,4,1)
}

# identificar melhor parâmetro
kfold = StratifiedKFold(n_splits=10, shuffle=True)
grid_search = GridSearchCV(xgb_smo_scaled, param_grid, scoring="re-
call", n_jobs=-1, cv=kfold)
grid_result = grid_search.fit(X_train_smo_scaled, y_train_smo_scaled)

# ver resultados
print("Melhor: {} para {}".format(grid_result.best_score_, grid_re-
sult.best_params_))
```

```
Melhor: 0.8773985705759019 para {'max_depth': 3, 'min_child_weight': 3}
```

```
# Modelo a ser otimizado
xgb_smo_scaled = XGBClassifier(learning_rate=0.1, n_estimators=50,
max_depth=3, min_child_weight=3)

# Parâmetros a serem otimizados
param_grid = {
    'gamma':[0, 0.5, 1, 3]
}

# Identificar melhores parâmetros
kfold = StratifiedKFold(n_splits=10, shuffle=True)
grid_search = GridSearchCV(xgb_smo_scaled, param_grid, scoring='re-
call', n_jobs=-1, cv=kfold)
grid_result = grid_search.fit(X_train_smo_scaled, y_train_smo_scaled)

# Ver resultados
print(f'Melhor: {grid_result.best_score_} para {grid_result.best_pa-
rams_}')
```

```
Melhor: 0.8777077179718706 para {'gamma': 1}
```

```
xgb_smo_scaled = XGBClassifier(n_estimators=50, max_depth=3,
min_child_weight=3, gamma=1)

param_grid = {
 'learning_rate':[0.001, 0.01, 0.015, 0.1]
}

# identificar melhor parâmetro
kfold = StratifiedKFold(n_splits=10, shuffle=True)
grid_search = GridSearchCV(xgb_smo_scaled, param_grid, scoring="re-
call", n_jobs=-1, cv=kfold)
grid_result = grid_search.fit(X_train_smo_scaled, y_train_smo_scaled)
```

```
# ver resultados
print("Melhor: {} para {}".format(grid_result.best_score_, grid_re-
sult.best_params_))
```

```
Melhor: 0.8730077592935522 para {'learning_rate': 0.1}
```

```
# modelo final
xgb_smo_scaled = XGBClassifier(learning_rate=0.1 , n_estimators=50,
max_depth=3, min_child_weight=3, gamma=1)

# treinando o modelo
xgb_smo_scaled.fit(X_train_smo_scaled, y_train_smo_scaled)

# fazer a previsão
# X_test_Rscaled = scaler.transform(X_test)
# X_test_scaled = scaler.transform(X_test)
y_pred_xgb_smo_scaled = xgb_smo_scaled.predict(X_test_scaled)

# Classification Report
print(classification_report(y_test, y_pred_xgb_smo_scaled))

# imprimir a área sob a curva e Recall score
print("AUC: {:.4f}\n".format(roc_auc_score(y_test, y_pred_xgb_smo_sca-
led)))
print(f'Recall:\t\t{recall_score(y_test, y_pred_xgb_smo_scaled, pos_la-
bel=1):0.4f}')

# plotar matriz de confusão
plot_confusion_matrix(y_test, y_pred_xgb_smo_scaled, normalize=True)
plt.show();
```

```
precision    recall  f1-score   support

           0       0.89      0.77      0.83      1549
           1       0.54      0.75      0.63       561

    accuracy                           0.77      2110
   macro avg       0.72      0.76      0.73      2110
weighted avg       0.80      0.77      0.78      2110

AUC: 0.7599
```

```
# modelo a ser otimizado
xgb_ada_Rscaled = XGBClassifier(learning_rate=0.1)

# parâmetros a serem otimizados
param_grid = {
      'n_estimators':[0,10,50,100],
}

# identificar os melhores parâmetros
kfold = StratifiedKFold(n_splits=10, shuffle=True)
grid_search = GridSearchCV(xgb_ada_Rscaled, param_grid, scoring='re-
call', n_jobs=-1, cv=kfold)
```

```
grid_result = grid_search.fit(X_train_ada_Rscaled, y_train_ada_Rscaled)

# ver resultados
print(f'Melhor resultado: {grid_result.best_score_} para {grid_re-
sult.best_params_}')
```
```
Melhor resultado: 0.8812248357702904 para {'n_estimators': 50}
```
```
xgb_ada_Rscaled = XGBClassifier(learning_rate=0.1, n_estimators=50)

param_grid = {
 'max_depth': range(1,4,1),
 'min_child_weight': range(1,4,1)
}

# identificar melhor parâmetro
kfold = StratifiedKFold(n_splits=10, shuffle=True)
grid_search = GridSearchCV(xgb_ada_Rscaled, param_grid, scoring="re-
call", n_jobs=-1, cv=kfold)
grid_result = grid_search.fit(X_train_ada_Rscaled, y_train_ada_Rscaled)

# ver resultados
print("Melhor: {} para {}".format(grid_result.best_score_, grid_re-
sult.best_params_))
```
```
Melhor: 0.8782111827566373 para {'max_depth': 3, 'min_child_weight': 1}
```
```
# Modelo a ser otimizado
xgb_ada_Rscaled = XGBClassifier(learning_rate=0.1, n_estimators=50,
max_depth=3, min_child_weight=1)

# Parâmetros a serem otimizados
param_grid = {
    'gamma':[0, 0.5, 1, 3]
}

# Identificar melhores parâmetros
kfold = StratifiedKFold(n_splits=10, shuffle=True)
grid_search = GridSearchCV(xgb_ada_Rscaled, param_grid, scoring='re-
call', n_jobs=-1, cv=kfold)
grid_result = grid_search.fit(X_train_ada_Rscaled, y_train_ada_Rscaled)

# Ver resultados
print(f'Melhor: {grid_result.best_score_} para {grid_result.best_pa-
rams_}')
```
```
Melhor: 0.8798436412072774 para {'gamma': 0}
```
```
xgb_ada_Rscaled = XGBClassifier(n_estimators=50, max_depth=3,
min_child_weight=1, gamma=0)

param_grid = {
 'learning_rate':[0.001, 0.01, 0.015, 0.1]
}
```

```
# identificar melhor parâmetro
kfold = StratifiedKFold(n_splits=10, shuffle=True)
grid_search = GridSearchCV(xgb_ada_Rscaled, param_grid, scoring="re-
call", n_jobs=-1, cv=kfold)
grid_result = grid_search.fit(X_train_ada_Rscaled, y_train_ada_Rscaled)

# ver resultados
print("Melhor: {} para {}".format(grid_result.best_score_, grid_re-
sult.best_params_))
```

```
Melhor: 0.878479096660915 para {'learning_rate': 0.1}
```

```
# modelo final
xgb_ada_Rscaled = XGBClassifier(learning_rate=0.1 , n_estimators=50,
max_depth=3, min_child_weight=3, gamma=3)

# treinando o modelo
xgb_ada_Rscaled.fit(X_train_ada_Rscaled, y_train_ada_Rscaled)

# fazer a previsão
# X_test_Rscaled = scaler.transform(X_test)
# X_test_scaled = scaler.transform(X_test)
y_pred_xgb_ada_Rscaled = xgb_ada_Rscaled.predict(X_test_Rscaled)

# Classification Report
print(classification_report(y_test, y_pred_xgb_ada_Rscaled))

# imprimir a área sob a curva e Recall score
print("AUC: {:.4f}\n".format(roc_auc_score(y_test, y_pred_xgb_ada_Rsca-
led)))
print(f'Recall:\t\t{recall_score(y_test, y_pred_xgb_ada_Rscaled,
pos_label=1):0.4f}')

# plotar matriz de confusão
plot_confusion_matrix(y_test, y_pred_xgb_ada_Rscaled, normalize=True)
plt.show();
```

```
precision    recall  f1-score   support

           0       0.91      0.68      0.78      1549
           1       0.48      0.82      0.61       561

    accuracy                           0.72      2110
   macro avg       0.70      0.75      0.69      2110
weighted avg       0.80      0.72      0.74      2110

AUC: 0.7504
```

```
# comparando os modelos finais
print('Métricas - XGBoost_Rscaled - Random Under Sampling')
print(f'Recall:\t\t{recall_score(y_test, y_pred_xgb_rus_Rscaled,
pos_label=1):0.4f}')
print(f'AUC:\t\t{roc_auc_score(y_test, y_pred_xgb_rus_Rscaled):0.4f}')
```

```
print('\nMétricas - XGBoost_scaled - SMOTE')
print(f'Recall:\t\t{recall_score(y_test, y_pred_xgb_smo_scaled, pos_la-
bel=1):0.4f}')
print(f'AUC:\t\t{roc_auc_score(y_test, y_pred_xgb_smo_scaled):0.4f}')

print('\nMétricas - XGBoost_rscaled - ADASYN')
print(f'Recall:\t\t{recall_score(y_test, y_pred_xgb_ada_Rscaled,
pos_label=1):0.4f}')
print(f'AUC:\t\t{roc_auc_score(y_test, y_pred_xgb_ada_Rscaled):0.4f}')
```

```
Métricas - XGBoost_Rscaled - Random Under Sampling
Recall:         0.8717
AUC:        0.7108

Métricas - XGBoost_scaled - SMOTE
Recall:         0.7451
AUC:        0.7599

Métricas - XGBoost_rscaled - ADASYN
Recall:         0.8164
AUC:        0.7504
```

```
from scikitplot.metrics import confusion_matrix
import sklearn
from matplotlib import colorbar


fig, ax = plt.subplots(1, 3,figsize=(17,5))

skplt.metrics.plot_confusion_matrix(y_test, y_pred_xgb_rus_Rscaled,
normalize=True, title='XGBoost_Rscaled - Random Under Sampling',
ax=ax[1])
skplt.metrics.plot_confusion_matrix(y_test, y_pred_xgb_smo_scaled, nor-
malize=True, title='XGBoost_scaled - SMOTE',ax=ax[2], cmap= plt.cm.Gre-
ens)
skplt.metrics.plot_confusion_matrix(y_test, y_pred_xgb_ada_Rscaled,
normalize=True, title='XGBoost_rscaled - ADASYN',ax=ax[0], cmap=
plt.cm.Purples)


plt.tight_layout();
```