```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.metrics import (
    confusion_matrix, classification_report,
    roc_curve, auc, precision_recall_curve,
    f1_score, accuracy_score, roc_auc_score
)
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.ensemble import (
    RandomForestClassifier,
    GradientBoostingClassifier,
    AdaBoostClassifier
)
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import cross_val_score, StratifiedKFold
from sklearn.pipeline import Pipeline
from sklearn.base import BaseEstimator, TransformerMixin
from scipy.stats import ttest_ind, chi2_contingency
import warnings
warnings.filterwarnings('ignore')
%matplotlib inline
%matplotlib inline
plt.rcParams['figure.figure.dpi'] = 100
plt.rcParams['figure.figure.dpi'] = 100
sns.set_style('whitegrid')
sns.set_style('whitegrid')
np.random.seed(42)
np.random.seed(42)
print("Bibliotecas importadas com sucesso!")
print("Bibliotecas importadas com sucesso!")
```

```
Bibliotecas importadas com sucesso!
Bibliotecas importadas com sucesso!
```

```
# ═══ 1. Carregamento do dataset ═══
# Substituir pelo caminho do arquivo
df = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/Classificação/Classificação_Binária/Classificação_Binária_Aula_01/Classificação_Binária_Aula_01_Dataset.csv')
df = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/Classificação/Classificação_Binária/Classificação_Binária_Aula_01/Classificação_Binária_Aula_01_Dataset.csv')

# Criar alvo binário
df['alvo'] = (df['churn'] == 1).astype(int)
df['alvo'] = (df['churn'] == 1).astype(int)

# ═══ 2. Exploração inicial ═══
print("═══ 1. CARREGAMENTO ═══")
print("═══ 1. CARREGAMENTO ═══")
print(f"Dimensões: {df.shape[0]:,} linhas × {df.shape[1]} colunas")
print(f"Dimensões: {df.shape[0]:,} linhas × {df.shape[1]} colunas")
```

```
═══ 1. CARREGAMENTO ═══
Dimensões: 7,043 linhas × 22 colunas
═══ 1. CARREGAMENTO ═══
Dimensões: 7,043 linhas × 22 colunas
```

```
customerID  gender  SeniorCitizen  Partner  Dependents  tenure  PhoneService  MultipleLines  InternetService  OnlineSecurity  OnlineBackup  DeviceProtection  TechSupport  StreamingTV  StreamingMovies  Contract  PaperlessBilling  PaymentMethod  MonthlyCharges  TotalCharges  Churn  alvo
0  7590-VHVEG  Female  0  Yes  No  1  No  No phone service  DSL  No  Yes  No  No  No  No  Month-to-month  Yes  Electronic check  29.85  29.85  No  0
1  5575-GNVDE    Male  0  No  No  34  Yes  No  DSL  Yes  No  Yes  No  No  One year  No  Mailed check  56.95  1734.65  No  0
2  3668-QPYBK    Male  0  No  No  2  Yes  No  DSL  Yes  Yes  No  No  No  Month-to-month  Yes  Mailed check  53.85  108.15  Yes  1
3  7795-CFOCW    Male  0  No  No  45  No  No phone service  DSL  Yes  No  Yes  Yes  No  One year  No  Bank transfer (automatic)  42.30  1840.75  No  0
4  9237-HQITU  Female  0  No  No  2  Yes  No  Fiber optic  No  No  No  No  No  Month-to-month  Yes  Electronic check  70.70  151.65  Yes  1
```

```
# ═══ 3. Tipos e colunas ═══
print("═══ 2. TIPOS E COLUNAS ═══")
print("═══ 2. TIPOS E COLUNAS ═══")
df.info()
df.info()
```

```
═══ 2. TIPOS E COLUNAS ═══
═══ 2. TIPOS E COLUNAS ═══
<class pandas.core.frame.DataFrame>
RangeIndex: 7,043 entries, 0 to 7,042
Data columns (total 23 entries):
 #   Column            Non-Non-Count  Dtype
---  ------            --------------  -----
 0   customerID        7,043 non-Non-  object
 1   gender            7,043 non-Non-  object
 2   SeniorCitizen     7,043 non-Non-  int64
 3   Partner           7,043 non-Non-  object
 4   Dependents        7,043 non-Non-  object
 5   tenure            7,043 non-Non-  int64
 6   PhoneService      7,043 non-Non-  object
 7   MultipleLines     7,043 non-Non-  object
 8   InternetService   7,043 non-Non-  object
 9   OnlineSecurity    7,043 non-Non-  object
 10  OnlineBackup      7,043 non-Non-  object
 11  DeviceProtection  7,043 non-Non-  object
 12  TechSupport       7,043 non-Non-  object
 13  StreamingTV       7,043 non-Non-  object
 14  StreamingMovies   7,043 non-Non-  object
 15  Contract          7,043 non-Non-  object
 16  PaperlessBilling  7,043 non-Non-  object
 17  PaymentMethod     7,043 non-Non-  object
 18  MonthlyCharges    7,043 non-Non-  float64
 19  TotalCharges      7,043 non-Non-  object
 20  Churn             7,043 non-Non-  object
 21  alvo              7,043 non-Non-  int64
dtypes: float64(1), int64(4), object(17)
memory usage: 1,265.6 KB
```

Se ainda observar que no tipo dados subiu o tamanho.

```
# Corrigindo TotalCharges (espaços vazios em alguns casos)
df['TotalCharges'] = pd.to_numeric(df['TotalCharges'], errors='coerce')
df['TotalCharges'] = df['TotalCharges'].fillna(df['TotalCharges'].median())

# Remover coluna de ID
df['gender'] = df['gender'].astype(int)
df['Partner'] = df['Partner'].astype(int)
df['Dependents'] = df['Dependents'].astype(int)
df['PhoneService'] = df['PhoneService'].astype(int)
df['PaperlessBilling'] = df['PaperlessBilling'].astype(int)
df['Churn'] = df['Churn'].astype(int)
df['alvo'] = df['alvo'].astype(int)
```

Se ainda observar que no tipo dados subiu o tamanho.

```
# ═══ 4. Distribuição do alvo ═══
print("═══ 3. DISTRIBUIÇÃO DO ALVO ═══")
print("═══ 3. DISTRIBUIÇÃO DO ALVO ═══")
alvo_dist = df['alvo'].value_counts()
alvo_dist = df['alvo'].value_counts()
for i, v in alvo_dist.items:
    print(f"  {i}: {v:,} ({v/len(df)*100:.1f%})")
    print(f"  {i}: {v:,} ({v/len(df)*100:.1f%})")
```

```
═══ 3. DISTRIBUIÇÃO DO ALVO ═══
═══ 3. DISTRIBUIÇÃO DO ALVO ═══
  0: 5,174 (73.4%)
  0: 5,174 (73.4%)
  1: 1,869 (26.5%)
  1: 1,869 (26.5%)
```

Se ainda observar que o tipo dados subiu o tamanho.

```
# ═══ 5. Colunas numéricas e categóricas ═══
print("═══ 4. COLUNAS NUMÉRICAS E CATEGÓRICAS ═══")
print("═══ 4. COLUNAS NUMÉRICAS E CATEGÓRICAS ═══")
num_cols = df.select_dtype(include=[np.number]).columns.tolist
num_cols = df.select_dtype(include=[np.number]).columns.tolist
cat_cols = df.select_dtype(include=['object']).columns.tolist
cat_cols = df.select_dtype(include=['object']).columns.tolist
print(f"Numéricas ({len(num_cols)}): {num_cols}")
print(f"Numéricas ({len(num_cols)}): {num_cols}")
print(f"Categóricas ({len(cat_cols)}): {cat_cols}")
print(f"Categóricas ({len(cat_cols)}): {cat_cols}")
```

```
═══ 4. COLUNAS NUMÉRICAS E CATEGÓRICAS ═══
═══ 4. COLUNAS NUMÉRICAS E CATEGÓRICAS ═══
Numéricas (6): ['gender', 'SeniorCitizen', 'Partner', 'Dependents', 'tenure', 'PhoneService', 'PaperlessBilling', 'MonthlyCharges', 'TotalCharges', 'Churn', 'alvo']
Numéricas (6): ['gender', 'SeniorCitizen', 'Partner', 'Dependents', 'tenure', 'PhoneService', 'PaperlessBilling', 'MonthlyCharges', 'TotalCharges', 'Churn', 'alvo']
Categóricas (16): ['customerID', 'MultipleLines', 'InternetService', 'OnlineSecurity', 'OnlineBackup', 'DeviceProtection', 'TechSupport', 'StreamingTV', 'StreamingMovies', 'Contract', 'PaymentMethod']
Categóricas (16): ['customerID', 'MultipleLines', 'InternetService', 'OnlineSecurity', 'OnlineBackup', 'DeviceProtection', 'TechSupport', 'StreamingTV', 'StreamingMovies', 'Contract', 'PaymentMethod']
```

Se ainda observar que o tipo dados subiu o tamanho.

```
# ═══ 6. Estatísticas descritivas ═══
print("═══ 5. ESTATÍSTICAS DESCRITIVAS ═══")
print("═══ 5. ESTATÍSTICAS DESCRITIVAS ═══")
display(df[num_cols].describe.round(2).T)
display(df[num_cols].describe.round(2).T)
```

	count	mean	std	min	25%	50%	75%	max
SeniorCitizen	7,043.00	0.16	0.37	0.00	0.00	0.00	0.00	1.00
tenure	7,043.00	32.37	24.56	0.00	9.00	29.00	55.00	72.00
MonthlyCharges	7,043.00	64.76	30.09	18.25	35.50	70.35	89.85	118.75
TotalCharges	7,043.00	2,279.73	2,266.79	18.80	402.30	1,397.47	3,786.60	8,684.80
Churn	7,043.00	0.27	0.44	0.00	0.00	0.00	1.00	1.00
alvo	7,043.00	0.27	0.44	0.00	0.00	0.00	1.00	1.00

Se ainda observar que o tipo dados subiu o tamanho.

```
# ═══ 7. Correlações com o alvo ═══
print("═══ 6. CORRELAÇÕES COM O ALVO ═══")
print("═══ 6. CORRELAÇÕES COM O ALVO ═══")
corr = df[num_cols].corr.round(2)
corr = df[num_cols].corr.round(2)
corr_alvo = corr['alvo'].sort_values(ascending=False)
corr_alvo = corr['alvo'].sort_values(ascending=False)
for c, v in corr_alvo.items:
    if c != 'alvo':
        print(f"  {c:{max(len(c) for c in num_cols)}}: {v:+.2}")
        print(f"  {c:{max(len(c) for c in num_cols)}}: {v:+.2}")
```

```
═══ 6. CORRELAÇÕES COM O ALVO ═══
═══ 6. CORRELAÇÕES COM O ALVO ═══
  Churn            : +0.35
  Churn            : +0.35
  SeniorCitizen    : +0.15
  SeniorCitizen    : +0.15
  MonthlyCharges   : +0.19
  MonthlyCharges   : +0.19
  PaperlessBilling : +0.19
  PaperlessBilling : +0.19
  Dependents       : -0.16
  Dependents       : -0.16
  Partner          : -0.15
  Partner          : -0.15
  gender           : -0.01
  gender           : -0.01
  PhoneService     : -0.01
  PhoneService     : -0.01
  TotalCharges     : -0.20
  TotalCharges     : -0.20
  tenure           : -0.35
  tenure           : -0.35
```

Se ainda observar que o tipo dados subiu o tamanho.

```
# ═══ 8. Gráfico de correlações com o alvo ═══
fig, ax = plt.subplots(1, 1, figure(8, 6))
fig, ax = plt.subplots(1, 1, figure(8, 6))
colors = ['#ff6b6b' if v > 0 else '#4ecdc4' for v in corr_alvo.values[1:]]
colors = ['#ff6b6b' if v > 0 else '#4ecdc4' for v in corr_alvo.values[1:]]
bars = ax.barh(corr_alvo.index[1:], corr_alvo.values[1:], color=colors, edgecolor='white')
bars = ax.barh(corr_alvo.index[1:], corr_alvo.values[1:], color=colors, edgecolor='white')
ax.axvline(0, color='black', linewidth=0.8)
ax.axvline(0, color='black', linewidth=0.8)
ax.set_title('Correlações com o alvo (Churn)', fontsize=14, fontweight='bold')
ax.set_title('Correlações com o alvo (Churn)', fontsize=14, fontweight='bold')
ax.set_xlabel('Correlação', fontsize=12)
ax.set_xlabel('Correlação', fontsize=12)
plt.tight_layout
plt.tight_layout
plt.show
plt.show
```

Se ainda observar que o tipo dados subiu o tamanho.

```
# ═══ 9. Preparação de conversão de tipos de dados ═══
# Converter binárias para numérico
# Target encoding para categóricas
# Criar dummies para categóricas com poucas categorias
df['gender'] = df['gender'].astype(int)
df['Partner'] = df['Partner'].astype(int)
df['Dependents'] = df['Dependents'].astype(int)
df['PhoneService'] = df['PhoneService'].astype(int)
df['PaperlessBilling'] = df['PaperlessBilling'].astype(int)
df['Churn'] = df['Churn'].astype(int)
df['alvo'] = df['alvo'].astype(int)
```

7. Preparação de conversão de tipos de dados

- Converter binárias para numérico
- Target encoding para categóricas
- Criar dummies para categóricas com poucas categorias

Se ainda observar que o tipo dados subiu o tamanho.

```
# ═══ 10. Target encoding para categóricas ═══
# Calcular taxa de churn por categoria
cat_cols = df.select_dtype(include=['object']).columns.tolist
cat_cols = df.select_dtype(include=['object']).columns.tolist
for c in cat_cols:
    if c != 'customerID':
        mapa = df.groupby(c)['alvo'].mean.round(3).to_dict
        df[f'{c}_target_enc'] = df[c].map(mapa)
        df[f'{c}_target_enc'] = df[c].map(mapa)
        print(f"  {c:{max(len(c) for c in cat_cols)}}: {len(mapa)} categorias")
        print(f"  {c:{max(len(c) for c in cat_cols)}}: {len(mapa)} categorias")
```

```
═══ 10. Target encoding para categóricas ═══
═══ 10. Target encoding para categóricas ═══
  MultipleLines    : 3 categorias
  MultipleLines    : 3 categorias
  InternetService  : 3 categorias
  InternetService  : 3 categorias
  OnlineSecurity   : 3 categorias
  OnlineSecurity   : 3 categorias
  OnlineBackup     : 3 categorias
  OnlineBackup     : 3 categorias
  DeviceProtection : 3 categorias
  DeviceProtection : 3 categorias
  TechSupport      : 3 categorias
  TechSupport      : 3 categorias
  StreamingTV      : 3 categorias
  StreamingTV      : 3 categorias
  StreamingMovies  : 3 categorias
  StreamingMovies  : 3 categorias
  Contract         : 3 categorias
  Contract         : 3 categorias
  PaymentMethod    : 4 categorias
  PaymentMethod    : 4 categorias
```

Se ainda observar que o tipo dados subiu o tamanho.

```
# ═══ 11. Dummies para categóricas com poucas categorias ═══
# Criar dummies para colunas com 2-4 categorias
dummy_cols = ['MultipleLines', 'InternetService', 'OnlineSecurity', 'OnlineBackup', 'DeviceProtection', 'TechSupport', 'StreamingTV', 'StreamingMovies', 'Contract', 'PaymentMethod']
dummy_cols = ['MultipleLines', 'InternetService', 'OnlineSecurity', 'OnlineBackup', 'DeviceProtection', 'TechSupport', 'StreamingTV', 'StreamingMovies', 'Contract', 'PaymentMethod']
df = pd.get_dummies(df, columns=dummy_cols, drop_first=True)
df = pd.get_dummies(df, columns=dummy_cols, drop_first=True)
print(f"Após dummies: {df.shape[1]} colunas")
print(f"Após dummies: {df.shape[1]} colunas")
```

```
Após dummies: 38 colunas
Após dummies: 38 colunas
```

Se ainda observar que o tipo dados subiu o tamanho.

```
# ═══ 12. Preparação final ═══
# Remover colunas não-numéricas
drop_cols = ['customerID', 'Churn', 'alvo']
drop_cols = ['customerID', 'Churn', 'alvo']
X = df[[c for c in df.columns if c not in drop_cols]]
X = df[[c for c in df.columns if c not in drop_cols]]
y = df['alvo']
y = df['alvo']
print(f"X: {X.shape[0]:,} × {X.shape[1]}")
print(f"X: {X.shape[0]:,} × {X.shape[1]}")
print(f"y: {y.shape[0]:,}")
print(f"y: {y.shape[0]:,}")
```

```
X: 7,043 × 35
X: 7,043 × 35
y: 7,043
y: 7,043
```

Se ainda observar que o tipo dados subiu o tamanho.

```
# ═══ 13. Divisão treino/teste ═══
X_train, X_test, y_train, y_test = train_test_split(X, y, test=0.20, random_state=42, stratify=y)
X_train, X_test, y_train, y_test = train_test_split(X, y, test=0.20, random_state=42, stratify=y)
print(f"Treino: {X_train.shape[0]:,}")
print(f"Treino: {X_train.shape[0]:,}")
print(f"Teste: {X_test.shape[0]:,}")
print(f"Teste: {X_test.shape[0]:,}")
print(f"Churn treino: {y_train.mean.round(3)}")
print(f"Churn treino: {y_train.mean.round(3)}")
print(f"Churn teste: {y_test.mean.round(3)}")
print(f"Churn teste: {y_test.mean.round(3)}")
```

```
Treino: 5,634
Treino: 5,634
Teste: 1,409
Teste: 1,409
Churn treino: 0.265
Churn treino: 0.265
Churn teste: 0.265
Churn teste: 0.265
```

Se ainda observar que o tipo dados subiu o tamanho.

```
# ═══ 14. Escala ═══
scaler = StandardScaler
X_train_s = scaler.fit_transform(X_train)
X_train_s = scaler.fit_transform(X_train)
X_test_s = scaler.transform(X_test)
X_test_s = scaler.transform(X_test)
print("Scaler aplicado")
print("Scaler aplicado")
```

```
Scaler aplicado
Scaler aplicado
```

Se ainda observar que o tipo dados subiu o tamanho.

```
# ═══ 15. Modelos ═══
models = {
    'Logistic Regression': LogisticRegression(max_iter=1000, random_state=42),
    'Decision Tree': DecisionTreeClassifier(max_depth=5, random_state=42),
    'Random Forest': RandomForestClassifier(n_estimators=100, random_state=42),
    'Gradient Boosting': GradientBoostingClassifier(n_estimators=100, random_state=42),
    'AdaBoost': AdaBoostClassifier(n_estimators=100, random_state=42),
    'SVC': SVC(kernel='rbf', probability=True, random_state=42),
    'KNN': KNeighborsClassifier(n_neighbors=5),
    'Naive Bayes': GaussianNB,
}
models = {
    'Logistic Regression': LogisticRegression(max_iter=1000, random_state=42),
    'Decision Tree': DecisionTreeClassifier(max_depth=5, random_state=42),
    'Random Forest': RandomForestClassifier(n_estimators=100, random_state=42),
    'Gradient Boosting': GradientBoostingClassifier(n_estimators=100, random_state=42),
    'AdaBoost': AdaBoostClassifier(n_estimators=100, random_state=42),
    'SVC': SVC(kernel='rbf', probability=True, random_state=42),
    'KNN': KNeighborsClassifier(n_neighbors=5),
    'Naive Bayes': GaussianNB,
}
print("Modelos definidos")
print("Modelos definidos")
```

```
Modelos definidos
Modelos definidos
```

Se ainda observar que o tipo dados subiu o tamanho.

```
# ═══ 16. Treinamento ═══
results = {}
for name, model in models:
    model.fit(X_train_s, y_train)
    y_pred = model.predict(X_test_s)
    y_prob = model.predict_proba(X_test_s)[:, 1]
    acc = accuracy_score(y_test, y_pred).round(4)
    f1 = f1_score(y_test, y_pred).round(4)
    auc = roc_auc_score(y_test, y_prob).round(4)
    results[name] = {'acc': acc, 'f1': f1, 'auc': auc, 'y_pred': y_pred, 'y_prob': y_prob}
    print(f"{name:{max(len(n) for n in models)}}: Acc={acc:.4} F1={f1:.4} AUC={auc:.4}")
    print(f"{name:{max(len(n) for n in models)}}: Acc={acc:.4} F1={f1:.4} AUC={auc:.4}")
```

```
Logistic Regression: Acc=0.8052 F1=0.5893 AUC=0.8419
Logistic Regression: Acc=0.8052 F1=0.5893 AUC=0.8419
Decision Tree      : Acc=0.7846 F1=0.5511 AUC=0.8169
Decision Tree      : Acc=0.7846 F1=0.5511 AUC=0.8169
Random Forest      : Acc=0.7892 F1=0.5470 AUC=0.8232
Random Forest      : Acc=0.7892 F1=0.5470 AUC=0.8232
Gradient Boosting  : Acc=0.8019 F1=0.5900 AUC=0.8457
Gradient Boosting  : Acc=0.8019 F1=0.5900 AUC=0.8457
AdaBoost           : Acc=0.7974 F1=0.5859 AUC=0.8452
AdaBoost           : Acc=0.7974 F1=0.5859 AUC=0.8452
SVC                : Acc=0.7960 F1=0.5730 AUC=0.8382
SVC                : Acc=0.7960 F1=0.5730 AUC=0.8382
KNN                : Acc=0.7660 F1=0.5331 AUC=0.7823
KNN                : Acc=0.7660 F1=0.5331 AUC=0.7823
Naive Bayes        : Acc=0.7344 F1=0.6208 AUC=0.8416
Naive Bayes        : Acc=0.7344 F1=0.6208 AUC=0.8416
```

Se ainda observar que o tipo dados subiu o tamanho.

```
# ═══ 17. Comparação ═══
print("═══ 8. COMPARAÇÃO ═══")
print("═══ 8. COMPARAÇÃO ═══")
comp = pd.DataFrame({n: {'Acc': r['acc'], 'F1': r['f1'], 'AUC': r['auc']} for n, r in results})
comp = pd.DataFrame({n: {'Acc': r['acc'], 'F1': r['f1'], 'AUC': r['auc']} for n, r in results})
display(comp.sort_values('AUC', ascending=False).T)
display(comp.sort_values('AUC', ascending=False).T)
```

	Logistic Regression	Gradient Boosting	AdaBoost	Random Forest	SVC	Decision Tree	KNN	Naive Bayes
Acc	0.8052	0.8019	0.7974	0.7892	0.7960	0.7846	0.7660	0.7344
F1	0.5893	0.5900	0.5859	0.5470	0.5730	0.5511	0.5331	0.6208
AUC	0.8419	0.8457	0.8452	0.8232	0.8382	0.8169	0.7823	0.8416

Se ainda observar que o tipo dados subiu o tamanho.

```
# ═══ 18. Gráfico ═══
fig, axes = plt.subplots(1, 3, figure(18, 5))
fig, axes = plt.subplots(1, 3, figure(18, 5))
for i, m in enumerate(['Acc', 'F1', 'AUC']):
    for i, m in enumerate(['Acc', 'F1', 'AUC']):
    axes[i].barh(comp.index, comp[m], color=plt.cm.viridis(np.linspace(0.3, 0.8, len(comp))))
    axes[i].barh(comp.index, comp[m], color=plt.cm.viridis(np.linspace(0.3, 0.8, len(comp))))
    axes[i].set_title(m, fontsize=14, fontweight='bold')
    axes[i].set_title(m, fontsize=14, fontweight='bold')
plt.tight_layout
plt.tight_layout
plt.show
plt.show
```

Se ainda observar que o tipo dados subiu o tamanho.

```
# ═══ 19. Curvas ROC ═══
fig, ax = plt.subplots(1, 1, figure(8, 6))
fig, ax = plt.subplots(1, 1, figure(8, 6))
for name, r in results:
    for name, r in results:
    fpr, tpr, _ = roc_curve(y_test, r['y_prob'])
    fpr, tpr, _ = roc_curve(y_test, r['y_prob'])
    ax.plot(fpr, tpr, label=f'{name} (AUC={r['auc']:.4})')
    ax.plot(fpr, tpr, label=f'{name} (AUC={r['auc']:.4})')
ax.plot(np.linspace(0, 1), np.linspace(0, 1), 'k--', alpha=0.5)
ax.plot(np.linspace(0, 1), np.linspace(0, 1), 'k--', alpha=0.5)
ax.set_title('Curvas ROC', fontsize=14, fontweight='bold')
ax.set_title('Curvas ROC', fontsize=14, fontweight='bold')
ax.set_xlabel('FPR', fontsize=12)
ax.set_xlabel('FPR', fontsize=12)
ax.set_ylabel('TPR', fontsize=12)
ax.set_ylabel('TPR', fontsize=12)
ax.legend(fontsize=10)
ax.legend(fontsize=10)
plt.tight_layout
plt.tight_layout
plt.show
plt.show
```

Se ainda observar que o tipo dados subiu o tamanho.

```
# ═══ 20. Matrizes de confusão ═══
fig, axes = plt.subplots(2, 4, figure(20, 10))
fig, axes = plt.subplots(2, 4, figure(20, 10))
for i, name, r in enumerate(results):
    for i, name, r in enumerate(results):
    cm = confusion_matrix(y_test, r['y_pred'])
    cm = confusion_matrix(y_test, r['y_pred'])
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', ax=axes[i], cbar=False)
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', ax=axes[i], cbar=False)
    axes[i].set_title(name, fontsize=12)
    axes[i].set_title(name, fontsize=12)
plt.tight_layout
plt.tight_layout
plt.show
plt.show
```

Se ainda observar que o tipo dados subiu o tamanho.

```
# ═══ 21. Relatórios ═══
for name, r in results:
    for name, r in results:
    print(f{name:{max(len(n) for n in models)}}:")
    print(f{name:{max(len(n) for n in models)}}:")
    print(classification_report(y_test, r['y_pred'], target_names=['Não', 'Sim']))
    print(classification_report(y_test, r['y_pred'], target_names=['Não', 'Sim']))
```

```
Logistic Regression:
Logistic Regression:
              precision    recall  f1-score   support
              precision    recall  f1-score   support

         Não       0.84      0.89      0.86      1,033
         Não       0.84      0.89      0.86      1,033
         Sim       0.66      0.54      0.59       376
         Sim       0.66      0.54      0.59       376

    accuracy                           0.80      1,409
    accuracy                           0.80      1,409
   macro avg       0.75      0.71      0.73      1,409
   macro avg       0.75      0.71      0.73      1,409
weighted avg       0.79      0.80      0.79      1,409
weighted avg       0.79      0.80      0.79      1,409

Gradient Boosting:
Gradient Boosting:
              precision    recall  f1-score   support
              precision    recall  f1-score   support

         Não       0.85      0.90      0.87      1,033
         Não       0.85      0.90      0.87      1,033
         Sim       0.67      0.55      0.60       376
         Sim       0.67      0.55      0.60       376

    accuracy                           0.80      1,409
    accuracy                           0.80      1,409
   macro avg       0.76      0.72      0.74      1,409
   macro avg       0.76      0.72      0.74      1,409
weighted avg       0.79      0.80      0.79      1,409
weighted avg       0.79      0.80      0.79      1,409

AdaBoost:
AdaBoost:
              precision    recall  f1-score   support
              precision    recall  f1-score   support

         Não       0.85      0.89      0.87      1,033
         Não       0.85      0.89      0.87      1,033
         Sim       0.66      0.55      0.60       376
         Sim       0.66      0.55      0.60       376

    accuracy                           0.80      1,409
    accuracy                           0.80      1,409
   macro avg       0.75      0.72      0.73      1,409
   macro avg       0.75      0.72      0.73      1,409
weighted avg       0.79      0.80      0.79      1,409
weighted avg       0.79      0.80      0.79      1,409

Random Forest:
Random Forest:
              precision    recall  f1-score   support
              precision    recall  f1-score   support

         Não       0.83      0.91      0.86      1,033
         Não       0.83      0.91      0.86      1,033
         Sim       0.65      0.47      0.55       376
         Sim       0.65      0.47      0.55       376

    accuracy                           0.79      1,409
    accuracy                           0.79      1,409
   macro avg       0.74      0.69      0.70      1,409
   macro avg       0.74      0.69      0.70      1,409
weighted avg       0.78      0.79      0.78      1,409
weighted avg       0.78      0.79      0.78      1,409

SVC:
SVC:
              precision    recall  f1-score   support
              precision    recall  f1-score   support

         Não       0.84      0.90      0.87      1,033
         Não       0.84      0.90      0.87      1,033
         Sim       0.67      0.54      0.60       376
         Sim       0.67      0.54      0.60       376

    accuracy                           0.80      1,409
    accuracy                           0.80      1,409
   macro avg       0.75      0.72      0.73      1,409
   macro avg       0.75      0.72      0.73      1,409
weighted avg       0.79      0.80      0.79      1,409
weighted avg       0.79      0.80      0.79      1,409

Decision Tree:
Decision Tree:
              precision    recall  f1-score   support
              precision    recall  f1-score   support

         Não       0.83      0.89      0.86      1,033
         Não       0.83      0.89      0.86      1,033
         Sim       0.63      0.52      0.57       376
         Sim       0.63      0.52      0.57       376

    accuracy                           0.78      1,409
    accuracy                           0.78      1,409
   macro avg       0.73      0.70      0.71      1,409
   macro avg       0.73      0.70      0.71      1,409
weighted avg       0.77      0.78      0.77      1,409
weighted avg       0.77      0.78      0.77      1,409

KNN:
KNN:
              precision    recall  f1-score   support
              precision    recall  f1-score   support

         Não       0.82      0.87      0.84      1,033
         Não       0.82      0.87      0.84      1,033
         Sim       0.57      0.49      0.53       376
         Sim       0.57      0.49      0.53       376

    accuracy                           0.76      1,409
    accuracy                           0.76      1,409
   macro avg       0.69      0.68      0.68      1,409
   macro avg       0.69      0.68      0.68      1,409
weighted avg       0.75      0.76      0.75      1,409
weighted avg       0.75      0.76      0.75      1,409

Naive Bayes:
Naive Bayes:
              precision    recall  f1-score   support
              precision    recall  f1-score   support

         Não       0.91      0.73      0.81      1,033
         Não       0.91      0.73      0.81      1,033
         Sim       0.49      0.82      0.61       376
         Sim       0.49      0.82      0.61       376

    accuracy                           0.74      1,409
    accuracy                           0.74      1,409
   macro avg       0.70      0.77      0.71      1,409
   macro avg       0.70      0.77      0.71      1,409
weighted avg       0.81      0.74      0.76      1,409
weighted avg       0.81      0.74      0.76      1,409
```

Se ainda observar que o tipo dados subiu o tamanho.

```
# ═══ 22. Importância ═══
rf = models['Random Forest']
rf = models['Random Forest']
imp = pd.Series(rf.feature_importances_, index=X.columns).sort_values(ascending=False)
imp = pd.Series(rf.feature_importances_, index=X.columns).sort_values(ascending=False)
for c, v in imp.head(15):
    for c, v in imp.head(15):
    print(f{c:{max(len(c) for c in X.columns)}}: {v:.4}")
    print(f{c:{max(len(c) for c in X.columns)}}: {v:.4}")
```

```
TotalCharges     : 0.1847
TotalCharges     : 0.1847
MonthlyCharges   : 0.1802
MonthlyCharges   : 0.1802
tenure           : 0.1700
tenure           : 0.1700
Contract         : 0.0723
Contract         : 0.0723
PaymentMethod    : 0.0420
PaymentMethod    : 0.0420
OnlineSecurity   : 0.0311
OnlineSecurity   : 0.0311
InternetService  : 0.0298
InternetService  : 0.0298
TechSupport      : 0.0289
TechSupport      : 0.0289
PaperlessBilling : 0.0270
PaperlessBilling : 0.0270
OnlineBackup     : 0.0264
OnlineBackup     : 0.0264
DeviceProtection : 0.0263
DeviceProtection : 0.0263
StreamingTV      : 0.0258
StreamingTV      : 0.0258
```

Se ainda observar que o tipo dados subiu o tamanho.

```
# ═══ 23. Gráfico ═══
fig, ax = plt.subplots(1, 1, figure(8, 6))
fig, ax = plt.subplots(1, 1, figure(8, 6))
top = imp.head(15)
top = imp.head(15)
ax.barh(top.index, top.values, color=plt.cm.viridis(np.linspace(0.3, 0.8, 15)))
ax.barh(top.index, top.values, color=plt.cm.viridis(np.linspace(0.3, 0.8, 15)))
ax.set_title('Importância (Random Forest)', fontsize=14, fontweight='bold')
ax.set_title('Importância (Random Forest)', fontsize=14, fontweight='bold')
plt.tight_layout
plt.tight_layout
plt.show
plt.show
```


Se ainda observar que o tipo dados subiu o tamanho.

```
# ═══ 24. Salvar ═══
import joblib
import joblib
joblib.dump(rf, 'modelo_churn.pkl')
joblib.dump(rf, 'modelo_churn.pkl')
joblib.dump(scaler, 'scaler.pkl')
joblib.dump(scaler, 'scaler.pkl')
print("Salvo!")
print("Salvo!")
```

```
Salvo!
Salvo!
```

Se ainda observar que o tipo dados subiu o tamanho.

```
# ═══ 25. Resumo ═══
print("═══ 9. RESUMO ═══")
print("═══ 9. RESUMO ═══")
print(f"Dados: {df.shape[0]:,} × {df.shape[1]}")
print(f"Dados: {df.shape[0]:,} × {df.shape[1]}")
print(f"Churn: {y.mean.round(3)}")
print(f"Churn: {y.mean.round(3)}")
print(f"Features: {X.shape[1]}")
print(f"Features: {X.shape[1]}")
print(f"Melhor AUC: {max(r['auc'] for r in results)}")
print(f"Melhor AUC: {max(r['auc'] for r in results)}")
```

```
═══ 9. RESUMO ═══
═══ 9. RESUMO ═══
Dados: 7,043 × 38
Dados: 7,043 × 38
Churn: 0.265
Churn: 0.265
Features: 35
Features: 35
Melhor AUC: 0.8457
Melhor AUC: 0.8457
```

8. Análise de dados

Se ainda observar que o tipo dados subiu o tamanho.

```
# ═══ 26. Distribuição do alvo ═══
print("═══ 10. DISTRIBUIÇÃO DO ALVO ═══")
print("═══ 10. DISTRIBUIÇÃO DO ALVO ═══")
alvo_dist = df['alvo'].value_counts
alvo_dist = df['alvo'].value_counts
for i, v in alvo_dist.items:
    for i, v in alvo_dist.items:
    print(f"  {i}: {v:,} ({v/len(df)*100:.1f%})")
    print(f"  {i}: {v:,} ({v/len(df)*100:.1f%})")
```

```
═══ 10. DISTRIBUIÇÃO DO ALVO ═══
═══ 10. DISTRIBUIÇÃO DO ALVO ═══
  0: 5,174 (73.4%)
  0: 5,174 (73.4%)
  1: 1,869 (26.5%)
  1: 1,869 (26.5%)
```

Se ainda observar que o tipo dados subiu o tamanho.

```
# ═══ 27. Gráfico ═══
fig, ax = plt.subplots(1, 1, figure(8, 6))
fig, ax = plt.subplots(1, 1, figure(8, 6))
ax.pie(alvo_dist.values, labels=['Não Churn', 'Churn'], autopct='%1.1%', colors=['#4ecdc4', '#ff6b6b'], explode=(0, 0.05), shadow=True)
ax.pie(alvo_dist.values, labels=['Não Churn', 'Churn'], autopct='%1.1%', colors=
```