

projeto_churn

September 3, 2025

1 Projeto: Prevendo o Churn de Clientes

1.1 Análise Explanatoria

```
[3]: ## Pacotes

import pandas as pd
import matplotlib.pyplot as plt

from scipy import stats

from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.metrics import ConfusionMatrixDisplay
from sklearn.metrics import accuracy_score
from sklearn.metrics import recall_score
from sklearn.metrics import precision_score
from sklearn.model_selection import GridSearchCV

from sklearn.inspection import permutation_importance
from sklearn.metrics import roc_curve, roc_auc_score

## Modelos

from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import plot_tree
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from xgboost import XGBClassifier
## Função para Ajudar

[4]: from funcoes ajuda import descritiva, avalia_clf, relatorio_missing, ↵
      ↪analise_churn_completa

[5]: dados = pd.read_excel('Churn.xlsx')
```

```
## Target: 1 => deixou de ser cliente
## 0 >= caso contrario
```

```
[6]: ## informações gerais
print(dados.head())
print(dados.columns)
```

```
   idade      sexo  cidade categoria_plano  acesso_mes \
0     56  masculino  capital             sem           1
1     46  masculino  interior          basico          22
2     32   feminino  interior          prata           7
3     60   feminino  interior          prata          13
4     25   feminino  interior             sem          18

   valor_medio_transacao  reclamacao  churn
0                4518.37          sim      0
1                2479.51          sim      0
2                 782.91         nao      0
3                2638.41         nao      0
4                5335.34          sim      0
Index(['idade', 'sexo', 'cidade', 'categoria_plano', 'acesso_mes',
       'valor_medio_transacao', 'reclamacao', 'churn'],
      dtype='object')
```

```
[7]: dados.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 8 columns):
#   Column                Non-Null Count  Dtype
---  -
0   idade                 10000 non-null  int64
1   sexo                  10000 non-null  object
2   cidade                10000 non-null  object
3   categoria_plano       10000 non-null  object
4   acesso_mes            10000 non-null  int64
5   valor_medio_transacao 10000 non-null  float64
6   reclamacao            10000 non-null  object
7   churn                 10000 non-null  int64
dtypes: float64(1), int64(3), object(4)
memory usage: 625.1+ KB
```

```
[8]: # Análise Univariada
for variavel in dados.columns:
    print(f'\n\nAnálise univariada de {variavel}:')
    print(dados[variavel].describe())
```

Análise univariada de idade:

```
count    10000.000000
mean      41.054000
std       13.484104
min       18.000000
25%       30.000000
50%       41.000000
75%       53.000000
max       64.000000
```

Name: idade, dtype: float64

Análise univariada de sexo:

```
count      10000
unique       2
top    masculino
freq       5002
```

Name: sexo, dtype: object

Análise univariada de cidade:

```
count      10000
unique       2
top    capital
freq       5012
```

Name: cidade, dtype: object

Análise univariada de categoria_plano:

```
count      10000
unique       5
top    prata
freq       2437
```

Name: categoria_plano, dtype: object

Análise univariada de acesso_mes:

```
count    10000.000000
mean     12.463800
std       6.976472
min       1.000000
25%       6.000000
50%      12.000000
75%      19.000000
max      24.000000
```

Name: acesso_mes, dtype: float64

Análise univariada de valor_medio_transacao:

```
count    10000.000000
mean      5091.757660
std       2864.796975
min       100.920000
25%       2616.132500
50%       5110.245000
75%       7582.575000
max       9998.110000
```

Name: valor_medio_transacao, dtype: float64

Análise univariada de reclamacao:

```
count    10000
unique      2
top       nao
freq      6081
```

Name: reclamacao, dtype: object

Análise univariada de churn:

```
count    10000.000000
mean      0.288700
std       0.453181
min       0.000000
25%       0.000000
50%       0.000000
75%       1.000000
max       1.000000
```

Name: churn, dtype: float64

```
[9]: print(dados['churn'].value_counts())
      print(dados['sexo'].value_counts())
      print(dados['cidade'].value_counts())
      print(dados['categoria_plano'].value_counts())
      print(dados['reclamacao'].value_counts())
```

churn

```
0    7113
1     2887
```

Name: count, dtype: int64

sexo

```
masculino    5002
feminino     4998
```

Name: count, dtype: int64

cidade

```
capital      5012
interior     4988
```

```

Name: count, dtype: int64
categoria_plano
prata      2437
ouro       2113
basico     2009
sem        1952
platinum   1489
Name: count, dtype: int64
reclamacao
nao        6081
sim        3919
Name: count, dtype: int64

```

```
[10]: dados.columns
```

```
[10]: Index(['idade', 'sexo', 'cidade', 'categoria_plano', 'acesso_mes',
            'valor_medio_transacao', 'reclamacao', 'churn'],
            dtype='object')
```

```
[11]: for variavel in ['sexo', 'cidade', 'categoria_plano', 'churn', 'reclamacao']:
        print(f'\n\nFrequencias da variável: {variavel}')
        print(dados[variavel].value_counts(dropna=False).sort_index())
```

```

Frequencias da variável: sexo
sexo
feminino    4998
masculino   5002
Name: count, dtype: int64

```

```

Frequencias da variável: cidade
cidade
capital     5012
interior    4988
Name: count, dtype: int64

```

```

Frequencias da variável: categoria_plano
categoria_plano
basico        2009
ouro          2113
platinum      1489
prata         2437
sem           1952
Name: count, dtype: int64

```

```
Frequencias da variável: churn
churn
0      7113
1       2887
Name: count, dtype: int64
```

```
Frequencias da variável: reclamacao
reclamacao
nao      6081
sim       3919
Name: count, dtype: int64
```

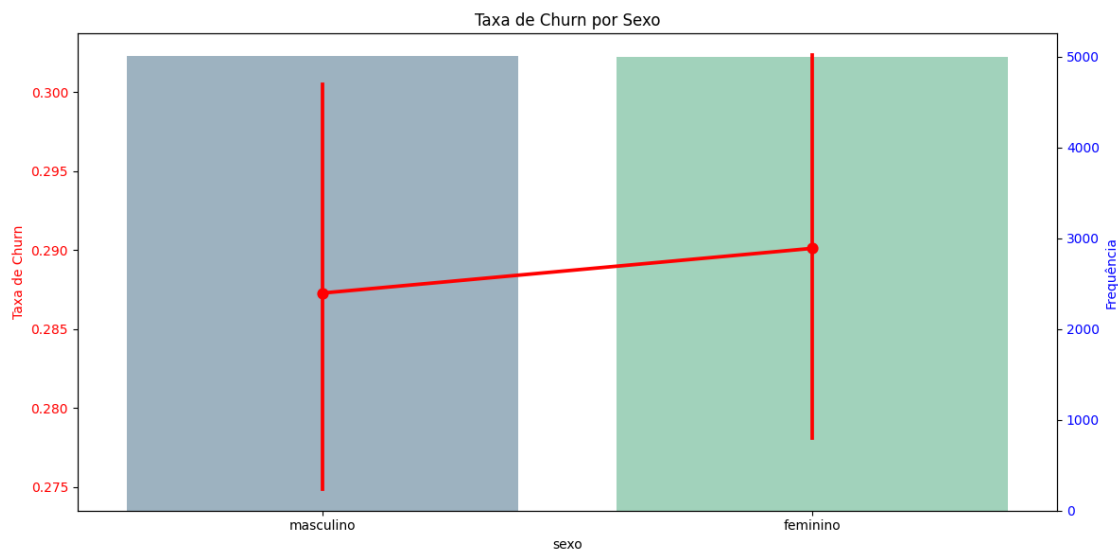
```
[12]: from funcoes_ajuda import descritiva
```

```
[13]: descritiva(dados, 'sexo', vresp='churn')
      descritiva(dados, 'cidade', vresp='churn')
      descritiva(dados, 'categoria_plano', vresp='churn')
      descritiva(dados, 'reclamacao', vresp='churn')
```

```
c:\Users\Usuário\Desktop\projetos_link\churn_clientes\funcoes_ajuda.py:39:
FutureWarning:
```

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

```
sns.countplot(data=df, x=var, palette='viridis', alpha=0.5, ax=ax2)
```



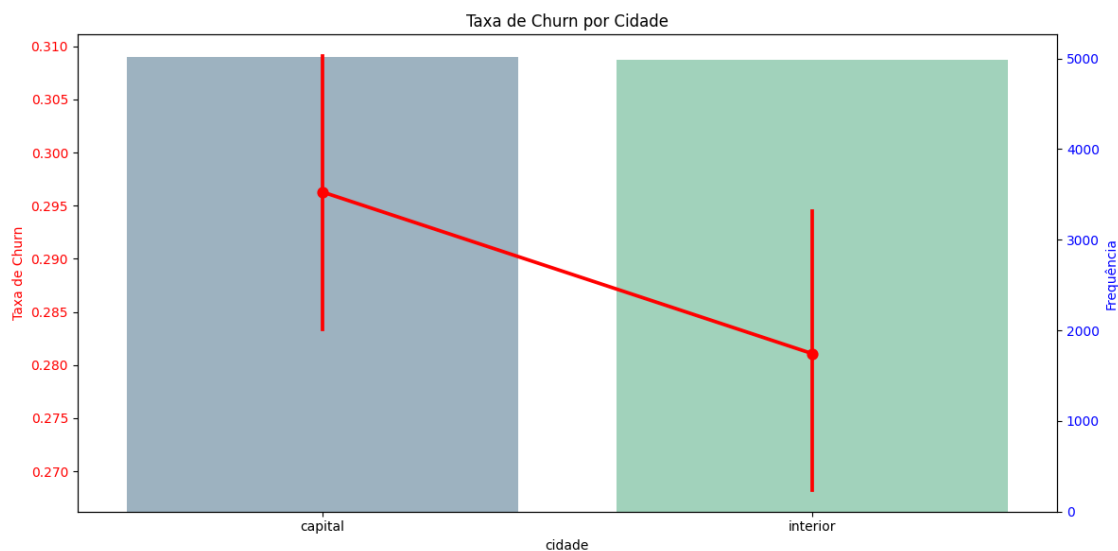
Estatísticas de churn por sexo:

	mean	count
sexo		
feminino	0.290	4998
masculino	0.287	5002

c:\Users\Usuário\Desktop\projetos_link\churn_clientes\funcoes_ajuda.py:39:
FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

```
sns.countplot(data=df, x=var, palette='viridis', alpha=0.5, ax=ax2)
```



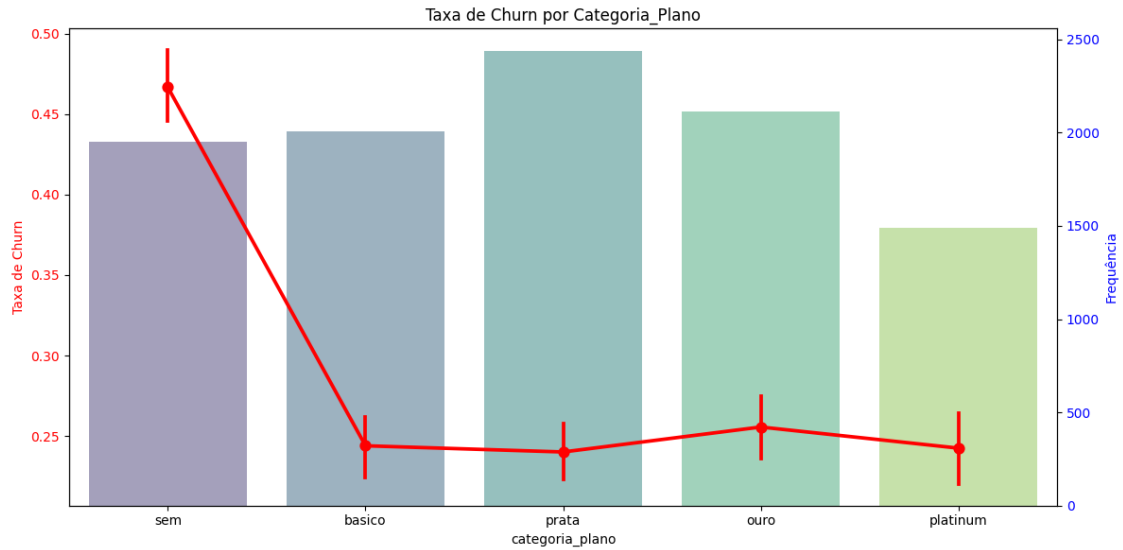
Estatísticas de churn por cidade:

	mean	count
cidade		
capital	0.296	5012
interior	0.281	4988

c:\Users\Usuário\Desktop\projetos_link\churn_clientes\funcoes_ajuda.py:39:
FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

```
sns.countplot(data=df, x=var, palette='viridis', alpha=0.5, ax=ax2)
```



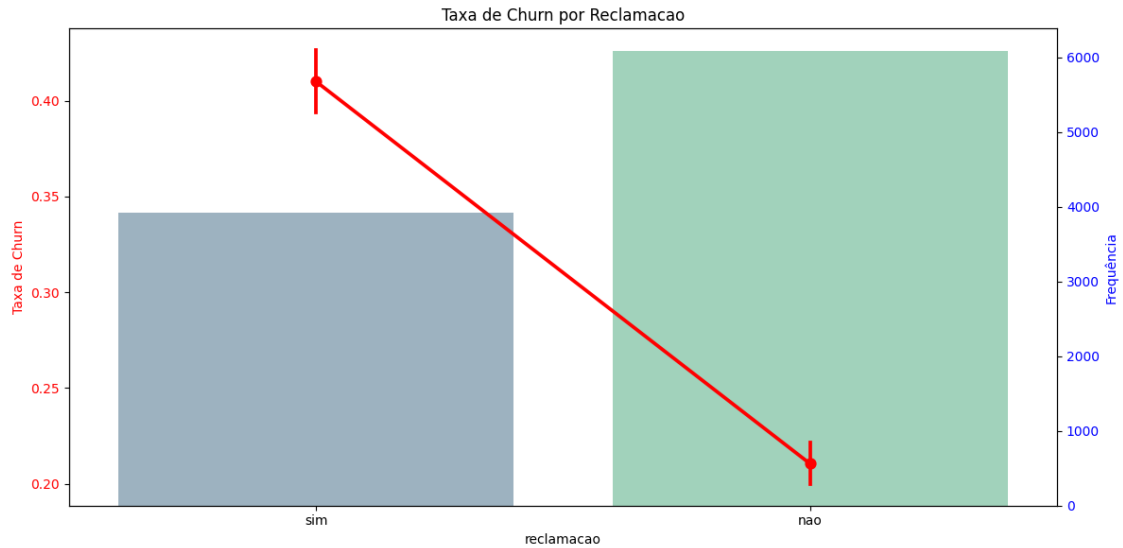
Estatísticas de churn por categoria_plano:

	mean	count
categoria_plano		
basico	0.244	2009
ouro	0.256	2113
platinum	0.242	1489
prata	0.240	2437
sem	0.467	1952

c:\Users\Usuário\Desktop\projetos_link\churn_clientes\funcoes_ajuda.py:39:
FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

```
sns.countplot(data=df, x=var, palette='viridis', alpha=0.5, ax=ax2)
```



Estatísticas de churn por reclamacao:

	mean	count
reclamacao		
nao	0.21	6081
sim	0.41	3919

```
[13]:
```

	mean	count
reclamacao		
nao	0.210492	6081
sim	0.410054	3919

```
[14]: from funcoes_ajuda import relatorio_missing
relatorio_missing(dados)
```

Número de linhas: 10,000 | Número de colunas: 8

Total de valores missing: 0

Nenhum valor missing encontrado!

```
[14]:
```

	Pct_missing	Freq_missing
idade	0.0%	0
sexo	0.0%	0
cidade	0.0%	0
categoria_plano	0.0%	0
acesso_mes	0.0%	0
valor_medio_transacao	0.0%	0
reclamacao	0.0%	0
churn	0.0%	0

```
[15]: def print_churn_analysis(crosstab_df, title):
        df_formatted = (crosstab_df.rename(columns={0: 'Não Cancelou', 1:
        ↪ 'Cancelou'}) * 100).round(2)
        print(f" {title}:")
        print(df_formatted)
        print("-" * 50)

    print("=== ANÁLISE DE TAXA DE CHURN ===")
    print_churn_analysis(pd.crosstab(dados['sexo'], dados['churn'],
    ↪ normalize='index'), "Churn por Sexo")
    print_churn_analysis(pd.crosstab(dados['cidade'], dados['churn'],
    ↪ normalize='index'), "Churn por Cidade")
    print_churn_analysis(pd.crosstab(dados['categoria_plano'], dados['churn'],
    ↪ normalize='index'), "Churn por Categoria do Plano")
    print_churn_analysis(pd.crosstab(dados['reclamacao'], dados['churn'],
    ↪ normalize='index'), "Churn por Reclamação")
```

=== ANÁLISE DE TAXA DE CHURN ===

Churn por Sexo:

churn	Não Cancelou	Cancelou
sexo		
feminino	70.99	29.01
masculino	71.27	28.73

Churn por Cidade:

churn	Não Cancelou	Cancelou
cidade		
capital	70.37	29.63
interior	71.89	28.11

Churn por Categoria do Plano:

churn	Não Cancelou	Cancelou
categoria_plano		
basico	75.61	24.39
ouro	74.44	25.56
platinum	75.76	24.24
prata	76.00	24.00
sem	53.33	46.67

Churn por Reclamação:

churn	Não Cancelou	Cancelou
reclamacao		
nao	78.95	21.05
sim	58.99	41.01

1.2 Preparando os Dados

```
[16]: dados = pd.read_excel('Churn.xlsx')

[17]: dados = pd.get_dummies(dados, columns=['sexo', 'cidade', 'categoria_plano', 'reclamacao'], drop_first=False, dtype='float')

[18]: dados.columns

[18]: Index(['idade', 'acesso_mes', 'valor_medio_transacao', 'churn',
          'sexo_feminino', 'sexo_masculino', 'cidade_capital', 'cidade_interior',
          'categoria_plano_basico', 'categoria_plano_ouro',
          'categoria_plano_platinum', 'categoria_plano_prata',
          'categoria_plano_sem', 'reclamacao_nao', 'reclamacao_sim'],
          dtype='object')

[19]: X = dados.drop(columns=['churn', 'sexo_feminino', 'cidade_interior', 'reclamacao_nao'])
      y = dados['churn']

[20]: X.columns
      dados.info()
      dados.head()
```

```
<class 'pandas.core.frame.DataFrame'>
```

```
RangeIndex: 10000 entries, 0 to 9999
```

```
Data columns (total 15 columns):
```

#	Column	Non-Null Count	Dtype
0	idade	10000 non-null	int64
1	acesso_mes	10000 non-null	int64
2	valor_medio_transacao	10000 non-null	float64
3	churn	10000 non-null	int64
4	sexo_feminino	10000 non-null	float64
5	sexo_masculino	10000 non-null	float64
6	cidade_capital	10000 non-null	float64
7	cidade_interior	10000 non-null	float64
8	categoria_plano_basico	10000 non-null	float64
9	categoria_plano_ouro	10000 non-null	float64
10	categoria_plano_platinum	10000 non-null	float64
11	categoria_plano_prata	10000 non-null	float64
12	categoria_plano_sem	10000 non-null	float64
13	reclamacao_nao	10000 non-null	float64
14	reclamacao_sim	10000 non-null	float64

```
dtypes: float64(12), int64(3)
```

```
memory usage: 1.1 MB
```

```
[20]:
```

	idade	acesso_mes	valor_medio_transacao	churn	sexo_feminino	\
0	56	1	4518.37	0	0.0	
1	46	22	2479.51	0	0.0	
2	32	7	782.91	0	1.0	
3	60	13	2638.41	0	1.0	
4	25	18	5335.34	0	1.0	

	sexo_masculino	cidade_capital	cidade_interior	categoria_plano_basico	\
0	1.0	1.0	0.0	0.0	
1	1.0	0.0	1.0	1.0	
2	0.0	0.0	1.0	0.0	
3	0.0	0.0	1.0	0.0	
4	0.0	0.0	1.0	0.0	

	categoria_plano_ouro	categoria_plano_platinum	categoria_plano_prata	\
0	0.0	0.0	0.0	
1	0.0	0.0	0.0	
2	0.0	0.0	1.0	
3	0.0	0.0	1.0	
4	0.0	0.0	0.0	

	categoria_plano_sem	reclamacao_nao	reclamacao_sim
0	1.0	0.0	1.0
1	0.0	0.0	1.0
2	0.0	1.0	0.0
3	0.0	1.0	0.0
4	1.0	0.0	1.0

```
[21]: dados.to_pickle('dados_churn.pkl')
      dados.to_parquet('dados_churn.parquet')
```

1.3 Modelos e Avaliações

1.3.1 Separando os Dados para Modelagem

```
[22]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
      ↪ random_state=42, stratify=y)
```

```
[23]: X.columns
```

```
[23]: Index(['idade', 'acesso_mes', 'valor_medio_transacao', 'sexo_masculino',
      'cidade_capital', 'categoria_plano_basico', 'categoria_plano_ouro',
      'categoria_plano_platinum', 'categoria_plano_prata',
      'categoria_plano_sem', 'reclamacao_sim'],
      dtype='object')
```

```
[24]: ### Padronização das variáveis métricas
```

```

# Vamos armazenar as informações de média e desvio padrão para previsões
media_idade, dp_idade = (X_train['idade'].mean(), X_train['idade'].std(ddof=1))
media_acessos, dp_acessos = (X_train['acesso_mes'].mean(),
    ↪X_train['acesso_mes'].std(ddof=1))
media_valor, dp_valor = (X_train['valor_medio_transacao'].mean(),
    ↪X_train['valor_medio_transacao'].std(ddof=1))

# No banco de dados de treino
X_train['idade'] = stats.zscore(X_train['idade'], ddof=1)
X_train['acesso_mes'] = stats.zscore(X_train['acesso_mes'], ddof=1)
X_train['valor_medio_transacao'] = stats.
    ↪zscore(X_train['valor_medio_transacao'], ddof=1)

# No banco de dados de teste
X_test['idade'] = stats.zscore(X_test['idade'], ddof=1)
X_test['acesso_mes'] = stats.zscore(X_test['acesso_mes'], ddof=1)
X_test['valor_medio_transacao'] = stats.zscore(X_test['valor_medio_transacao'],
    ↪ddof=1)

```

1.3.2 Modelos Baseados em Arvore

```

[25]: # definindo

arvore = DecisionTreeClassifier(criterion='gini', max_depth=3, random_state=42)
xgb = XGBClassifier(use_label_encoder=False, eval_metric='logloss',
    ↪random_state=42)
random_forest = RandomForestClassifier(n_estimators=100, random_state=42)

# Treinando

arvore.fit(X_train, y_train)
xgb.fit(X_train, y_train)
random_forest.fit(X_train, y_train)

```

C:\Users\Usuário\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.11_qbz5n2kfra8p0\LocalCache\local-packages\Python311\site-packages\xgboost\training.py:183: UserWarning: [11:36:46] WARNING: C:\actions-runner_work\xgboost\xgboost\src\learner.cc:738: Parameters: { "use_label_encoder" } are not used.

```
bst.update(dtrain, iteration=i, fobj=obj)
```

```
[25]: RandomForestClassifier(random_state=42)
```

1.4 Modelos SVM

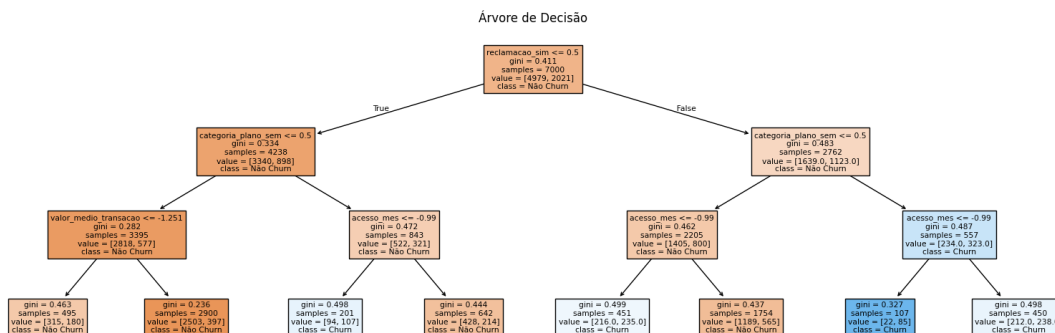
```
[26]: ## Definindo
svm_linear = SVC(kernel='linear', probability=True, random_state=42)
svm_pol = SVC(kernel='poly', degree=3, probability=True, random_state=42)
svm_rbf = SVC(kernel='rbf', probability=True, random_state=42)

## Treinando
svm_linear.fit(X_train, y_train)
svm_pol.fit(X_train, y_train)
svm_rbf.fit(X_train, y_train)
```

```
[26]: SVC(probability=True, random_state=42)
```

```
[27]: from funcoes ajuda import avalia_clf
```

```
[28]: plt.figure(figsize=(20, 6))
plot_tree(arvore, filled=True, feature_names=X.columns, class_names=['Não Churn', 'Churn'])
plt.title('Árvore de Decisão')
plt.show()
```



1.5 Métricas de Avaliação do Modelo

Ao avaliar modelos de classificação, é importante utilizar diferentes métricas, pois cada uma fornece uma perspectiva distinta sobre a qualidade das previsões. Abaixo estão as principais métricas:

- **Acurácia**

Proporção de previsões corretas sobre o total de amostras.

- Boa quando: as classes estão balanceadas.

- Ruim quando: há desbalanceamento, pois pode dar uma falsa impressão de bom desempenho.

- **Acurácia Balanceada**

Média das acurácias obtidas em cada classe. Corrige o problema da acurácia em bases des-

balanceadas.

- Boa quando: há equilíbrio no desempenho entre classes.

- Ruim quando: o modelo acerta muito uma classe, mas erra muito a outra.

- **Precisão (Precision)**

Proporção de verdadeiros positivos entre todas as previsões positivas. Mede a qualidade das previsões positivas.

- Boa quando: poucos falsos positivos são importantes (ex.: diagnóstico de doença grave).

- Ruim quando: muitos negativos são classificados incorretamente como positivos.

- **Recall (Sensibilidade)**

Proporção de verdadeiros positivos detectados entre todos os casos positivos reais. Mede a capacidade do modelo de encontrar todos os positivos.

- Boa quando: não perder positivos é crucial (ex.: detectar fraudes ou doenças).

- Ruim quando: o modelo deixa escapar muitos positivos.

- **F1-Score**

Média harmônica entre precisão e recall. Balanceia os dois aspectos.

- Boa quando: há necessidade de equilíbrio entre precisão e recall.

- Ruim quando: o modelo tem desempenho muito desigual entre precisão e recall.

- **AUC-ROC (Área sob a Curva ROC)**

Mede a capacidade do modelo em diferenciar entre classes, variando o limiar de decisão.

- Boa quando: próximo de 1 (excelente separação).

- Ruim quando: próximo de 0.5 (sem poder discriminativo, equivalente a sorteio aleatório).

- **Índice de Gini**

Derivado da AUC, usado principalmente em modelos de crédito. Calculado como $GINI = 2 * AUC - 1$.

- Boa quando: próximo de 1 (modelo tem alto poder discriminativo).

- Ruim quando: próximo de 0 (modelo não diferencia as classes).

1.5.1 Modelos Arvores

[29]: *## Treino*

```
avalia_clf(arvore, y_train, X_train, base='treino')
avalia_clf(random_forest, y_train, X_train, base='treino')
avalia_clf(xgb, y_train, X_train, base='treino')
```

DECISIONTREECLASSIFIER - BASE DE TREINO:

=====

Acurácia: 72.9%

Acurácia Balanceada: 61.0%

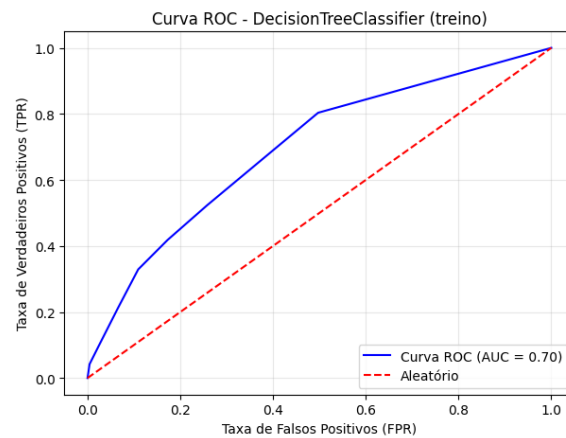
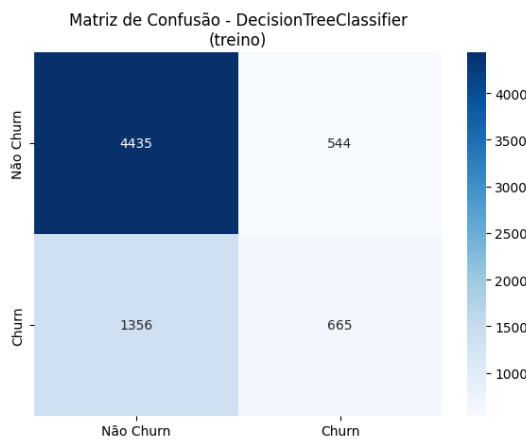
Precision: 55.0%

Recall: 32.9%

F1-Score: 41.2%
AUC-ROC: 69.57%
GINI: 39.13%

RELATÓRIO DE CLASSIFICAÇÃO:

	precision	recall	f1-score	support
Não Churn	0.77	0.89	0.82	4979
Churn	0.55	0.33	0.41	2021
accuracy			0.73	7000
macro avg	0.66	0.61	0.62	7000
weighted avg	0.70	0.73	0.70	7000



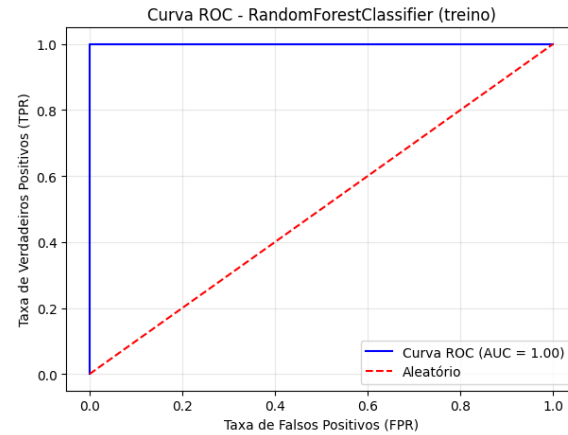
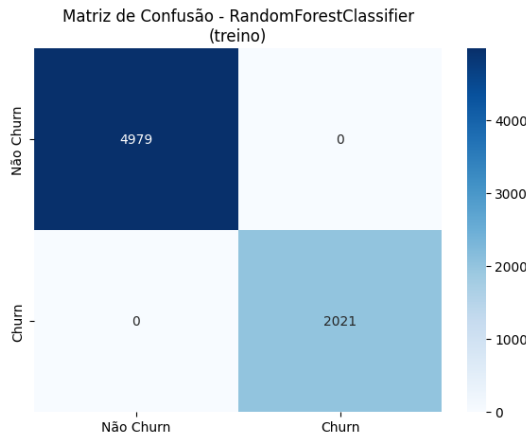
RANDOMFORESTCLASSIFIER - BASE DE TREINO:

Acurácia: 100.0%
Acurácia Balanceada: 100.0%
Precision: 100.0%
Recall: 100.0%
F1-Score: 100.0%
AUC-ROC: 100.00%
GINI: 100.00%

RELATÓRIO DE CLASSIFICAÇÃO:

	precision	recall	f1-score	support
Não Churn	1.00	1.00	1.00	4979

Churn	1.00	1.00	1.00	2021
accuracy			1.00	7000
macro avg	1.00	1.00	1.00	7000
weighted avg	1.00	1.00	1.00	7000



XGBCLASSIFIER - BASE DE TREINO:

=====

Acurácia: 90.4%

Acurácia Balanceada: 84.5%

Precision: 95.1%

Recall: 70.4%

F1-Score: 80.9%

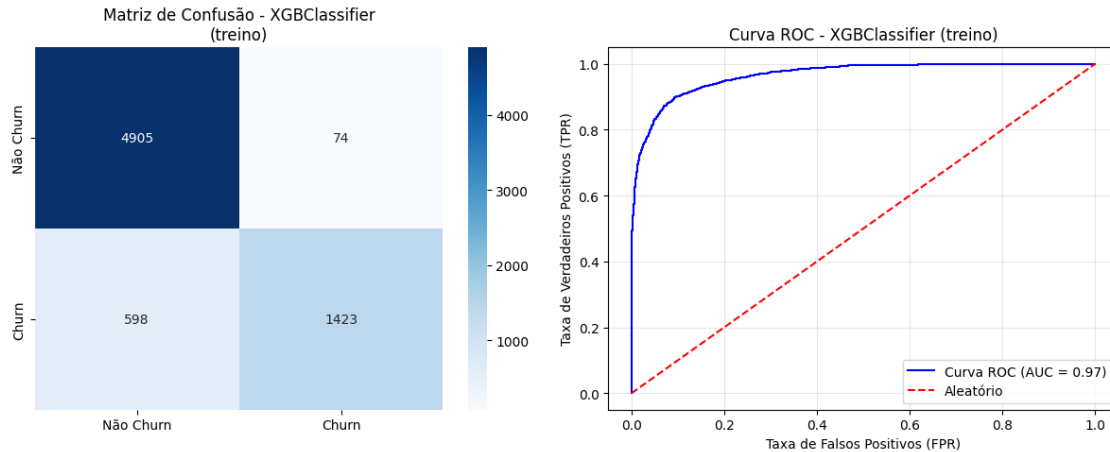
AUC-ROC: 96.50%

GINI: 93.00%

RELATÓRIO DE CLASSIFICAÇÃO:

=====

	precision	recall	f1-score	support
Não Churn	0.89	0.99	0.94	4979
Churn	0.95	0.70	0.81	2021
accuracy			0.90	7000
macro avg	0.92	0.84	0.87	7000
weighted avg	0.91	0.90	0.90	7000



```
[29]: {'modelo': 'XGBClassifier',
      'base': 'treino',
      'acuracia': 0.904,
      'acuracia_balanceada': 0.8446222278050742,
      'precision': 0.9505678022712091,
      'recall': 0.7041068777832756,
      'f1': 0.808982376350199,
      'auc': 0.9650135715974435,
      'gini': 0.930027143194887}
```

```
[30]: ### Base de Teste
avalia_clf(arvore, y_train, X_train, base='teste')
avalia_clf(random_forest, y_train, X_train, base='teste')
avalia_clf(xgb, y_train, X_train, base='teste')
```

DECISIONTREECLASSIFIER - BASE DE TESTE:

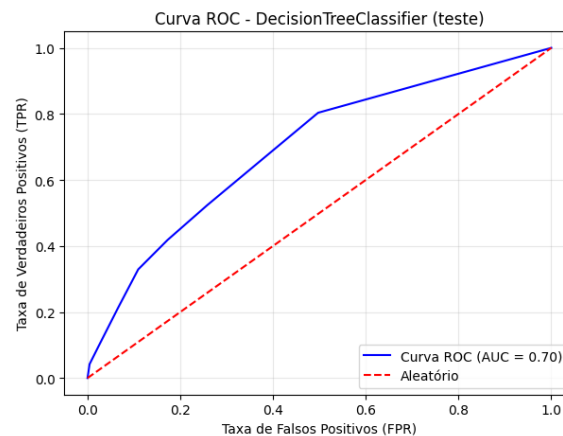
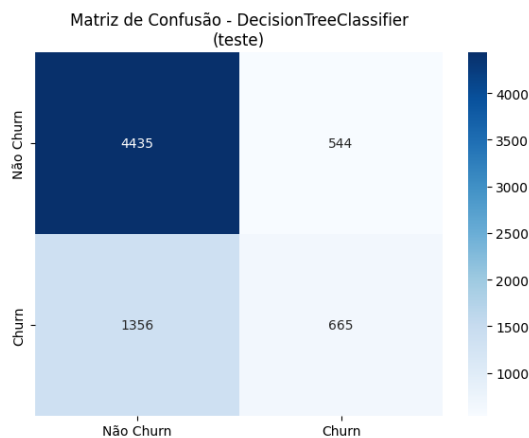
```
=====
Acurácia: 72.9%
Acurácia Balanceada: 61.0%
Precision: 55.0%
Recall: 32.9%
F1-Score: 41.2%
AUC-ROC: 69.57%
GINI: 39.13%
```

RELATÓRIO DE CLASSIFICAÇÃO:

```
=====
precision    recall  f1-score   support

Não Churn    0.77     0.89     0.82     4979
Churn        0.55     0.33     0.41     2021
```

accuracy			0.73	7000
macro avg	0.66	0.61	0.62	7000
weighted avg	0.70	0.73	0.70	7000



RANDOMFORESTCLASSIFIER - BASE DE TESTE:

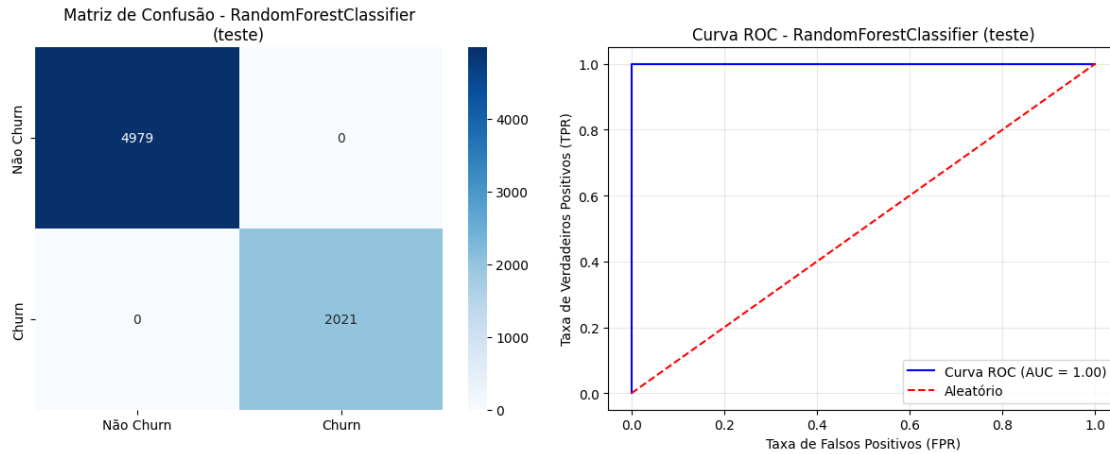
```
=====
Acurácia: 100.0%
Acurácia Balanceada: 100.0%
Precision: 100.0%
Recall: 100.0%
F1-Score: 100.0%
AUC-ROC: 100.00%
GINI: 100.00%
```

RELATÓRIO DE CLASSIFICAÇÃO:

```
=====
              precision    recall  f1-score   support

   Não Churn         1.00      1.00      1.00     4979
     Churn          1.00      1.00      1.00     2021

   accuracy              1.00              7000
   macro avg              1.00              7000
   weighted avg           1.00              7000
```

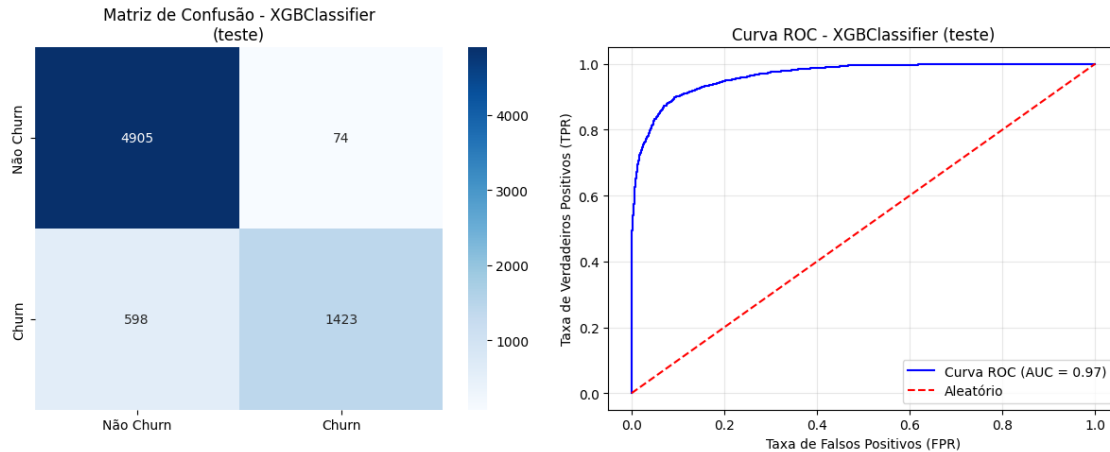


XGBCLASSIFIER - BASE DE TESTE:

Acurácia: 90.4%
 Acurácia Balanceada: 84.5%
 Precision: 95.1%
 Recall: 70.4%
 F1-Score: 80.9%
 AUC-ROC: 96.50%
 GINI: 93.00%

RELATÓRIO DE CLASSIFICAÇÃO:

	precision	recall	f1-score	support
Não Churn	0.89	0.99	0.94	4979
Churn	0.95	0.70	0.81	2021
accuracy			0.90	7000
macro avg	0.92	0.84	0.87	7000
weighted avg	0.91	0.90	0.90	7000



```
[30]: {'modelo': 'XGBClassifier',
      'base': 'teste',
      'acuracia': 0.904,
      'acuracia_balanceada': 0.8446222278050742,
      'precision': 0.9505678022712091,
      'recall': 0.7041068777832756,
      'f1': 0.808982376350199,
      'auc': 0.9650135715974435,
      'gini': 0.930027143194887}
```

1.5.2 Modelos SVM

```
[31]: avalia_clf(svm_linear, y_train, X_train, base='treino')
      avalia_clf(svm_pol, y_train, X_train, base='treino')

      avalia_clf(svm_rbf, y_train, X_train, base='treino')
```

```
C:\Users\Usuário\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.11_qbz
5n2kfra8p0\LocalCache\local-packages\Python311\site-
packages\sklearn\metrics\_classification.py:1517: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 due to no predicted samples. Use
`zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
C:\Users\Usuário\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.11_qbz
5n2kfra8p0\LocalCache\local-packages\Python311\site-
packages\sklearn\metrics\_classification.py:1517: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
C:\Users\Usuário\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.11_qbz
5n2kfra8p0\LocalCache\local-packages\Python311\site-
packages\sklearn\metrics\_classification.py:1517: UndefinedMetricWarning:
```

Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero\_division` parameter to control this behavior.

```
_warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
C:\Users\Usuário\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.11_qbz5n2kfra8p0\LocalCache\local-packages\Python311\site-packages\sklearn\metrics\_classification.py:1517: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
_warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

SVC - BASE DE TREINO:

=====

Acurácia: 71.1%

Acurácia Balanceada: 50.0%

Precision: 0.0%

Recall: 0.0%

F1-Score: 0.0%

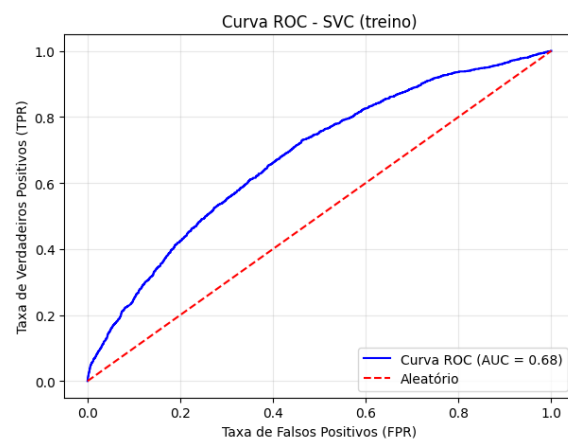
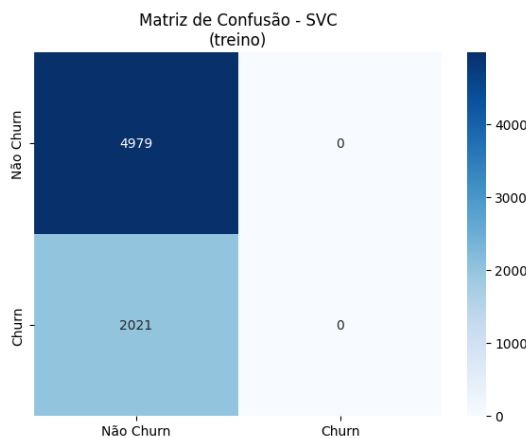
AUC-ROC: 67.85%

GINI: 35.70%

RELATÓRIO DE CLASSIFICAÇÃO:

=====

	precision	recall	f1-score	support
Não Churn	0.71	1.00	0.83	4979
Churn	0.00	0.00	0.00	2021
accuracy			0.71	7000
macro avg	0.36	0.50	0.42	7000
weighted avg	0.51	0.71	0.59	7000

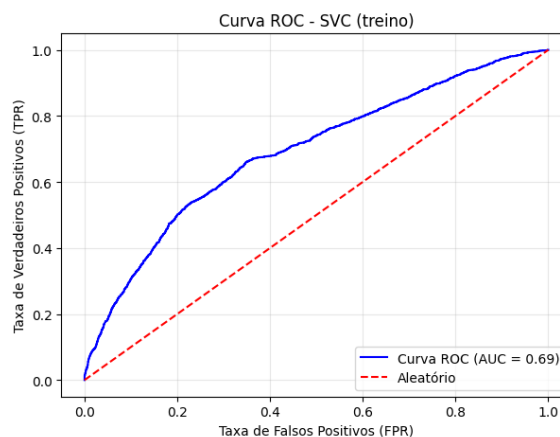
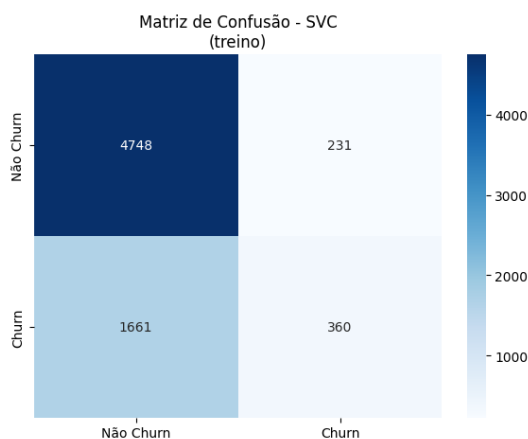


SVC - BASE DE TREINO:

Acurácia: 73.0%
Acurácia Balanceada: 56.6%
Precision: 60.9%
Recall: 17.8%
F1-Score: 27.6%
AUC-ROC: 69.15%
GINI: 38.29%

RELATÓRIO DE CLASSIFICAÇÃO:

	precision	recall	f1-score	support
Não Churn	0.74	0.95	0.83	4979
Churn	0.61	0.18	0.28	2021
accuracy			0.73	7000
macro avg	0.67	0.57	0.55	7000
weighted avg	0.70	0.73	0.67	7000



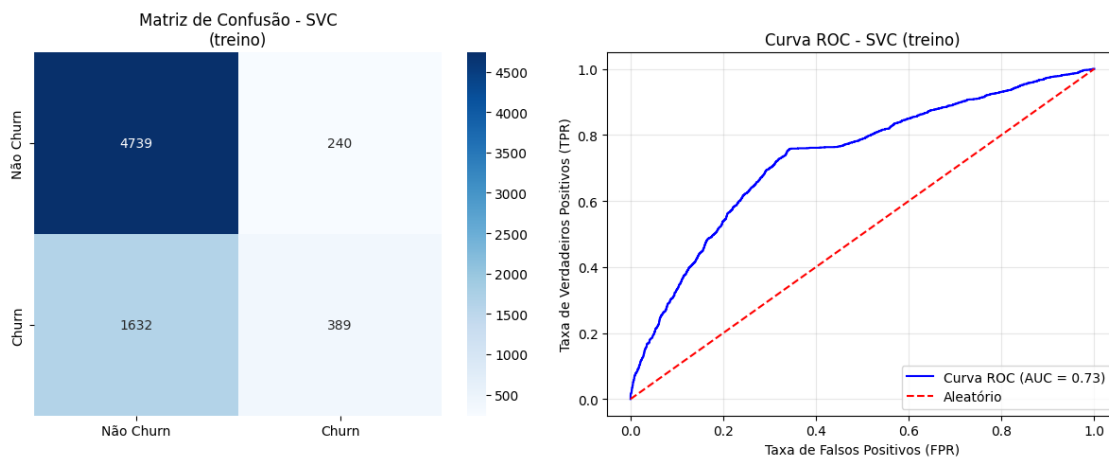
SVC - BASE DE TREINO:

Acurácia: 73.3%
Acurácia Balanceada: 57.2%
Precision: 61.8%
Recall: 19.2%
F1-Score: 29.4%
AUC-ROC: 73.14%

GINI: 46.28%

RELATÓRIO DE CLASSIFICAÇÃO:

	precision	recall	f1-score	support
Não Churn	0.74	0.95	0.84	4979
Churn	0.62	0.19	0.29	2021
accuracy			0.73	7000
macro avg	0.68	0.57	0.56	7000
weighted avg	0.71	0.73	0.68	7000



```
[31]: {'modelo': 'SVC',  
      'base': 'treino',  
      'acuracia': 0.7325714285714285,  
      'acuracia_balanceada': 0.5721382602576541,  
      'precision': 0.6184419713831478,  
      'recall': 0.19247897080653142,  
      'f1': 0.29358490566037737,  
      'auc': 0.7314028171164015,  
      'gini': 0.4628056342328031}
```

```
[32]: ### Base de Teste
```

```
avalia_clf(svm_linear, y_train, X_train, base='teste')  
avalia_clf(svm_pol, y_train, X_train, base='teste')  
avalia_clf(svm_rbf, y_train, X_train, base='teste')
```

C:\Users\Usuário\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.11_qbz5n2kfra8p0\LocalCache\local-packages\Python311\site-

```

packages\sklearn\metrics\_classification.py:1517: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 due to no predicted samples. Use
`zero_division` parameter to control this behavior.
    _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
C:\Users\Usuário\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.11_qbz
5n2kfra8p0\LocalCache\local-packages\Python311\site-
packages\sklearn\metrics\_classification.py:1517: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
    _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
C:\Users\Usuário\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.11_qbz
5n2kfra8p0\LocalCache\local-packages\Python311\site-
packages\sklearn\metrics\_classification.py:1517: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
    _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
C:\Users\Usuário\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.11_qbz
5n2kfra8p0\LocalCache\local-packages\Python311\site-
packages\sklearn\metrics\_classification.py:1517: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
    _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))

```

SVC - BASE DE TESTE:

```
=====
```

Acurácia: 71.1%

Acurácia Balanceada: 50.0%

Precision: 0.0%

Recall: 0.0%

F1-Score: 0.0%

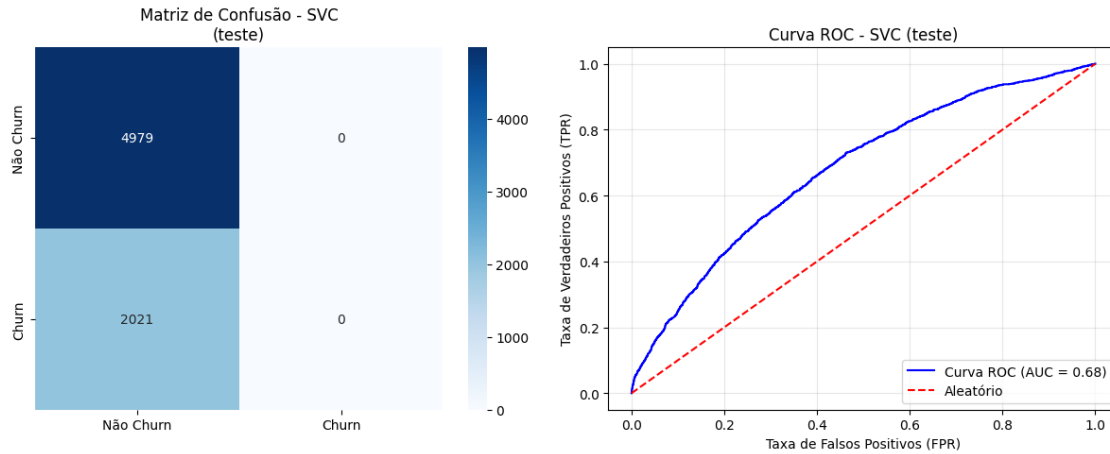
AUC-ROC: 67.85%

GINI: 35.70%

RELATÓRIO DE CLASSIFICAÇÃO:

```
=====
```

	precision	recall	f1-score	support
Não Churn	0.71	1.00	0.83	4979
Churn	0.00	0.00	0.00	2021
accuracy			0.71	7000
macro avg	0.36	0.50	0.42	7000
weighted avg	0.51	0.71	0.59	7000



SVC - BASE DE TESTE:

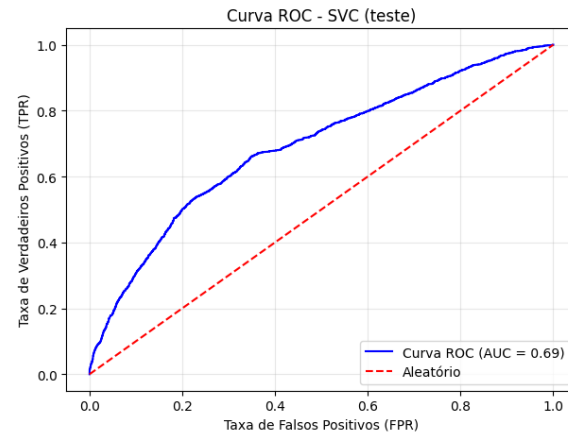
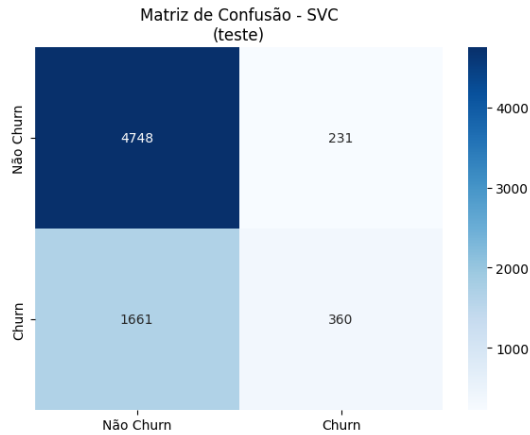
=====

Acurácia: 73.0%
Acurácia Balanceada: 56.6%
Precision: 60.9%
Recall: 17.8%
F1-Score: 27.6%
AUC-ROC: 69.15%
GINI: 38.29%

RELATÓRIO DE CLASSIFICAÇÃO:

=====

	precision	recall	f1-score	support
Não Churn	0.74	0.95	0.83	4979
Churn	0.61	0.18	0.28	2021
accuracy			0.73	7000
macro avg	0.67	0.57	0.55	7000
weighted avg	0.70	0.73	0.67	7000

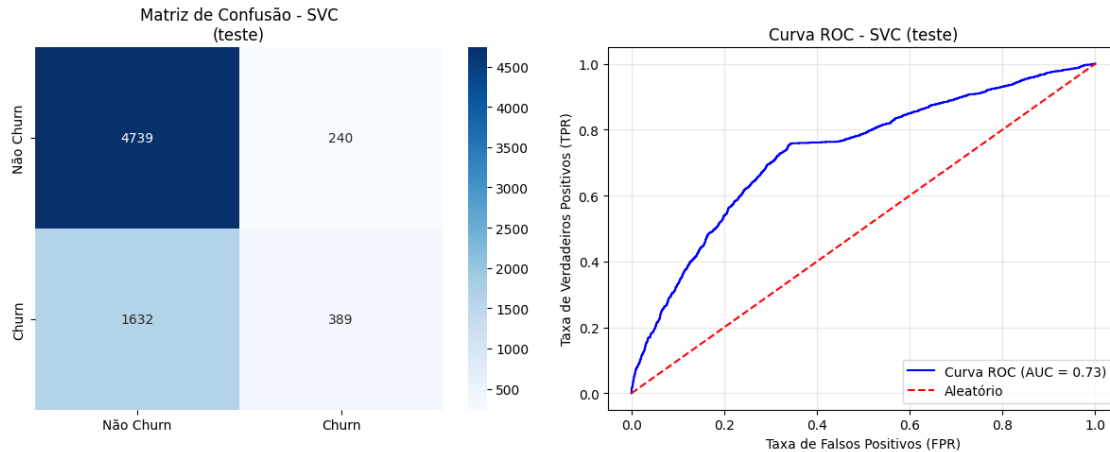


SVC - BASE DE TESTE:

Acurácia: 73.3%
 Acurácia Balanceada: 57.2%
 Precision: 61.8%
 Recall: 19.2%
 F1-Score: 29.4%
 AUC-ROC: 73.14%
 GINI: 46.28%

RELATÓRIO DE CLASSIFICAÇÃO:

	precision	recall	f1-score	support
Não Churn	0.74	0.95	0.84	4979
Churn	0.62	0.19	0.29	2021
accuracy			0.73	7000
macro avg	0.68	0.57	0.56	7000
weighted avg	0.71	0.73	0.68	7000



```
[32]: {'modelo': 'SVC',
      'base': 'teste',
      'acuracia': 0.7325714285714285,
      'acuracia_balanceada': 0.5721382602576541,
      'precision': 0.6184419713831478,
      'recall': 0.19247897080653142,
      'f1': 0.29358490566037737,
      'auc': 0.7314028171164015,
      'gini': 0.4628056342328031}
```

1.6 Comparando Os Resultados

```
[33]: # Execute esta célula - é a mais simples e funciona!
modelos_dict = {
    'Árvore': arvore,
    'SVM Linear': svm_linear,
    'Random Forest': random_forest,
    'XGBoost': xgb,
    'SVM Polinomial': svm_pol,
    'SVM RBF': svm_rbf,
}

# Coletar e plotar automaticamente
modelos_lista = []
resultados_lista = []

for nome, modelo in modelos_dict.items():
    try:
        acc = accuracy_score(y_test, modelo.predict(X_test))
        modelos_lista.append(nome)
        resultados_lista.append(acc)
```

```

        print(f" {nome}: {acc:.3f}")
    except:
        print(f" {nome}: erro na predição")

# Plotar
plt.figure(figsize=(12, 10))
plt.barh(modelos_lista, resultados_lista, color='lightblue')
plt.title('Acurácia dos Modelos na Base de Teste')
plt.xlabel('Acurácia')
plt.xlim(0, 1)

for i, acc in enumerate(resultados_lista):
    plt.text(acc + 0.01, i, f'{acc:.1%}', va='center', fontweight='bold')

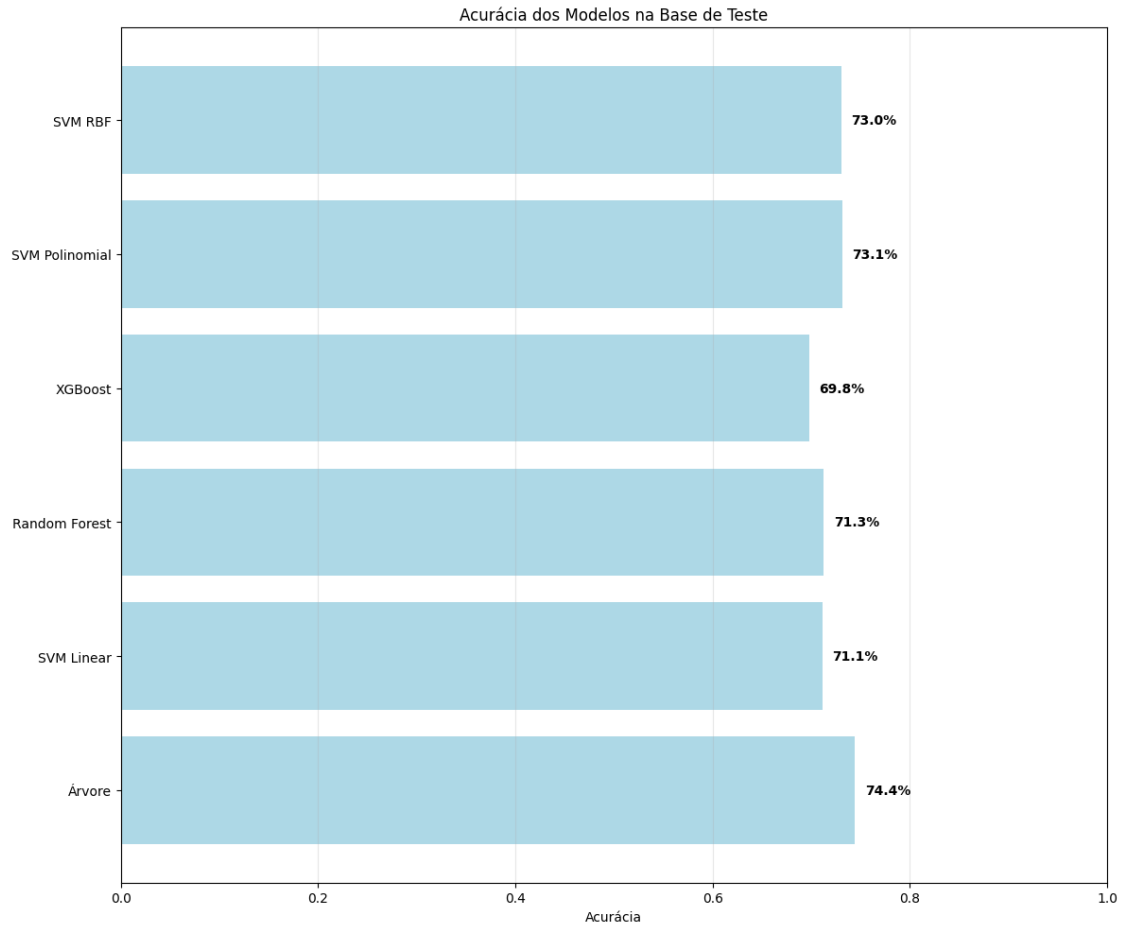
plt.grid(axis='x', alpha=0.3)
plt.tight_layout()
plt.show()

```

```

Árvore: 0.744
SVM Linear: 0.711
Random Forest: 0.713
XGBoost: 0.698
SVM Polinomial: 0.731
SVM RBF: 0.730

```



1.7 Validação Cruzada

1.7.1 SVM

```
[34]: param_grid = [  
    {'C': [0.1, 1, 10], 'kernel': ['linear']},  
    {'C': [0.1, 1, 10], 'degree': [2, 3], 'coef0': [0, 1], 'kernel': ['poly']},  
    {'C': [0.1, 1, 10], 'gamma': [0.1, 1, 10, 100], 'kernel': ['rbf']},  
]
```

```
[ ]:
```

```
[35]: svm_grid = SVC()  
  
svm_grid = GridSearchCV(estimator=svm_grid,  
                        param_grid= param_grid,  
                        scoring='accuracy',  
                        cv=5,
```

```

        verbose=2,
    )

    svm_grid.fit(X_train, y_train)

```

Fitting 5 folds for each of 27 candidates, totalling 135 fits

```

[CV] END ...C=0.1, kernel=linear; total time= 0.3s
[CV] END ...C=0.1, kernel=linear; total time= 0.3s
[CV] END ...C=0.1, kernel=linear; total time= 0.3s
[CV] END ...C=0.1, kernel=linear; total time= 0.3s
[CV] END ...C=0.1, kernel=linear; total time= 0.3s
[CV] END ...C=1, kernel=linear; total time= 0.4s
[CV] END ...C=1, kernel=linear; total time= 0.5s
[CV] END ...C=1, kernel=linear; total time= 0.4s
[CV] END ...C=1, kernel=linear; total time= 0.4s
[CV] END ...C=1, kernel=linear; total time= 0.4s
[CV] END ...C=10, kernel=linear; total time= 0.8s
[CV] END ...C=10, kernel=linear; total time= 0.8s
[CV] END ...C=10, kernel=linear; total time= 0.7s
[CV] END ...C=10, kernel=linear; total time= 0.8s
[CV] END ...C=10, kernel=linear; total time= 0.8s
[CV] END ...C=0.1, coef0=0, degree=2, kernel=poly; total time= 0.5s
[CV] END ...C=0.1, coef0=0, degree=2, kernel=poly; total time= 0.4s
[CV] END ...C=0.1, coef0=0, degree=2, kernel=poly; total time= 0.4s
[CV] END ...C=0.1, coef0=0, degree=2, kernel=poly; total time= 0.4s
[CV] END ...C=0.1, coef0=0, degree=2, kernel=poly; total time= 0.5s
[CV] END ...C=0.1, coef0=0, degree=3, kernel=poly; total time= 0.5s
[CV] END ...C=0.1, coef0=0, degree=3, kernel=poly; total time= 0.5s
[CV] END ...C=0.1, coef0=0, degree=3, kernel=poly; total time= 0.5s
[CV] END ...C=0.1, coef0=0, degree=3, kernel=poly; total time= 0.5s
[CV] END ...C=0.1, coef0=0, degree=3, kernel=poly; total time= 0.5s
[CV] END ...C=0.1, coef0=1, degree=2, kernel=poly; total time= 0.5s
[CV] END ...C=0.1, coef0=1, degree=2, kernel=poly; total time= 0.5s
[CV] END ...C=0.1, coef0=1, degree=2, kernel=poly; total time= 0.4s
[CV] END ...C=0.1, coef0=1, degree=2, kernel=poly; total time= 0.5s
[CV] END ...C=0.1, coef0=1, degree=2, kernel=poly; total time= 0.5s
[CV] END ...C=0.1, coef0=1, degree=3, kernel=poly; total time= 0.6s
[CV] END ...C=0.1, coef0=1, degree=3, kernel=poly; total time= 0.8s
[CV] END ...C=0.1, coef0=1, degree=3, kernel=poly; total time= 0.6s
[CV] END ...C=0.1, coef0=1, degree=3, kernel=poly; total time= 0.7s
[CV] END ...C=0.1, coef0=1, degree=3, kernel=poly; total time= 0.6s
[CV] END ...C=1, coef0=0, degree=2, kernel=poly; total time= 0.4s
[CV] END ...C=1, coef0=0, degree=2, kernel=poly; total time= 0.4s
[CV] END ...C=1, coef0=0, degree=2, kernel=poly; total time= 0.4s
[CV] END ...C=1, coef0=0, degree=2, kernel=poly; total time= 0.4s
[CV] END ...C=1, coef0=0, degree=2, kernel=poly; total time= 0.4s
[CV] END ...C=1, coef0=0, degree=3, kernel=poly; total time= 0.8s
[CV] END ...C=1, coef0=0, degree=3, kernel=poly; total time= 0.9s

```

[illegible]

```

[CV] END ...C=0.1, gamma=100, kernel=rbf; total time= 2.8s
[CV] END ...C=0.1, gamma=100, kernel=rbf; total time= 2.8s
[CV] END ...C=0.1, gamma=100, kernel=rbf; total time= 2.8s
[CV] END ...C=0.1, gamma=100, kernel=rbf; total time= 2.8s
[CV] END ...C=0.1, gamma=100, kernel=rbf; total time= 2.7s
[CV] END ...C=1, gamma=0.1, kernel=rbf; total time= 1.2s
[CV] END ...C=1, gamma=0.1, kernel=rbf; total time= 1.2s
[CV] END ...C=1, gamma=0.1, kernel=rbf; total time= 1.2s
[CV] END ...C=1, gamma=0.1, kernel=rbf; total time= 1.2s
[CV] END ...C=1, gamma=0.1, kernel=rbf; total time= 1.2s
[CV] END ...C=1, gamma=1, kernel=rbf; total time= 1.3s
[CV] END ...C=1, gamma=1, kernel=rbf; total time= 1.3s
[CV] END ...C=1, gamma=1, kernel=rbf; total time= 1.3s
[CV] END ...C=1, gamma=1, kernel=rbf; total time= 1.3s
[CV] END ...C=1, gamma=1, kernel=rbf; total time= 1.3s
[CV] END ...C=1, gamma=10, kernel=rbf; total time= 1.8s
[CV] END ...C=1, gamma=10, kernel=rbf; total time= 1.8s
[CV] END ...C=1, gamma=10, kernel=rbf; total time= 1.8s
[CV] END ...C=1, gamma=10, kernel=rbf; total time= 1.8s
[CV] END ...C=1, gamma=10, kernel=rbf; total time= 1.8s
[CV] END ...C=1, gamma=100, kernel=rbf; total time= 2.9s
[CV] END ...C=1, gamma=100, kernel=rbf; total time= 2.9s
[CV] END ...C=1, gamma=100, kernel=rbf; total time= 2.9s
[CV] END ...C=1, gamma=100, kernel=rbf; total time= 2.9s
[CV] END ...C=1, gamma=100, kernel=rbf; total time= 2.9s
[CV] END ...C=10, gamma=0.1, kernel=rbf; total time= 1.5s
[CV] END ...C=10, gamma=0.1, kernel=rbf; total time= 1.6s
[CV] END ...C=10, gamma=0.1, kernel=rbf; total time= 1.6s
[CV] END ...C=10, gamma=0.1, kernel=rbf; total time= 1.5s
[CV] END ...C=10, gamma=0.1, kernel=rbf; total time= 1.6s
[CV] END ...C=10, gamma=1, kernel=rbf; total time= 1.5s
[CV] END ...C=10, gamma=1, kernel=rbf; total time= 1.4s
[CV] END ...C=10, gamma=1, kernel=rbf; total time= 1.5s
[CV] END ...C=10, gamma=1, kernel=rbf; total time= 1.5s
[CV] END ...C=10, gamma=1, kernel=rbf; total time= 1.5s
[CV] END ...C=10, gamma=10, kernel=rbf; total time= 1.9s
[CV] END ...C=10, gamma=10, kernel=rbf; total time= 1.9s
[CV] END ...C=10, gamma=10, kernel=rbf; total time= 1.9s
[CV] END ...C=10, gamma=10, kernel=rbf; total time= 1.9s
[CV] END ...C=10, gamma=10, kernel=rbf; total time= 1.9s
[CV] END ...C=10, gamma=100, kernel=rbf; total time= 3.1s
[CV] END ...C=10, gamma=100, kernel=rbf; total time= 3.1s
[CV] END ...C=10, gamma=100, kernel=rbf; total time= 3.1s
[CV] END ...C=10, gamma=100, kernel=rbf; total time= 3.1s
[CV] END ...C=10, gamma=100, kernel=rbf; total time= 3.1s

```

C:\Users\Usuário\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.11_qbz5n2kfra8p0\LocalCache\local-packages\Python311\site-

```
packages\numpy\ma\core.py:2820: RuntimeWarning: invalid value encountered in  
cast
```

```
_data = np.array(data, dtype=dtype, copy=copy,
```

```
[35]: GridSearchCV(cv=5, estimator=SVC(),  
                param_grid=[{'C': [0.1, 1, 10], 'kernel': ['linear']},  
                            {'C': [0.1, 1, 10], 'coef0': [0, 1], 'degree': [2, 3],  
                             'kernel': ['poly']},  
                            {'C': [0.1, 1, 10], 'gamma': [0.1, 1, 10, 100],  
                             'kernel': ['rbf']}],  
                scoring='accuracy', verbose=2)
```

```
[36]: print("Melhores parâmetros para SVM:", svm_grid.best_params_)
```

```
Melhores parâmetros para SVM: {'C': 1, 'coef0': 0, 'degree': 3, 'kernel':  
'poly'}
```

```
[37]: svm_best = svm_grid.best_estimator_
```

```
[38]: avalia_clf(svm_best, y_test, X_test, base='teste')
```

```
Modelo não possui predict_proba()
```

```
SVC - BASE DE TESTE:
```

```
=====
```

```
Acurácia: 73.1%
```

```
Acurácia Balanceada: 56.5%
```

```
Precision: 62.7%
```

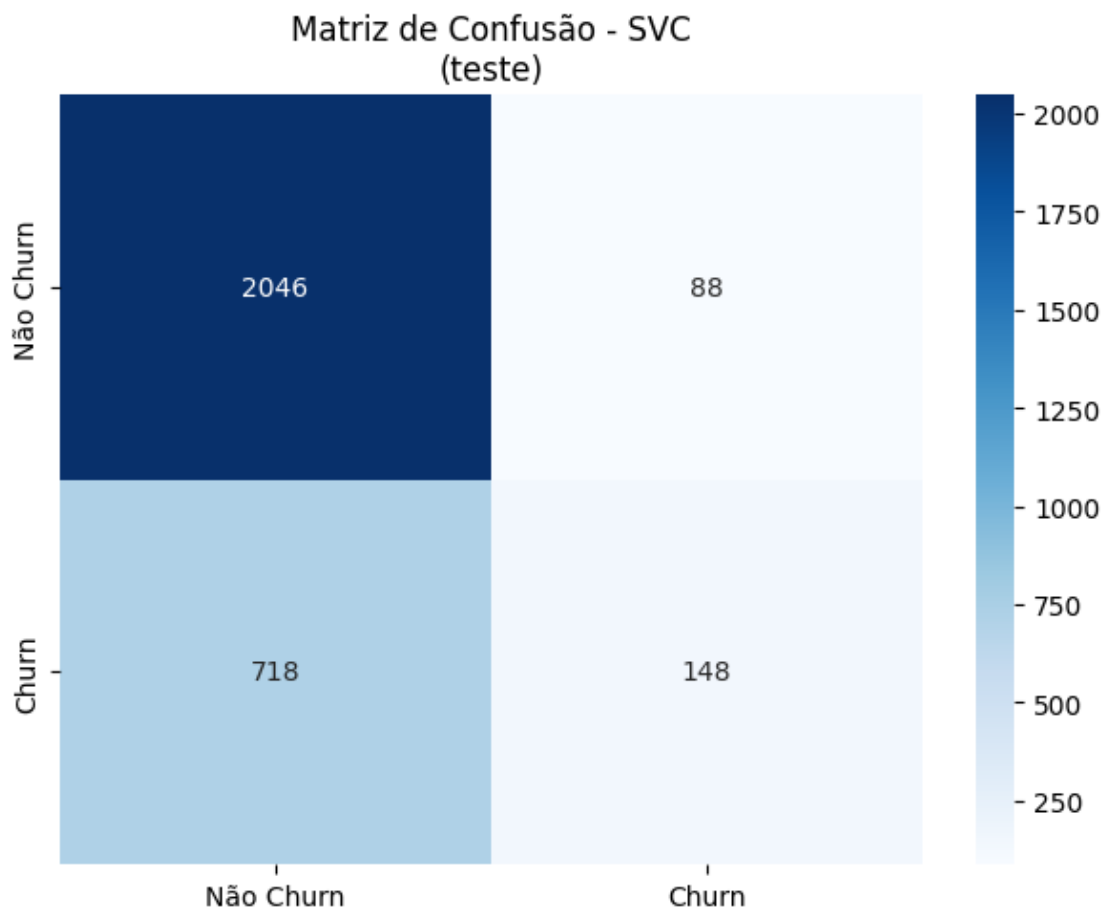
```
Recall: 17.1%
```

```
F1-Score: 26.9%
```

```
RELATÓRIO DE CLASSIFICAÇÃO:
```

```
=====
```

	precision	recall	f1-score	support
Não Churn	0.74	0.96	0.84	2134
Churn	0.63	0.17	0.27	866
accuracy			0.73	3000
macro avg	0.68	0.56	0.55	3000
weighted avg	0.71	0.73	0.67	3000



```
[38]: {'modelo': 'SVC',
      'base': 'teste',
      'acuracia': 0.7313333333333333,
      'acuracia_balanceada': 0.5648317897192924,
      'precision': 0.6271186440677966,
      'recall': 0.17090069284064666,
      'f1': 0.26860254083484575,
      'auc': None,
      'gini': None}
```

1.7.2 Árvores

```
[39]: melhores_modelos = {}
```

```
[40]: ## Arvore
      ## 1) Arvore de Decisao
```

```

param_grid_arvore = {
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
}

grid_arvore = GridSearchCV(estimator=arvore,
                           param_grid=param_grid_arvore,
                           scoring='accuracy',
                           cv=5,
                           n_jobs=-1,
                           verbose=1)

grid_arvore.fit(X_train, y_train)

## 2) XGB

param_grid = {
    'n_estimators': [50, 100, 150],
    'max_depth': [2, 3],
    'gamma': [0],
    'learning_rate': [0.1, 0.4],
    'colsample_bytree': [0.6, 0.8],
    'min_child_weight': [1],
    'subsample': [0.75, 1]
}

grid_xgb = GridSearchCV(estimator=xgb,
                        param_grid=param_grid,
                        scoring='roc_auc',
                        cv=5,
                        n_jobs=-1)
grid_xgb.fit(X_train, y_train)

## 3) Random Forest

param_grid_rf = {
    'n_estimators': [50, 100, 200],
    'max_depth': [5, 10, 20, None],
    'min_samples_split': [2, 5, 10],
    'class_weight': [None, 'balanced', {0: 1, 1: 3}],
}

```

```

grid_rf = GridSearchCV(estimator=random_forest,
                        param_grid=param_grid_rf,
                        scoring='accuracy',
                        cv=5,
                        n_jobs=-1,
                        verbose=1)

grid_rf.fit(X_train, y_train)

```

Fitting 5 folds for each of 36 candidates, totalling 180 fits

```

C:\Users\Usuário\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.11_qbz
5n2kfra8p0\LocalCache\local-packages\Python311\site-
packages\xgboost\training.py:183: UserWarning: [11:41:07] WARNING: C:\actions-
runner\_work\xgboost\xgboost\src\learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

```

```
bst.update(dtrain, iteration=i, fobj=obj)
```

Fitting 5 folds for each of 108 candidates, totalling 540 fits

```

[40]: GridSearchCV(cv=5, estimator=RandomForestClassifier(random_state=42), n_jobs=-1,
                  param_grid={'class_weight': [None, 'balanced', {0: 1, 1: 3}],
                              'max_depth': [5, 10, 20, None],
                              'min_samples_split': [2, 5, 10],
                              'n_estimators': [50, 100, 200]},
                  scoring='accuracy', verbose=1)

```

```

[41]: best_arvore = grid_arvore.best_estimator_
      best_xgb = grid_xgb.best_estimator_
      best_rf = grid_rf.best_estimator_

```

```

[42]: print("Melhores parâmetros para Árvore de Decisão:", grid_arvore.best_params_)
      print("Melhores parâmetros para Random Forest:", grid_rf.best_params_)
      print("Melhores parâmetros para XGBoost:", grid_xgb.best_params_)

```

```

Melhores parâmetros para Árvore de Decisão: {'max_depth': 10,
'min_samples_leaf': 2, 'min_samples_split': 10}

```

```

Melhores parâmetros para Random Forest: {'class_weight': None, 'max_depth': 10,
'min_samples_split': 5, 'n_estimators': 100}

```

```

Melhores parâmetros para XGBoost: {'colsample_bytree': 0.6, 'gamma': 0,
'learning_rate': 0.1, 'max_depth': 2, 'min_child_weight': 1, 'n_estimators': 50,
'subsample': 0.75}

```

```

[49]: avalia_clf(best_arvore, y_test, X_test, base='teste')
      avalia_clf(best_rf, y_test, X_test, base='teste')
      avalia_clf(best_xgb, y_test, X_test, base='teste')

```

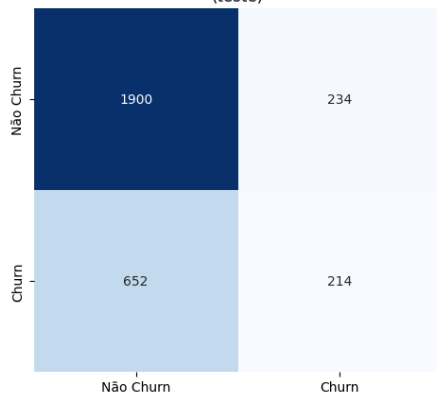
DECISIONTREECLASSIFIER - BASE DE TESTE:

Acurácia: 70.5%
 Acurácia Balanceada: 56.9%
 Precision: 47.8%
 Recall: 24.7%
 F1-Score: 32.6%
 AUC-ROC: 65.01%
 GINI: 30.02%

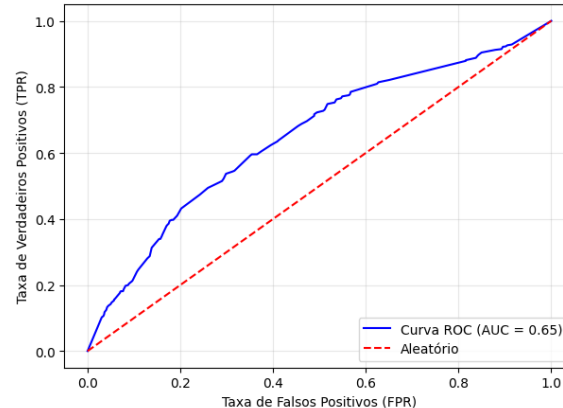
RELATÓRIO DE CLASSIFICAÇÃO:

	precision	recall	f1-score	support
Não Churn	0.74	0.89	0.81	2134
Churn	0.48	0.25	0.33	866
accuracy			0.70	3000
macro avg	0.61	0.57	0.57	3000
weighted avg	0.67	0.70	0.67	3000

Matriz de Confusão - DecisionTreeClassifier (teste)



Curva ROC - DecisionTreeClassifier (teste)



RANDOMFORESTCLASSIFIER - BASE DE TESTE:

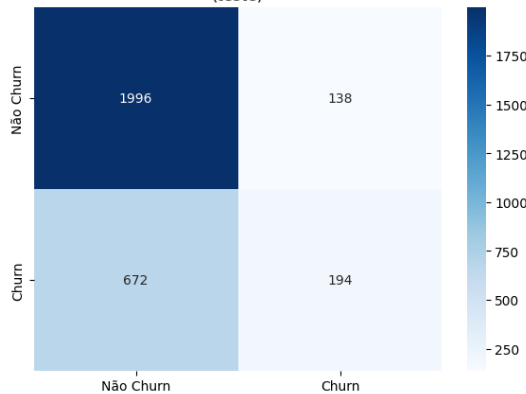
Acurácia: 73.0%
 Acurácia Balanceada: 58.0%
 Precision: 58.4%
 Recall: 22.4%
 F1-Score: 32.4%
 AUC-ROC: 70.48%

GINI: 40.96%

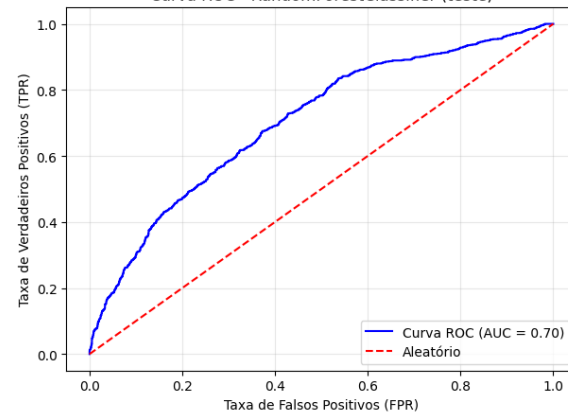
RELATÓRIO DE CLASSIFICAÇÃO:

	precision	recall	f1-score	support
Não Churn	0.75	0.94	0.83	2134
Churn	0.58	0.22	0.32	866
accuracy			0.73	3000
macro avg	0.67	0.58	0.58	3000
weighted avg	0.70	0.73	0.68	3000

Matriz de Confusão - RandomForestClassifier (teste)



Curva ROC - RandomForestClassifier (teste)



XGBCLASSIFIER - BASE DE TESTE:

Acurácia: 72.7%

Acurácia Balanceada: 55.8%

Precision: 60.4%

Recall: 15.7%

F1-Score: 24.9%

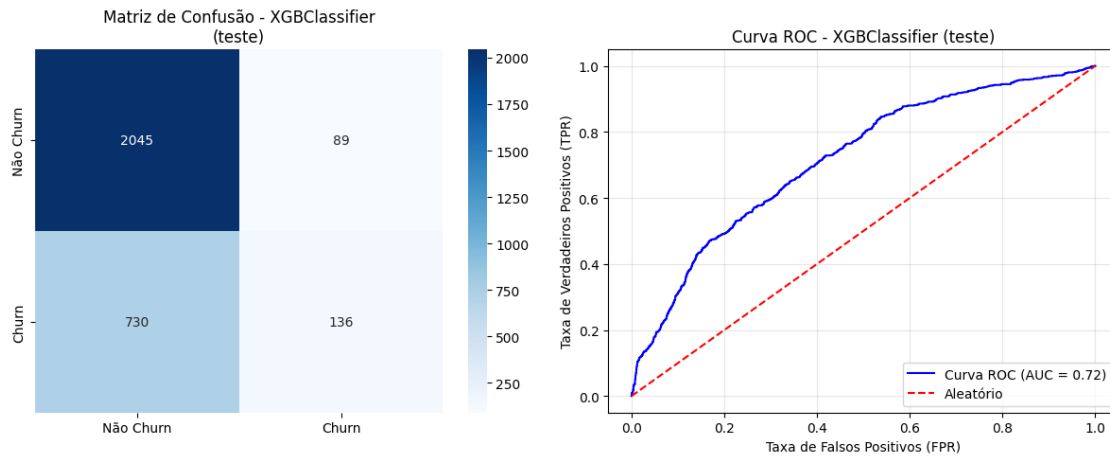
AUC-ROC: 71.64%

GINI: 43.28%

RELATÓRIO DE CLASSIFICAÇÃO:

	precision	recall	f1-score	support
Não Churn	0.74	0.96	0.83	2134
Churn	0.60	0.16	0.25	866

accuracy			0.73	3000
macro avg	0.67	0.56	0.54	3000
weighted avg	0.70	0.73	0.66	3000



```
[49]: {'modelo': 'XGBClassifier',
      'base': 'teste',
      'acuracia': 0.727,
      'acuracia_balanceada': 0.5576690814720862,
      'precision': 0.6044444444444445,
      'recall': 0.15704387990762125,
      'f1': 0.24931255728689275,
      'auc': 0.7164085378919549,
      'gini': 0.4328170757839098}
```

1.8 Comparando Os Resultados

```
[44]: import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import roc_curve, auc, accuracy_score
import numpy as np

def comparar_modelos_graficos(modelos_dict, X_test, y_test):
    """
    Cria gráficos comparativos para todos os modelos
    """
    fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2, figsize=(20, 16))
    fig.suptitle('COMPARAÇÃO DE MODELOS - ANÁLISE COMPLETA', fontsize=16,
fontweight='bold')

    # Dados para os gráficos
```

```

resultados = {}
cores = plt.cm.Set3(np.linspace(0, 1, len(modelos_dict)))

# 1. CURVA ROC
for i, (nome, modelo) in enumerate(modelos_dict.items()):
    try:
        if hasattr(modelo, 'predict_proba'):
            y_proba = modelo.predict_proba(X_test)[: , 1]
        else:
            y_proba = modelo.decision_function(X_test)

        fpr, tpr, _ = roc_curve(y_test, y_proba)
        roc_auc = auc(fpr, tpr)

        ax1.plot(fpr, tpr, color=cores[i], lw=2,
                 label=f'{nome} (AUC = {roc_auc:.3f})')

        # Coletar métricas
        y_pred = modelo.predict(X_test)
        resultados[nome] = {
            'acuracia': accuracy_score(y_test, y_pred),
            'auc': roc_auc,
            'fpr': fpr,
            'tpr': tpr
        }

    except Exception as e:
        print(f" Erro no modelo {nome}: {e}")
        continue

# Configurar gráfico ROC
ax1.plot([0, 1], [0, 1], 'k--', lw=2)
ax1.set_xlim([0.0, 1.0])
ax1.set_ylim([0.0, 1.05])
ax1.set_xlabel('False Positive Rate')
ax1.set_ylabel('True Positive Rate')
ax1.set_title('CURVAS ROC - COMPARAÇÃO', fontweight='bold')
ax1.legend(loc="lower right")
ax1.grid(alpha=0.3)

# 2. ACURÁCIAS
nomes = list(resultados.keys())
acuracias = [resultados[nome]['acuracia'] for nome in nomes]

bars = ax2.barh(nomes, acuracias, color=cores[:len(nomes)])
ax2.set_xlim(0, 1)
ax2.set_title('ACURÁCIA POR MODELO', fontweight='bold')

```

```

ax2.set_xlabel('Acurácia')
ax2.grid(axis='x', alpha=0.3)

# Adicionar valores nas barras
for bar, acc in zip(bars, acuracias):
    ax2.text(bar.get_width() + 0.01, bar.get_y() + bar.get_height()/2,
             f'{acc:.3f}', va='center', fontweight='bold')

# 3. AUC-ROC
auc_values = [resultados[nome]['auc'] for nome in nomes]

bars = ax3.barh(nomes, auc_values, color=cores[:len(nomes)])
ax3.set_xlim(0, 1)
ax3.set_title('AUC-ROC POR MODELO', fontweight='bold')
ax3.set_xlabel('AUC Score')
ax3.grid(axis='x', alpha=0.3)

for bar, auc_val in zip(bars, auc_values):
    ax3.text(bar.get_width() + 0.01, bar.get_y() + bar.get_height()/2,
             f'{auc_val:.3f}', va='center', fontweight='bold')

# 4. COMPARAÇÃO ACURÁCIA vs AUC
ax4.scatter(acuracias, auc_values, s=100, color=cores[:len(nomes)], alpha=0.
↪7)

# Adicionar labels
for i, nome in enumerate(nomes):
    ax4.annotate(nome, (acuracias[i], auc_values[i]),
                 xytext=(5, 5), textcoords='offset points', fontsize=9)

ax4.set_xlabel('Acurácia')
ax4.set_ylabel('AUC-ROC')
ax4.set_title('ACURÁCIA vs AUC-ROC', fontweight='bold')
ax4.grid(alpha=0.3)
ax4.set_xlim(0, 1)
ax4.set_ylim(0, 1)

plt.tight_layout()
plt.show()

return resultados

# Dicionário com todos os modelos
modelos_dict = {
    'Árvore': arvore,
    'XGBoost': xgb,
    'Random Forest': random_forest,

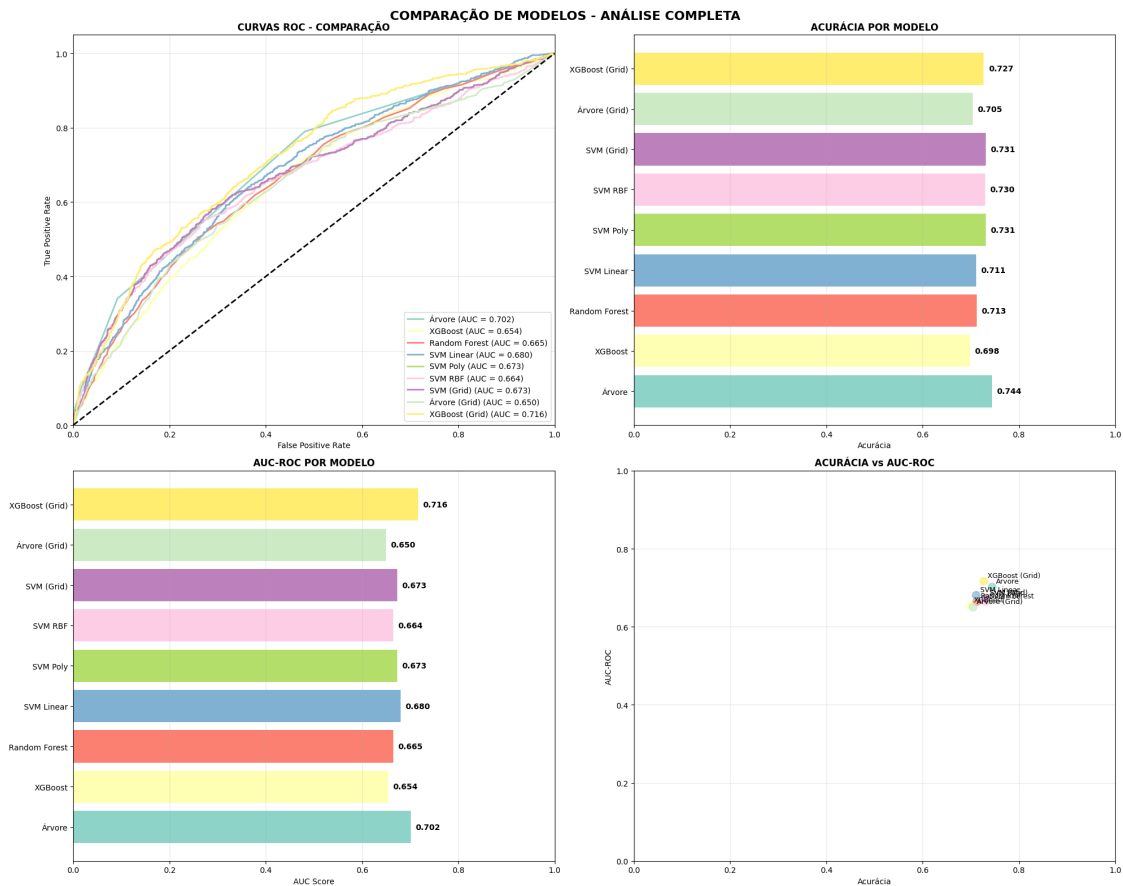
```

```

'SVM Linear': svm_linear,
'SVM Poly': svm_pol,
'SVM RBF': svm_rbf,
'SVM (Grid)': svm_grid.best_estimator_ if hasattr(svm_grid, '
↳ 'best_estimator_') else svm_rbf,
'Árvore (Grid)': grid_arvore.best_estimator_ if hasattr(grid_arvore, '
↳ 'best_estimator_') else arvore,
'XGBoost (Grid)': grid_xgb.best_estimator_ if hasattr(grid_xgb, '
↳ 'best_estimator_') else xgb
}

# Gerar visualização
resultados = comparar_modelos_graficos(modelos_dict, X_test, y_test)

```



```

[45]: def plot_desempenho_detalhado(resultados):
      """
      Gráfico detalhado com múltiplas métricas
      """
      modelos = list(resultados.keys())

```

```

metricas = ['acuracia', 'auc']

fig, axes = plt.subplots(1, 2, figsize=(15, 8))
fig.suptitle('DESEMPENHO DETALHADO DOS MODELOS', fontsize=16,
fontweight='bold')

# Criar dados para o gráfico
dados_plot = []
for modelo, metrics in resultados.items():
    dados_plot.append({
        'Modelo': modelo,
        'Acurácia': metrics['acuracia'],
        'AUC-ROC': metrics['auc']
    })

df = pd.DataFrame(dados_plot)

# Gráfico de barras agrupadas
x = np.arange(len(modelos))
width = 0.35

axes[0].bar(x - width/2, df['Acurácia'], width, label='Acurácia', alpha=0.8)
axes[0].bar(x + width/2, df['AUC-ROC'], width, label='AUC-ROC', alpha=0.8)

axes[0].set_xlabel('Modelos')
axes[0].set_ylabel('Score')
axes[0].set_title('Acurácia vs AUC-ROC', fontweight='bold')
axes[0].set_xticks(x)
axes[0].set_xticklabels(modelos, rotation=45, ha='right')
axes[0].legend()
axes[0].grid(alpha=0.3)

# Adicionar valores
for i, (acc, auc_val) in enumerate(zip(df['Acurácia'], df['AUC-ROC'])):
    axes[0].text(i - width/2, acc + 0.01, f'{acc:.3f}', ha='center',
va='bottom', fontsize=8)
    axes[0].text(i + width/2, auc_val + 0.01, f'{auc_val:.3f}',
ha='center', va='bottom', fontsize=8)

# Gráfico de ranking
df['Score_Total'] = (df['Acurácia'] + df['AUC-ROC']) / 2
df = df.sort_values('Score_Total', ascending=True)

axes[1].barh(df['Modelo'], df['Score_Total'], color='lightgreen')
axes[1].set_xlabel('Score Médio (Acurácia + AUC) / 2')
axes[1].set_title('RANKING DE MODELOS', fontweight='bold')
axes[1].grid(axis='x', alpha=0.3)

```

```

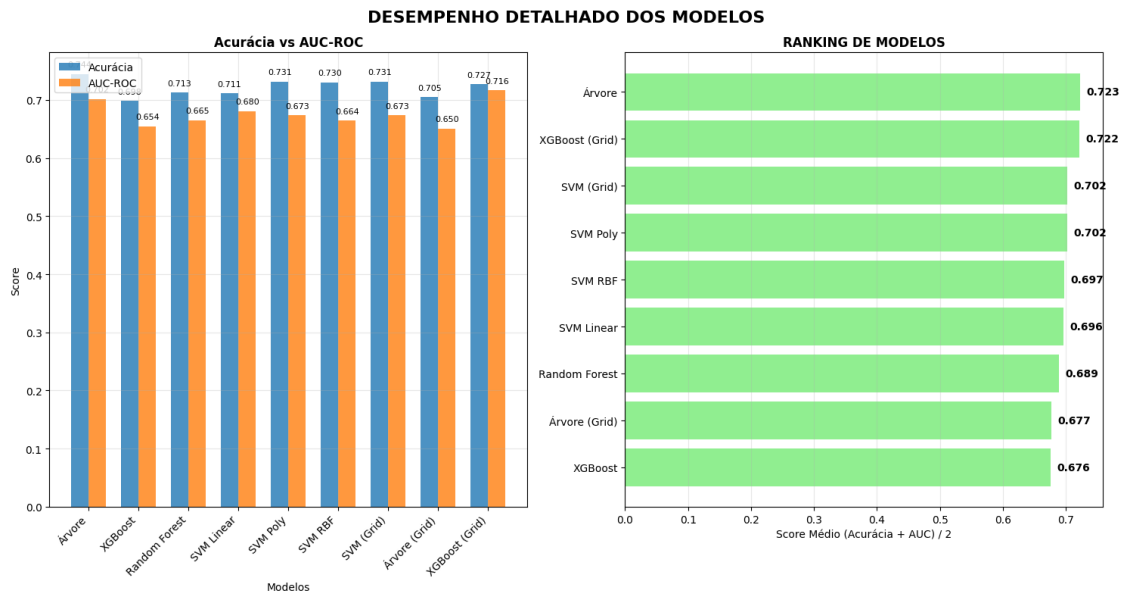
for i, (modelo, score) in enumerate(zip(df['Modelo'], df['Score_Total'])):
    axes[1].text(score + 0.01, i, f'{score:.3f}', va='center',
fontweight='bold')

plt.tight_layout()
plt.show()

return df

# Gerar gráfico detalhado
df_ranking = plot_desempenho_detalhado(resultados)
print(" RANKING FINAL:")
print(df_ranking[['Modelo', 'Acurácia', 'AUC-ROC', 'Score_Total']].round(3))

```



RANKING FINAL:

	Modelo	Acurácia	AUC-ROC	Score_Total
1	XGBoost	0.698	0.654	0.676
7	Árvore (Grid)	0.705	0.650	0.677
2	Random Forest	0.713	0.665	0.689
3	SVM Linear	0.711	0.680	0.696
5	SVM RBF	0.730	0.664	0.697
4	SVM Poly	0.731	0.673	0.702
6	SVM (Grid)	0.731	0.673	0.702
8	XGBoost (Grid)	0.727	0.716	0.722
0	Árvore	0.744	0.702	0.723

```
[46]: def plot_roc_individual(resultados):
    """
    Plot individual de cada curva ROC com zoom
    """
    n_modelos = len(resultados)
    n_cols = 3
    n_rows = (n_modelos + n_cols - 1) // n_cols

    fig, axes = plt.subplots(n_rows, n_cols, figsize=(15, 5*n_rows))
    axes = axes.flatten()

    for i, (nome, metrics) in enumerate(resultados.items()):
        ax = axes[i]

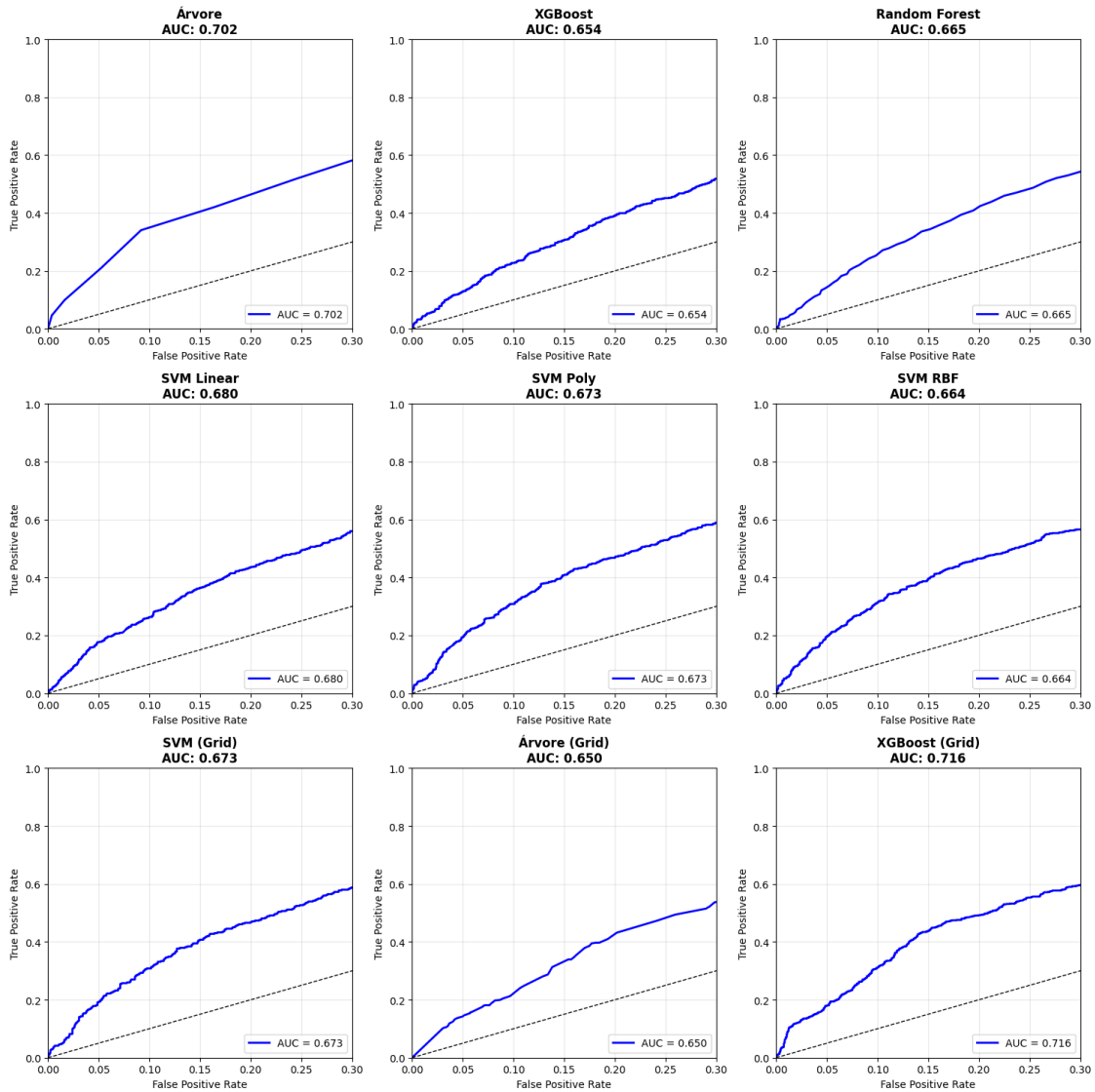
        # Curva ROC completa
        ax.plot(metrics['fpr'], metrics['tpr'], color='blue', lw=2,
                label=f'AUC = {metrics["auc"]:.3f}')
        ax.plot([0, 1], [0, 1], 'k--', lw=1)

        # Zoom na área importante (0.0 a 0.3 FPR)
        ax.set_xlim([0.0, 0.3])
        ax.set_ylim([0.0, 1.0])
        ax.set_title(f'{nome}\nAUC: {metrics["auc"]:.3f}', fontweight='bold')
        ax.set_xlabel('False Positive Rate')
        ax.set_ylabel('True Positive Rate')
        ax.legend(loc="lower right")
        ax.grid(alpha=0.3)

    # Ocultar eixos vazios
    for j in range(i + 1, len(axes)):
        axes[j].set_visible(False)

    plt.tight_layout()
    plt.show()

# Plot individual
plot_roc_individual(resultados)
```



```
[47]: def criar_tabela_resumo(resultados):
    """
    Cria tabela resumo com todas as métricas
    """
    dados = []
    for nome, metrics in resultados.items():
        dados.append({
            'Modelo': nome,
            'Acurácia': f'{metrics["acuracia"]:.3f}',
            'AUC-ROC': f'{metrics["auc"]:.3f}',
            'GINI': f'{(2*metrics["auc"]-1):.3f}',
            'Performance': 'Boa' if metrics["auc"] > 0.7 else 'Regular' if
↪ metrics["auc"] > 0.6 else 'Ruim'
```

```

    })

    df = pd.DataFrame(dados)
    df = df.sort_values('AUC-ROC', ascending=False)

    print(" TABELA RESUMO - DESEMPENHO DOS MODELOS")
    print("=" * 80)
    print(df.to_string(index=False))
    print("=" * 80)

    return df

# Gerar tabela resumo
tabela_final = criar_tabela_resumo(resultados)

```

```

TABELA RESUMO - DESEMPENHO DOS MODELOS
=====
      Modelo Acurácia AUC-ROC  GINI Performance
XGBoost (Grid)    0.727    0.716 0.433         Boa
      Árvore      0.744    0.702 0.403         Boa
      SVM Linear   0.711    0.680 0.361     Regular
      SVM Poly     0.731    0.673 0.346     Regular
      SVM (Grid)   0.731    0.673 0.346     Regular
Random Forest     0.713    0.665 0.329     Regular
      SVM RBF      0.730    0.664 0.329     Regular
      XGBoost      0.698    0.654 0.308     Regular
      Árvore (Grid) 0.705    0.650 0.300     Regular
=====

```

1.9 Explicabilidade dos Modelos

1.9.1 SVM

[]:

1.9.2 Arvores

```

[ ]: resultado = permutation_importance(best_rf, X_test, y_test, n_repeats=30,
    ↪ random_state=42)
importancia = resultado.importances_mean.argsort()

```