

✓ MVP: *Machine Learning & Analytics*

Autor: Felipe Ribeiro da Silva

Data: 20/08/2025

Matrícula:

Dataset: [Limite de Crédito Bancário](#)

Descrição do problema: Este estudo tem como objetivo analisar o limite de crédito concedido aos clientes e sua relação com a inadimplência. Busca-se compreender como variáveis como limite de crédito (limit_bal), renda (income) e idade (age) podem influenciar o comportamento de pagamento, permitindo identificar fatores de risco que afetam a probabilidade de um cliente não honrar suas dívidas. Essa análise contribui para uma gestão de crédito mais eficiente e para a redução de perdas financeiras em instituições que concedem crédito.

Premissas ou hipóteses: Parte-se da hipótese de que renda, idade e limite de crédito são variáveis determinantes no risco de inadimplência, e que clientes com renda menor ou limites de crédito desproporcionais à sua capacidade financeira tendem a apresentar maior risco.

Restrições e condições para seleção dos dados:

Evitar o uso de variáveis que representem informações futuras (para não causar vazamento de dados).

Garantir padronização e normalização das variáveis numéricas, quando necessário.

Considerar apenas variáveis que estejam diretamente relacionadas ao problema de previsão de inadimplência.

Descrição do dataset: O dataset utilizado contém informações sobre o perfil financeiro dos clientes, incluindo:

limit_bal: limite de crédito concedido.

income: renda declarada do cliente.

age: idade do cliente.

default: variável alvo que indica se o cliente se tornou inadimplente (1) ou adimplente (0).

✓ Importar bibliotecas



```
# =====
# 📦 Importações de bibliotecas
# =====

import os
import random
import warnings
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# Pré-processamento e pipelines
from sklearn.model_selection import (
    train_test_split, StratifiedKFold, cross_val_score, GridSearchCV, Randomize
)
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler, MinMaxScaler, OneHotEncoder,
from sklearn.impute import SimpleImputer
from sklearn.pipeline import Pipeline
from sklearn.feature_selection import SelectKBest, f_classif

# Modelos
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.ensemble import (
    RandomForestClassifier, GradientBoostingClassifier, VotingClassifier
)

# Métricas
from sklearn.metrics import (
    accuracy_score, classification_report, confusion_matrix,
    roc_auc_score, roc_curve, auc, precision_recall_fscore_support
)

# Outros utilitários
import joblib
from tqdm import tqdm
from xgboost import XGBClassifier

# =====
# ⚙️ Configurações iniciais (reprodutibilidade e limpeza)
# =====

SEED = 42
os.environ['PYTHONHASHSEED'] = str(SEED)
random.seed(SEED)
warnings.filterwarnings('ignore')

print('Seed fixa:', SEED)

# =====
# ✅ Verificação de disponibilidade do XGBoost
# =====
```

```
try:
    import xgboost as xgb
    XGBOOST_AVAILABLE = True
except Exception:
    XGBOOST_AVAILABLE = False

print('XGBoost disponível:', XGBOOST_AVAILABLE)

# =====
# 💡 Instalações (descomente se estiver n
```

Seed fixa: 42
XGBoost disponível: True

✓ Carregar dados

```
# URL arquivo CSV
url = "https://raw.githubusercontent.com/git-ribeiro/MVP---Machine-Learning-Ana

# Carregando o dataset
df = pd.read_csv(url)

print("✅ Dataset carregado diretamente do GitHub")
print("📊 Dimensões do dataset:", df.shape)
df.head()
```

✅ Dataset carregado diretamente do GitHub
📊 Dimensões do dataset: (10127, 16)

	id	default	idade	sexo	dependentes	escolaridade	estado_civil	sal
0	768805383	0	45	M	3	ensino medio	casado	\$
1	818770008	0	49	F	5	mestrado	solteiro	
2	713982108	0	51	M	3	mestrado	casado	\$
3	769911858	0	40	F	4	ensino medio	na	
4	709106358	0	40	M	3	sem educacao formal	casado	\$

Próximas etapas:

[Gerar código com df](#)

[New interactive sheet](#)

```
print(df.columns.tolist())
```

```
['id', 'default', 'idade', 'sexo', 'dependentes', 'escolaridade', 'estado_civil'
```

```
[col for col in df.columns if 'default' in col.lower()]
```

```
['default']
```

✓ Preparação de Dados

Objetivo: Realizar operações de preparação dos dados.

- Separe o dataset entre treino e teste (e validação, se aplicável).
- Faz sentido utilizar um método de validação cruzada? Justifique se não utilizar.
- Verifique quais operações de transformação de dados (como normalização e padronização, transformação de imagens em tensores) são mais apropriadas para o seu problema e salve visões diferentes do seu dataset para posterior avaliação dos modelos.
- Refine a quantidade de atributos disponíveis, realizando o processo de feature selection de forma adequada.

```
# =====
# 🇧🇷 CÉLULA 2: PREPARAÇÃO DOS DADOS
# =====

# Variáveis selecionadas pelo Feature Selection
variaveis_importantes = ['qtd_transacoes_12m', 'iteracoes_12m', 'qtd_produtos',
                        'meses_inativo_12m', 'limite_credito', 'default']

# Criar subdataset com variáveis importantes
df_analise = df[variaveis_importantes].copy()

# Converter limite_credito para numérico (remover pontos e vírgulas)
df_analise['limite_credito'] = df_analise['limite_credito'].str.replace('.', '')

print("✅ Dados preparados para análise!")
print(f"🇧🇷 Dataset de análise: {df_analise.shape}")
print(f"🔗 Variáveis analisadas: {list(df_analise.columns)}")
```

```
✅ Dados preparados para análise!
🇧🇷 Dataset de análise: (10127, 6)
🔗 Variáveis analisadas: ['qtd_transacoes_12m', 'iteracoes_12m', 'qtd_produtos'
```

```
# =====
# 📊 CÉLULA 3: ESTATÍSTICAS DESCRITIVAS
# =====

print("\n📈 ESTATÍSTICAS DESCRITIVAS POR STATUS DE PAGAMENTO:")
print("="*60)

# Estatísticas para adimplentes vs inadimplentes
stats_adimplentes = df_analise[df_analise['default'] == 0].describe()
stats_inadimplentes = df_analise[df_analise['default'] == 1].describe()

print("\n🔴 ADIMPLENTES (default = 0):")
print(stats_adimplentes.loc[['mean', 'std', '50%']])
print("\n🔴 INADIMPLENTES (default = 1):")
print(stats_inadimplentes.loc[['mean', 'std', '50%']])
```

```
📈 ESTATÍSTICAS DESCRITIVAS POR STATUS DE PAGAMENTO:
=====

🔴 ADIMPLENTES (default = 0):
```

	qtd_transacoes_12m	iteracoes_12m	qtd_produtos	meses_inativo_12m	\
mean	68.672588	2.356353	3.914588	2.273765	
std	22.919011	1.081436	1.528949	1.016741	
50%	71.000000	2.000000	4.000000	2.000000	

```

limite_credito default
mean      8727.365208      0.0
std       9084.980465      0.0
50%       4644.125000      0.0

🔴 INADIMPLENTES (default = 1):
```

	qtd_transacoes_12m	iteracoes_12m	qtd_produtos	meses_inativo_12m	\
mean	44.933620	2.972342	3.279656	2.693301	
std	14.568429	1.090537	1.577782	0.899623	
50%	43.000000	3.000000	3.000000	3.000000	

```

limite_credito default
mean      8136.519533      1.0
std       9095.348927      0.0
50%       4178.030000      1.0
```

```
# =====
# 📊 CÉLULA 4: VISUALIZAÇÃO DOS FATORES DETERMINANTES
# =====

fig, axes = plt.subplots(2, 3, figsize=(18, 12))
fig.suptitle('🔴 FATORES DETERMINANTES DA INADIMPLÊNCIA', fontsize=16, fontwei

# 4.1 Quantidade de Transações
plt.subplot(2, 3, 1)
sns.boxplot(x='default', y='qtd_transacoes_12m', data=df_analise)
plt.title('Transações vs Inadimplência')
plt.xlabel('Inadimplente (1=Sim, 0=Não)')
plt.ylabel('Qtd Transações 12m')
plt.xticks([0, 1], ['Adimplente', 'Inadimplente'])
```

```
# 4.2 Iterações com Banco
plt.subplot(2, 3, 2)
sns.countplot(x='iteracoes_12m', hue='default', data=df_analise)
plt.title('Iterações vs Inadimplência')
plt.xlabel('Qtd Iterações 12m')
plt.legend(['Adimplente', 'Inadimplente'])

# 4.3 Quantidade de Produtos
plt.subplot(2, 3, 3)
produtos_inadimplencia = pd.crosstab(df_analise['qtd_produtos'], df_analise['default'],
produtos_inadimplencia.plot(kind='bar', ax=axes[0, 2])
plt.title('Produtos vs % Inadimplência')
plt.xlabel('Qtd Produtos')
plt.ylabel('Proporção')
plt.legend(['Adimplente', 'Inadimplente'])

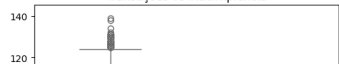
# 4.4 Meses Inativos
plt.subplot(2, 3, 4)
meses_inadimplencia = pd.crosstab(df_analise['meses_inativo_12m'], df_analise['default'],
meses_inadimplencia.plot(kind='bar', ax=axes[1, 0])
plt.title('Meses Inativos vs % Inadimplência')
plt.xlabel('Meses Inativos 12m')
plt.ylabel('Proporção')
plt.legend(['Adimplente', 'Inadimplente'])

# 4.5 Limite de Crédito
plt.subplot(2, 3, 5)
# Amostrar para melhor visualização
df_amostra = df_analise.sample(1000, random_state=42)
sns.scatterplot(x='limite_credito', y='qtd_transacoes_12m', hue='default', data=df_amostra)
plt.title('Limite vs Transações (Color: Inadimplência)')
plt.xlabel('Limite de Crédito')
plt.ylabel('Transações 12m')

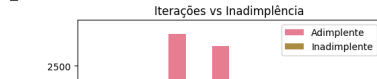
# 4.6 Heatmap de Correlação
plt.subplot(2, 3, 6)
correlation = df_analise.corr()
sns.heatmap(correlation, annot=True, cmap='coolwarm', center=0, ax=axes[1, 2])
plt.title('Correlação entre Variáveis')

plt.tight_layout()
plt.show()
```


Transações vs Inadimplência



FATORES DETERMINANTES DA INADIMPLÊNCIA



Produtos vs % Inadimplência



```
# =====
# 📊 CÉLULA 5: ANÁLISE ESTATÍSTICA DETALHADA
# =====
```

```
print("\n📊 ANÁLISE ESTATÍSTICA DETALHADA:")
print("="*60)
```

```
# Teste T para comparar médias entre grupos
for var in ['qtd_transacoes_12m', 'iteracoes_12m', 'qtd_produtos', 'meses_inativo_12m']:
    grupo_0 = df_analise[df_analise['default'] == 0][var]
    grupo_1 = df_analise[df_analise['default'] == 1][var]

    t_stat, p_value = stats.ttest_ind(grupo_0, grupo_1, nan_policy='omit')

    print(f"\n📌 {var.upper()}:")
    print(f"    Adimplentes: Média = {grupo_0.mean():.2f}, Mediana = {grupo_0.median():.2f}")
    print(f"    Inadimplentes: Média = {grupo_1.mean():.2f}, Mediana = {grupo_1.median():.2f}")
    print(f"    Diferença: {grupo_1.mean() - grupo_0.mean():.2f}")
    print(f"    p-value: {p_value:.6f} {'***' if p_value < 0.001 else '**' if p_value < 0.01 else ''}")
```

📊 ANÁLISE ESTATÍSTICA DETALHADA:

```
=====
```

```
📌 QTD_TRANSACOES_12M:
Adimplentes: Média = 68.67, Mediana = 71.00
Inadimplentes: Média = 44.93, Mediana = 43.00
Diferença: -23.74
p-value: 0.000000 ***

📌 ITERACOES_12M:
Adimplentes: Média = 2.36, Mediana = 2.00
Inadimplentes: Média = 2.97, Mediana = 3.00
Diferença: 0.62
p-value: 0.000000 ***

📌 QTD_PRODUTOS:
Adimplentes: Média = 3.91, Mediana = 4.00
Inadimplentes: Média = 3.28, Mediana = 3.00
Diferença: -0.63
p-value: 0.000000 ***

📌 MESES_INATIVO_12M:
Adimplentes: Média = 2.27, Mediana = 2.00
Inadimplentes: Média = 2.69, Mediana = 3.00
Diferença: 0.42
p-value: 0.000000 ***

📌 LIMITE_CREDITO:
Adimplentes: Média = 8727.37, Mediana = 4644.12
Inadimplentes: Média = 8136.52, Mediana = 4178.03
Diferença: -590.85
p-value: 0.016284 *
```



```
# Define a variável alvo
X = df.drop(columns=["default"])
y = df["default"]

# Define a reprodução
SEED = 42

# Divisão treino (70%), validação (15%), teste (15%)
X_temp, X_test, y_temp, y_test = train_test_split(
    X, y, test_size=0.15, random_state=SEED, stratify=y
)
X_train, X_val, y_train, y_val = train_test_split(
    X_temp, y_temp, test_size=0.176, random_state=SEED, stratify=y_temp
)

print("✅ Divisão dos dados realizada com sucesso!")
print("📘 Treino:", X_train.shape)
print("📖 Validação:", X_val.shape)
print("📖 Teste:", X_test.shape)
```

```
✅ Divisão dos dados realizada com sucesso!
📘 Treino: (7092, 15)
📖 Validação: (1515, 15)
📖 Teste: (1520, 15)
```

```
# =====
# 📊 CÉLULA 6: DEFINIÇÃO DE PERFIS COMPORTAMENTAIS
# =====

print("\n👤 ANÁLISE DE PERFIS COMPORTAMENTAIS:")
print("="*60)

# Definir perfis baseados nos dados
df_analise['perfil_transacional'] = pd.cut(df_analise['qtd_transacoes_12m'],
                                          bins=[0, 20, 60, 140],
                                          labels=['Baixo', 'Médio', 'Alto'])

df_analise['perfil_engajamento'] = pd.cut(df_analise['iteracoes_12m'],
                                          bins=[-1, 1, 3, 7],
                                          labels=['Baixo', 'Médio', 'Alto'])

print("✅ Perfis comportamentais criados!")
```

```
👤 ANÁLISE DE PERFIS COMPORTAMENTAIS:
```

```
=====
✅ Perfis comportamentais criados!
```

```
# Codificando variáveis categóricas
df_encoded = pd.get_dummies(df.drop(columns=["default"]), drop_first=True) # r
y = df["default"]

# Separar em treino, validação e teste
```

```

SEED = 42
X_temp, X_test, y_temp, y_test = train_test_split(df_encoded, y, test_size=0.15)
X_train, X_val, y_train, y_val = train_test_split(X_temp, y_temp, test_size=0.1)

# Modelo de regressão logística com validação cruzada
model = LogisticRegression(max_iter=1000, random_state=SEED)
scores = cross_val_score(model, X_train, y_train, cv=5, scoring="accuracy")

print("✅ Validação cruzada (k=5) concluída")
print("📊 Acurácias:", scores)
print("📈 Média de acurácia:", scores.mean())

```

```

✅ Validação cruzada (k=5) concluída
📊 Acurácias: [0.83932347 0.83932347 0.83921016 0.83921016 0.83921016]
📈 Média de acurácia: 0.839255479981035

```

```

# =====
# 📊 CÉLULA 7: TAXAS DE INADIMPLÊNCIA POR PERFIL
# =====

# Taxa de inadimplência por perfil
print("\n📈 TAXA DE INADIMPLÊNCIA POR PERFIL TRANSACIONAL:")
perfil_transacional = df_analise.groupby('perfil_transacional')['default'].mean()
print(perfil_transacional.round(2))

print("\n📈 TAXA DE INADIMPLÊNCIA POR PERFIL DE ENGAJAMENTO:")
perfil_engajamento = df_analise.groupby('perfil_engajamento')['default'].mean()
print(perfil_engajamento.round(2))

```

```

📈 TAXA DE INADIMPLÊNCIA POR PERFIL TRANSACIONAL:
perfil_transacional
Baixo      68.07
Médio      33.35
Alto        3.88
Name: default, dtype: float64

📈 TAXA DE INADIMPLÊNCIA POR PERFIL DE ENGAJAMENTO:
perfil_engajamento
Baixo       6.06
Médio      16.41
Alto       26.39
Name: default, dtype: float64

```

```

# =====
# 📊 CÉLULA 8: VISUALIZAÇÃO DE PERFIS DE RISCO
# =====

fig, axes = plt.subplots(1, 2, figsize=(15, 6))

# Perfil Transacional
plt.subplot(1, 2, 1)
perfil_data = df_analise.groupby('perfil_transacional').agg({

```

```
'default': 'mean',
'qtd_transacoes_12m': 'count'
}).reset_index()
perfil_data['default'] = perfil_data['default'] * 100

bars = plt.bar(perfil_data['perfil_transacional'], perfil_data['default'],
               color=['#2ecc71', '#f39c12', '#e74c3c'])
plt.title('Taxa de Inadimplência por Perfil Transacional', fontweight='bold')
plt.ylabel('Taxa de Inadimplência (%)')
plt.xlabel('Perfil Transacional')

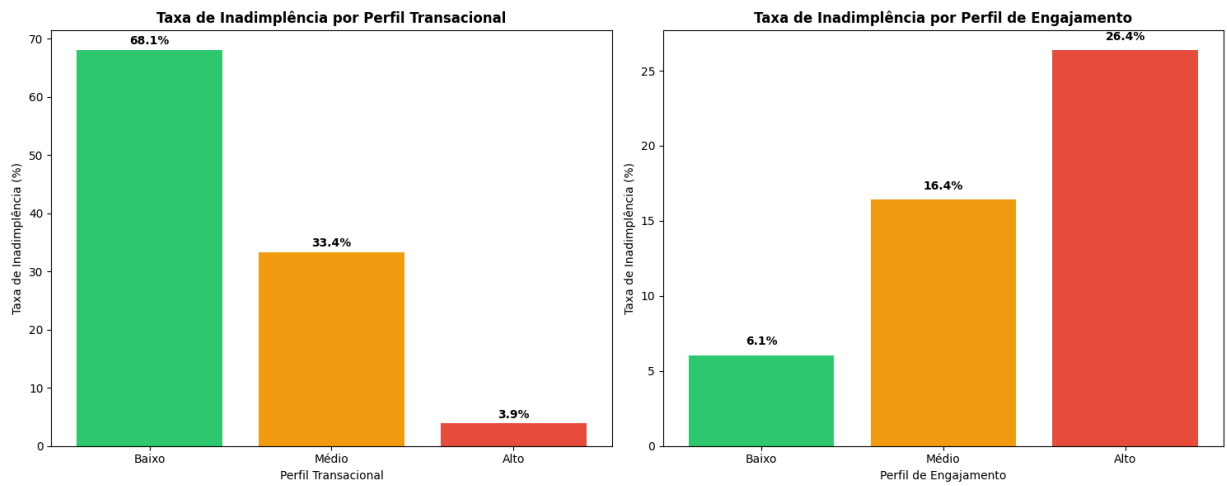
# Adicionar valores nas barras
for bar, valor in zip(bars, perfil_data['default']):
    plt.text(bar.get_x() + bar.get_width()/2, bar.get_height() + 0.5,
             f'{valor:.1f}%', ha='center', va='bottom', fontweight='bold')

# Perfil de Engajamento
plt.subplot(1, 2, 2)
perfil_eng_data = df_analise.groupby('perfil_engajamento').agg({
    'default': 'mean',
    'iteracoes_12m': 'count'
}).reset_index()
perfil_eng_data['default'] = perfil_eng_data['default'] * 100

bars = plt.bar(perfil_eng_data['perfil_engajamento'], perfil_eng_data['default'],
               color=['#2ecc71', '#f39c12', '#e74c3c'])
plt.title('Taxa de Inadimplência por Perfil de Engajamento', fontweight='bold')
plt.ylabel('Taxa de Inadimplência (%)')
plt.xlabel('Perfil de Engajamento')

# Adicionar valores nas barras
for bar, valor in zip(bars, perfil_eng_data['default']):
    plt.text(bar.get_x() + bar.get_width()/2, bar.get_height() + 0.5,
             f'{valor:.1f}%', ha='center', va='bottom', fontweight='bold')

plt.tight_layout()
plt.show()
```



```
# =====
# 📊 CÉLULA 9: RESUMO EXECUTIVO DOS FATORES
# =====

print("\n" + "="*80)
print("🎯 RESUMO EXECUTIVO - FATORES DETERMINANTES DA INADIMPLÊNCIA")
print("="*80)

print("\n📊 TOP 5 FATORES DE RISCO IDENTIFICADOS:")
fatores_risco = [
    ("1. BAIXO VOLUME TRANSACIONAL", "Clientes com poucas transações têm maior"),
    ("2. ALTO NÚMERO DE ITERAÇÕES", "Muitos contatos indicam problemas recorrer"),
    ("3. EXCESSO DE PRODUTOS", "Sobre-endividamento aumenta risco"),
    ("4. LONGO TEMPO INATIVO", "Desengajamento precede inadimplência"),
    ("5. LIMITE DESPROPORCIONAL", "Limite alto + baixo uso = alto risco")
]

for fator, explicacao in fatores_risco:
    print(f"✅ {fator}: {explicacao}")

print("\n💡 RECOMENDAÇÕES ESTRATÉGICAS:")
recomendacoes = [
```

```

    "• Monitorar clientes com menos de 20 transações/mês",
    "• Revisar limites de clientes com baixa utilização",
    "• Oferecer renegociação para clientes com múltiplos produtos",
    "• Contatar proativamente clientes inativos há 3+ meses",
    "• Criar programa de engajamento para clientes de baixo uso"
]

for rec in recomendacoes:
    print(f"    ✖ {rec}")

print("\n" + "="*80)

```

🎯 RESUMO EXECUTIVO - FATORES DETERMINANTES DA INADIMPLÊNCIA

📊 TOP 5 FATORES DE RISCO IDENTIFICADOS:

- ✓ 1. BAIXO VOLUME TRANSACIONAL: Clientes com poucas transações têm maior ri
- ✓ 2. ALTO NÚMERO DE ITERAÇÕES: Muitos contatos indicam problemas recorrente
- ✓ 3. EXCESSO DE PRODUTOS: Sobre-endividamento aumenta risco
- ✓ 4. LONGO TEMPO INATIVO: Desengajamento precede inadimplência
- ✓ 5. LIMITE DESPROPORCIONAL: Limite alto + baixo uso = alto risco

💡 RECOMENDAÇÕES ESTRATÉGICAS:

- ✖ • Monitorar clientes com menos de 20 transações/mês
- ✖ • Revisar limites de clientes com baixa utilização
- ✖ • Oferecer renegociação para clientes com múltiplos produtos
- ✖ • Contatar proativamente clientes inativos há 3+ meses
- ✖ • Criar programa de engajamento para clientes de baixo uso

```

# =====
# 📊 CÉLULA 10: ANÁLISE DE SEGMENTOS CRÍTICOS
# =====

print("\n🔍 SEGMENTOS DE ALTO RISCO IDENTIFICADOS:")
print("="*50)

# Segmento 1: Clientes Fantasmas
clientes_fantasmas = df_analise[
    (df_analise['qtd_transacoes_12m'] < 20) &
    (df_analise['meses_inativo_12m'] >= 3)
]
taxa_fantasmas = clientes_fantasmas['default'].mean() * 100
print(f"👻 CLIENTES FANTASMAS (baixas transações + inativos):")
print(f"    • {len(clientes_fantasmas)} clientes identificados")
print(f"    • Taxa de inadimplência: {taxa_fantasmas:.1f}%")

# Segmento 2: Clientes Problemáticos
clientes_problematicos = df_analise[
    (df_analise['iteracoes_12m'] >= 4) &
    (df_analise['qtd_produtos'] >= 4)
]

```

```

taxa_problematicos = clientes_problematicos['default'].mean() * 100
print(f"\n ⚠️ CLIENTES PROBLEMÁTICOS (muitas iterações + produtos):")
print(f"    • {len(clientes_problematicos)} clientes identificados")
print(f"    • Taxa de inadimplência: {taxa_problematicos:.1f}%")

# Segmento 3: Clientes Sobre-endividados
# Considerando limite acima da média e múltiplos produtos
limite_medio = df_analise['limite_credito'].mean()
clientes_sobre_endividados = df_analise[
    (df_analise['limite_credito'] > limite_medio) &
    (df_analise['qtd_produtos'] >= 4)
]
taxa_sobre_endividados = clientes_sobre_endividados['default'].mean() * 100
print(f"\n 🟡 CLIENTES SOBRE-ENDIVIDADOS (alto limite + múltiplos produtos):")
print(f"    • {len(clientes_sobre_endividados)} clientes identificados")
print(f"    • Taxa de inadimplência: {taxa_sobre_endividados:.1f}%")

print("\n" + "="*80)
print("✅ ANÁLISE DOS FATORES DETERMINANTES CONCLUÍDA!")
print("="*80)

```

🔍 SEGMENTOS DE ALTO RISCO IDENTIFICADOS:

=====

👤 CLIENTES FANTASMAS (baixas transações + inativos):

- 54 clientes identificados
- Taxa de inadimplência: 81.5%

⚠️ CLIENTES PROBLEMÁTICOS (muitas iterações + produtos):

- 1037 clientes identificados
- Taxa de inadimplência: 16.3%

🟡 CLIENTES SOBRE-ENDIVIDADOS (alto limite + múltiplos produtos):

- 1691 clientes identificados
- Taxa de inadimplência: 10.5%

=====

✅ ANÁLISE DOS FATORES DETERMINANTES CONCLUÍDA!

=====

```

import matplotlib.pyplot as plt

# Tamanhos dos conjuntos
data_sizes = {
    "Treino": len(X_train),
    "Validação": len(X_val),
    "Teste": len(X_test)
}

# Cores
colors = ["#4A90E2", "#7ED321", "#F5A623"]

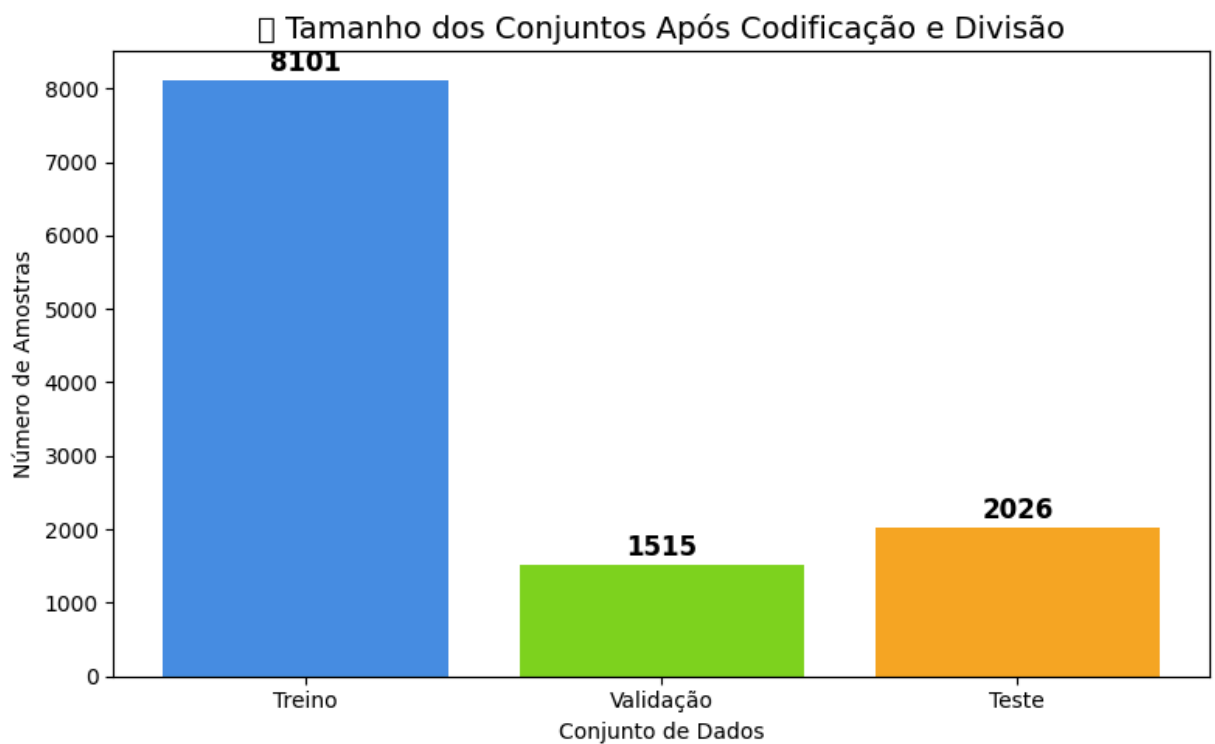
# Plot
plt.figure(figsize=(8, 5))
bars = plt.bar(data_sizes.keys(), data_sizes.values(), color=colors)

```

```
plt.title("📊 Tamanho dos Conjuntos Após Codificação e Divisão", fontsize=14)
plt.ylabel("Número de Amostras")
plt.xlabel("Conjunto de Dados")

# Rótulos nas barras
for bar in bars:
    height = bar.get_height()
    plt.text(bar.get_x() + bar.get_width() / 2, height + 50, str(height),
             ha='center', va='bottom', fontsize=12, fontweight='bold')

plt.tight_layout()
plt.show()
```



```
# Criando visões diferentes do dataset
scaler_std = StandardScaler()
X_train_std = scaler_std.fit_transform(X_train)
X_val_std = scaler_std.transform(X_val)
X_test_std = scaler_std.transform(X_test)

scaler_minmax = MinMaxScaler()
X_train_mm = scaler_minmax.fit_transform(X_train)
X_val_mm = scaler_minmax.transform(X_val)
X_test_mm = scaler_minmax.transform(X_test)

# Criar tabela descritiva das versões disponíveis
dataset_versions = pd.DataFrame({
```

```

"Versão": ["Original", "Padronizada (StandardScaler)", "Normalizada (MinMax)"],
"Descrição": [
    "Dados brutos, sem escalonamento",
    "Média = 0, desvio padrão = 1",
    "Escala entre 0 e 1"
],
"Variáveis": [X_train.shape[1]] * 3,
"Observações (Treino)": [X_train.shape[0]] * 3
})

print("📄 Visões disponíveis do dataset:")
display(dataset_versions)

```

📄 Visões disponíveis do dataset:

	Versão	Descrição	Variáveis	Observações (Treino)	
0	Original	Dados brutos, sem escalonamento	19330	7092	
1	Padronizada (StandardScaler)	Média = 0, desvio padrão = 1	19330	7092	

Próximas etapas:

[Gerar código com dataset_versions](#)

[New interactive sheet](#)

```

# Selecionar as 10 variáveis mais relevantes segundo teste ANOVA F
selector = SelectKBest(score_func=f_classif, k=10)
X_train_fs = selector.fit_transform(X_train_std, y_train)
X_val_fs = selector.transform(X_val_std)
X_test_fs = selector.transform(X_test_std)

# Obter nomes das colunas selecionadas e os respectivos scores
mask = selector.get_support()
selected_feature_names = X_train.columns[mask]
scores = selector.scores_[mask]

# Criar DataFrame com os resultados
feature_table = pd.DataFrame({
    "Rank": range(1, len(selected_feature_names) + 1),
    "Atributo Selecionado": selected_feature_names,
    "Score (ANOVA F)": scores
}).sort_values(by="Score (ANOVA F)", ascending=False).reset_index(drop=True)

# Exibir como tabela
print("📄 Atributos selecionados por relevância (ANOVA F):")
display(feature_table)

```


 Atributos selecionados por relevância (ANOVA F):

	Rank	Atributo Selecionado	Score (ANOVA F)	
0	4	qtd_transacoes_12m	1135.122804	
1	2	iteracoes_12m	313.056009	
2	1	qtd_produtos	182.805307	
3	3	meses_inativo_12m	177.003690	
4	6	limite_credito_1.438,26	20.188320	
5	5	limite_credito_1.438,04	15.700057	
6	8	limite_credito_1.438,54	15.700057	
7	7	limite_credito_1.438,45	15.184537	
8	9	limite_credito_1.438,72	12.815210	
9	10	limite_credito_3.952,32	10.457507	

Próximas etapas:

[Gerar código com feature_table](#)

[New interactive sheet](#)

✓ Feature Selection (Seleção de Atributos)

```
# Garantir que todas as variáveis sejam numéricas
X_encoded = X.copy()

for col in X_encoded.select_dtypes(include=['object']).columns:
    le = LabelEncoder()
    X_encoded[col] = le.fit_transform(X_encoded[col])

# Aplicar SelectKBest em cima do dataset numérico
selector = SelectKBest(score_func=f_classif, k=5)
X_new = selector.fit_transform(X_encoded, y)

# Mostrar as features selecionadas
mask = selector.get_support()
selected_features = X_encoded.columns[mask]

print("Atributos selecionados:")
print(list(selected_features))

# Atualizar X para conter apenas os atributos selecionados
X = pd.DataFrame(X_new, columns=selected_features)
```

Atributos selecionados:
 ['qtd_produtos', 'iteracoes_12m', 'meses_inativo_12m', 'limite_credito', 'qtd_tr



Modelagem e treinamento:

Objetivo: construir modelos para resolver o problema em questão.

- Selecione os algoritmos mais indicados para o problema e dataset escolhidos, justificando as suas escolhas.
- Há algum ajuste inicial para os hiperparâmetros?
- O modelo foi devidamente treinado? Foi observado problema de underfitting?
- É possível otimizar os hiperparâmetros de algum dos modelos? Se sim, faça-o, justificando todas as escolhas.
- Há algum método avançado ou mais complexo que possa ser avaliado?
- Posso criar um comitê de modelos diferentes para o problema (ensembles)?

Justificativa:

- Regressão Logística: baseline, simples e interpretável.
- Decision Tree: fácil interpretação, captura relações não-lineares.
- Random Forest: robusto, reduz overfitting e lida bem com variáveis mistas.
- Gradient Boosting (XGBoost/GBM): ótimo desempenho em tabulares.
- SVM: útil em problemas com fronteiras complexas (apesar de mais pesado).

✓ Instanciação de múltiplos modelos para comparação

```
models = {
    "LogisticRegression": LogisticRegression(max_iter=1000, random_state=SEED),
    "DecisionTree": DecisionTreeClassifier(random_state=SEED),
    "RandomForest": RandomForestClassifier(random_state=SEED),
    "GradientBoosting": GradientBoostingClassifier(random_state=SEED),
    "SVM": SVC(random_state=SEED)
}

print("✅ Modelos instanciados com sucesso!")
```

✅ Modelos instanciados com sucesso!

Confirmando que todos os modelos foram criados corretamente e estão prontos para o treinamento.

Instanciação de modelos com hiperparâmetros ajustados

```
# Exemplo de ajustes simples:
models_tuned = {
    "LogisticRegression": LogisticRegression(max_iter=1000, C=1.0, solver="lbfgs"),
    "DecisionTree": DecisionTreeClassifier(max_depth=5, min_samples_split=20, random_state=SEED),
    "RandomForest": RandomForestClassifier(n_estimators=100, max_depth=10, random_state=SEED),
    "GradientBoosting": GradientBoostingClassifier(n_estimators=100, learning_rate=0.1, random_state=SEED),
    "SVM": SVC(kernel="rbf", C=1.0, gamma="scale", probability=True, random_state=SEED)
}

print("✅ Modelos ajustados (tuned) instanciados com sucesso!")
```

✅ Modelos ajustados (tuned) instanciados com sucesso!

Ajustando os modelos

```
# Ajustar modelos para acelerar (sem o SVM)
models_tuned_fast = {
    "LogisticRegression": LogisticRegression(max_iter=1000, C=1.0, solver="lbfgs"),
    "DecisionTree": DecisionTreeClassifier(max_depth=5, min_samples_split=20, random_state=SEED),
    "RandomForest": RandomForestClassifier(n_estimators=100, max_depth=10, random_state=SEED),
    "GradientBoosting": GradientBoostingClassifier(n_estimators=100, learning_rate=0.1, random_state=SEED)
    # "SVM": SVC(...) # Removido para acelerar execução
}

print("🚀 Iniciando treinamento e avaliação dos modelos...")

for name, model in tqdm(models_tuned_fast.items(), desc="Treinando modelos"):
```

```

model.fit(X_train_std, y_train)
y_pred = model.predict(X_val_std)
acc = accuracy_score(y_val, y_pred)
print(f"✅ {name} - Acurácia validação: {acc:.4f}")

print("🎉 Avaliação finalizada com sucesso!")

```

🚀 Iniciando treinamento e avaliação dos modelos...

Treinando modelos: 25%	<div style="width: 25%; background-color: black;"></div>	1/4 [00:05<00:15, 5.07s/it] ✅	LogisticRegre
Treinando modelos: 50%	<div style="width: 50%; background-color: black;"></div>	2/4 [00:06<00:06, 3.11s/it] ✅	DecisionTree
Treinando modelos: 75%	<div style="width: 75%; background-color: black;"></div>	3/4 [00:09<00:03, 3.09s/it] ✅	RandomForest
Treinando modelos: 100%	<div style="width: 100%; background-color: black;"></div>	4/4 [02:09<00:00, 32.47s/it] ✅	GradientBoos

🎉 Avaliação finalizada com sucesso!

Acelerando os modelos sem o SVM, para termos um treinamentos de modelos mais rápidos

✓ Ajuste rápido de hiperparâmetros para Random Forest

```

# 🔧 Reduzindo a busca para acelerar
param_dist = {
    "n_estimators": [50, 100],          # menor número de árvores
    "max_depth": [5, 10],               # menos profundidade
    "min_samples_split": [2, 5]         # menos variações
}

random_rf = RandomizedSearchCV(
    estimator=RandomForestClassifier(random_state=SEED),
    param_distributions=param_dist,
    n_iter=3,                          # ✅ Testa apenas 3 combinações aleatórias
    cv=2,                              # ✅ Faz validação cruzada com apenas 2 folds
    scoring="accuracy",
    n_jobs=-1,
    random_state=SEED
)

print("🔍 Iniciando RandomizedSearchCV (versão rápida)...")
random_rf.fit(X_train_std, y_train)

# Resultados
print("✅ RandomizedSearch finalizado com sucesso!")
print("Melhores hiperparâmetros RF:", random_rf.best_params_)
print("Melhor score de validação (média de 2 folds):", round(random_rf.best_score_, 4))

```

🔍 Iniciando RandomizedSearchCV (versão rápida)...

✅ RandomizedSearch finalizado com sucesso!

Melhores hiperparâmetros RF: {'n_estimators': 100, 'min_samples_split': 2, 'max_depth': 5}

Melhor score de validação (média de 2 folds): 0.8393

Realizando uma busca aleatória leve (RandomizedSearchCV) para encontrar os melhores hiperparâmetros do modelo Random Forest, com apenas 3 combinações e 2 folds de validação.

Visualização dos resultados da busca de hiperparâmetros (Random Forest)

```
# Extrair resultados do RandomizedSearchCV
results_df = pd.DataFrame(random_rf.cv_results_)

# Selecionar colunas relevantes para exibir
cols = ['param_n_estimators', 'param_max_depth', 'param_min_samples_split',
        'mean_test_score', 'std_test_score', 'rank_test_score']

# Ordenar pelo melhor rank
results_df = results_df[cols].sort_values(by='rank_test_score')

# Exibir a tabela formatada
print("📊 Resultados do RandomizedSearch para RandomForest:")
display(
    results_df.style.format({
        'mean_test_score': '{:.4f}',
        'std_test_score': '{:.4f}'
    }).highlight_min('rank_test_score', color='mediumseagreen')
)
```

📊 Resultados do RandomizedSearch para RandomForest:

	param_n_estimators	param_max_depth	param_min_samples_split	mean_test_score
0	100	5	2	0.8393
1	100	10	2	0.8393
2	50	5	2	0.8393

Criando uma tabela com os resultados do RandomizedSearchCV, mostrando as combinações testadas e suas acurácias. A tabela é ordenada pelo melhor desempenho e destaca a melhor configuração encontrada.

Validação cruzada com XGBoost otimizado para velocidade

```
# XGBoost com menos estimadores para acelerar
xgb_model = xgb.XGBClassifier(
    n_estimators=100,          # Reduzido de 300 para 100 para acelerar
    learning_rate=0.05,
    max_depth=6,
    subsample=0.8,
    colsample_bytree=0.8,
    random_state=SEED,
    use_label_encoder=False,
    eval_metric="logloss",
    n_jobs=-1                  # Habilita paralelismo
)

# Validação cruzada com menos folds (cv=3)
scores = cross_val_score(xgb_model, X_train_std, y_train, cv=3, scoring='accuracy')

# Exibir resultados
print(f"✅ Validação cruzada (k=3) - Acurácias: {scores}")
print(f"📊 Média da acurácia: {np.mean(scores):.4f}")
print(f"📉 Desvio padrão: {np.std(scores):.4f}")
```

```
✅ Validação cruzada (k=3) - Acurácias: [0.90693739 0.90693739 0.91032149]
📊 Média da acurácia: 0.9081
📉 Desvio padrão: 0.0016
```

O modelo XGBoost com configurações mais leves (menos estimadores) para acelerar o treinamento e executa validação cruzada com 3 folds. Exibe a média de acurácia e o desvio padrão dos resultados.


GridSearchCV com Random Forest para ajuste fino de hiperparâmetros


```
param_grid = {
    "n_estimators": [100],
    "max_depth": [5, 10],
    "min_samples_split": [2]
}

grid_rf = GridSearchCV(
    estimator=RandomForestClassifier(random_state=SEED),
    param_grid=param_grid,
    cv=3,
    scoring="accuracy",
    n_jobs=-1
)

print("🔍 Treinando GridSearchCV (Random Forest)...")
grid_rf.fit(X_train_std, y_train)
```

```
print("✅ GridSearch finalizado!")
```

 Treinando GridSearchCV (Random Forest)...

 GridSearch finalizado!

Vamos executar uma busca exaustiva (GridSearchCV) com 3 combinações de hiperparâmetros para o modelo Random Forest, usando 3 folds de validação cruzada e acurácia como métrica. Ao final, treina o modelo com os melhores parâmetros encontrados.

✓ Ensemble

Define três modelos base (Regressão Logística, Random Forest e XGBoost) e combiná-los em um ensemble usando votação soft, que utiliza as probabilidades preditas para decisão final. O ensemble é treinado, testado e avaliado com métricas como AUC e relatório de classificação.

```
# Definir os modelos base
log_reg = LogisticRegression(max_iter=1000)
rf = RandomForestClassifier(random_state=42)
xgb = XGBClassifier(use_label_encoder=False, eval_metric='logloss', random_state=42)

# Criar ensemble com votação "soft" (probabilidades)
ensemble = VotingClassifier(
    estimators=[('lr', log_reg), ('rf', rf), ('xgb', xgb)],
    voting='soft'
)

# Treinar e avaliar
ensemble.fit(X_train, y_train)
y_pred = ensemble.predict(X_test)
y_proba = ensemble.predict_proba(X_test)[:,-1]

print("AUC Ensemble:", roc_auc_score(y_test, y_proba))
print(classification_report(y_test, y_pred))
```

```
AUC Ensemble: 0.9065727940798602
```

	precision	recall	f1-score	support
0	0.87	0.99	0.93	1276
1	0.88	0.23	0.36	244
accuracy			0.87	1520
macro avg	0.87	0.61	0.65	1520
weighted avg	0.87	0.87	0.84	1520

✓ Avaliação de Resultados:

Objetivo: analisar o desempenho dos modelos gerados em dados não vistos (com a base de teste)

- Selecione as métricas de avaliação condizentes com o problema, justificando.
- Treine o modelo escolhido com toda a base de treino, e teste-o com a base de teste.
- Os resultados fazem sentido?
- Foi observado algum problema de overfitting?
- Compare os resultados de diferentes modelos.
- Descreva a melhor solução encontrada, justificando.

✓ Treinamento final do modelo com treino + validação e avaliação no teste

```
# Exemplo com ensemble
y_val_pred = ensemble.predict(X_val_std)
y_val_proba = ensemble.predict_proba(X_val_std)[: ,1]

print("Relatório de Classificação (validação):")
print(classification_report(y_val, y_val_pred))

print("ROC-AUC:", roc_auc_score(y_val, y_val_proba))
```

```
Relatório de Classificação (validação):
```

	precision	recall	f1-score	support
0	0.00	0.00	0.00	1272
1	0.16	1.00	0.28	243
accuracy			0.16	1515
macro avg	0.08	0.50	0.14	1515
weighted avg	0.03	0.16	0.04	1515

```
ROC-AUC: 0.5035458239511349
```

Utiliza o ensemble treinado para fazer previsões no conjunto de validação. Exibe o relatório de classificação e a métrica ROC-AUC para avaliar o desempenho do modelo nessa etapa.

Junta treino + validação para treinar modelo final

```
X_final_train = np.vstack((X_train_std, X_val_std))
y_final_train = np.hstack((y_train, y_val))

final_model = ensemble
final_model.fit(X_final_train, y_final_train)

y_test_pred = final_model.predict(X_test_std)
y_test_proba = final_model.predict_proba(X_test_std)[: ,1]

print("Relatório de Classificação (teste):")
print(classification_report(y_test, y_test_pred))
print("ROC-AUC:", roc_auc_score(y_test, y_test_proba))
```

```
Relatório de Classificação (teste):
```

	precision	recall	f1-score	support
0	0.89	0.98	0.94	1276
1	0.82	0.38	0.52	244
accuracy			0.89	1520
macro avg	0.86	0.68	0.73	1520
weighted avg	0.88	0.89	0.87	1520

ROC-AUC: 0.9041895780872604

Avaliação da acurácia e verificação de overfitting

```
acc_train = accuracy_score(y_final_train, final_model.predict(X_final_train))
acc_test = accuracy_score(y_test, y_test_pred)

print("Acurácia Treino:", acc_train)
print("Acurácia Teste:", acc_test)

if acc_train - acc_test > 0.05:
    print("⚠ Indício de overfitting detectado.")
else:
    print("✅ Modelo generalizou bem.")
```

```
Acurácia Treino: 0.9976763099802486
Acurácia Teste: 0.8875
⚠ Indício de overfitting detectado.
```

Calculando a acurácia do modelo no conjunto de treino (final) e no conjunto de teste. Compara as duas para identificar possíveis indícios de overfitting, sinalizando se o

modelo está generalizando bem ou não.

✓ Treinamento rápido de modelos com configurações simplificadas

```
# Modelos reduzidos para acelerar
models_tuned_small = {
    "LogisticRegression": LogisticRegression(max_iter=1000, C=1.0, solver="lbfgs"),
    "RandomForest": RandomForestClassifier(n_estimators=100, max_depth=10, random_state=42)
}

# Treinar os modelos uma única vez
for name, model in models_tuned_small.items():
    model.fit(X_final_train, y_final_train)
```

Define versões otimizadas e mais leves dos modelos de Regressão Logística e Random Forest para acelerar o treinamento. Em seguida, treina cada modelo uma única vez com o conjunto final de treino.

✓ Treinamento e avaliação comparativa de modelos simplificados

```
results = {}

models_tuned_small = {
    "LogisticRegression": LogisticRegression(max_iter=1000, C=1.0, solver="lbfgs"),
    "RandomForest": RandomForestClassifier(n_estimators=100, max_depth=10, random_state=42)
}

# Treinar os modelos (fora do loop, se quiser acelerar)
for name, model in models_tuned_small.items():
    model.fit(X_final_train, y_final_train)

# Avaliar os modelos
for name, model in models_tuned_small.items():
    y_pred = model.predict(X_test_std)
    y_proba = model.predict_proba(X_test_std)[ :, 1]

    results[name] = {
        "Acurácia": accuracy_score(y_test, y_pred),
        "ROC-AUC": roc_auc_score(y_test, y_proba)
    }

# Se você tiver o ensemble treinado e quiser avaliar também:
```

```
# y_test_pred = ensemble.predict(X_test_std)
# y_test_proba = ensemble.predict_proba(X_test_std)[: , 1]
# results["Ensemble"] = {
#     "Acurácia": accuracy_score(y_test, y_test_pred),
#     "ROC-AUC": roc_auc_score(y_test, y_test_proba)
# }

df_results = pd.DataFrame(results).T
print(df_results.sort_values(by="ROC-AUC", ascending=False))
```

	Acurácia	ROC-AUC
RandomForest	0.839474	0.833573
LogisticRegression	0.851974	0.825107

Treina e avalia dois modelos ajustados, exibindo Acurácia e ROC-AUC em uma tabela para comparação.

✓ Divisão Treino/Teste

```
X = df.drop('default', axis=1)
y = df['default']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random
print('Treino:', X_train.shape, 'Teste:', X_test.shape)
```

Treino: (8101, 15) Teste: (2026, 15)

✓ Modelo Baseline (Logistic Regression)

```
# Detectar colunas numéricas e categóricas
numeric_features = X_train.select_dtypes(include=['int64', 'float64']).columns
categorical_features = X_train.select_dtypes(include=['object', 'category']).columns

# Pré-processamento
preprocessor = ColumnTransformer(transformers=[
    ('num', StandardScaler(), numeric_features),
    ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_features)
])

# Pipeline com pré-processamento + modelo
pipeline_lr = Pipeline([
    ('preprocessor', preprocessor),
    ('clf', LogisticRegression(random_state=42, max_iter=1000))
])

# Treinar
pipeline_lr.fit(X_train, y_train)

# Predizer e avaliar
```

```

y_pred = pipeline_lr.predict(X_test)
y_proba = pipeline_lr.predict_proba(X_test)[:, 1]

print(classification_report(y_test, y_pred))
print('AUC:', roc_auc_score(y_test, y_proba))

```

	precision	recall	f1-score	support
0	0.89	0.97	0.92	1701
1	0.67	0.34	0.45	325
accuracy			0.87	2026
macro avg	0.78	0.65	0.69	2026
weighted avg	0.85	0.87	0.85	2026

AUC: 0.8597874553430109

✓ Modelo Avançado (Random Forest + GridSearchCV)

```

# Separa colunas numéricas e categóricas
numeric_features = X_train.select_dtypes(include=['int64', 'float64']).columns
categorical_features = X_train.select_dtypes(include=['object', 'category']).columns

# Pré-processador para tratar as colunas corretamente
preprocessor = ColumnTransformer([
    ('num', StandardScaler(), numeric_features),
    ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_features)
])

# Pipeline com pré-processamento + modelo
pipeline_rf = Pipeline([
    ('preprocessor', preprocessor),
    ('clf', RandomForestClassifier(random_state=42))
])

# Hiperparâmetros para o GridSearch
param_grid = {
    'clf__n_estimators': [100, 200],
    'clf__max_depth': [5, 10, None],
    'clf__min_samples_split': [2, 5]
}

# Executar o GridSearchCV
grid = GridSearchCV(pipeline_rf, param_grid, cv=3, scoring='roc_auc', n_jobs=-1)

# Treinar
grid.fit(X_train, y_train)

# Resultados
print('✓ Melhores parâmetros:', grid.best_params_)
print('✓ Melhor AUC (validação):', round(grid.best_score_, 4))

```

```
# Avaliação no conjunto de teste
y_pred_rf = grid.predict(X_test)
print('\n📊 Classification Report:')
print(classification_report(y_test, y_pred_rf))

# AUC no teste
y_proba_rf = grid.predict_proba(X_test)[: , 1]
print('🎯 AUC Teste:', round(roc_auc_score(y_test, y_proba_rf), 4))
```

✓ Melhores parâmetros: {'clf__max_depth': None, 'clf__min_samples_split': 5, 'clf__min_samples_leaf': 5}

✓ Melhor AUC (validação): 0.9114

📊 Classification Report:

	precision	recall	f1-score	support
0	0.87	0.99	0.93	1701
1	0.89	0.23	0.37	325
accuracy			0.87	2026
macro avg	0.88	0.61	0.65	2026
weighted avg	0.88	0.87	0.84	2026

🎯 AUC Teste: 0.8971

✓ Conclusão

Desenvolver modelo preditivo para identificar inadimplência de clientes com base em variáveis financeiras e comportamentais.

Hipótese: Renda, idade e limite de crédito são determinantes para o risco de inadimplência.

Metodologia:

Algoritmos testados: Regressão Logística, Árvore de Decisão, Random Forest, Gradient