

Projeto: Reduzindo os cancelamentos - Telecomunicações

Autora: Georgia A. Cavallaro

Data: 06/03/2025

Objetivo

Desenvolver um modelo de Machine Learning para a empresa de comunicação *Interconnect*, visando identificar clientes propensos ao cancelamento e oferecer incentivos para retenção.

- **Métrica principal:** AUC-ROC > 0.75
- **Métrica adicional:** Acurácia

Serviços oferecidos:

- **Principais:** Telefonia fixa e Internet (*DSL ou fibra óptica*).
- **Secundários:** Segurança na Internet, suporte técnico, armazenamento na nuvem e streamings (*TV e filmes*).

Tabela de conteúdo

- [Objetivo](#)
- [Configurando ambiente de trabalho](#)
- [Importando Dados](#)
- [Entendendo os dataframes](#)
- [Análise Exploratória](#)
- [Plano de Resolução da Tarefa](#)
- [Solução da tarefa](#)
- [Testando o modelo com conjunto de dados completo](#)
- [Criando um pipeline para utilização do modelo recomendado](#)
- [Conclusão](#)

Configurando ambiente de trabalho

Tabela de conteúdo

- [Importando bibliotecas](#)
- [Suprimindo avisos desnecessários](#)
- [Personalizando opções de visualização de retorno](#)
- [Banco de Funções úteis](#)

Importando bibliotecas

```
In [2]: import timeit
import warnings
import zipfile
```

```
import pandas as pd
import seaborn as sns
import numpy as np

from tqdm import tqdm
from itertools import product
from matplotlib import pyplot as plt

from sklearn.preprocessing import OneHotEncoder, RobustScaler, OrdinalEncoder
from sklearn.model_selection import train_test_split, StratifiedKFold, cross_val_score
from sklearn.metrics import roc_auc_score, accuracy_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier

from catboost import CatBoostClassifier
```

Suprimindo avisos desnecessários

```
In [ ]: pd.options.mode.chained_assignment = None
warnings.simplefilter("ignore", category=FutureWarning)
```

Personalizando opções de visualização de retorno

```
In [ ]: pd.set_option("display.max_columns", None)
pd.set_option("display.max_colwidth", None)
```

Banco de Funções úteis

Todas as funções utilizadas neste caderno são armazenadas aqui.

Tabela de conteúdo:

- Função para exploração estrutural de pastas de dados
- Função para plotagem de histogramas de dados temporais
- Função para criação de gráficos em grid com o mesmo objetivo em comum
- Função para realizar Validação Cruzada em modelos

Função para exploração estrutural de pastas de dados

Esta função ajuda a entender a localização dos arquivos de interesse sem necessariamente precisar visualizar ou mesmo descompactar a pasta com os dados.

```
In [ ]: def show_structure(structure, level=0):
    indent = " " * 4 * level
    for key in sorted(structure):
        print(f"{indent}📁 {key}" if structure[key] else f"{indent}📄 {key}")
        show_structure(structure[key], level + 1)
```

Função para plotagem de histogramas de dados temporais

```
In [ ]: def plot_date_histogram(
    df,
    title="Distribuição de Churns ao Longo do Tempo",
    column="BeginDate",
    bins=30,
    xlabel="Data de Início do Contrato",
```

```

        ylabel="Número de Cancelamentos",
    ):
        """
        Plota um histograma com KDE de datas de início de contrato para clientes que cancelaram.

        Parâmetros:
        df (pandas.DataFrame): DataFrame contendo os dados a serem analisados
        title (str): Título do gráfico
        column (str): Coluna a ser utilizada pelo eixo x
        bins (int): Número de intervalos para o histograma
        xlabel (str): Legenda do eixo x
        ylabel (str): Legenda do eixo y
        title (str): Título do gráfico
        """

        sns.histplot(
            data=df,
            x=column,
            bins=bins,
            edgecolor="black",
            alpha=0.3,
            kde=True,
            stat="count",
        )

        plt.title(title, fontsize=14, pad=20)
        plt.xlabel(xlabel, fontsize=12)
        plt.ylabel(ylabel, fontsize=12)
        plt.xticks(rotation=45)
        plt.tight_layout()

        return plt

```

Função para criação de gráficos em grid com o mesmo objetivo em comum

```

In [ ]: def target_tax_barplot(df, column, target="Churn", ax=None):
        churns = df.groupby([column, target]).size().unstack(fill_value=0)
        churns.columns = ["0", "1"]
        churn_tax = churns.apply(lambda x: x["1"] / (x["0"] + x["1"]), axis=1).reset_index(
            name="churn_tax"
        )

        sns.barplot(
            data=churn_tax, x=column, y="churn_tax", edgecolor="black", alpha=0.7, ax=ax
        )
        ax.set_title(f"Taxa de {target} para cada valor de {column}")
        ax.set_ylabel("Cancelamentos")

        ticks = ax.get_xticks()
        ax.set_xticks(ticks)
        ax.set_xticklabels(ax.get_xticklabels(), rotation=10)

```

Função para realizar Validação Cruzada em modelos

```

In [ ]: def cv_models(model, X, y, n_splits=5, n_jobs=14):
        cv = StratifiedKFold(n_splits=n_splits, shuffle=True, random_state=333)
        cv_scores = cross_val_score(model, X, y, cv=cv, scoring="roc_auc", n_jobs=n_jobs)
        return cv_scores

```

Importando Dados

Tabela de conteúdo:

- Explorando arquivos
- Criando DataFrames

Explorando arquivos

```
In [35]: zip_path = "final_provider.zip"

with zipfile.ZipFile(zip_path, "r") as zip_ref:
    file_list = zip_ref.namelist()
    structure = {}
    for file in file_list:
        lots = file.split("/")
        current = structure
        for lot in lots:
            if lot not in current:
                current[lot] = {}
            current = current[lot]

show_structure(structure)
```

```

└─ __MACOSX
   └─ final_provider
      ├── ._personal.csv
      └── ._phone.csv
final_provider
├── contract.csv
├── internet.csv
├── personal.csv
└── phone.csv
```

Criando DataFrames

```
In [ ]: file_path = "final_provider/"
names = ["contract", "internet", "personal", "phone"]
dfs = {}

with zipfile.ZipFile(zip_path, "r") as zip_ref:
    for name in names:
        with zip_ref.open(file_path + name + ".csv") as file:
            dfs[name] = pd.read_csv(file)

df_contract = dfs["contract"]
df_internet = dfs["internet"]
df_personal = dfs["personal"]
df_phone = dfs["phone"]
```

Entendendo os dataframes

Tabela de conteúdo:

- Primeira análise superficial de todos os dataframes
- Criando uma *Master Table*
- Conferindo se há valores nulos ou ausentes
- Tratando valores ausentes
- Conferindo tipologia dos dados

- Criando coluna-objetivo: 'Churn'
- Convertendo as colunas para a tipologia correta
- Ordenando os dados por 'BeginDate'
- Conferindo se será útil manter 'BeginDate'
- Criando coluna exclusiva para os anos de início dos contratos
- Conferindo integridade dos dados
- Tratando os valores nulos de 'TotalCharges'
- Apagando colunas que não serão utilizadas no processo

Primeira análise superficial de todos os dataframes

Farei uma primeira análise rápida visando identificar como posso unir os diferentes dataframes em um só.

In [37]: `df_contract.info()`

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7043 entries, 0 to 7042
Data columns (total 8 columns):
#   Column                Non-Null Count  Dtype
---  -
0   customerID            7043 non-null   object
1   BeginDate              7043 non-null   object
2   EndDate                7043 non-null   object
3   Type                   7043 non-null   object
4   PaperlessBilling       7043 non-null   object
5   PaymentMethod          7043 non-null   object
6   MonthlyCharges         7043 non-null   float64
7   TotalCharges           7043 non-null   object
dtypes: float64(1), object(7)
memory usage: 440.3+ KB
```

In [38]: `df_internet.info()`

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5517 entries, 0 to 5516
Data columns (total 8 columns):
#   Column                Non-Null Count  Dtype
---  -
0   customerID            5517 non-null   object
1   InternetService        5517 non-null   object
2   OnlineSecurity         5517 non-null   object
3   OnlineBackup           5517 non-null   object
4   DeviceProtection       5517 non-null   object
5   TechSupport            5517 non-null   object
6   StreamingTV            5517 non-null   object
7   StreamingMovies        5517 non-null   object
dtypes: object(8)
memory usage: 344.9+ KB
```

In [39]: `df_personal.info()`

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7043 entries, 0 to 7042
Data columns (total 5 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
dtypes: int64(1), object(4)
memory usage: 275.2+ KB
```

```
In [40]: df_phone.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6361 entries, 0 to 6360
Data columns (total 2 columns):
#   Column          Non-Null Count  Dtype
---  -
0   customerID      6361 non-null   object
1   MultipleLines    6361 non-null   object
dtypes: object(2)
memory usage: 99.5+ KB
```

Todos os dataframes possuem uma coluna em comum chamada 'customerID', com a mesma tipagem, esta será a chave para junção dos dados.

Criando uma *Master Table*

Utilizarei `merge()` com parâmetro *outer* para garantir que nenhum dado será perdido no processo.

```
In [ ]: datas = [df_internet, df_personal, df_phone]
data = df_contract.copy()
for d in datas:
    data = data.merge(d, how="outer", on="customerID")
```

Conferindo se há valores nulos ou ausentes

```
In [42]: data.info()
```

```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7043 entries, 0 to 7042
Data columns (total 20 columns):
#   Column                Non-Null Count  Dtype
---  -
0   customerID            7043 non-null   object
1   BeginDate             7043 non-null   object
2   EndDate               7043 non-null   object
3   Type                  7043 non-null   object
4   PaperlessBilling       7043 non-null   object
5   PaymentMethod         7043 non-null   object
6   MonthlyCharges        7043 non-null   float64
7   TotalCharges          7043 non-null   object
8   InternetService       5517 non-null   object
9   OnlineSecurity        5517 non-null   object
10  OnlineBackup          5517 non-null   object
11  DeviceProtection      5517 non-null   object
12  TechSupport           5517 non-null   object
13  StreamingTV           5517 non-null   object
14  StreamingMovies       5517 non-null   object
15  gender                7043 non-null   object
16  SeniorCitizen         7043 non-null   int64
17  Partner               7043 non-null   object
18  Dependents            7043 non-null   object
19  MultipleLines         6361 non-null   object
dtypes: float64(1), int64(1), object(18)
memory usage: 1.1+ MB

```

As colunas que possuem valores nulos ou ausentes são colunas contendo serviços relacionado a *Internet* e quantificações das *linhas de telefone* que o cliente pode ter, sugerindo que a ausência destes valores está relacionada ao fato do cliente ter um dos serviços principais contratados (*telefonia ou internet*) e não possuir o outro, logo não entrou no dataframe contendo os dados sobre os serviços que não possui em seu *contrato*.

Tratando valores ausentes

Tabela de conteúdo:

- Verificando como posso preencher os valores ausentes
- Substituindo os valores ausentes por 'No'

Verificando como posso preencher os valores ausentes

```

In [43]: null_cols = data.columns[data.isnull().sum() > 0]
for col in null_cols:
    print(data[col].value_counts())
    print()

```

```
InternetService
Fiber optic    3096
DSL            2421
Name: count, dtype: int64
```

```
OnlineSecurity
No            3498
Yes           2019
Name: count, dtype: int64
```

```
OnlineBackup
No            3088
Yes           2429
Name: count, dtype: int64
```

```
DeviceProtection
No            3095
Yes           2422
Name: count, dtype: int64
```

```
TechSupport
No            3473
Yes           2044
Name: count, dtype: int64
```

```
StreamingTV
No            2810
Yes           2707
Name: count, dtype: int64
```

```
StreamingMovies
No            2785
Yes           2732
Name: count, dtype: int64
```

```
MultipleLines
No            3390
Yes           2971
Name: count, dtype: int64
```

Todos os valores ausentes podem ser substituídos por 'No', pois realmente representam que o serviço não é prestado para aquele cliente.

Substituindo os valores ausentes por 'No'

```
In [ ]: data.fillna("No", inplace=True)
```

Conferindo tipologia dos dados

```
In [45]: data.info()
```



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7043 entries, 0 to 7042
Data columns (total 20 columns):
#   Column                Non-Null Count  Dtype
---  -
0   customerID            7043 non-null   object
1   BeginDate             7043 non-null   object
2   EndDate               7043 non-null   object
3   Type                  7043 non-null   object
4   PaperlessBilling       7043 non-null   object
5   PaymentMethod         7043 non-null   object
6   MonthlyCharges        7043 non-null   float64
7   TotalCharges          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  gender                7043 non-null   object
16  SeniorCitizen         7043 non-null   int64
17  Partner               7043 non-null   object
18  Dependents            7043 non-null   object
19  MultipleLines         7043 non-null   object
dtypes: float64(1), int64(1), object(18)
memory usage: 1.1+ MB
```

In [46]: `data.sample(5, random_state=333)`

Out[46]:

	customerID	BeginDate	EndDate	Type	PaperlessBilling	PaymentMethod	MonthlyCharges
601	0880-FVFWF	2015-06-01	No	Month-to-month	Yes	Electronic check	86.40
6311	8931-GJJIQ	2019-01-01	2019-10-01 00:00:00	Month-to-month	Yes	Electronic check	83.30
5229	7359-SSBJK	2014-06-01	2019-10-01 00:00:00	Two year	Yes	Credit card (automatic)	70.20
5497	7740-BTPUX	2015-07-01	No	Two year	Yes	Electronic check	113.60
631	0927-LCSMG	2019-06-01	2020-01-01 00:00:00	Month-to-month	No	Mailed check	74.65



Há colunas contendo **datas**, que precisam ser convertidas para *datetime*, são elas: 'BeginDate', 'EndDate'.

Não parece ser útil manter a coluna 'EndDate' por ser tratar de um dado *pós* objetivo da tarefa, preciso criar uma nova coluna binária contendo os valores 'Churn'.

A coluna 'TotalCharges' sugere que seja a soma de todos os pagamentos mensais ('MonthlyCharges'), logo, precisa ser convertida para *float*.

Criando coluna-objetivo: 'Churn'

A coluna-objetivo será baseada em 'EndDate', 'No' são os contratos ainda ativos no momento de extração dos dados e as *datas* são contratos já cancelados que terão suas características analisadas.

```
In [ ]: data["Churn"] = data["EndDate"].apply(lambda x: 0 if x == "No" else 1)
```

Convertendo as colunas para a tipologia correta

- 'BeginDate' para *datetime*;
- 'MonthlyCharges' para *float*.

```
In [ ]: data["BeginDate"] = pd.to_datetime(data["BeginDate"], errors="coerce")
data["TotalCharges"] = pd.to_numeric(data["TotalCharges"], errors="coerce")
```

Ordenando os dados por 'BeginDate'

Para a análise exploratória dos dados será útil entender a tendência dos acontecimentos e se há algum tipo de sazonalidade.

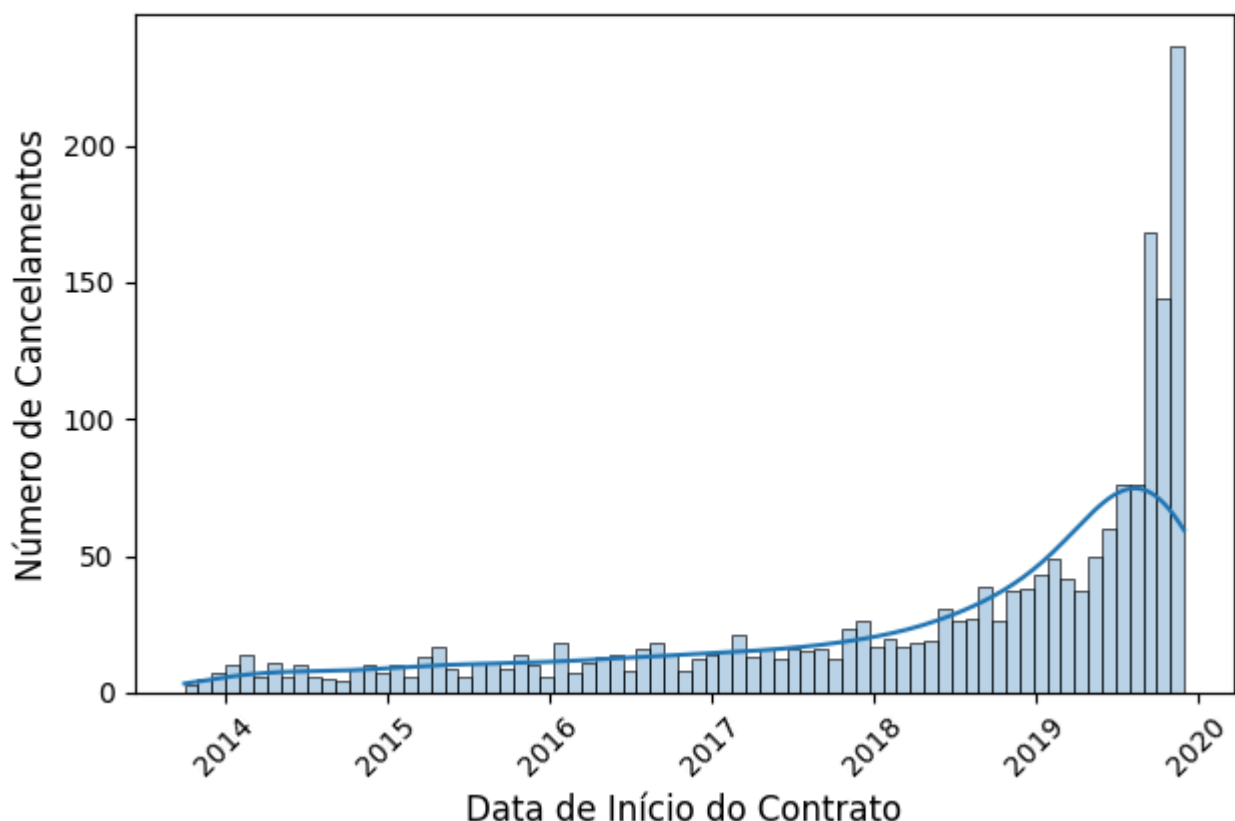
```
In [ ]: data = data.sort_values(by="BeginDate")
```

Conferindo se será útil manter 'BeginDate'

Plotarei um histograma para melhor visualização dos dados.

```
In [50]: churned_customers = data[data["Churn"] == 1]
title = "Contagem de cancelamentos por Data de Início do contrato"
bins = 73
plot_date_histogram(churned_customers, title, bins=bins)
plt.show()
```

Contagem de cancelamentos por Data de Início do contrato



Há muitos contratos *antigos* ainda ativos e os maiores cancelamentos são claramente nos contratos mais recentes, a partir de **2018**. Será útil criar features contendo os anos de início dos contratos.

Criando coluna exclusiva para os anos de início dos contratos

```
In [ ]: data["BeginYear"] = data["BeginDate"].dt.year.astype(object)
```

Conferindo integridade dos dados

Última conferência para garantir que o modelo rodará sem surpresas.


```
In [52]: data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 7043 entries, 6055 to 1731
Data columns (total 22 columns):
#   Column                Non-Null Count  Dtype
---  -
0   customerID            7043 non-null   object
1   BeginDate              7043 non-null   datetime64[ns]
2   EndDate                7043 non-null   object
3   Type                  7043 non-null   object
4   PaperlessBilling       7043 non-null   object
5   PaymentMethod         7043 non-null   object
6   MonthlyCharges        7043 non-null   float64
7   TotalCharges          7032 non-null   float64
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  gender                7043 non-null   object
16  SeniorCitizen         7043 non-null   int64
17  Partner               7043 non-null   object
18  Dependents            7043 non-null   object
19  MultipleLines         7043 non-null   object
20  Churn                 7043 non-null   int64
21  BeginYear             7043 non-null   object
dtypes: datetime64[ns](1), float64(2), int64(2), object(17)
memory usage: 1.2+ MB
```

```
In [53]: data.tail()
```

Out[53]:

	customerID	BeginDate	EndDate	Type	PaperlessBilling	PaymentMethod	MonthlyCharges	Tot
2250	3213-VVOLG	2020-02-01	No	Two year	No	Mailed check	25.35	
2855	4075-WKNIU	2020-02-01	No	Two year	No	Mailed check	73.35	
3118	4472-LVYGI	2020-02-01	No	Two year	Yes	Bank transfer (automatic)	52.55	
945	1371-DWPAZ	2020-02-01	No	Two year	No	Credit card (automatic)	56.05	
1731	2520-SGTTA	2020-02-01	No	Two year	No	Mailed check	20.00	



Agora 'TotalCharges' possui valores NaN nos contratos mais *recentes*, o que sugere que não tenham pagamentos o suficiente para o sistema calcular um *total*. Serão preenchidos então com Zero.

Tratando os valores nulos de 'TotalCharges'

```
In [ ]: data["TotalCharges"] = data.apply(  
    lambda row: (0 if pd.isna(row["TotalCharges"]) else row["TotalCharges"]),  
    axis=1,  
)
```

Apagando colunas que não serão utilizadas no processo

Não precisarei mais das colunas 'customerID' e 'EndDate' para análise exploratória nem para o modelo preditivo.

```
In [ ]: end_date = data["EndDate"]  
customer_id = data["customerID"]  
  
data = data.drop(["EndDate", "customerID"], axis=1)
```

Análise Exploratória

Tabela de conteúdo:

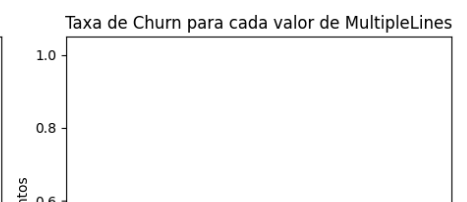
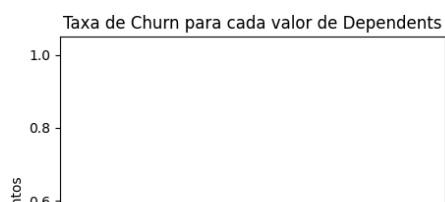
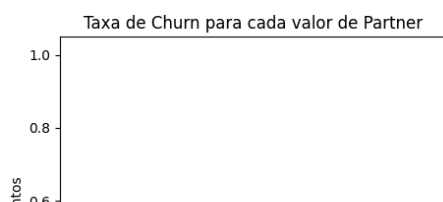
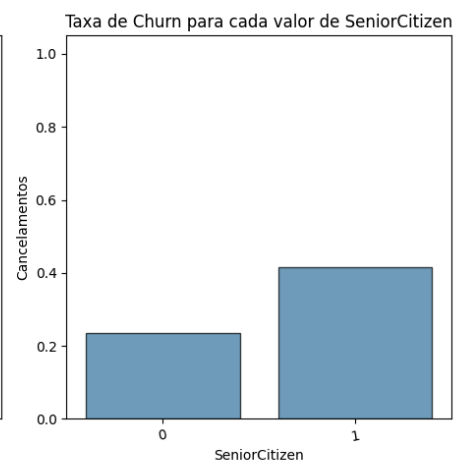
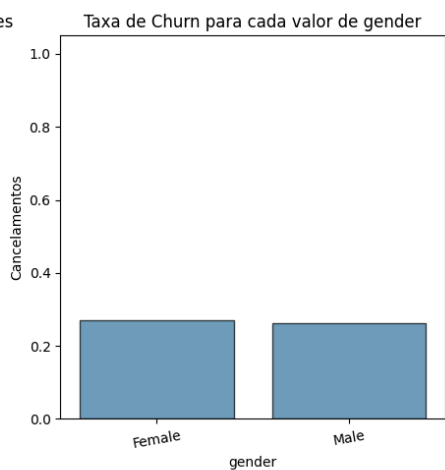
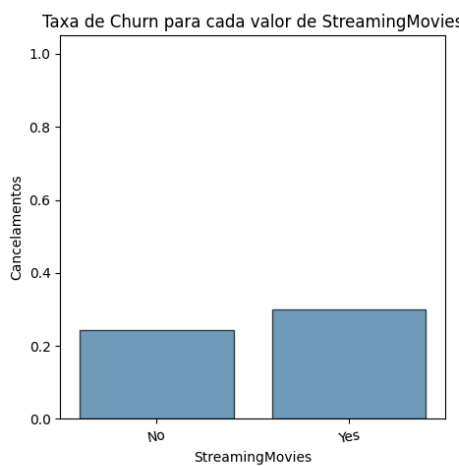
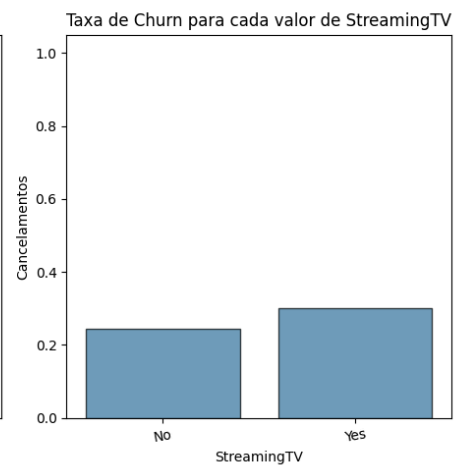
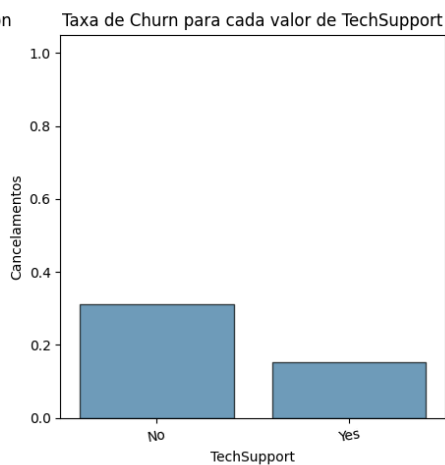
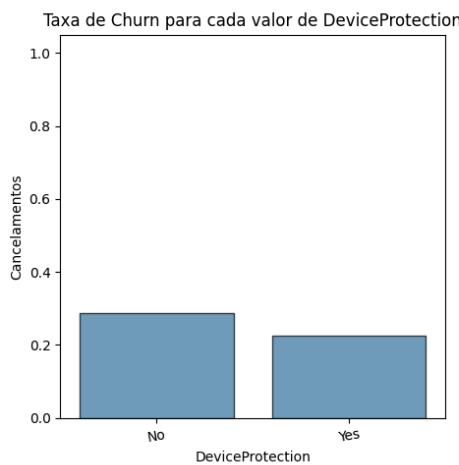
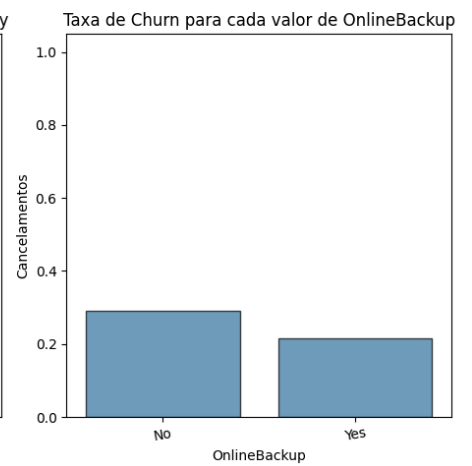
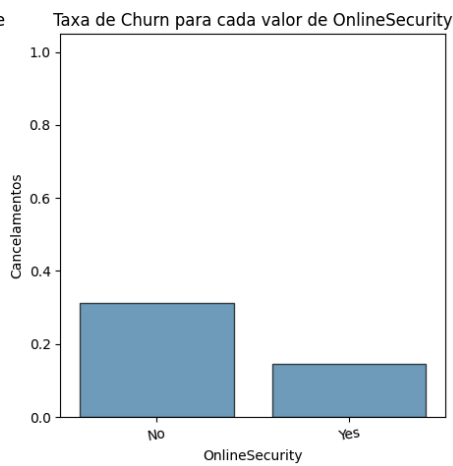
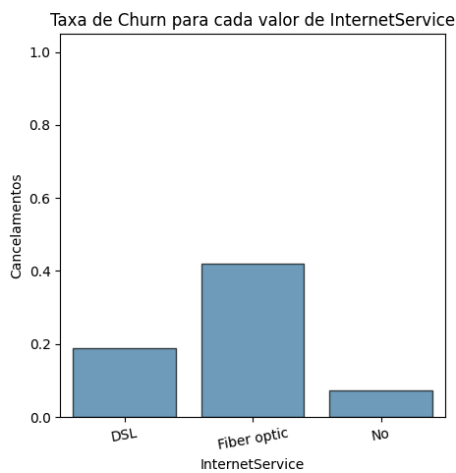
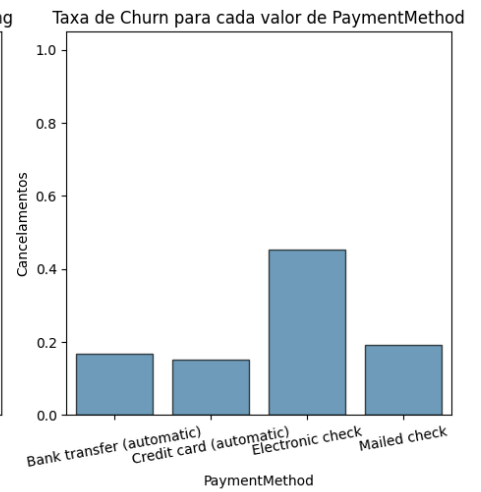
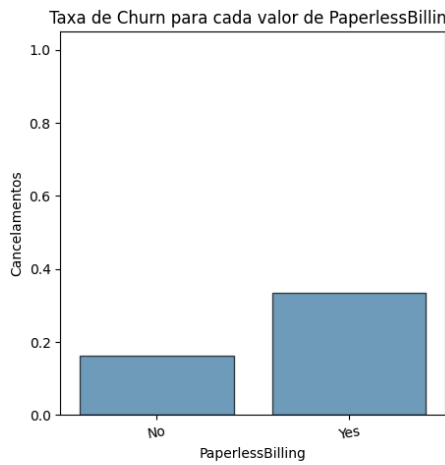
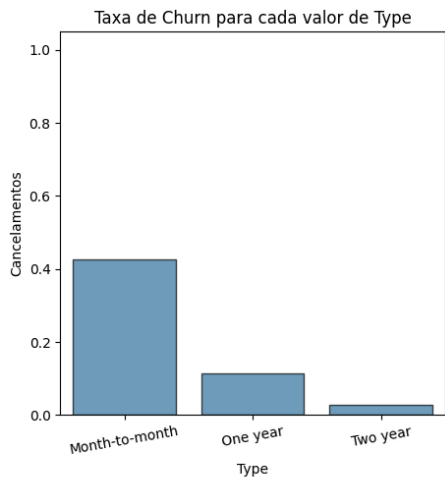
- Conferindo distribuição e outliers dos dados numéricos
 - Ampliando o boxplot de 'MonthlyCharges'

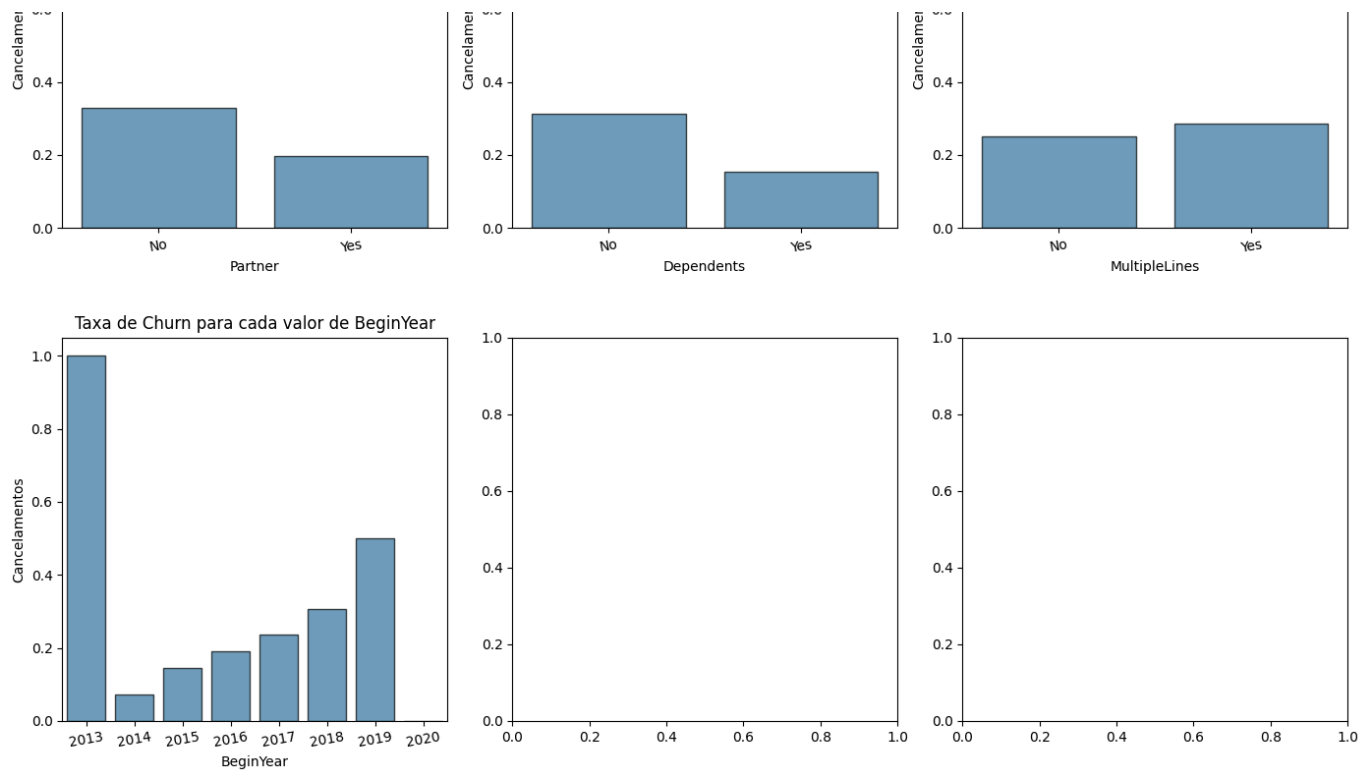
Antes de adaptar os dados para uso em Machine Learning irei analisar as colunas que mais têm correlação com os cancelamentos.

```
In [ ]: columns = [  
    "Type",  
    "PaperlessBilling",  
    "PaymentMethod",  
    "InternetService",  
    "OnlineSecurity",  
    "OnlineBackup",  
    "DeviceProtection",  
    "TechSupport",  
]
```

```
"StreamingTV",  
"StreamingMovies",  
"gender",  
"SeniorCitizen",  
"Partner",  
"Dependents",  
"MultipleLines",  
"BeginYear",  
]
```

```
In [57]: n_cols = 3  
n_rows = (len(columns) + n_cols - 1) // n_cols  
fig, axes = plt.subplots(n_rows, n_cols, figsize=(14, 30))  
axes = axes.flatten()  
y_limit = (0, 1.05)  
  
for i, column in enumerate(columns):  
    target_tax_barplot(data, column, "Churn", axes[i])  
    axes[i].set_ylim(y_limit)  
  
plt.tight_layout()  
plt.show()
```





Dentre as *features* analisadas as que parecem não ser relevantes ao modelo por haver pouca diferença entre as taxas de *churn* são:

1. **DeviceProtection**: (cerca de 0.28 para 0.22);
2. **StreamingTV**: (cerca de 0.25 para 0.30);
3. **StreamingMovies**: (cerca de 0.25 para 0.30);
4. **Gender**: (cerca de 0.27 para 0.26);
5. **MultipleLines**: (cerca de 0.25 para 0.29).

Serão excluídas na primeira versão do modelo para otimização dos treinamentos, porém as revisitarei para garantir a melhor calibragem possível.

Conferindo distribuição e outliers dos dados numéricos

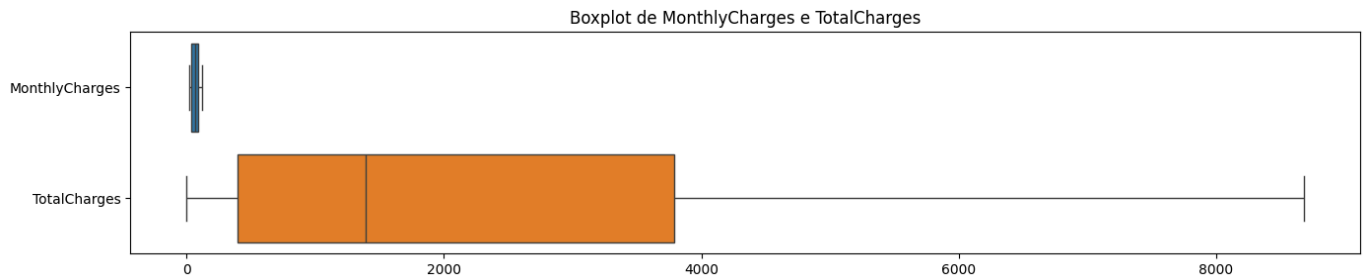
O *scaler* a ser utilizado dependerá da distribuição dos dados, do tamanho dos pavios e da presença de *outliers*.

Tabela de conteúdo:

- Ampliando o boxplot de 'MonthlyCharges'

```
In [58]: plt.figure(figsize=(14, 3))
sns.boxplot(data=data[["MonthlyCharges", "TotalCharges"]], orient="h")

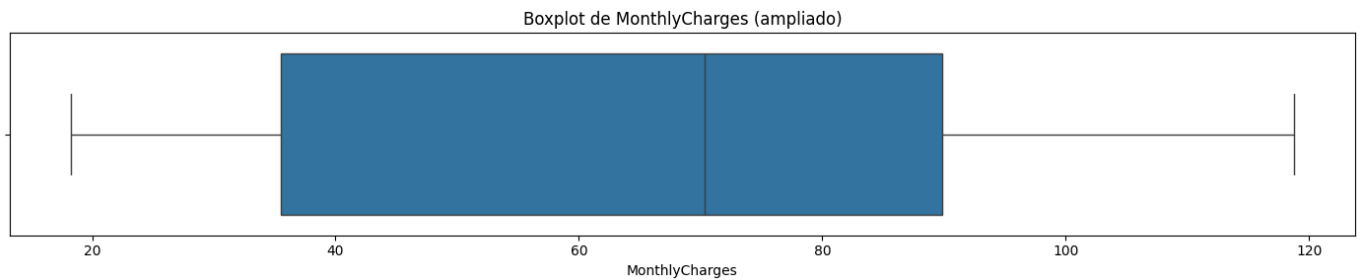
plt.title("Boxplot de MonthlyCharges e TotalCharges")
plt.tight_layout()
plt.show()
```



Ampliando o boxplot de 'MonthlyCharges'

```
In [59]: plt.figure(figsize=(14, 3))
sns.boxplot(data=data["MonthlyCharges"], orient="h")

plt.title("Boxplot de MonthlyCharges (ampliado)")
plt.tight_layout()
plt.show()
```



Considerando a diferença entre as escalas e o tamanho do pavio de 'TotalCharges', usarei o **RobustScaler** para tratamento dos dados com menor distorção.

Plano de Resolução da Tarefa

1. Separarei os dados de trabalho (*exclusão das colunas irrelevantes*);
2. Escalarei os dados numéricos com *RobustScaler* e codificarei as colunas categóricas com *OneHotEncoder*, a coluna 'BeginYear' será codificada com *OrdinalEncoder*;
3. Treinarei 3 diferentes modelos em busca dos melhores: *tempo de execução* e *taxa AUC-ROC*;
4. Criarei um pipeline de tratamento e execução do modelo para rodar em dados futuros com segurança.

Solução da tarefa

Tabela de conteúdo:

- Preparando os dados
- Treinando modelos
- Rodando modelos no Conjunto de Teste

Preparando os dados

Tabela de conteúdo:

- Separando os dados de trabalho
- Dividindo conjuntos de Treinamento, Validação e Testes
- Padronizando as escalas

- Codificando colunas categóricas Não-Ordinais
- Codificando 'BeginYear'

Separando os dados de trabalho

```
In [ ]: cols = [
    "BeginDate",
    "OnlineBackup",
    "DeviceProtection",
    "StreamingTV",
    "StreamingMovies",
    "gender",
    "MultipleLines",
]

review_later = data[cols]
work_data = data.drop(columns=cols, axis=1)
```

Dividindo conjuntos de Treinamento, Validação e Testes

Será usada a proporção 3:1 na divisão dos dados.

```
In [65]: X = work_data.drop("Churn", axis=1)
y = work_data["Churn"]

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.25, random_state=333
)

print(X_train.shape)
print(X_test.shape)
print()
print(y_train.shape)
print(y_test.shape)
```

(5282, 12)

(1761, 12)

(5282,)

(1761,)

Padronizando as escalas

```
In [ ]: num_cols = ["MonthlyCharges", "TotalCharges"]
scaler = RobustScaler().fit(X_train[num_cols])

X_train[num_cols] = scaler.transform(X_train[num_cols])
X_test[num_cols] = scaler.transform(X_test[num_cols])
```

Codificando colunas categóricas Não-Ordinais

As colunas categóricas não-ordinais serão codificadas com *OneHotEncoder()*.

```
In [ ]: cat_cols = X_train.drop("BeginYear", axis=1).select_dtypes(include="object").columns

encoder = OneHotEncoder(
    handle_unknown="ignore", drop="if_binary", sparse_output=False
).fit(X_train[cat_cols])
col_names = encoder.get_feature_names_out(cat_cols)
```

```
X_dadas = [X_train, X_test]

for i, df in enumerate(X_dadas):
    df_encoded = pd.DataFrame(
        encoder.transform(df[cat_cols]), columns=col_names, index=df.index
    )
    df = df.drop(columns=cat_cols)
    df = pd.concat([df, df_encoded], axis=1)
    X_dadas[i] = df

X_train, X_test = X_dadas
```

```
In [68]: print("Conferindo se todos os dataframes de features possuem as mesmas features:")
print(
    f"      ",
    all(X_train.columns) == all(X_test.columns),
)
```

Conferindo se todos os dataframes de features possuem as mesmas features:
True

Codificando 'BeginYear'

Para 'BeginYear' será utilizado `OrdinalEncoder()`.

```
In [ ]: ordinal_encoder = OrdinalEncoder()
ordinal_encoder.fit(pd.DataFrame(X_train["BeginYear"]))

X_dadas = [X_train, X_test]

for i, df in enumerate(X_dadas):
    df_encoded = pd.DataFrame(
        ordinal_encoder.transform(df[["BeginYear"]]),
        columns=["BeginYearEncoded"],
        index=df.index,
    )
    X_dadas[i] = pd.concat([df, df_encoded], axis=1)
    X_dadas[i] = X_dadas[i].drop("BeginYear", axis=1)

X_train, X_test = X_dadas
```

Treinando modelos

1. Procurarei a melhor configuração de modelo com *Validação Cruzada*;
2. Rodarei o modelo no conjunto de testes para conferir *métrica final* e *velocidade de predição*.

Tabela de conteúdo:

- Random Forest Classifier
- K Neighbors Classifier
- CatBoost Classifier

Random Forest Classifier

```
In [70]: auc_train = []
ests = [200, 300]
depths = [5, 7, 10, 12]
ssplit = [2, 5, 7, 10]
cweight = ["balanced", "balanced_subsample"]
```

```

for est, depth, split, cw in tqdm(list(product(ests, depths, ssplit, cweight))):
    model = RandomForestClassifier(
        random_state=333,
        n_jobs=-1,
        n_estimators=est,
        max_depth=depth,
        min_samples_split=split,
        class_weight=cw,
    )
    train_score = min(cv_models(model, X_train, y_train))
    auc_train.append(train_score)

```

100% |██████████| 64/64 [01:15<00:00, 1.18s/it]

```

In [71]: params = pd.DataFrame(
    list(product(ests, depths, ssplit, cweight)),
    columns=["n_estimators", "max_depth", "min_samples_split", "class_weight"],
)
idx = pd.Series(auc_train).idxmax()
print(f"Melhor AUC-ROC no Conjunto de Treino: {auc_train[idx]:.2f}")
print(f" - Ciclo: {idx}")
print()
print(f"Parâmetros:")
print(params.iloc[idx])

```

Melhor AUC-ROC no Conjunto de Treino: 0.86
- Ciclo: 23

Parâmetros:

n_estimators	200
max_depth	10
min_samples_split	10
class_weight	balanced_subsample

Name: 23, dtype: object

```

In [72]: rf_model = RandomForestClassifier(
    random_state=333,
    n_jobs=14,
    n_estimators=200,
    max_depth=10,
    min_samples_split=10,
    class_weight="balanced_subsample",
)

rf_model.fit(X_train, y_train)

```

```

Out[72]: ▼ RandomForestClassifier ⓘ ?
RandomForestClassifier(class_weight='balanced_subsample', max_depth=10,
    min_samples_split=10, n_estimators=200, n_jobs=14,
    random_state=333)

```

O modelo *Random Forest Classifier* performou muito bem com o Conjunto de Treino, **AUC-ROC score de 0.86**.

Hiperparâmetros:

- `n_estimators` = 200;
- `max_depth` = 10;
- `min_samples_split` = 10;

- `class_weight = 'balanced_subsample'`.

K Neighbors Classifier

```
In [73]: auc_train = []
auc_valid = []
nn = [3, 5, 9, 12, 15, 18, 21, 24, 27, 30, 33, 35, 36, 37, 39, 42, 45, 48]
metrics = ["cityblock", "cosine", "euclidean"]
weights = ["uniform", "distance"]

for n, metric, weight in tqdm(list(product(nn, metrics, weights))):
    model = KNeighborsClassifier(
        n_jobs=14,
        n_neighbors=n,
        metric=metric,
        weights=weight,
    )
    train_score = min(cv_models(model, X_train, y_train))
    auc_train.append(train_score)
```

100% |██████████| 108/108 [00:18<00:00, 5.92it/s]

```
In [74]: params = pd.DataFrame(
    list(product(nn, metrics, weights)),
    columns=["n_neighbors", "metric", "weights"],
)
idx = pd.Series(auc_train).idxmax()
print(f"Melhor AUC-ROC no Conjunto de Treino: {auc_train[idx]:.2f}")
print(f" - Ciclo: {idx}")
print()
print(f"Parâmetros:")
print(params.iloc[idx])
```

Melhor AUC-ROC no Conjunto de Treino: 0.83
- Ciclo: 107

Parâmetros:
n_neighbors 48
metric euclidean
weights distance
Name: 107, dtype: object

```
In [75]: knn_model = KNeighborsClassifier(
    n_jobs=-1,
    n_neighbors=48,
    metric="euclidean",
    weights="distance",
)

knn_model.fit(X_train, y_train)
```

```
Out[75]: KNeighborsClassifier
KNeighborsClassifier(metric='euclidean', n_jobs=-1, n_neighbors=48,
    weights='distance')
```

O modelo *K Neighbors Classifier* teve uma performance ligeiramente inferior com o Conjunto de Treino, **AUC-ROC score de 0.83**.

Hiperparâmetros:

- `n_neighbors = 48;`
- `metric = 'euclidean';`
- `weights = 'distance'.`

CatBoost Classifier

```
In [ ]: cat_model = CatBoostClassifier(  
    random_seed=333,  
    task_type="GPU",  
    verbose=100,  
    eval_metric="Logloss",  
    auto_class_weights="SqrtBalanced",  
)
```

```
In [77]: auc_train = []  
  
train_score = min(cv_models(cat_model, X_train, y_train, n_jobs=None))  
auc_train.append(train_score)
```

Learning rate set to 0.031142

0:	learn: 0.6727065	total: 127ms	remaining: 2m 7s
100:	learn: 0.4021945	total: 3.75s	remaining: 33.4s
200:	learn: 0.3881067	total: 7.3s	remaining: 29s
300:	learn: 0.3791138	total: 10.8s	remaining: 25.2s
400:	learn: 0.3732251	total: 14.3s	remaining: 21.4s
500:	learn: 0.3704610	total: 17.7s	remaining: 17.7s
600:	learn: 0.3680469	total: 21.2s	remaining: 14.1s
700:	learn: 0.3648192	total: 24.8s	remaining: 10.6s
800:	learn: 0.3624689	total: 28.4s	remaining: 7.06s
900:	learn: 0.3590247	total: 32s	remaining: 3.51s
999:	learn: 0.3561815	total: 35.5s	remaining: 0us

Learning rate set to 0.031142

0:	learn: 0.6735296	total: 42.6ms	remaining: 42.6s
100:	learn: 0.4014006	total: 3.72s	remaining: 33.1s
200:	learn: 0.3862722	total: 7.15s	remaining: 28.4s
300:	learn: 0.3787670	total: 10.7s	remaining: 24.9s
400:	learn: 0.3727875	total: 14.2s	remaining: 21.2s
500:	learn: 0.3675015	total: 17.7s	remaining: 17.6s
600:	learn: 0.3641648	total: 21.2s	remaining: 14.1s
700:	learn: 0.3610869	total: 24.9s	remaining: 10.6s
800:	learn: 0.3571400	total: 28.3s	remaining: 7.04s
900:	learn: 0.3547290	total: 31.8s	remaining: 3.49s
999:	learn: 0.3513525	total: 35.4s	remaining: 0us

Learning rate set to 0.031142

0:	learn: 0.6736280	total: 38.3ms	remaining: 38.3s
100:	learn: 0.4085447	total: 3.53s	remaining: 31.5s
200:	learn: 0.3956317	total: 7.06s	remaining: 28.1s
300:	learn: 0.3874505	total: 10.5s	remaining: 24.4s
400:	learn: 0.3815494	total: 14s	remaining: 20.8s
500:	learn: 0.3775837	total: 17.4s	remaining: 17.3s
600:	learn: 0.3739874	total: 21s	remaining: 14s
700:	learn: 0.3687859	total: 24.7s	remaining: 10.5s
800:	learn: 0.3654102	total: 28.5s	remaining: 7.08s
900:	learn: 0.3615280	total: 32s	remaining: 3.52s
999:	learn: 0.3562113	total: 35.5s	remaining: 0us

Learning rate set to 0.031142

0:	learn: 0.6732142	total: 36.2ms	remaining: 36.2s
100:	learn: 0.3982168	total: 3.65s	remaining: 32.5s
200:	learn: 0.3807083	total: 7.07s	remaining: 28.1s
300:	learn: 0.3719871	total: 10.6s	remaining: 24.7s
400:	learn: 0.3657068	total: 14.2s	remaining: 21.2s
500:	learn: 0.3591911	total: 17.7s	remaining: 17.6s
600:	learn: 0.3551337	total: 21.2s	remaining: 14.1s
700:	learn: 0.3511508	total: 24.9s	remaining: 10.6s
800:	learn: 0.3472587	total: 28.3s	remaining: 7.04s
900:	learn: 0.3436091	total: 31.9s	remaining: 3.5s
999:	learn: 0.3405139	total: 35.3s	remaining: 0us

Learning rate set to 0.031142

0:	learn: 0.6742420	total: 43.8ms	remaining: 43.7s
100:	learn: 0.4083434	total: 3.71s	remaining: 33.1s
200:	learn: 0.3985003	total: 7.28s	remaining: 28.9s
300:	learn: 0.3918204	total: 10.8s	remaining: 25.1s
400:	learn: 0.3895006	total: 14.5s	remaining: 21.7s
500:	learn: 0.3870310	total: 18.2s	remaining: 18.1s
600:	learn: 0.3858019	total: 22s	remaining: 14.6s
700:	learn: 0.3850116	total: 25.7s	remaining: 11s
800:	learn: 0.3832751	total: 29.3s	remaining: 7.28s
900:	learn: 0.3820768	total: 33s	remaining: 3.63s
999:	learn: 0.3809148	total: 36.5s	remaining: 0us

```
In [78]: print(f"Melhor AUC-ROC no Conjunto de Treino: {auc_train[0]:.2f}")
```

Melhor AUC-ROC no Conjunto de Treino: 0.87

```
In [79]: cat_model.fit(  
        X_train,  
        y_train,  
        use_best_model=True,  
        eval_set=(X_test, y_test),  
        verbose=100,  
        early_stopping_rounds=50,  
        )
```

Learning rate set to 0.061814

```
0:   learn: 0.6536066      test: 0.6560213 best: 0.6560213 (0)   total: 41.6ms   remain  
ing: 41.6s  
100:   learn: 0.3897931    test: 0.4152857 best: 0.4152569 (97)   total: 3.65s    remain  
ing: 32.5s  
200:   learn: 0.3686355    test: 0.4069563 best: 0.4069563 (200)   total: 7.37s    remain  
ing: 29.3s  
300:   learn: 0.3563727    test: 0.4051372 best: 0.4047115 (294)   total: 11s      remain  
ing: 25.6s  
bestTest = 0.4047115493  
bestIteration = 294  
Shrink model to first 295 iterations.
```

```
Out[79]: <catboost.core.CatBoostClassifier at 0x1cf007ca750>
```

O modelo *Cat Boost Classifier* teve performance significativamente superior com o Conjunto de Treino, **AUC-ROC score de 0.87**.

Hiperparâmetros listados na tabela abaixo. ▼

```
In [80]: cat_params = pd.DataFrame(  
        list(cat_model.get_all_params().items()), columns=["Parameter", "Value"]  
        )  
cat_params
```

Out[80]:

	Parameter	Value
0	nan_mode	Min
1	gpu_ram_part	0.95
2	eval_metric	Logloss
3	iterations	1000
4	fold_permutation_block	64
5	leaf_estimation_method	Newton
6	observations_to_bootstrap	TestOnly
7	od_pval	0
8	random_score_type	NormalWithModelSizeDecrease
9	grow_policy	SymmetricTree
10	penalties_coefficient	1
11	boosting_type	Ordered
12	feature_border_type	GreedyLogSum
13	bayesian_matrix_reg	0.1
14	devices	-1
15	eval_fraction	0
16	pinned_memory_bytes	104857600
17	force_unit_auto_pair_weights	False
18	l2_leaf_reg	3
19	random_strength	1
20	od_type	lter
21	rsm	1
22	boost_from_average	False
23	gpu_cat_features_storage	GpuRam
24	fold_size_loss_normalization	False
25	model_size_reg	0.5
26	pool_metainfo_options	{'tags': {}}
27	use_best_model	True
28	meta_l2_frequency	0
29	od_wait	50
30	class_names	[0, 1]
31	random_seed	333
32	depth	6
33	has_time	False
34	fold_len_multiplier	2

	Parameter	Value
35	border_count	128
36	min_fold_size	100
37	class_weights	[1, 1.6790573596954346]
38	data_partition	FeatureParallel
39	bagging_temperature	1
40	classes_count	0
41	auto_class_weights	SqrtBalanced
42	leaf_estimation_backtracking	AnyImprovement
43	best_model_min_trees	1
44	min_data_in_leaf	1
45	add_ridge_penalty_to_loss_function	False
46	loss_function	Logloss
47	learning_rate	0.061814
48	meta_l2_exponent	1
49	score_function	Cosine
50	task_type	GPU
51	leaf_estimation_iterations	10
52	bootstrap_type	Bayesian
53	max_leaves	64
54	permutation_count	4

Rodando modelos no Conjunto de Teste

Neste momento avaliarei *performance vs tempo de predição* de cada um modelos para o Conjunto de Teste.

```
In [81]: models = [rf_model, knn_model, cat_model]
names = ["RandomForestClassifier", "KNeighborsClassifier", "CatBoostClassifier"]

for name, model in zip(names, models):
    avg_time = timeit.timeit(lambda: model.predict(X_test), number=10) / 10
    preds = model.predict_proba(X_test)[: , 1]
    auc_roc = roc_auc_score(y_test, preds)

    print(f"Modelo: {name}")
    print(f"Tempo médio de previsão do modelo: {avg_time:.6f} segundos")
    print(f"AUC-ROC score no Conjunto de Testes: {auc_roc:.2f}")
    print()
```

Modelo: RandomForestClassifier
Tempo médio de previsão do modelo: 0.053288 segundos
AUC-ROC score no Conjunto de Testes: 0.87

Modelo: KNeighborsClassifier
Tempo médio de previsão do modelo: 0.065324 segundos
AUC-ROC score no Conjunto de Testes: 0.84

Modelo: CatBoostClassifier
Tempo médio de previsão do modelo: 0.002638 segundos
AUC-ROC score no Conjunto de Testes: 0.88

CatBoostClassifier superou todos os modelos, tanto em *tempo de execução* quanto em *AUC-ROC score*!

Avaliação final:

- *CatBoost Classifier*: Score **0.88** em 0.003 segundos;
- *Random Forest Classifier*: Score **0.87** em 0.056 segundos;
- *K Neighbors Classifier*: Score **0.84** em 0.065 segundos.

Testando o modelo com conjunto de dados completo

Irei utilizar o melhor modelo (*CatBoostClassifier*) para testar possíveis melhorias com as colunas retiradas no começo do estudo.

Será aplicado o mesmo preprocessamento dos dados utilizado anteriormente.

Tabela de conteúdo:

- Retirando coluna '*BeginDate*'
- Dividindo conjuntos de Treinamento e Teste
- Padronizando as escalas
- Codificando colunas categóricas Não-Ordinais
- Codificando '*BeginYear*'
- Treinando modelo *CatBoost Classifier*
- Prosseguindo o treinamento com '*eval_set*'
- Rodando o modelo no Conjunto de Testes

Retirando coluna '*BeginDate*'

Esta coluna ainda será retirada pois os dados relevantes dela já estão em '*BeginYear*'.

```
In [ ]: data2 = data.drop("BeginDate", axis=1)
```

Dividindo conjuntos de Treinamento e Teste

```
In [103... X2 = data2.drop("Churn", axis=1)
y2 = data2["Churn"]

X_train2, X_test2, y_train2, y_test2 = train_test_split(
    X2, y2, test_size=0.25, random_state=333
)

print(X_train2.shape)
print(X_test2.shape)
```

```
print()
print(y_train2.shape)
print(y_test2.shape)
```

```
(5282, 18)
(1761, 18)
```

```
(5282,)
(1761,)
```

Padronizando as escalas

```
In [ ]: num_cols = ["MonthlyCharges", "TotalCharges"]
        scaler = RobustScaler().fit(X_train2[num_cols])

X_train2[num_cols] = scaler.transform(X_train2[num_cols])
X_test2[num_cols] = scaler.transform(X_test2[num_cols])
```

Codificando colunas categóricas Não-Ordinais

```
In [ ]: cat_cols = X_train2.drop("BeginYear", axis=1).select_dtypes(include="object").columns

encoder = OneHotEncoder(
    handle_unknown="ignore", drop="if_binary", sparse_output=False
).fit(X_train2[cat_cols])
col_names = encoder.get_feature_names_out(cat_cols)

X_dadas2 = [X_train2, X_test2]

for i, df in enumerate(X_dadas2):
    df_encoded = pd.DataFrame(
        encoder.transform(df[cat_cols]), columns=col_names, index=df.index
    )
    df = df.drop(columns=cat_cols)
    df = pd.concat([df, df_encoded], axis=1)
    X_dadas2[i] = df

X_train2, X_test2 = X_dadas2
```

Codificando 'BeginYear'

```
In [ ]: ordinal_encoder = OrdinalEncoder()
        ordinal_encoder.fit(pd.DataFrame(X_train2["BeginYear"]))

X_dadas2 = [X_train2, X_test2]

for i, df in enumerate(X_dadas2):
    df_encoded = pd.DataFrame(
        ordinal_encoder.transform(df[["BeginYear"]]),
        columns=["BeginYearEncoded"],
        index=df.index,
    )
    X_dadas2[i] = pd.concat([df, df_encoded], axis=1)
    X_dadas2[i] = X_dadas2[i].drop("BeginYear", axis=1)

X_train2, X_test2 = X_dadas2
```

Treinando modelo *CatBoost Classifier*

```
In [ ]: cat_model2 = CatBoostClassifier(  
    random_seed=333,  
    task_type="GPU",  
    verbose=100,  
    eval_metric="Logloss",  
    auto_class_weights="SqrtBalanced",  
)
```

```
In [109... auc_train = []  
  
train_score = min(cv_models(cat_model2, X_train2, y_train2, n_jobs=None))  
auc_train.append(train_score)
```

Learning rate set to 0.031142

0:	learn: 0.6727065	total: 38.1ms	remaining: 38.1s
100:	learn: 0.3996977	total: 3.63s	remaining: 32.3s
200:	learn: 0.3829795	total: 7.25s	remaining: 28.8s
300:	learn: 0.3733618	total: 10.8s	remaining: 25.1s
400:	learn: 0.3636873	total: 14.5s	remaining: 21.6s
500:	learn: 0.3585260	total: 18.1s	remaining: 18s
600:	learn: 0.3543882	total: 21.6s	remaining: 14.4s
700:	learn: 0.3503011	total: 25.2s	remaining: 10.8s
800:	learn: 0.3478171	total: 28.8s	remaining: 7.16s
900:	learn: 0.3451730	total: 32.3s	remaining: 3.55s
999:	learn: 0.3418175	total: 35.9s	remaining: 0us

Learning rate set to 0.031142

0:	learn: 0.6734421	total: 35ms	remaining: 35s
100:	learn: 0.3988499	total: 3.55s	remaining: 31.6s
200:	learn: 0.3822782	total: 7.26s	remaining: 28.9s
300:	learn: 0.3745471	total: 11s	remaining: 25.5s
400:	learn: 0.3658250	total: 14.5s	remaining: 21.7s
500:	learn: 0.3585816	total: 18s	remaining: 18s
600:	learn: 0.3533516	total: 21.5s	remaining: 14.3s
700:	learn: 0.3470985	total: 25.1s	remaining: 10.7s
800:	learn: 0.3422419	total: 28.8s	remaining: 7.15s
900:	learn: 0.3381429	total: 32.3s	remaining: 3.55s
999:	learn: 0.3346641	total: 35.9s	remaining: 0us

Learning rate set to 0.031142

0:	learn: 0.6736280	total: 38.8ms	remaining: 38.7s
100:	learn: 0.4093056	total: 3.66s	remaining: 32.6s
200:	learn: 0.3934517	total: 7.26s	remaining: 28.9s
300:	learn: 0.3849447	total: 10.9s	remaining: 25.3s
400:	learn: 0.3773061	total: 14.4s	remaining: 21.5s
500:	learn: 0.3698384	total: 18s	remaining: 17.9s
600:	learn: 0.3629382	total: 21.8s	remaining: 14.4s
700:	learn: 0.3567650	total: 25.8s	remaining: 11s
800:	learn: 0.3530718	total: 29.9s	remaining: 7.42s
900:	learn: 0.3508875	total: 33.6s	remaining: 3.69s
999:	learn: 0.3459784	total: 37.3s	remaining: 0us

Learning rate set to 0.031142

0:	learn: 0.6732142	total: 37.6ms	remaining: 37.5s
100:	learn: 0.3947897	total: 3.72s	remaining: 33.1s
200:	learn: 0.3769501	total: 7.25s	remaining: 28.8s
300:	learn: 0.3677761	total: 10.7s	remaining: 24.8s
400:	learn: 0.3612772	total: 14.2s	remaining: 21.2s
500:	learn: 0.3554166	total: 17.7s	remaining: 17.7s
600:	learn: 0.3503354	total: 21.2s	remaining: 14.1s
700:	learn: 0.3449518	total: 24.7s	remaining: 10.5s
800:	learn: 0.3421564	total: 28.2s	remaining: 7.01s
900:	learn: 0.3399082	total: 31.8s	remaining: 3.5s
999:	learn: 0.3366977	total: 35.3s	remaining: 0us

Learning rate set to 0.031142

0:	learn: 0.6742420	total: 38.2ms	remaining: 38.2s
100:	learn: 0.4076768	total: 3.55s	remaining: 31.6s
200:	learn: 0.3960042	total: 7.03s	remaining: 27.9s
300:	learn: 0.3910680	total: 10.5s	remaining: 24.3s
400:	learn: 0.3876714	total: 13.9s	remaining: 20.8s
500:	learn: 0.3827586	total: 17.3s	remaining: 17.3s
600:	learn: 0.3767998	total: 20.9s	remaining: 13.9s
700:	learn: 0.3732067	total: 24.4s	remaining: 10.4s
800:	learn: 0.3712022	total: 28s	remaining: 6.95s
900:	learn: 0.3687894	total: 31.6s	remaining: 3.47s
999:	learn: 0.3677145	total: 35.1s	remaining: 0us

In [111...

```
print(f"Melhor AUC-ROC no Conjunto de Treino: {auc_train[0]:.2f}")
```

Melhor AUC-ROC no Conjunto de Treino: 0.87

O modelo *não apresentou melhora* no conjunto de treinamento com os parâmetros que haviam sido retirados, vou continuar com o treinamento utilizando `eval_set` a procura de melhora na performance com os novos dados.

Prosseguindo o treinamento com 'eval_set'

```
In [112... cat_model2.fit(  
    X_train2,  
    y_train2,  
    use_best_model=True,  
    eval_set=(X_test2, y_test2),  
    verbose=100,  
    early_stopping_rounds=50,  
)
```

Learning rate set to 0.061814

```
0:      learn: 0.6536066      test: 0.6560213 best: 0.6560213 (0)      total: 38.4ms      remain  
ing: 38.4s  
100:    learn: 0.3857279      test: 0.4139128 best: 0.4139128 (100)    total: 3.67s      remain  
ing: 32.7s  
200:    learn: 0.3627532      test: 0.4058648 best: 0.4058648 (200)    total: 7.21s      remain  
ing: 28.7s  
300:    learn: 0.3460195      test: 0.4029633 best: 0.4029633 (300)    total: 10.8s      remain  
ing: 25s  
bestTest = 0.4024306799  
bestIteration = 346  
Shrink model to first 347 iterations.
```

```
Out[112... <catboost.core.CatBoostClassifier at 0x1cf007daab0>
```

Rodando o modelo no Conjunto de Testes

```
In [113... avg_time = timeit.timeit(lambda: cat_model2.predict(X_test2), number=10) / 10  
preds = cat_model2.predict_proba(X_test2)[: , 1]  
auc_roc = roc_auc_score(y_test2, preds)  
  
print(f"Modelo: CatBoost Classifier - com conjunto completo de features")  
print(f"Tempo médio de previsão do modelo: {avg_time:.6f} segundos")  
print(f"AUC-ROC score no Conjunto de Testes: {auc_roc:.2f}")
```

Modelo: CatBoost Classifier - com conjunto completo de features

Tempo médio de previsão do modelo: 0.002445 segundos

AUC-ROC score no Conjunto de Testes: 0.89

No conjunto de testes o modelo com o conjunto completo de features superou o modelo com as features filtradas, tanto em *tempo de execução* quanto em *AUC-ROC score*!

Avaliação final - CatBoost Classifier:

- Com features filtradas: Score **0.88** em 0.003 segundos;
- Com conjunto de features completo: Score **0.89** em 0.002 segundos.

Considerando que uma das preocupações em filtrar o conjunto de features era de deixar a execução do modelo *mais leve e rápida*, não faz mais sentido a exclusão de colunas, o modelo performa melhor com todas as features.

Hiperparâmetros listados na tabela abaixo. 

In [119...

```
cat_params = pd.DataFrame(  
    list(cat_model2.get_all_params().items()), columns=["Parameter", "Value"]  
)  
cat_params
```

Out[119...

	Parameter	Value
0	nan_mode	Min
1	gpu_ram_part	0.95
2	eval_metric	Logloss
3	iterations	1000
4	fold_permutation_block	64
5	leaf_estimation_method	Newton
6	observations_to_bootstrap	TestOnly
7	od_pval	0
8	random_score_type	NormalWithModelSizeDecrease
9	grow_policy	SymmetricTree
10	penalties_coefficient	1
11	boosting_type	Ordered
12	feature_border_type	GreedyLogSum
13	bayesian_matrix_reg	0.1
14	devices	-1
15	eval_fraction	0
16	pinned_memory_bytes	104857600
17	force_unit_auto_pair_weights	False
18	l2_leaf_reg	3
19	random_strength	1
20	od_type	Iter
21	rsm	1
22	boost_from_average	False
23	gpu_cat_features_storage	GpuRam
24	fold_size_loss_normalization	False
25	model_size_reg	0.5
26	pool_metainfo_options	{'tags': {}}
27	use_best_model	True
28	meta_l2_frequency	0
29	od_wait	50
30	class_names	[0, 1]
31	random_seed	333
32	depth	6
33	has_time	False
34	fold_len_multiplier	2

	Parameter	Value
35	border_count	128
36	min_fold_size	100
37	class_weights	[1, 1.6790573596954346]
38	data_partition	FeatureParallel
39	bagging_temperature	1
40	classes_count	0
41	auto_class_weights	SqrtBalanced
42	leaf_estimation_backtracking	AnyImprovement
43	best_model_min_trees	1
44	min_data_in_leaf	1
45	add_ridge_penalty_to_loss_function	False
46	loss_function	Logloss
47	learning_rate	0.061814
48	meta_l2_exponent	1
49	score_function	Cosine
50	task_type	GPU
51	leaf_estimation_iterations	10
52	bootstrap_type	Bayesian
53	max_leaves	64
54	permutation_count	4

Criando um pipeline para utilização do modelo recomendado

Criarei uma *pipeline* simples baseada em **funções**, replicando todo o processo de tratamento de dados e treinamento de modelo executado neste caderno, para uso em arquivos .csv.

Tabela de conteúdo:

- [Função para carregar e unir os dados](#)
- [Função para tratamento inicial dos dados](#)
- [Função para Criação de Pipeline](#)
- [Função para novos treinamentos do modelo](#)
- [Função para fazer previsões a partir dos dados em .csv](#)

Função para carregar e unir os dados

```
In [ ]: def load_join_data(contract_path, internet_path, personal_path, phone_path):
        """
        Carrega e une os dados dos diferentes arquivos
        """
```

```

df_contract = pd.read_csv(contract_path)
df_internet = pd.read_csv(internet_path)
df_personal = pd.read_csv(personal_path)
df_phone = pd.read_csv(phone_path)

data = df_contract.copy()
for df in [df_internet, df_personal, df_phone]:
    data = data.merge(df, how="outer", on="customerID")

data.fillna("No", inplace=True)

return data

```

Função para tratamento inicial dos dados

```

In [ ]: def data_prep(data, train_mode=True):
        """
        Prepara os dados para treino ou predição
        """

        if train_mode and "EndDate" in data.columns:
            data["Churn"] = data["EndDate"].apply(lambda x: 0 if x == "No" else 1)

        customer_ids = data["customerID"].copy() if "customerID" in data.columns else None

        if "BeginDate" in data.columns:
            data["BeginDate"] = pd.to_datetime(data["BeginDate"], errors="coerce")
            data["BeginYear"] = data["BeginDate"].dt.year.astype(object)

        if "TotalCharges" in data.columns:
            data["TotalCharges"] = pd.to_numeric(data["TotalCharges"], errors="coerce")

            if "MonthlyCharges" in data.columns:
                data["TotalCharges"] = data.apply(
                    lambda row: (
                        0 if pd.isna(row["TotalCharges"]) else row["TotalCharges"]
                    ),
                    axis=1,
                )

        return data, customer_ids

```

Função para Criação de Pipeline

```

In [ ]: def pipeline_create():
        """
        Cria um pipeline para o modelo de churn
        """

        num_cols = ["MonthlyCharges", "TotalCharges"]

        year_cols = ["BeginYear"]

        cat_cols = [
            "Type",
            "InternetService",
            "PaymentMethod",
            "Partner",
            "Dependents",
            "PhoneService",
            "PaperlessBilling",
            "SeniorCitizen",

```

```

        "TechSupport",
        "OnlineSecurity",
        "OnlineBackup",
        "DeviceProtection",
        "StreamingTV",
        "StreamingMovies",
        "gender",
        "MultipleLines",
    ]

    num_processor = Pipeline([("scaler", RobustScaler())])

    year_processor = Pipeline([("ordinal", OrdinalEncoder())])

    cat_processor = Pipeline(
        [
            (
                "onehot",
                OneHotEncoder(
                    handle_unknown="ignore", drop="if_binary", sparse_output=False
                ),
            )
        ]
    )

    col_transformer = ColumnTransformer(
        transformers=[
            ("num", num_processor, num_cols),
            ("cat", cat_processor, cat_cols),
            ("year", year_processor, year_cols),
        ],
        remainder="drop",
    )

    pipeline = Pipeline(
        [
            ("col_transformer", col_transformer),
            (
                "model",
                CatBoostClassifier(
                    verbose=100,
                    eval_metric="Logloss",
                    random_seed=333,
                    auto_class_weights="SqrtBalanced",
                ),
            ),
        ]
    )

    return pipeline

```

Função para novos treinamentos do modelo

```

In [ ]: def treinar_modelo(pipeline, X_train, y_train, X_valid=None, y_valid=None):
        """
        Treina o modelo com os dados fornecidos
        """
        fit_params = {}

        if X_valid is not None and y_valid is not None:
            fit_params = {
                "model__eval_set": [(X_valid, y_valid)],

```

```

        "model__early_stopping_rounds": 50,
    }

    pipeline.fit(X_train, y_train, **fit_params)

    return pipeline

```

Função para fazer previsões a partir dos dados em .csv

```

In [ ]: def predict(model_path, contract_path, internet_path, personal_path, phone_path):
        """
        Faz previsões com novos dados usando um modelo salvo
        """
        print("Carregando modelo...")

        try:
            pipeline = joblib.load(model_path)
        except FileNotFoundError:
            print(f"Modelo não encontrado em {model_path}")
            return None

        try:
            data = load_join_data(contract_path, internet_path, personal_path, phone_path)
        except FileNotFoundError:
            print("Arquivos de dados não encontrados. Verifique os caminhos.")
            return None

        prep_data, customer_ids = data_prep(data, train_mode=False)

        predicts = pipeline.predict(prepare_data)
        probas = pipeline.predict_proba(prepare_data)[: , 1]

        results = pd.DataFrame(
            {
                "customerID": customer_ids,
                "churn_preds": predicts,
                "churn_probas": probas,
            }
        )

        print(f"Previsões realizadas para {len(results)} clientes.")

        return results

```

Conclusão

Tabela de conteúdo:

- Resultados Obtidos
- Insights Relevantes
- Impacto e Aplicabilidade
- Modelo recomendado:

Resultados Obtidos

O objetivo do projeto foi alcançado: o modelo de Machine Learning desenvolvido consegue prever clientes propensos ao cancelamento.

O modelo *CatBoost Classifier* apresentou o melhor desempenho, alcançando um **AUC-ROC de 0.87** no conjunto de treino e **0.89** no conjunto de teste.

O *Random Forest Classifier* também teve bom desempenho, com **AUC-ROC de 0.86** no treino e **0.87** no teste.

O *K Neighbors Classifier* obteve um **AUC-ROC de 0.84** em ambos os conjuntos, sendo o modelo menos eficiente.

Além disso, desenvolvi uma **Pipeline** para automatizar o processo de tratamento, treinamento e predição de novos dados. A pipeline consiste em:

- Função para carregar e unir os dados;
- Função para tratamento inicial dos dados;
- Função para criação do pipeline de Machine Learning;
- Função para realizar novos treinamentos do modelo;
- Função para fazer predições a partir de dados em .csv.

Essa estrutura garante a reprodutibilidade e a escalabilidade do processo, tornando a aplicação do modelo em novos dados mais eficiente.

Insights Relevantes

O cancelamento de clientes ("*Churn*") está fortemente relacionado a fatores como:

- **Tempo de contrato:** contratos mais recentes têm maior taxa de cancelamento;
- **Tempo de renovação do contrato:** contratos com períodos de renovação mais curtos (como o *Mensal*), têm maior taxa de cancelamento;
- Contratos com **cobranças eletrônicas porém não automáticas** também são mais propensos ao cancelamento;
- Dos contratos com serviço de internet os de **Fibra Ótica** são os mais cancelados.

Impacto e Aplicabilidade

O modelo pode ser aplicado em campanhas de retenção para clientes com maior risco de cancelamento. A empresa pode criar incentivos direcionados (*descontos, serviços gratuitos por tempo limitado, suporte diferenciado*) para os perfis de clientes mais propensos a cancelar.

Possibilidade de otimizar os pacotes de serviço para reduzir cancelamentos, como oferecer planos de menor custo para clientes de alto risco.

Modelo recomendado:

O modelo mais recomendado para uso (*e utilizado no pipeline proposto*) é o **CatBoostClassifier** treinado com os Hiperparâmetros da tabela a seguir (*variável `cat_params`*).

In [120...

```
cat_params
```

Out[120...

	Parameter	Value
0	nan_mode	Min
1	gpu_ram_part	0.95
2	eval_metric	Logloss
3	iterations	1000
4	fold_permutation_block	64
5	leaf_estimation_method	Newton
6	observations_to_bootstrap	TestOnly
7	od_pval	0
8	random_score_type	NormalWithModelSizeDecrease
9	grow_policy	SymmetricTree
10	penalties_coefficient	1
11	boosting_type	Ordered
12	feature_border_type	GreedyLogSum
13	bayesian_matrix_reg	0.1
14	devices	-1
15	eval_fraction	0
16	pinned_memory_bytes	104857600
17	force_unit_auto_pair_weights	False
18	l2_leaf_reg	3
19	random_strength	1
20	od_type	lter
21	rsm	1
22	boost_from_average	False
23	gpu_cat_features_storage	GpuRam
24	fold_size_loss_normalization	False
25	model_size_reg	0.5
26	pool_metainfo_options	{'tags': {}}
27	use_best_model	True
28	meta_l2_frequency	0
29	od_wait	50
30	class_names	[0, 1]
31	random_seed	333
32	depth	6
33	has_time	False
34	fold_len_multiplier	2

	Parameter	Value
35	border_count	128
36	min_fold_size	100
37	class_weights	[1, 1.6790573596954346]
38	data_partition	FeatureParallel
39	bagging_temperature	1
40	classes_count	0
41	auto_class_weights	SqrtBalanced
42	leaf_estimation_backtracking	AnyImprovement
43	best_model_min_trees	1
44	min_data_in_leaf	1
45	add_ridge_penalty_to_loss_function	False
46	loss_function	Logloss
47	learning_rate	0.061814
48	meta_l2_exponent	1
49	score_function	Cosine
50	task_type	GPU
51	leaf_estimation_iterations	10
52	bootstrap_type	Bayesian
53	max_leaves	64
54	permutation_count	4