```
# %% [markdown]
# # 💻 Projeto Técnico – Previsão de Evasão de Clientes (Churn)
# **Nome do arquivo:** projeto_cloves_logica.ipynb
# **Disciplina:** Lógica e Algoritmos 2
# **Curso:** Análise e Desenvolvimento de Sistemas
# **Modelo Utilizado:** Árvore de Decisão (Decision Tree)
# **Dataset:** IBM Telco Customer Churn
#
# ---
#
# ## 1️⃣ Definição do Problema / Objetivos
#
# O problema de *Churn* (evasão de clientes) ocorre quando consumidores deixam de utilizar
# O objetivo deste projeto é desenvolver um modelo de Machine Learning capaz de prever, co
# quem tem maior probabilidade de cancelar o serviço.
#
# 💡 **Lógica do Negócio:**
# Minimizar **falsos negativos** é essencial, pois perder um cliente real que o modelo pre
# não agir preventivamente – algo que representa prejuízo direto para a empresa.
# ## 2️⃣ Coleta e Entendimento dos Dados
# Carregamos o dataset público da IBM com informações de clientes e se deram churn (sim/nã
import pandas as pd

# Carregar o dataset (coloque o arquivo na mesma pasta do notebook)
df = pd.read_csv("WA_Fn-UseC_-Telco-Customer-Churn.csv")

# Visualizar as 10 primeiras linhas
display(df.head(10))

# Tipos de dados
print("\nTipos de dados por coluna:\n", df.dtypes)

# Contagem de valores ausentes
print("\nValores ausentes por coluna:\n", df.isnull().sum())

# %% [markdown]
# ## 3️⃣ Tratamento e Preparação dos Dados
#
# ### a) Lógica de Tratamento de Nulos
#
# Implementamos uma estrutura de controle (if/else) para detectar e tratar colunas com val
# - Se a coluna for numérica → preenchemos com a média.
# - Se for categórica → preenchemos com o valor 'Desconhecido'.

for coluna in df.columns:
    if df[coluna].isnull().sum() > 0:
        if df[coluna].dtype in ['int64', 'float64']:
            df[coluna].fillna(df[coluna].mean(), inplace=True)
        else:
            df[coluna].fillna('Desconhecido', inplace=True)

# Verificação final
print("Valores ausentes após tratamento:", df.isnull().sum().sum())

# %% [markdown]
# ### b) Algoritmo de Codificação (One-Hot Encoding)
#
# O *One-Hot Encoding* transforma variáveis categóricas em colunas binárias (0 e 1),
# evitando que o modelo interprete categorias como se tivessem ordem numérica.

cat_cols = df.select_dtypes(include=['object']).columns
```

```
df_encoded = pd.get_dummies(df, columns=cat_cols, drop_first=True)

# %% [markdown]
# ### c) Normalização / Escalonamento
#
# Normalizamos as variáveis numéricas usando o algoritmo **MinMaxScaler**,
# garantindo que todas as features fiquem na mesma escala (0–1).

from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
num_cols = df_encoded.select_dtypes(include=['int64', 'float64']).columns
df_encoded[num_cols] = scaler.fit_transform(df_encoded[num_cols])

# %% [markdown]
# ## 4️⃣ Análise Exploratória dos Dados
# Verificamos a proporção entre clientes que deram churn e os que não deram.

churn_rate = df['Churn'].value_counts(normalize=True)
print("Distribuição das classes:\n", churn_rate)

# %% [markdown]
# **Lógica:**
# Se as classes estiverem desbalanceadas (por exemplo, 70% não churn e 30% churn),
# devemos priorizar métricas como **Recall** para avaliar o modelo, pois o negócio precisa


# ## 5️⃣ Modelagem do Algoritmo — Árvore de Decisão
#
# A Árvore de Decisão baseia-se em divisões lógicas (if/else) e usa o **índice Gini** para
#
# \[
# G = 1 - \sum (p_i)^2
# \]
#
# Quanto menor o Gini, mais puro o nó.

from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report

X = df_encoded.drop('Churn_Yes', axis=1)
y = df_encoded['Churn_Yes']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

modelo = DecisionTreeClassifier(criterion='gini', max_depth=5, random_state=42)
modelo.fit(X_train, y_train)

y_pred = modelo.predict(X_test)

print(classification_report(y_test, y_pred))

# %% [markdown]
# ## 6️⃣ Análise dos Resultados e Otimização
#
# ### a) Escolha da Métrica
#
# Como o foco é **reter clientes**, a métrica mais importante é o **Recall**,
# pois ela indica quantos clientes que realmente dariam churn foram corretamente identific
# ### b) Validação Cruzada (Cross-Validation)
#
```

```
# A validação cruzada divide o conjunto de dados em k partes (folds),
# garantindo uma avaliação mais robusta e menos dependente de uma única divisão aleatória.

from sklearn.model_selection import cross_val_score

scores = cross_val_score(modelo, X, y, cv=5, scoring='recall')
print("Recall médio com Cross-Validation:", scores.mean())

# %% [markdown]
# ## 7️⃣ Deploy Simulado – Função de Predição
#
# A função `prever_novo_cliente()` recebe um dicionário com os dados de um cliente,
# aplica o mesmo pré-processamento e retorna a probabilidade de churn e a decisão final.

def prever_novo_cliente(dados_cliente):
    import pandas as pd
    dados = pd.DataFrame([dados_cliente])
    dados_encoded = pd.get_dummies(dados)
    dados_encoded = dados_encoded.reindex(columns=X.columns, fill_value=0)

    prob = modelo.predict_proba(dados_encoded)[:, 1][0]
    decisao = "Churn" if prob >= 0.5 else "Não Churn"

    return round(prob, 3), decisao

# Exemplo de uso
novo_cliente = {
    'gender': 'Female',
    'SeniorCitizen': 0,
    'tenure': 6,
    'MonthlyCharges': 75.0,
    'Contract': 'Month-to-month'
}

prever_novo_cliente(novo_cliente)

# %% [markdown]
# ## ✅ Conclusão
#
# - O modelo de **Árvore de Decisão** foi capaz de prever clientes com alto risco de evasã
# - As etapas de **tratamento, codificação e normalização** garantiram a coerência lógica
# - A métrica **Recall** foi priorizada, refletindo o objetivo real do negócio: **reter cl
# - A **validação cruzada** reforçou a robustez do modelo.
#
# 💡 Este projeto demonstra a aplicação prática de **Lógica e Algoritmos** em um contexto
```

Comece a programar ou gere código com IA.