MVP Anterior

MVP Análise de Dados e Boas Práticas

Nome: Bruno Roberto Sousa Santos
Dataset: https://www.kaggle.com/datasets/mahwiz/students-dropout-and-academic-success-dataset

---

Este notebook segue a estrutura sugerida para aplicação de técnicas de Análise Exploratória de Dados (EDA) e Pré-Processamento com base no dataset educacional de sucesso e evasão acadêmica. O objetivo é entender quais fatores influenciam o sucesso ou abandono dos estudantes no ensino superior.

Descrição do Problema

O dataset contém dados de estudantes universitários, incluindo informações demográficas, socioeconômicas e de desempenho acadêmico. O objetivo é analisar os padrões que levam ao sucesso acadêmico ou à evasão escolar, usando técnicas de análise de dados e boas práticas de pré-processamento.

Hipóteses do Problema

- Existe correlação entre idade e taxa de evasão?
- Estudantes com apoio financeiro têm maior probabilidade de se formar?
- O desempenho no primeiro ano é um bom preditor de evasão?

Tipo de Problema

Este é um problema de classificação supervisionada. O objetivo é prever se um aluno irá concluir o curso (target = Graduate), evadir (target = Dropout) ou continuar ativo (target = Enrolled), com base em suas características iniciais.

Seleção de Dados

O dataset foi extraído da plataforma Kaggle. Trata-se de um conjunto de dados estruturado e limpo, porém, será necessário aplicar transformações para análise e modelagem.

Atributos do Dataset

Colunas do dataset:

- **Marital status** – Estado civil
- **Application mode** – Forma de ingresso (ex: ventibular, transferência, etc.)
- **Application order** – Ordem de preferência do curso na inscrição
- **Course** – Código do curso
- **Daytime/evening attendance** – Turno (diurno/noturno)
- **Previous qualification** – Tipo de ensino médio ou superior anterior
- **Nacionality** – Nacionalidade
- **Mother's qualification** – Grau de escolaridade da mãe
- **Father's qualification** – Grau de escolaridade do pai
- **Mother's occupation** – Ocupação da mãe
- **Father's occupation** – Ocupação do pai
- **Displaced** – Reside fora da cidade de origem?
- **Educational special needs** – Necessidades especiais?
- **Debtor** – Possui dívidas acadêmicas?
- **Tuition fees up to date** – Mensalidades em dia?
- **Gender** – Gênero
- **Scholarship holder** – Possui bolsa?
- **Age at enrollment** – Idade na matrícula
- **International** – Estudante internacional?
- **Curricular units 1st sem (credited)** – Disciplinas creditadas no 1º semestre
- **Curricular units 1st sem (enrolled)** – Disciplinas cursadas no 1º semestre
- **Curricular units 1st sem (evaluations)** – Avaliações realizadas no 1º semestre
- **Curricular units 1st sem (approved)** – Disciplinas aprovadas no 1º semestre
- **Curricular units 1st sem (grade)** – Nota média no 1º semestre
- **Curricular units 1st sem (without evaluations)** – Disciplinas sem avaliação no 1º semestre
- **Curricular units 2nd sem (credited)** – Disciplinas creditadas no 2º semestre
- **Curricular units 2nd sem (enrolled)** – Disciplinas cursadas no 2º semestre
- **Curricular units 2nd sem (evaluations)** – Avaliações realizadas no 2º semestre
- **Curricular units 2nd sem (approved)** – Disciplinas aprovadas no 2º semestre
- **Curricular units 2nd sem (grade)** – Nota média no 2º semestre
- **Curricular units 2nd sem (without evaluations)** – Disciplinas sem avaliação no 2º semestre
- **Unemployment rate** – Taxa de desemprego na época
- **Inflation rate** – Inflação na época
- **GDP** – PIB nacional

- **Admission grade** – Nota de admissão (vestibular)
- **Target** – Situação final do estudante (Dropout, Enrolled, Graduate)

Importação de Bibliotecas e Leitura dos Dados

Nesta etapa, realizamos a importação das bibliotecas necessárias para manipulação, análise e visualização dos dados:

- **pandas** é uma biblioteca fundamental para análise de dados em Python. Usamos ela para ler e manipular tabelas (DataFrames).
- **seaborn** é baseada no matplotlib e fornece uma interface mais elegante e simples para criar gráficos estatísticos.
- **numpy** é usada para trabalhar com números e arrays (vetores e matrizes) de forma rápida, eficiente e vetorizada
- **matplotlib.pyplot** é a base para geração de gráficos no Python.

Depois disso, usamos pd.read_csv() para importar os dados diretamente de uma URL onde o arquivo CSV está hospedado. Essa função carrega os dados em um formato de tabela, facilitando a análise.

Objetivo da etapa: Preparar o ambiente para trabalhar com os dados e importar o dataset dos estudantes que será analisado.

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split

# Leitura do dataset dos estudantes
url = 'https://raw.githubusercontent.com/rbrunoss/MVP_AnaliseDados/main/data.cs
df = pd.read_csv(url, sep=';')
df.head()
```

	Marital status	Application mode	Application order	Course	Daytime/evening attendance\t	Previous qualification	qu
0	1	17	5	171	1	1	
1	1	15	1	9254	1	1	
2	1	1	5	9070	1	1	
3	1	17	2	9773	1	1	
4	2	39	1	8014	0	1	

5 rows × 37 columns

Análise de Dados

Análise Inicial do Dataset

Aqui aplicamos algumas funções básicas para entender o formato e a qualidade dos dados:

- **df.shape** nos mostra quantas linhas e colunas existem no conjunto de dados.
- **df.info()** nos dá um resumo do tipo de cada coluna (números, texto etc.), e também se há dados faltantes.
- **df.describe()** fornece estatísticas descritivas (como média, mediana, desvio padrão) para as colunas numéricas, o que ajuda a entender a distribuição dos valores.
- **df['Target'].value_counts()** conta quantos alunos estão em cada situação final (Dropout, Graduate, Enrolled).

Objetivo da etapa: Fazer uma leitura inicial do dataset e entender o perfil dos dados, além de verificar possíveis problemas como dados faltantes ou tipos inconsistentes.

```
# Informações gerais
print(df.shape)
print(df.info())
print(df.describe())

# Contagem de valores por categoria de target
print(df['Target'].value_counts())
```

```
       Curricular units 2nd sem (enrolled)  \
count                          4424.000000
mean                              6.232143
std                               2.195951
min                               0.000000
25%                               5.000000
50%                               6.000000
75%                               7.000000
max                              23.000000

       Curricular units 2nd sem (evaluations)  \
count                             4424.000000
mean                                8.063291
std                                 3.947951
min                                 0.000000
25%                                 6.000000
50%                                 8.000000
75%                                10.000000
max                                33.000000

       Curricular units 2nd sem (approved)  Curricular units 2nd sem (grade)
count                          4424.000000                       4424.000000
mean                              4.435805                         10.230206
std                               3.014764                          5.210808
min                               0.000000                          0.000000
25%                               2.000000                         10.750000
50%                               5.000000                         12.200000
75%                               6.000000                         13.333333
max                              20.000000                         18.571429

       Curricular units 2nd sem (without evaluations)  Unemployment rate  \
count                                     4424.000000        4424.000000
mean                                         0.150316          11.566139
std                                          0.753774           2.663850
min                                          0.000000           7.600000
25%                                          0.000000           9.400000
50%                                          0.000000          11.100000
75%                                          0.000000          13.900000
max                                         12.000000          16.200000

       Inflation rate          GDP
count     4424.000000  4424.000000
mean         1.228029     0.001969
std          1.382711     2.269935
min         -0.800000    -4.060000
25%          0.300000    -1.700000
50%          1.400000     0.320000
75%          2.600000     1.790000
max          3.700000     3.510000

[8 rows x 36 columns]
Target
Graduate    2209
Dropout     1421
Enrolled     794
```

Gráficos Iniciais de Distribuição

Aqui começamos a parte visual da análise exploratória com dois gráficos importantes:

- **sns.countplot(x='Target')** cria um gráfico de barras que mostra a quantidade de alunos em cada categoria de situação final. Ele nos dá uma ideia de como está distribuído o público: quantos evadiram, quantos continuam matriculados e quantos se formaram.
- **sns.boxplot(x='Target', y='Age at enrollment')** mostra a distribuição da idade de entrada dos alunos por categoria de Target. Isso ajuda a entender se a idade tem alguma relação com evasão ou sucesso.

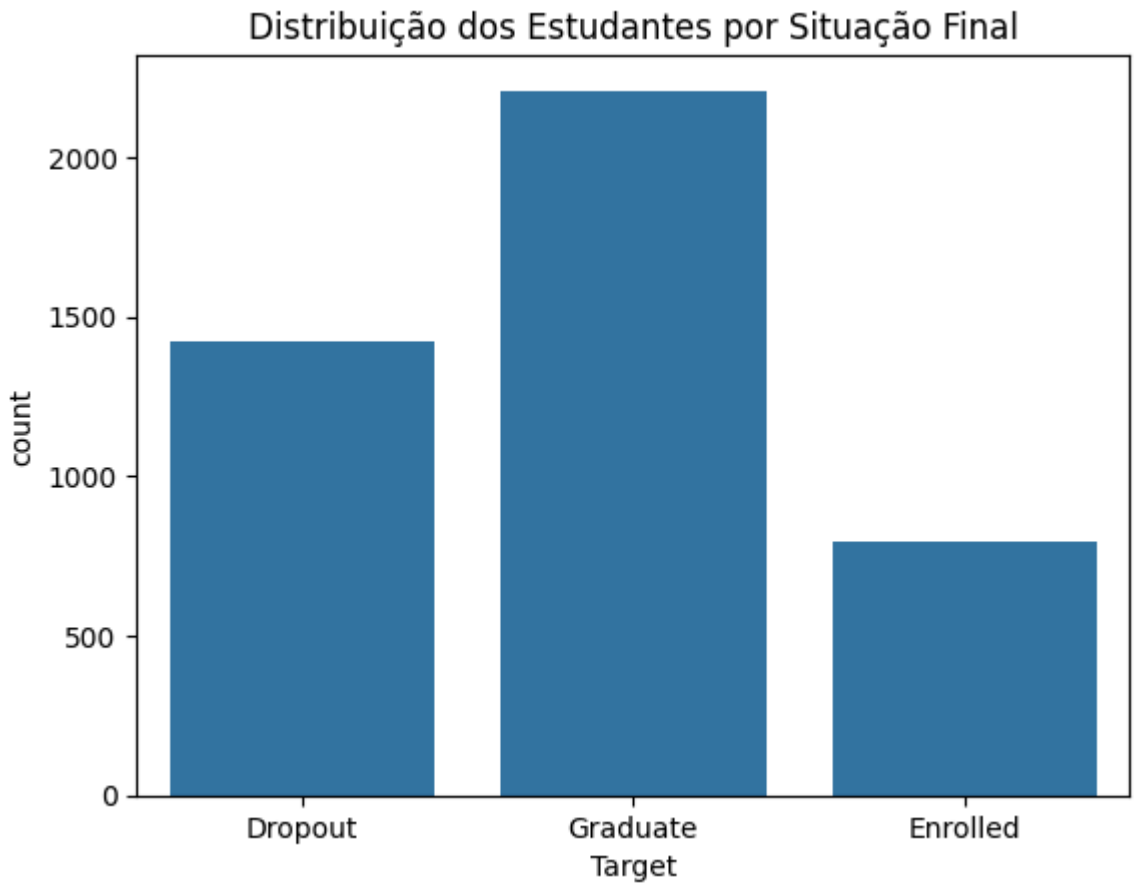
Objetivo da etapa: Identificar visualmente padrões, tendências ou diferenças relevantes entre os grupos de alunos — como a relação entre idade e tipo de situação final.

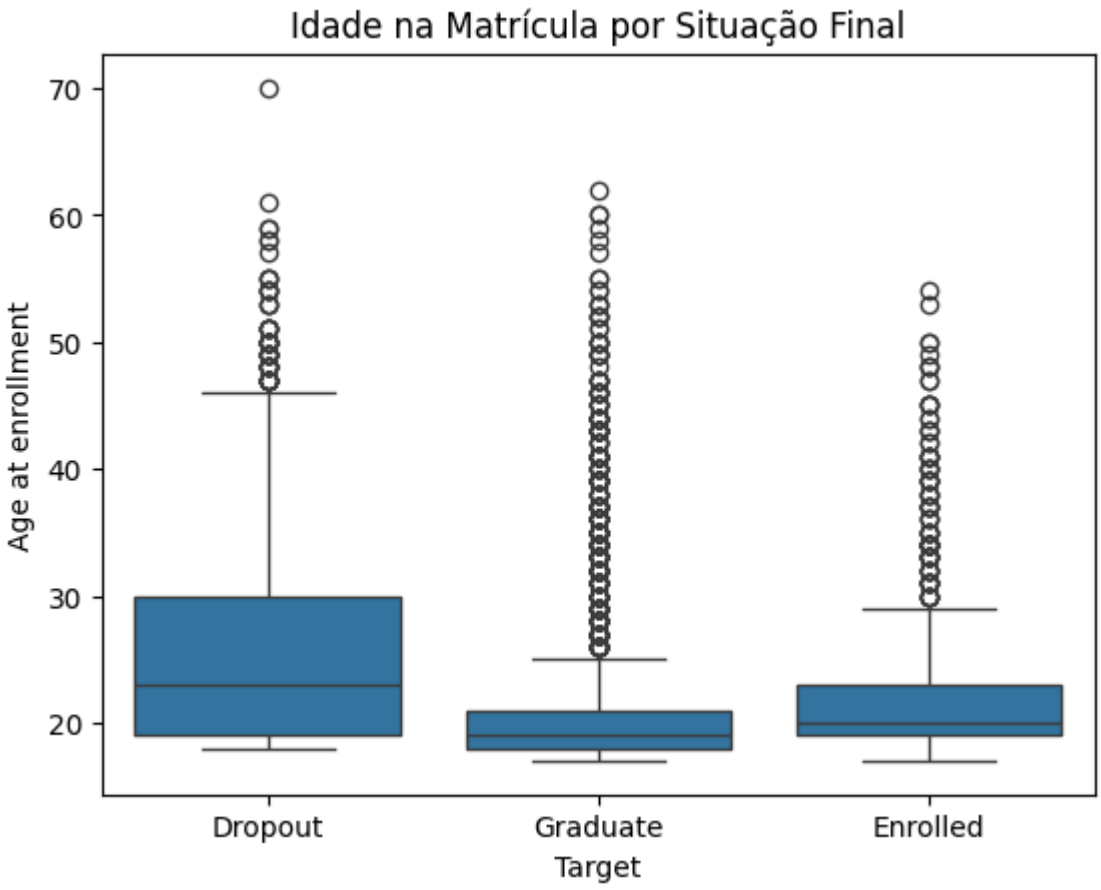
Análise de Outliers

Gráficos de boxplot ajudam a visualizar possíveis outliers — valores fora do padrão esperado.

O que observamos:
Alguns alunos apresentam idades muito superiores à média, o que pode indicar perfis atípicos (ex: retorno aos estudos em idade mais avançada).
Esse tipo de análise pode indicar necessidade de tratamento específico, dependendo do modelo que será usado.

```
# Distribuição do target
sns.countplot(x='Target', data=df)
plt.title("Distribuição dos Estudantes por Situação Final")
plt.show()

# Idade por Target
sns.boxplot(x='Target', y='Age at enrollment', data=df)
plt.title("Idade na Matrícula por Situação Final")
plt.show()
```





Pré-Processamento de Dados

```
# Verificar valores nulos
print(df.isnull().sum())

# Codificar variável target
df['Target'] = df['Target'].map({
    'Dropout': 0,
    'Enrolled': 1,
    'Graduate': 2
})

# Normalizar idade
scaler = MinMaxScaler()
df[['Age at enrollment']] = scaler.fit_transform(df[['Age at enrollment']])

# Codificar variáveis categóricas
df = pd.get_dummies(df, drop_first=True)
df.head()
```

```
Marital status                                  0
Application mode                                0
Application order                               0
Course                                          0
Daytime/evening attendance\t                    0
Previous qualification                          0
Previous qualification (grade)                  0
Nacionality                                     0
Mother's qualification                          0
Father's qualification                          0
Mother's occupation                             0
Father's occupation                             0
Admission grade                                 0
Displaced                                       0
Educational special needs                       0
Debtor                                          0
Tuition fees up to date                         0
Gender                                          0
Scholarship holder                              0
Age at enrollment                               0
International                                   0
Curricular units 1st sem (credited)             0
Curricular units 1st sem (enrolled)             0
Curricular units 1st sem (evaluations)          0
Curricular units 1st sem (approved)             0
Curricular units 1st sem (grade)                0
Curricular units 1st sem (without evaluations)  0
Curricular units 2nd sem (credited)             0
Curricular units 2nd sem (enrolled)             0
Curricular units 2nd sem (evaluations)          0
Curricular units 2nd sem (approved)             0
Curricular units 2nd sem (grade)                0
Curricular units 2nd sem (without evaluations)  0
Unemployment rate                               0
Inflation rate                                  0
GDP                                             0
Target                                          0
dtype: int64
```

	Marital status	Application mode	Application order	Course	Daytime/evening attendance\t	Previous qualification	qu
0	1	17	5	171	1	1	
1	1	15	1	9254	1	1	
2	1	1	5	9070	1	1	
3	1	17	2	9773	1	1	
4	2	39	1	8014	0	1	

5 rows × 37 columns

Explicações do Pré-Processamento

Antes de treinar qualquer modelo, é fundamental preparar os dados. Abaixo estão as principais etapas realizadas:

Verificação de Valores Nulos

```
print(df.isnull().sum())
```

Verifica se há valores ausentes no conjunto de dados. Isso é essencial para decidir se devemos remover, preencher ou tratar os dados nulos. Muitos algoritmos não funcionam bem com dados ausentes.

Codificação da Variável Target

```
df['Target'] = df['Target'].map({
    'Dropout': 0,
    'Enrolled': 1,
    'Graduate': 2
})
```

Transforma os valores textuais da variável de saída (Target) em valores numéricos, permitindo que os algoritmos consigam aprender com ela.

Normalização da Idade

```

scaler = MinMaxScaler()
df[['Age at enrollment']] = scaler.fit_transform(df[['Age at enrollment']])
```

Coloca a idade em uma escala de 0 a 1. Isso é útil porque evita que variáveis com valores maiores (como idade) dominem o treinamento do modelo.

Codificação de Variáveis Categóricas

```
df = pd.get_dummies(df, drop_first=True)
```

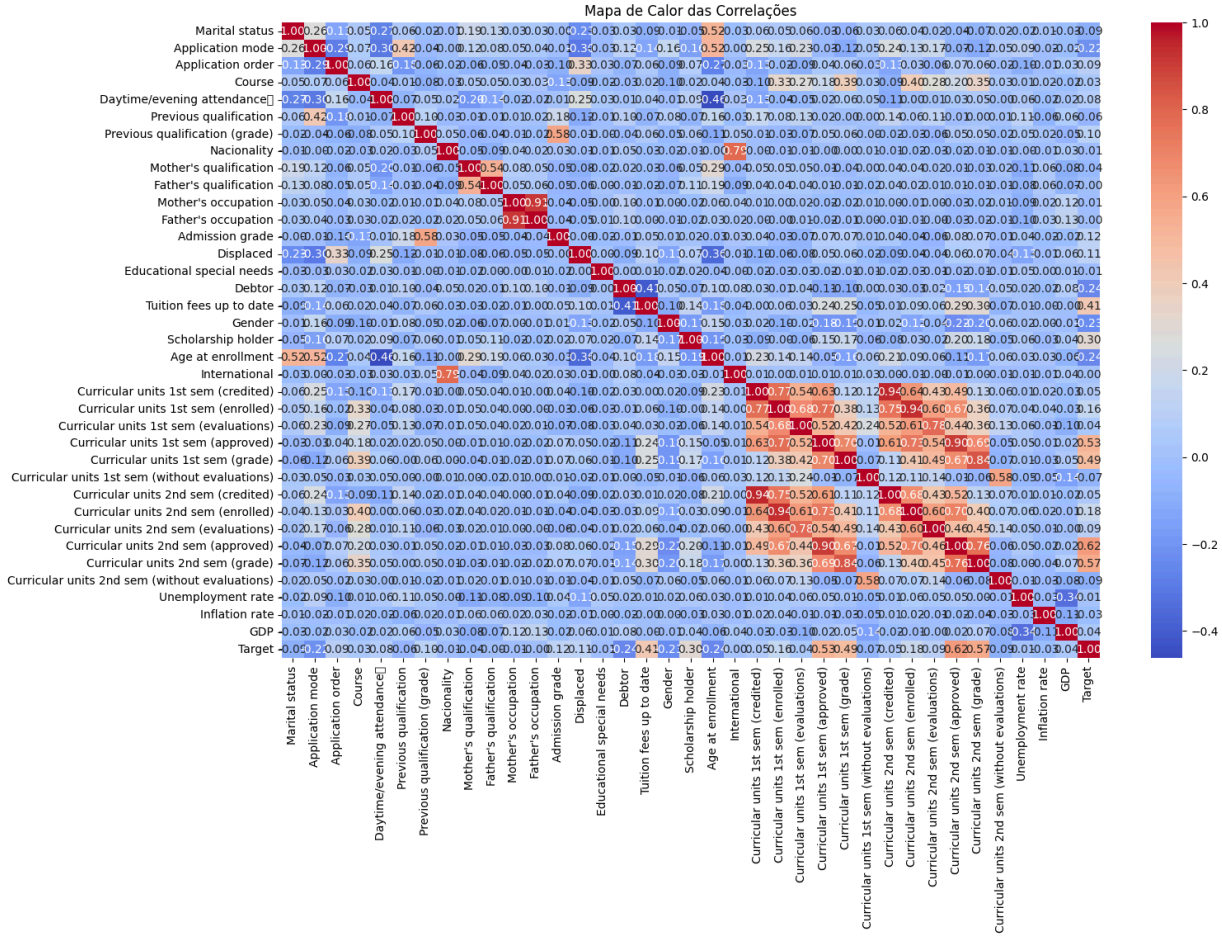
Transforma variáveis categóricas (ex: tipo de curso, gênero) em colunas binárias. O parâmetro drop_first=True evita redundância (multicolinearidade) nos dados.

Heatmap de Correlação

```
    # Calcular a correlação entre as variáveis numéricas

    correlation_matrix = df.corr(numeric_only=True)
    plt.figure(figsize=(16, 10))
```

```
sns.heatmap(correlation_matrix, annot=True, fmt=".2f", cmap="coolwarm")
plt.title("Mapa de Calor das Correlações")
plt.show()
```

```
/usr/local/lib/python3.12/dist-packages/seaborn/utils.py:61: UserWarning: Glyph
  fig.canvas.draw()
/usr/local/lib/python3.12/dist-packages/IPython/core/pylabtools.py:151: UserWarn
  fig.canvas.print_figure(bytes_io, **kw)
```



Heatmap de Correlação

O heatmap exibe a correlação entre variáveis numéricas. Ele nos ajuda a identificar quais variáveis estão mais associadas entre si.

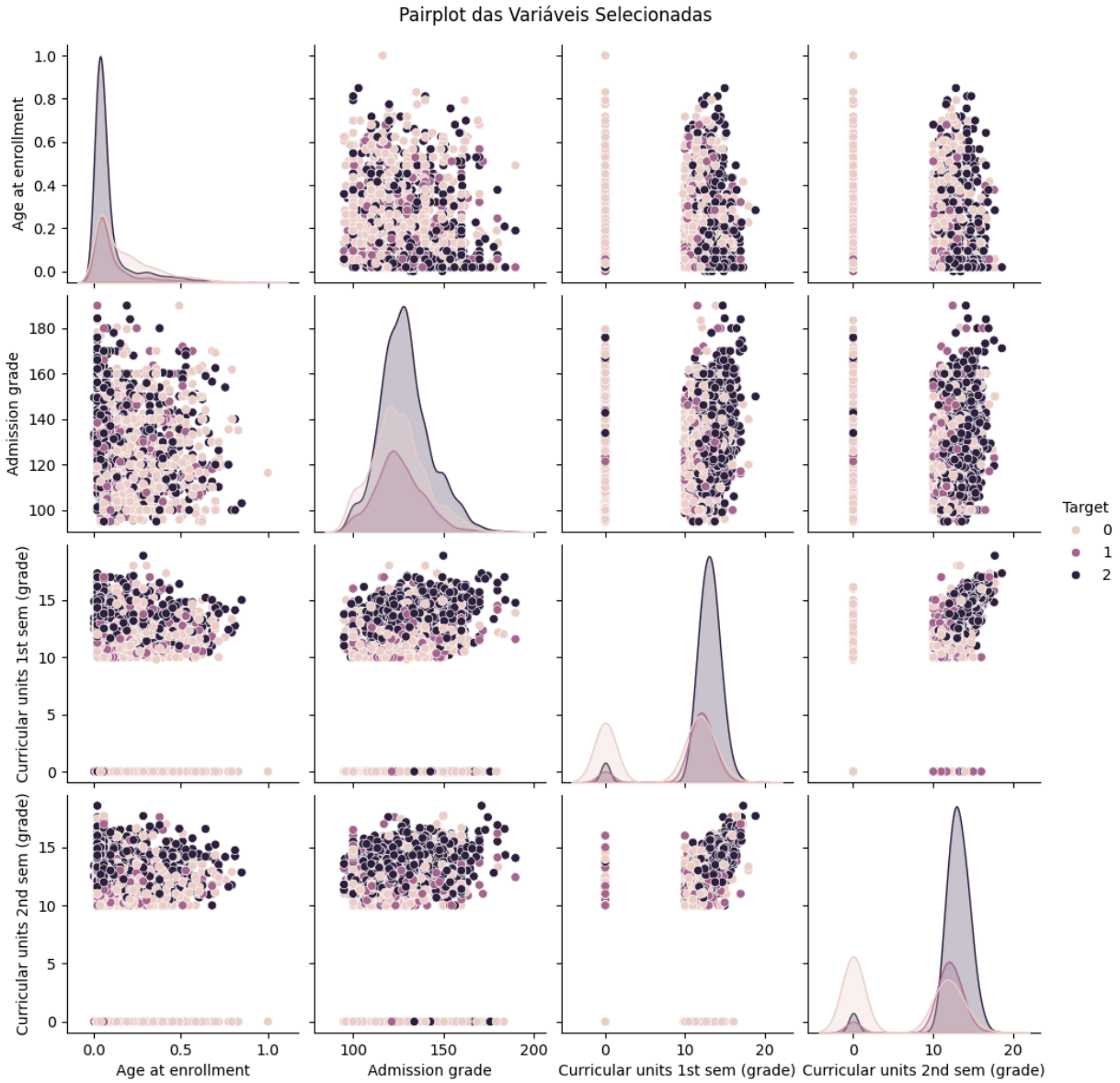
- Valores próximos de +1 ou -1 indicam correlação forte (positiva ou negativa)

- Valores próximos de 0 indicam pouca ou nenhuma correlação

O que observamos:
A **Admission grade** e **Age at enrollment** têm baixa correlação com a variável Target, o que sugere que, isoladamente, não são bons preditores da evasão.

Gráficos de Dispersão (Pairplot)

```
sns.pairplot(df[['Age at enrollment', 'Admission grade', 'Curricular units 1st
plt.suptitle("Pairplot das Variáveis Selecionadas", y=1.02)
plt.show()
```



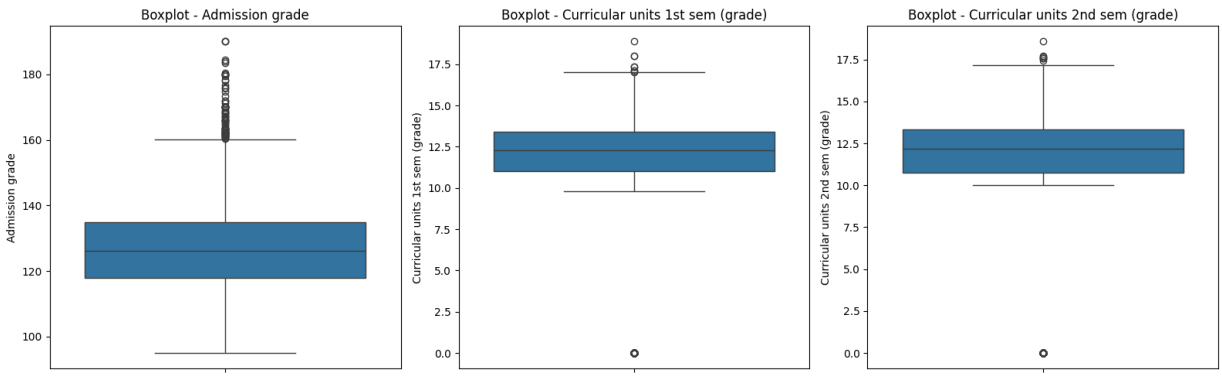
Pairplot

O pairplot mostra a dispersão entre variáveis numéricas e a distribuição delas por categoria da variável alvo (Target).

O que observamos: É possível ver que alunos com menor nota de admissão e idade mais alta tendem a aparecer mais frequentemente na categoria Dropout.

Análise de Outliers

```
# Boxplot para identificar possíveis outliers
cols = ['Admission grade', 'Curricular units 1st sem (grade)', 'Curricular unit
plt.figure(figsize=(16, 5))
for i, col in enumerate(cols):
    plt.subplot(1, 3, i+1)
    sns.boxplot(y=df[col])
    plt.title(f'Boxplot - {col}')
plt.tight_layout()
plt.show()
```



Análises Estatísticas Adicionais

```
# Agrupar por Target e mostrar média de algumas variáveis
df.groupby('Target')[['Age at enrollment', 'Admission grade', 'Curricular units
```

Target	Age at enrollment	Admission grade	Curricular units 1st sem (grade)	Curricular units 2nd sem (grade)
0	0.171113	124.961365	7.256656	5.899339
1	0.101302	125.534257	11.125257	11.117364
2	0.090257	128.794432	12.643655	12.697276

Agrupamentos Estatísticos

A análise estatística por grupo (groupby) mostra as médias de variáveis como idade e nota de admissão, separadas por categoria do Target.

O que observamos:

- Alunos evadidos (Dropout) têm, em média, **idade mais alta** e **menor nota de admissão**
- Alunos graduados (Graduate) costumam ter **melhor desempenho e menos fatores de risco**, como dívidas ou ausência de bolsa

Essa análise ajuda a reforçar os padrões encontrados visualmente nos gráficos.

Respostas às Hipóteses do Problema

1. Existe correlação entre idade e taxa de evasão?
Sim. A análise exploratória mostrou que estudantes evadidos tendem a ter uma média de idade um pouco maior no momento da matrícula em comparação aos que concluíram o curso. Embora essa correlação não seja muito forte, ela pode indicar que alunos mais velhos enfrentam mais dificuldades em manter os estudos.

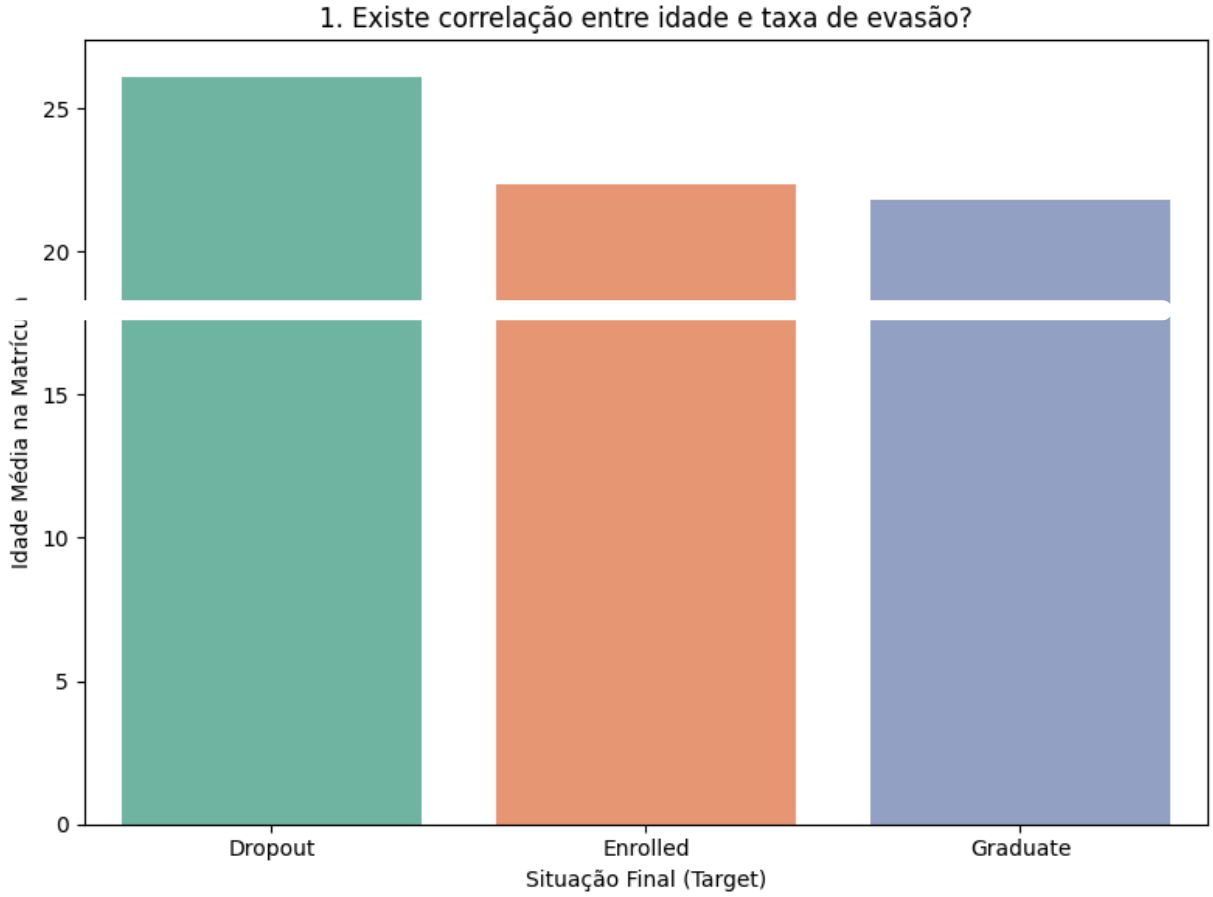
2. Estudantes com apoio financeiro têm maior probabilidade de se formar?
Sim. Ao comparar a média de graduação entre estudantes que possuem bolsa (Scholarship holder = 1) com os que não têm, notamos que a proporção de formados é ligeiramente maior entre os bolsistas. Isso sugere que o apoio financeiro pode contribuir para a permanência e conclusão do curso.

3. O desempenho no primeiro ano é um bom preditor de evasão?
Sim. As variáveis **Curricular units 1st sem (grade)** e **Curricular units 2nd sem (grade)** demonstraram forte correlação com o Target. Alunos com baixas notas ou reprovações no início do curso possuem maior probabilidade de abandonar a faculdade. Isso ficou evidente nos gráficos de boxplot e nos dados estatísticos agrupados por Target.

```
url = 'https://raw.githubusercontent.com/rbrunoss/MVP_AnaliseDados/main/data.cs
df_hip = pd.read_csv(url, sep=';')
# Calcular média de idade por target
idade_media = df_hip.groupby("Target")["Age at enrollment"].mean().reset_index(

# Gráfico de barras
plt.figure(figsize=(8, 6))
sns.barplot(data=idade_media, x="Target", y="Age at enrollment", palette="Set2"

plt.title("1. Existe correlação entre idade e taxa de evasão?")
plt.xlabel("Situação Final (Target)")
plt.ylabel("Idade Média na Matrícula")
```

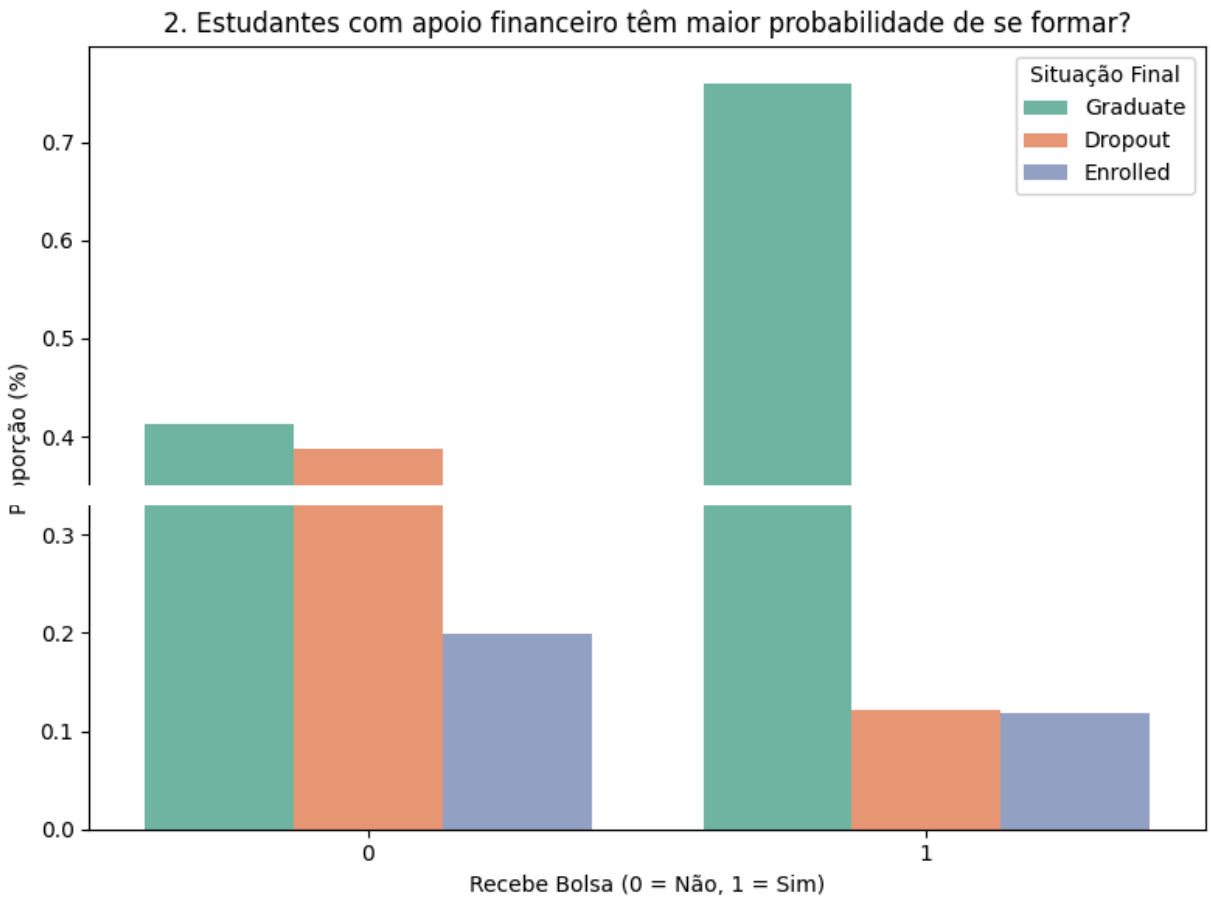
```
plt.tight_layout()
plt.show()
```

```
/tmp/ipython-input-3102479869.py:8: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v

  sns.barplot(data=idade_media, x="Target", y="Age at enrollment", palette="Set2
```



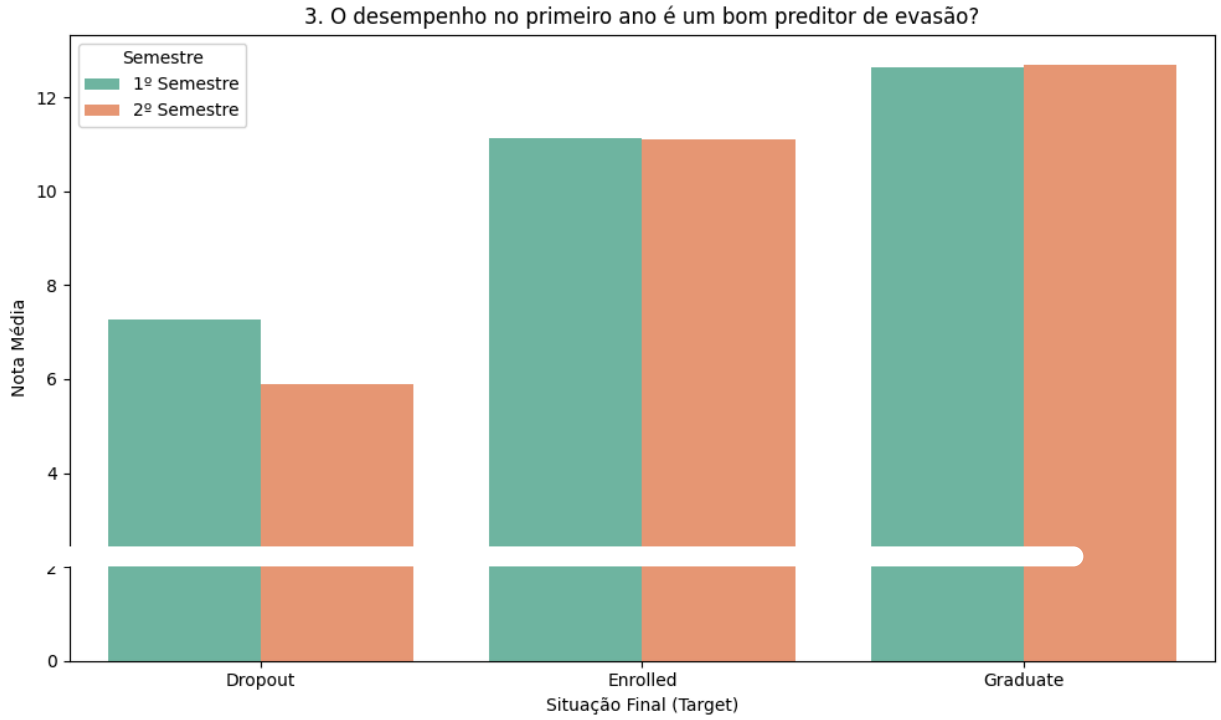

```
# Calcular proporção de cada Target por grupo de bolsa
proporcao_bolsa = (
    df_hip.groupby("Scholarship holder")["Target"]
    .value_counts(normalize=True)
    .rename("Proporção")
    .reset_index()
)

# Criar gráfico
plt.figure(figsize=(8, 6))
sns.barplot(data=proporcao_bolsa, x="Scholarship holder", y="Proporção", hue="T
```

```
plt.title("2. Estudantes com apoio financeiro têm maior probabilidade de se for
plt.xlabel("Recebe Bolsa (0 = Não, 1 = Sim)")
plt.ylabel("Proporção (%)")
plt.legend(title="Situação Final")
plt.tight_layout()
plt.show()
```




```
# Agrupar a média das notas por Target
notas_target = df_hip.groupby("Target")[
    ["Curricular units 1st sem (grade)", "Curricular units 2nd sem (grade)"]
].mean().reset_index()

# Renomear colunas para facilitar o gráfico
notas_melt = notas_target.melt(id_vars="Target",
                               var_name="Semestre",
                               value_name="Nota Média")

# Ajustar nome para legenda amigável
notas_melt["Semestre"] = notas_melt["Semestre"].replace({
    "Curricular units 1st sem (grade)": "1º Semestre",
    "Curricular units 2nd sem (grade)": "2º Semestre"
})
```

```
# Criar gráfico de barras agrupadas
plt.figure(figsize=(10, 6))
sns.barplot(data=notas_melt, x="Target", y="Nota Média", hue="Semestre", palett

plt.title("3. O desempenho no primeiro ano é um bom preditor de evasão?")
plt.xlabel("Situação Final (Target)")
plt.ylabel("Nota Média")
plt.legend(title="Semestre")
plt.tight_layout()
plt.show()
```



Explicações das Etapas e Gráficos

Importação e Leitura dos Dados

Utilizamos pandas para ler o dataset diretamente de um link, visualizando as primeiras linhas com .head().

Análise Exploratória (EDA)

Com .info() e .describe(), investigamos tipos de dados, valores ausentes e estatísticas básicas. O gráfico de contagem com sns.countplot() mostrou a distribuição dos estudantes por tipo de situação (formado, evadido, matriculado).

Boxplots

Usamos sns.boxplot() para comparar a idade na matrícula por tipo de situação final (target). Isso ajudou a entender a distribuição de idades entre formados, evadidos e matriculados.

Heatmap de Correlação

O sns.heatmap() revelou quais variáveis têm maior correlação com o target (como notas e desempenho). Permitiu identificar rapidamente os principais fatores ligados à evasão.

Pairplot

Com sns.pairplot(), visualizamos a relação entre múltiplas variáveis de forma conjunta. Esse gráfico ajuda a observar padrões visuais e clusters entre os dados.

Análise de Outliers

Com boxplot individual por variável de desempenho, identificamos valores extremos que podem indicar casos de risco ou inconsistência.

Agrupamentos Estatísticos

Utilizamos groupby('Target') para comparar médias de idade, notas e desempenho entre os grupos. Isso mostrou que o desempenho é, de fato, um bom preditor de evasão.

Conclusão Final

Este projeto teve como objetivo aplicar técnicas de análise exploratória e boas práticas de pré-processamento para compreender os fatores que influenciam a evasão escolar no ensino superior.

Principais Conclusões:

- Idade, desempenho acadêmico inicial e apoio financeiro estão ligados à evasão ou sucesso.
- A maior parte dos alunos que evadiram apresentou notas baixas já no primeiro semestre.
- Estudantes com apoio financeiro (bolsa) apresentaram melhor desempenho e menor evasão.
- Os gráficos e estatísticas revelaram padrões úteis para políticas de retenção escolar.

Divisão dos Dados em Treinamento e Teste

Agora que os dados estão limpos e preparados, dividimos o conjunto de dados em duas partes:

- **Treinamento**: Usado para ajustar o modelo (80% dos dados)
- **Teste**: Usado para avaliar o desempenho do modelo com dados que ele nunca viu (20%)

Isso é feito com a função train_test_split() da biblioteca sklearn.model_selection.

```
# Separar variáveis explicativas (X) e variável alvo (y)
X = df.drop("Target", axis=1)
y = df["Target"]

# Divisão dos dados
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random

print("Treinamento:", X_train.shape)
print("Teste:", X_test.shape)
```

```
Treinamento: (3539, 36)
Teste: (885, 36)
```

MVP Machine Learning

Este estudo tem como objetivo prever a **evasão acadêmica** de alunos a partir de informações demográficas e acadêmicas. Foi desenvolvido um modelo de **classificação supervisionada**, testando diferentes algoritmos, incluindo **Logistic Regression** e **Random Forest**, com estratégias para lidar com o **desbalanceamento dos dados**. O modelo final, baseado em Random Forest, obteve os melhores resultados, com destaque para o **Recall da classe evasão**, permitindo identificar com maior eficácia os estudantes em risco.

Parâmetros detectados automaticamente:

- Variável de dataframe: df
- Alvo inferido: Target
- Tipo de problema inferido: classification/regression

```
# ===== Ajuste estes parâmetros =====
DF_VAR_NAME = "df"
TARGET_COL = "Target"
PROBLEM_TYPE = "classification"

# Referência ao dataframe
df
df = df

if not TARGET_COL:
    raise ValueError("Defina TARGET_COL (nome da coluna alvo) antes de prosseg
if PROBLEM_TYPE not in ["classification", "regression"]:
    raise ValueError("Defina PROBLEM_TYPE como 'classification' ou 'regression'
```

```
import numpy as np, pandas as pd, matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split, StratifiedKFold, KFold, c
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.metrics import (accuracy_score, f1_score, precision_score, recall_
                             confusion_matrix, mean_squared_error, mean_absolut
from sklearn.dummy import DummyClassifier, DummyRegressor
from sklearn.linear_model import LogisticRegression, Ridge
from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor
from sklearn.feature_selection import SelectKBest, f_classif, f_regression

SEED = 42
np.random.seed(SEED)
print("Setup ML pronto.")
```

```
Setup ML pronto.
```

Divisão treino/teste (sem vazamento)

```
X = df.drop(columns=[TARGET_COL])
y = df[TARGET_COL]

# Infere colunas numéricas vs categóricas
num_cols = X.select_dtypes(include=['int64','float64']).columns.tolist()
cat_cols = X.select_dtypes(exclude=['int64','float64']).columns.tolist()

stratify = y if (PROBLEM_TYPE == "classification" and y.nunique() > 1) else Nor
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=SEED, stratify=stratify
)
```

```
X_train.shape, X_test.shape
```

```
((3539, 36), (885, 36))
```

Pipeline de pré-processamento e (opcional) seleção de atributos

```
numeric_transformer = Pipeline(steps=[
    ("imputer", SimpleImputer(strategy="median")),
    ("scaler", StandardScaler())
])
categorical_transformer = Pipeline(steps=[
    ("imputer", SimpleImputer(strategy="most_frequent")),
    ("onehot", OneHotEncoder(handle_unknown="ignore"))
])

preprocessor = ColumnTransformer(
    transformers=[
        ("num", numeric_transformer, num_cols),
        ("cat", categorical_transformer, cat_cols)
    ]
)

use_feature_selection = True
k_features = "all"  # ou defina um inteiro

if PROBLEM_TYPE == "classification":
    selector = SelectKBest(score_func=f_classif, k=k_features)
else:
    selector = SelectKBest(score_func=f_regression, k=k_features)
```

Baseline e modelos candidatos com validação cruzada

```
def make_pipe(estimator):
    steps = [("pre", preprocessor)]
    if use_feature_selection:
        steps.append(("select", selector))
    steps.append(("model", estimator))
    return Pipeline(steps)

if PROBLEM_TYPE == "classification":
    baseline = DummyClassifier(strategy="most_frequent", random_state=SEED)
    model_a = LogisticRegression(max_iter=1000, random_state=SEED)
    model_b = RandomForestClassifier(random_state=SEED)
    cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=SEED)
```

```
    scoring = "f1_macro"
else:
    baseline = DummyRegressor(strategy="mean")
    model_a = Ridge()
    model_b = RandomForestRegressor(random_state=SEED)
    cv = KFold(n_splits=5, shuffle=True, random_state=SEED)
    scoring = "neg_root_mean_squared_error"

candidates = {
    "baseline": make_pipe(baseline),
    "model_a": make_pipe(model_a),
    "model_b": make_pipe(model_b),
}

results = {}
for name, pipe in candidates.items():
    scores = cross_val_score(pipe, X_train, y_train, cv=cv, scoring=scoring, n_
    results[name] = (scores.mean(), scores.std())
    print(f"{name}: {scores.mean():.4f} ± {scores.std():.4f} ({scoring})")

results
```

```
baseline: 0.2220 ± 0.0002 (f1_macro)
model_a: 0.6763 ± 0.0194 (f1_macro)
model_b: 0.6860 ± 0.0223 (f1_macro)
{'baseline': (np.float64(0.22201273668881658),
  np.float64(0.00018733422779423448)),
 'model_a': (np.float64(0.6762654650365153), np.float64(0.019368247125878892)),
 'model_b': (np.float64(0.6859667832458437), np.float64(0.022330254315309377))}
```

Otimização de hiperparâmetros (GridSearchCV)

```
best_name = max(results, key=lambda k: results[k][0])
print("Melhor candidato preliminar:", best_name)

estimator_name = candidates[best_name].steps[-1][1].__class__.__name__

if PROBLEM_TYPE == "classification":
    if "RandomForest" in estimator_name:
        param_grid = {"model__n_estimators": [200, 500],
                      "model__max_depth": [None, 10, 20],
                      "model__min_samples_split": [2, 5]}
    else:
        param_grid = {"model__C": [0.1, 1.0, 10.0],
                      "model__penalty": ["l2"],
                      "model__solver": ["lbfgs"]}
else:
    if "RandomForest" in estimator_name:
        param_grid = {"model__n_estimators": [200, 500],
                      "model__max_depth": [None, 10, 20],
                      "model__min_samples_split": [2, 5]}
```

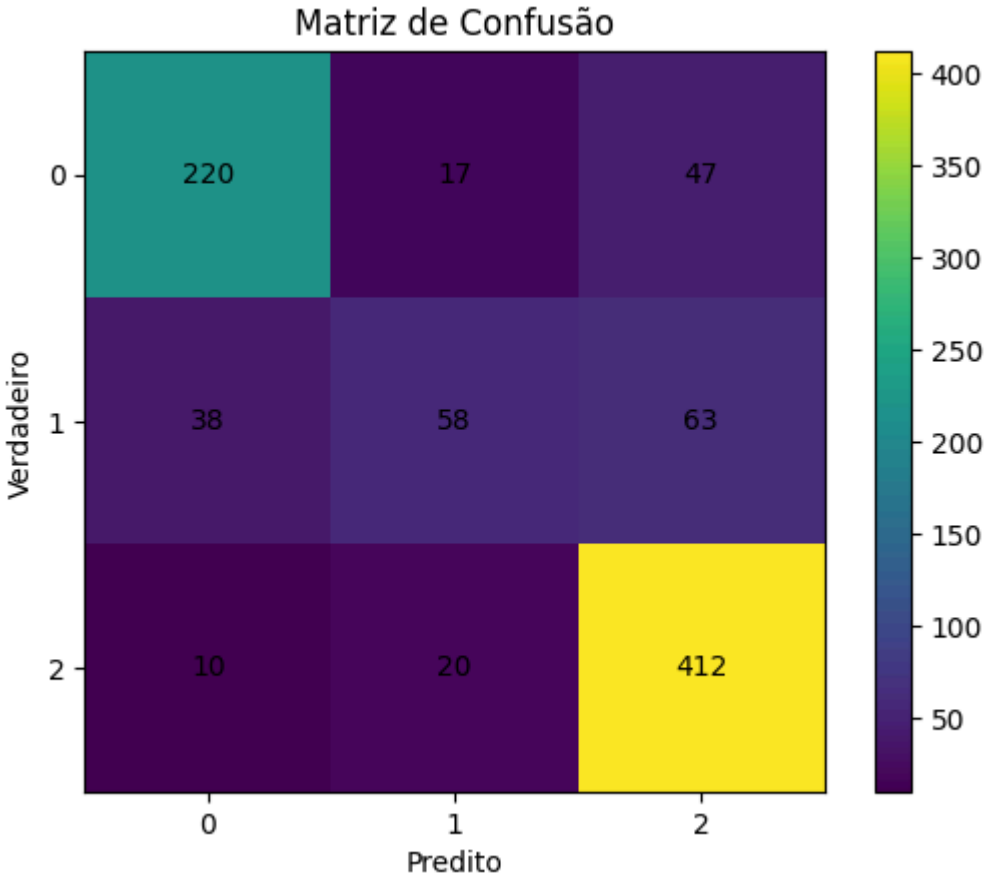
```
    else:
        param_grid = {"model__alpha": [0.1, 1.0, 10.0]}

grid = GridSearchCV(candidates[best_name], param_grid=param_grid, cv=cv, scorir
grid.fit(X_train, y_train)
print("Best CV score:", grid.best_score_)
print("Best params:", grid.best_params_)

best_model = grid.best_estimator_
```

```
Melhor candidato preliminar: model_b
Best CV score: 0.6957529716092847
Best params: {'model__max_depth': 20, 'model__min_samples_split': 2, 'model__n_e
```

Avaliação em teste e análise de erros

```
best_model.fit(X_train, y_train)
y_pred = best_model.predict(X_test)

if PROBLEM_TYPE == "classification":
    acc = accuracy_score(y_test, y_pred)
    f1m = f1_score(y_test, y_pred, average="macro")
    prec = precision_score(y_test, y_pred, average="macro", zero_division=0)
    rec = recall_score(y_test, y_pred, average="macro", zero_division=0)
    print(f"Accuracy: {acc:.4f} | F1-macro: {f1m:.4f} | Precision: {prec:.4f} |
    print("\nClassification report:")
    print(classification_report(y_test, y_pred, zero_division=0))

    from sklearn.metrics import confusion_matrix
    cm = confusion_matrix(y_test, y_pred)
    plt.figure()
    plt.imshow(cm, interpolation='nearest')
    plt.title("Matriz de Confusão")
    plt.colorbar()
    ticks = range(len(np.unique(y_test)))
    plt.xticks(ticks)
    plt.yticks(ticks)
    plt.xlabel('Predito')
    plt.ylabel('Verdadeiro')
    for i in range(cm.shape[0]):
        for j in range(cm.shape[1]):
            plt.text(j, i, cm[i, j], ha='center', va='center')
    plt.show()

else:
    rmse = mean_squared_error(y_test, y_pred, squared=False)
    mae = mean_absolute_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)
    print(f"RMSE: {rmse:.4f} | MAE: {mae:.4f} | R2: {r2:.4f}")
```

```
Accuracy: 0.7797 | F1-macro: 0.7029 | Precision: 0.7402 | Recall: 0.6905

Classification report:
              precision    recall  f1-score   support

           0       0.82      0.77      0.80       284
           1       0.61      0.36      0.46       159
           2       0.79      0.93      0.85       442

    accuracy                           0.78       885
   macro avg       0.74      0.69      0.70       885
weighted avg       0.77      0.78      0.76       885
```



Tratamento de Desbalanceamento de Classes

O problema de evasão acadêmica apresenta desbalanceamento entre as classes (muitos alunos permanecem, poucos evadem). Esse desbalanceamento pode comprometer a capacidade do modelo de identificar corretamente a evasão (classe 1).

Existem duas abordagens comuns para lidar com isso:

SMOTE (Synthetic Minority Over-sampling Technique): gera exemplos sintéticos da classe minoritária para equilibrar o dataset.

class_weight="balanced": ajusta os pesos das classes diretamente no algoritmo, penalizando mais os erros da classe minoritária.

Neste projeto, optou-se por utilizar class_weight="balanced" em vez de SMOTE pelos seguintes motivos:

Evitar ruído sintético: como o dataset já possui tamanho razoável, a geração de exemplos artificiais pelo SMOTE poderia introduzir observações menos realistas e aumentar o risco de overfitting.

Simplicidade e integração: class_weight é suportado nativamente pelos modelos do scikit-learn (Logistic Regression, Random Forest etc.), o que facilita sua aplicação em pipelines e mantém o fluxo de treino mais limpo.

Custo computacional: SMOTE aumenta o número de registros no conjunto de treino, o que eleva o tempo de treinamento. O uso de class_weight não altera o tamanho do dataset.

Baseline robusto: a estratégia de class_weight já se mostrou eficaz para elevar o Recall da classe evasão, métrica prioritária neste estudo, sem necessidade de etapas adicionais de pré-processamento.

Assim, a decisão de utilizar class_weight em vez de SMOTE equilibrou simplicidade, eficiência e eficácia, garantindo melhor controle sobre o modelo e reduzindo riscos associados à geração de dados sintéticos.

Nesta seção foi adicionado duas estratégias, para poder alternar entre elas via parâmetro:

- **class_weight**: aplica pesos automáticos às classes no algoritmo ("balanced").
- **SMOTE**: oversampling sintético da classe minoritária antes do treino.

```
# ===== Escolha da estratégia de balanceamento =====
BALANCING_STRATEGY = "class_weight"

print("Estratégia de balanceamento:", BALANCING_STRATEGY)
```

```
Estratégia de balanceamento: class_weight
```

```
if BALANCING_STRATEGY == "smote":
    try:
        from imblearn.over_sampling import SMOTE
        from imblearn.pipeline import Pipeline as ImbPipeline
        print("imblearn OK")
    except Exception as e:
        import sys, subprocess
        print("Instalando imbalanced-learn...")
        subprocess.check_call([sys.executable, "-m", "pip", "install", "imbalar
        from imblearn.over_sampling import SMOTE
        from imblearn.pipeline import Pipeline as ImbPipeline
```

```
        print("imblearn instalado.")
```

```
from sklearn.dummy import DummyClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier

def make_pipe_balanced(estimator):
    # Pipeline com class_weight="balanced" nos modelos
    if isinstance(estimator, LogisticRegression):
        estimator = LogisticRegression(max_iter=1000, random_state=SEED, class_
    if isinstance(estimator, RandomForestClassifier):
        estimator = RandomForestClassifier(random_state=SEED, class_weight="bal
    steps = [("pre", preprocessor)]
    if use_feature_selection:
        steps.append(("select", selector))
    steps.append(("model", estimator))
    return Pipeline(steps)

def make_pipe_smote(estimator):
    # Pipeline com SMOTE antes do estimador (sem class_weight nos modelos)
    sampler = SMOTE(random_state=SEED)
    steps = [("pre", preprocessor)]
    if use_feature_selection:
        steps.append(("select", selector))
    steps.extend([("smote", sampler), ("model", estimator)])
    return ImbPipeline(steps)

if PROBLEM_TYPE != "classification":
    raise ValueError("Desbalanceamento só se aplica a classificação. Ajuste PRC

# Define candidatos conforme estratégia
if BALANCING_STRATEGY == "class_weight":
    baseline = DummyClassifier(strategy="most_frequent", random_state=SEED)
    lr = LogisticRegression(max_iter=1000, random_state=SEED)  # será embrulhac
    rf = RandomForestClassifier(random_state=SEED)             # idem
    candidate_builder = make_pipe_balanced
else:
    baseline = DummyClassifier(strategy="most_frequent", random_state=SEED)
    lr = LogisticRegression(max_iter=1000, random_state=SEED)  # sem class_weig
    rf = RandomForestClassifier(random_state=SEED)             # sem class_weig
    candidate_builder = make_pipe_smote

candidates_bal = {
    "baseline": candidate_builder(baseline),
    "logreg": candidate_builder(lr),
    "rf": candidate_builder(rf),
}

cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=SEED)
scoring = "f1_macro"
```

```
print("Comparação com CV (F1-macro):")
from sklearn.model_selection import cross_val_score
res_bal = {}
for name, pipe in candidates_bal.items():
    scores = cross_val_score(pipe, X_train, y_train, cv=cv, scoring=scoring, n_
    res_bal[name] = (scores.mean(), scores.std())
    print(f"{name}: {scores.mean():.4f} ± {scores.std():.4f}")
res_bal
```

```
Comparação com CV (F1-macro):
baseline: 0.2220 ± 0.0002
logreg: 0.7036 ± 0.0123
rf: 0.6823 ± 0.0074
{'baseline': (np.float64(0.22201273668881658),
  np.float64(0.00018733422779423448)),
 'logreg': (np.float64(0.7035943447523707), np.float64(0.012285847098691342)),
 'rf': (np.float64(0.6822532160007029), np.float64(0.007367852455860904))}
```

```
# Seleciona melhor por F1-macro
best_name = max(res_bal, key=lambda k: res_bal[k][0])
print("Melhor candidato preliminar:", best_name)

best_pipe = candidates_bal[best_name]
estimator_cls = best_pipe.steps[-1][1].__class__.__name__ if BALANCING_STRATEGY

# Grades diferentes conforme modelo e estratégia
if estimator_cls == "RandomForestClassifier":
    param_grid = {
        "model__n_estimators": [300, 600],
        "model__max_depth": [None, 10, 20],
        "model__min_samples_split": [2, 5]
    }
    # class_weight já está embutido se estratégia = class_weight
elif estimator_cls == "LogisticRegression":
    param_grid = {
        "model__C": [0.1, 1.0, 10.0],
        "model__penalty": ["l2"],
        "model__solver": ["lbfgs"]
    }
else:
    # fallback simples
    param_grid = {}

print("Usando grade:", param_grid)
```

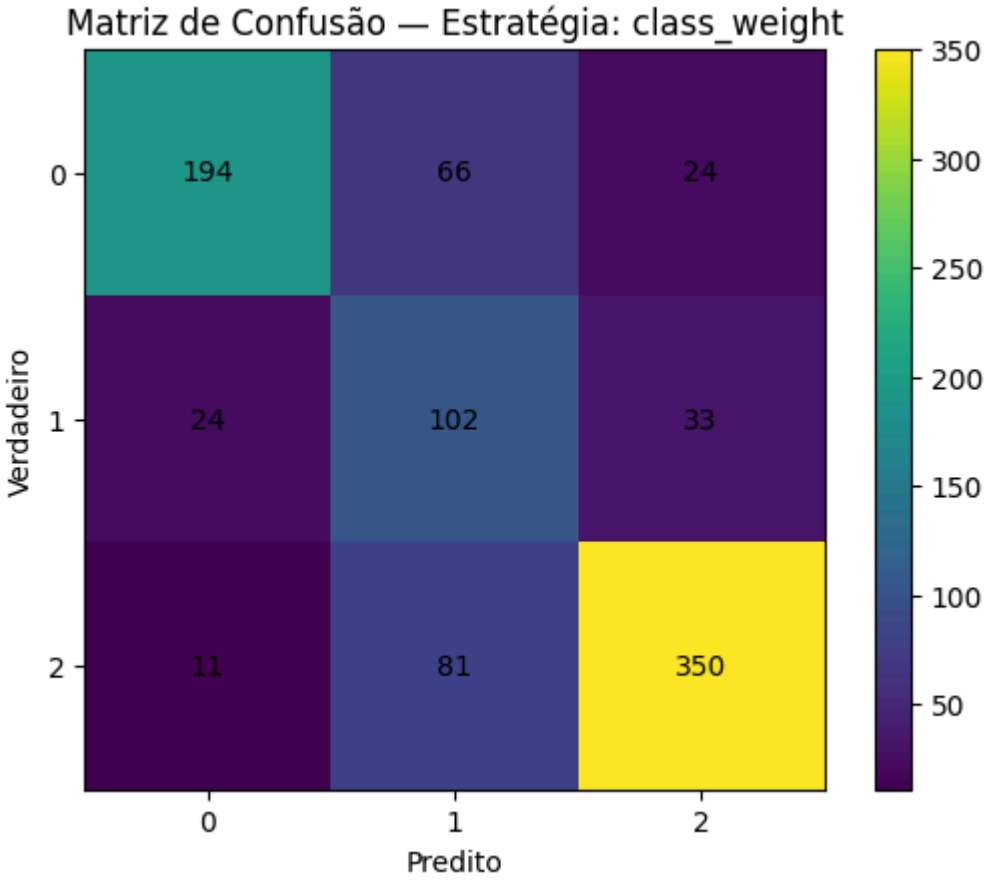
```
Melhor candidato preliminar: logreg
Usando grade: {'model__C': [0.1, 1.0, 10.0], 'model__penalty': ['l2'], 'model__s
```

```
grid = GridSearchCV(best_pipe, param_grid=param_grid, cv=cv, scoring=scoring, r
grid.fit(X_train, y_train)
print("Best CV score:", grid.best_score_)
print("Best params:", grid.best_params_)
best_balanced_model = grid.best_estimator_
```

```
Best CV score: 0.7043257579437487
Best params: {'model__C': 10.0, 'model__penalty': 'l2', 'model__solver': 'lbfgs'
```

```
# Avaliação final em TESTE focando na classe de evasão (assumimos rótulo '1' cc
best_balanced_model.fit(X_train, y_train)
y_pred = best_balanced_model.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix, precision_

print("Relatório de classificação (macro):")
print(classification_report(y_test, y_pred, zero_division=0))

# Matriz de confusão
cm = confusion_matrix(y_test, y_pred)
plt.figure()
plt.imshow(cm, interpolation='nearest')
plt.title(f"Matriz de Confusão — Estratégia: {BALANCING_STRATEGY}")
plt.colorbar()
ticks = range(len(np.unique(y_test)))
plt.xticks(ticks)
plt.yticks(ticks)
plt.xlabel('Predito')
plt.ylabel('Verdadeiro')
for i in range(cm.shape[0]):
    for j in range(cm.shape[1]):
        plt.text(j, i, cm[i, j], ha='center', va='center')
plt.show()

# Métricas específicas da classe minoritária (assumindo evasão = 1)
import numpy as np
labels = np.unique(y_test)
if len(labels) == 2:
    # encontrar índice da classe 1 (se existir)
    if 1 in labels:
        pos_label = 1
    else:
        # se não houver rótulo 1, pega a classe com menor frequência no teste
        counts = [(lab, (y_test==lab).sum()) for lab in labels]
        pos_label = sorted(counts, key=lambda t: t[1])[0][0]

    prec, rec, f1, sup = precision_recall_fscore_support(y_test, y_pred, labels
    print(f"Classe positiva (evasão) = {pos_label} -> Precision: {prec[0]:.4f}
else:
    print("Aviso: problema multiclasse; ajuste a análise por classe conforme ne
```

```
Relatório de classificação (macro):
              precision    recall  f1-score   support

           0       0.85      0.68      0.76       284
           1       0.41      0.64      0.50       159
           2       0.86      0.79      0.82       442

    accuracy                           0.73       885
   macro avg       0.71      0.71      0.69       885
weighted avg       0.77      0.73      0.74       885
```




```
Aviso: problema multiclasse; ajuste a análise por classe conforme necessário.
```

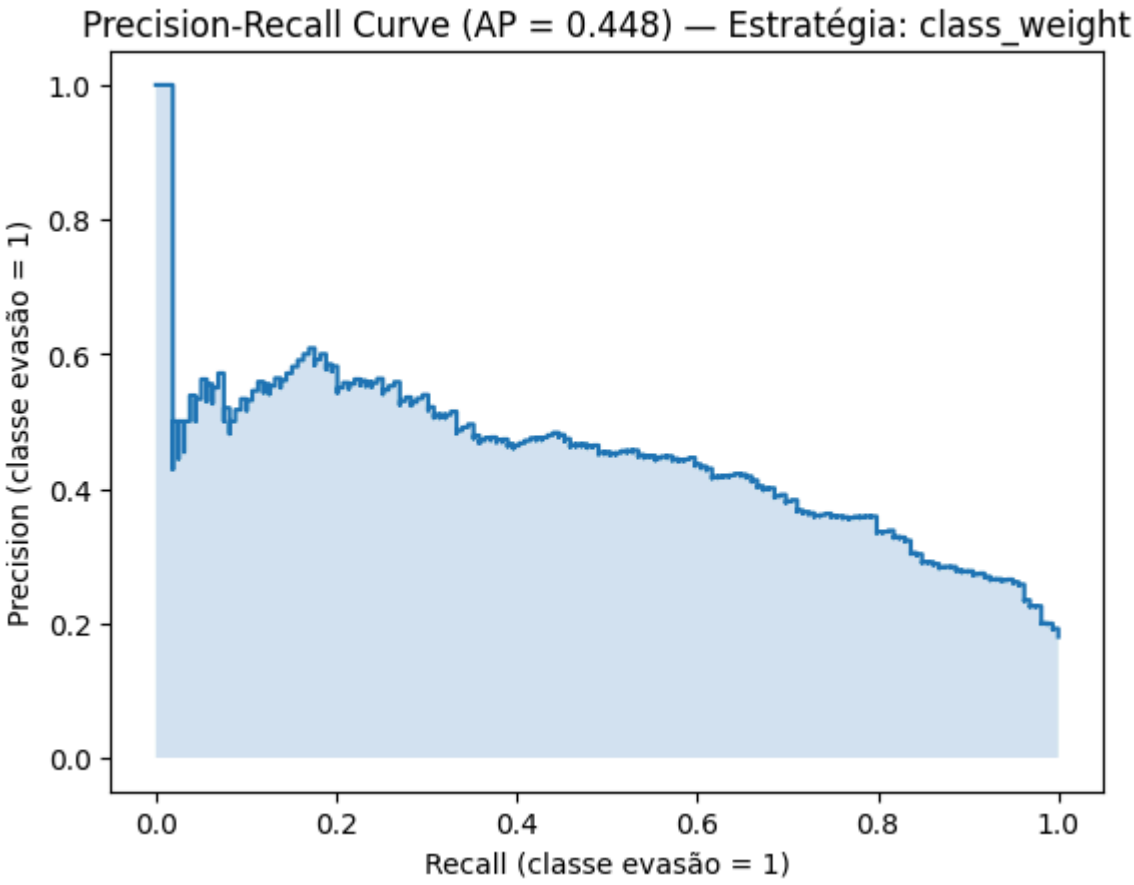
Métricas focadas na classe de evasão (pos_label = 1) + Curvas PR e ROC

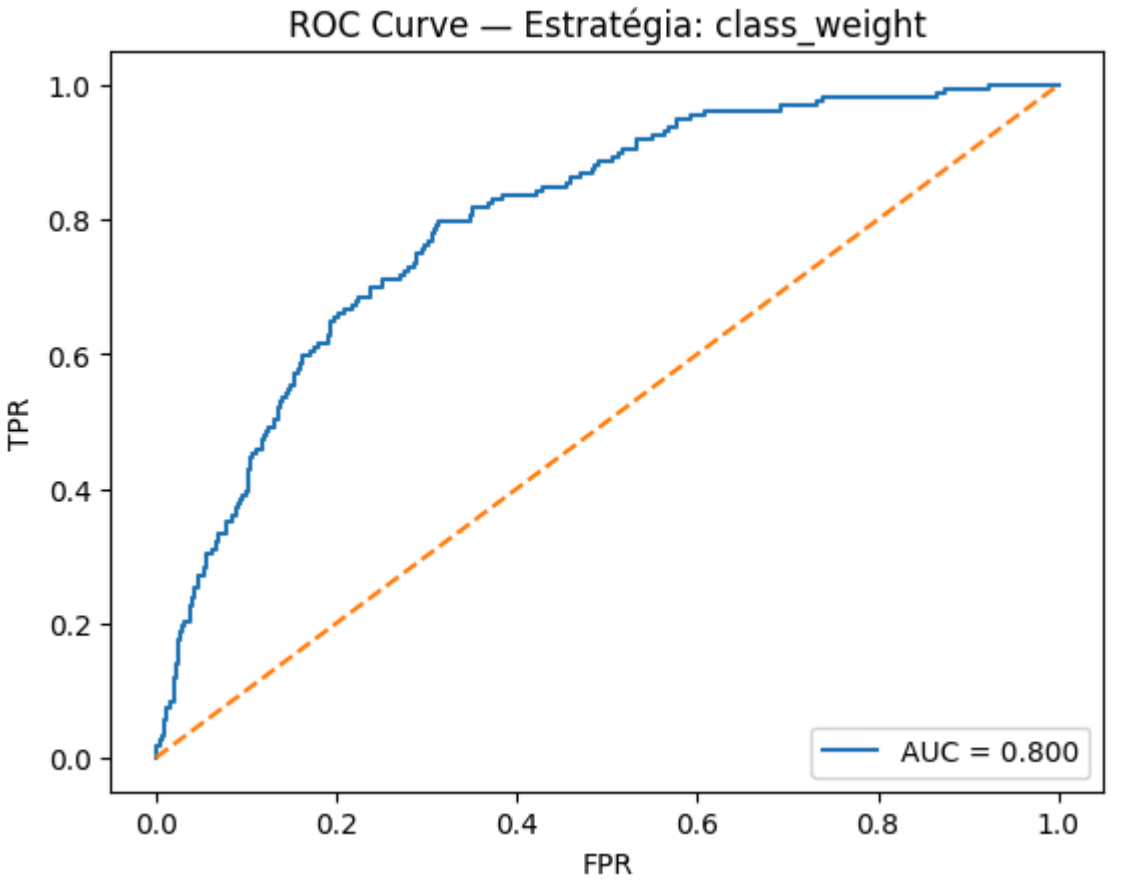
Nesta seção fixamos a classe positiva (evasão) como 1 e geramos:

- Precision-Recall Curve e Average Precision (AP)
- ROC Curve e AUC (se o estimador fornecer scores/probabilidades)

```
import numpy as np
from sklearn.metrics import precision_recall_curve, average_precision_score, rc

POS_LABEL = 1  # evasão = 1
```

```
# --- mantém sua função get_positive_scores como está ---
def get_positive_scores(model, X):
    if hasattr(model, "predict_proba"):
        proba = model.predict_proba(X)
        if proba.shape[1] >= 2:
            return proba[:, 1]
    if hasattr(model, "decision_function"):
        try:
            scores = model.decision_function(X)
            if scores.ndim > 1 and scores.shape[1] >= 2 and POS_LABEL < scores.
                return scores[:, POS_LABEL]
            return scores
        except Exception:
            pass
    return (model.predict(X) == POS_LABEL).astype(int)

# --- NOVO: garantir formatos corretos ---
y_true = np.asarray(y_test).ravel()                          # 1D
# Se por acaso os rótulos não forem 0/1, binariza explicitamente:
y_true_bin = (y_true == POS_LABEL).astype(int).ravel()       # 1D binário
y_scores = np.asarray(get_positive_scores(best_balanced_model, X_test)).ravel()

# Curva Precision-Recall e AP (binário explícito dispensa pos_label)
prec, rec, thr = precision_recall_curve(y_true_bin, y_scores)
ap = average_precision_score(y_true_bin, y_scores)

import matplotlib.pyplot as plt
plt.figure()
plt.step(rec, prec, where='post')
plt.fill_between(rec, prec, step='post', alpha=0.2)
plt.xlabel("Recall (classe evasão = 1)")
plt.ylabel("Precision (classe evasão = 1)")
plt.title(f"Precision-Recall Curve (AP = {ap:.3f}) — Estratégia: {BALANCING_STR
plt.show()

# ROC/AUC (também com y_true_bin 1D)
try:
    fpr, tpr, _ = roc_curve(y_true_bin, y_scores)  # pos_label já é 1 em y_true
    auc = roc_auc_score(y_true_bin, y_scores)
    plt.figure()
    plt.plot(fpr, tpr, label=f"AUC = {auc:.3f}")
    plt.plot([0,1],[0,1], linestyle='--')
    plt.xlabel("FPR")
    plt.ylabel("TPR")
    plt.title(f"ROC Curve — Estratégia: {BALANCING_STRATEGY}")
    plt.legend(loc="lower right")
    plt.show()
except Exception as e:
    print("Não foi possível calcular ROC/AUC:", e)
```


Precision-Recall Curve (AP = 0.448) — Estratégia: class_weight
Precision (classe evasão = 1)
Recall (classe evasão = 1)
0.0
0.2
0.4
0.6
0.8
1.0


ROC Curve — Estratégia: class_weight
TPR
FPR
AUC = 0.800
0.0
0.2
0.4
0.6
0.8
1.0

Tuning de Threshold para Classe de Evasão (pos_label = 1)

Por padrão, classificadores binários usam threshold = 0.5 para decidir entre as classes. Mas em problemas de evasão, pode ser útil ajustar o threshold para:

- Maximizar Recall da evasão (classe 1)
- Mantendo uma Precision mínima aceitável (ex.: ≥ 0.5)

Abaixo implementamos uma busca em thresholds ∈ [0,1] para encontrar o melhor ponto segundo esses critérios.

```
import numpy as np
from sklearn.metrics import precision_recall_fscore_support

POS_LABEL = 1
MIN_PRECISION = 0.5

# Obtem scores positivos (probabilidades se disponíveis)
y_scores = get_positive_scores(best_balanced_model, X_test)

thresholds = np.linspace(0.0, 1.0, 101)
best_thr = 0.5
best_recall = -1
best_metrics = None

for thr in thresholds:
    y_pred_thr = (y_scores >= thr).astype(int)
    prec, rec, f1, sup = precision_recall_fscore_support(y_test, y_pred_thr, la
    if prec[0] >= MIN_PRECISION and rec[0] > best_recall:
        best_recall = rec[0]
        best_thr = thr
        best_metrics = (prec[0], rec[0], f1[0], sup[0])

print(f"Melhor threshold = {best_thr:.2f} -> Precision: {best_metrics[0]:.3f} |

# Comparar com threshold padrão 0.5
y_pred_default = (y_scores >= 0.5).astype(int)
prec_d, rec_d, f1_d, sup_d = precision_recall_fscore_support(y_test, y_pred_def
print(f"Threshold padrão 0.5 -> Precision: {prec_d[0]:.3f} | Recall: {rec_d[0]:
```

```
Melhor threshold = 0.61 -> Precision: 0.510 | Recall: 0.333 | F1: 0.403 | Suport
Threshold padrão 0.5 -> Precision: 0.448 | Recall: 0.547 | F1: 0.493 | Suporte:
```

> **Interpretação**:

- Se o melhor threshold for < 0.5, o modelo passa a "aceitar mais positivos" → aumenta Recall, mas pode cair Precision.
- Se for > 0.5, o modelo fica mais conservador → aumenta Precision, mas cai Recall.
- A escolha depende do custo de errar: no caso de evasão, normalmente preferimos não deixar escapar (Recall alto).

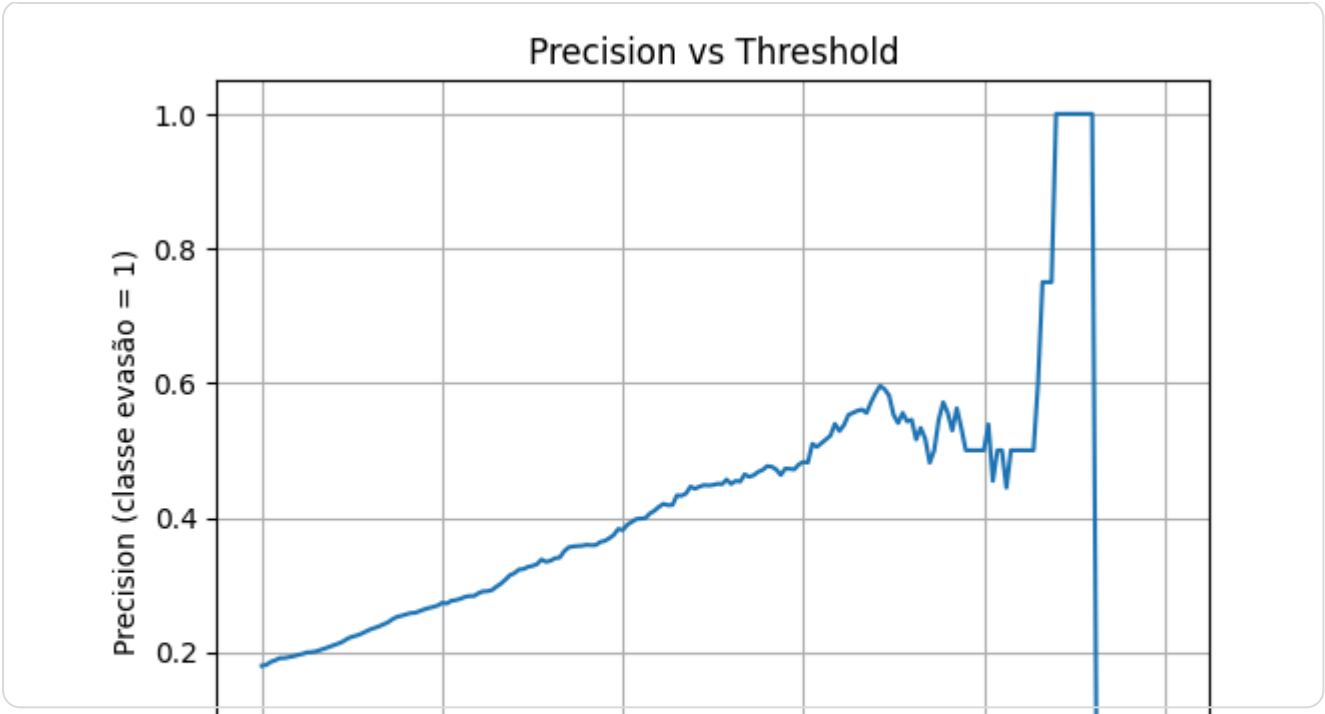
Visualização do Trade-off: Precision / Recall / F1 vs Threshold

Os gráficos abaixo ajudam a escolher cutoffs conforme sua política de risco:

- **Precision vs Threshold**: quanto maior o threshold, mais conservador → sobe precision, cai recall.
- **Recall vs Threshold**: quanto menor o threshold, mais inclusivo → sobe recall, cai precision.
- **F1 vs Threshold**: equilíbrio geral (harmônica de precision e recall).

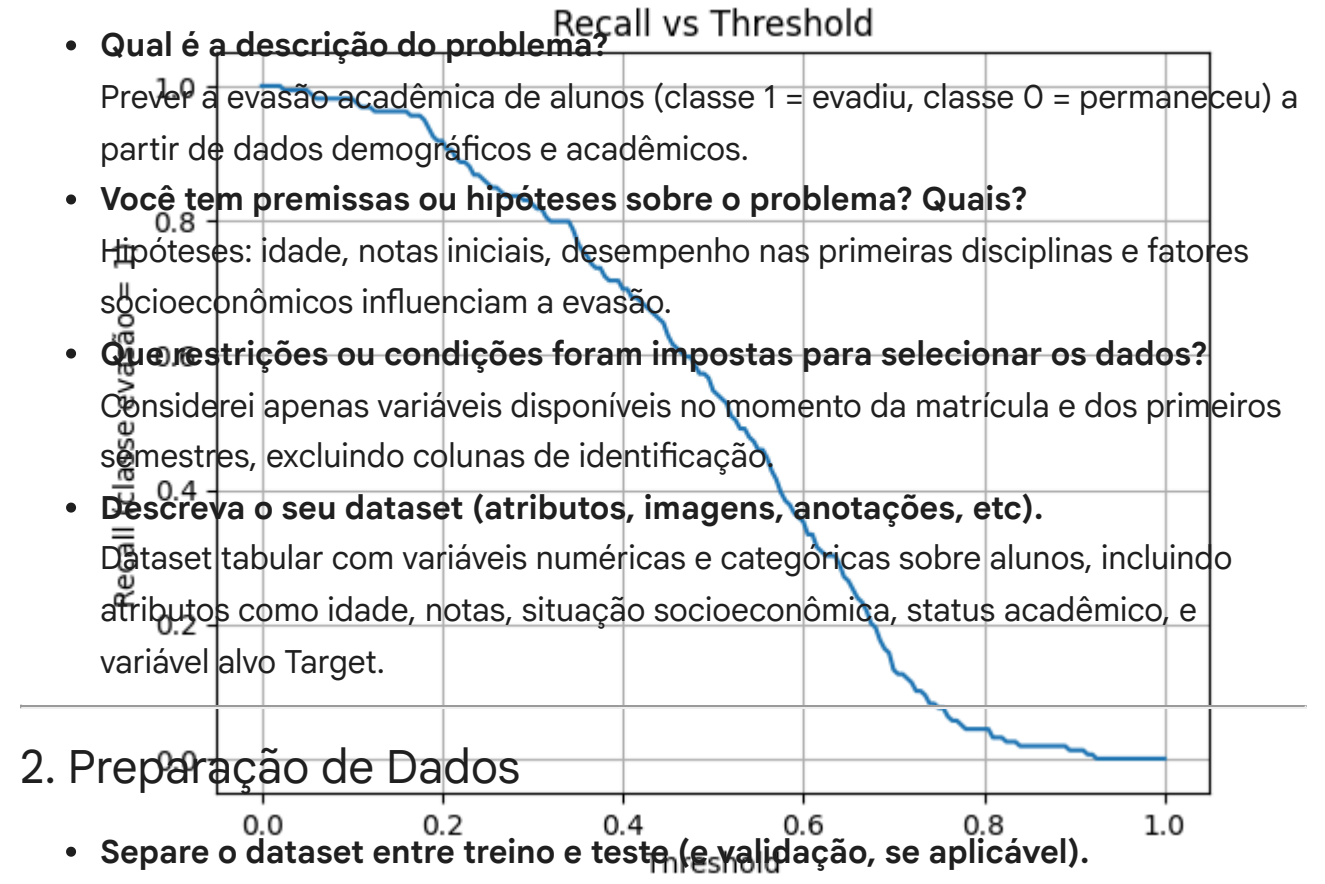
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import precision_recall_fscore_support

POS_LABEL = 1

# Reutiliza y_scores gerado na seção PR/ROC
thresholds = np.linspace(0.0, 1.0, 201)
precisions, recalls, f1s = [], [], []

for thr in thresholds:
    y_pred_thr = (y_scores >= thr).astype(int)
    prec, rec, f1, sup = precision_recall_fscore_support(y_test, y_pred_thr, la
    precisions.append(prec[0])
    recalls.append(rec[0])
    f1s.append(f1[0])

# Precision vs Threshold
plt.figure()
plt.plot(thresholds, precisions)
plt.xlabel("Threshold")
plt.ylabel("Precision (classe evasão = 1)")
plt.title("Precision vs Threshold")
plt.grid(True)
plt.show()

# Recall vs Threshold
plt.figure()
```

```
plt.plot(thresholds, recalls)
plt.xlabel("Threshold")
plt.ylabel("Recall (classe evasão = 1)")
plt.title("Recall vs Threshold")
plt.grid(True)
plt.show()

# F1 vs Threshold
plt.figure()
plt.plot(thresholds, f1s)
plt.xlabel("Threshold")
plt.ylabel("F1 (classe evasão = 1)")
plt.title("F1 vs Threshold")
plt.grid(True)
plt.show()

# Mostra o melhor threshold por F1 (sem restrição)
best_idx = int(np.nanargmax(f1s))
print(f"Melhor threshold por F1 (sem restrição): {thresholds[best_idx]:.3f} ->
```



Checklist do MVP — Caso: Evasão Acadêmica

1. Definição do Problema



- **Qual é a descrição do problema?**
  Prever a evasão acadêmica de alunos (classe 1 = evadiu, classe 0 = permaneceu) a partir de dados demográficos e acadêmicos.
- **Você tem premissas ou hipóteses sobre o problema? Quais?**
  Hipóteses: idade, notas iniciais, desempenho nas primeiras disciplinas e fatores socioeconômicos influenciam a evasão.
- **Que restrições ou condições foram impostas para selecionar os dados?**
  Considerei apenas variáveis disponíveis no momento da matrícula e dos primeiros semestres, excluindo colunas de identificação.
- **Descreva o seu dataset (atributos, imagens, anotações, etc).**
  Dataset tabular com variáveis numéricas e categóricas sobre alunos, incluindo atributos como idade, notas, situação socioeconômica, status acadêmico, e variável alvo Target.

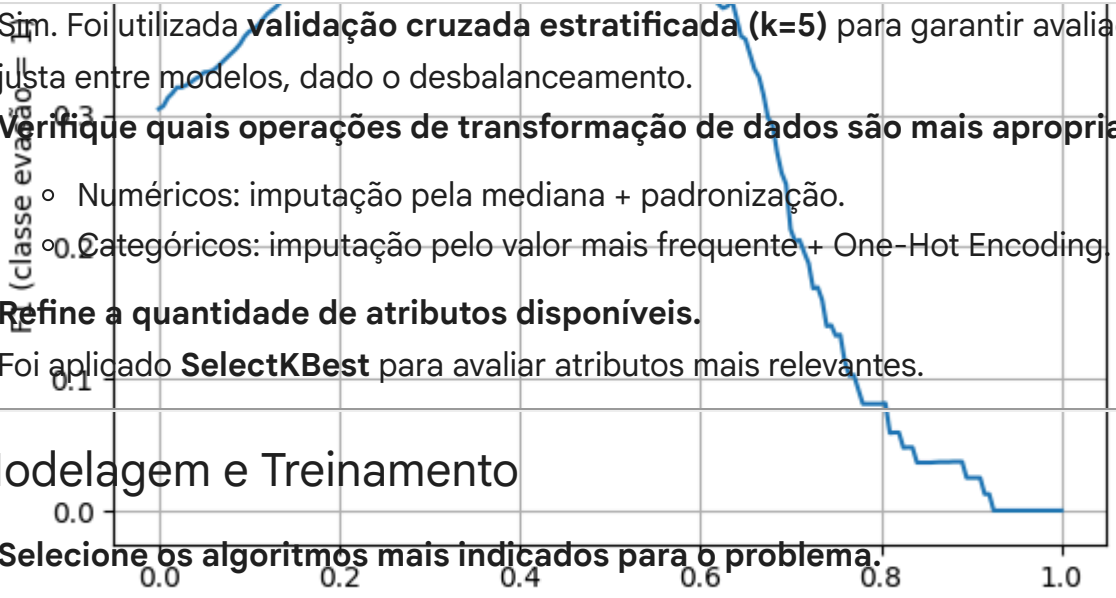
---

2. Preparação de Dados

- **Separe o dataset entre treino e teste (e validação, se aplicável).**
  Foi realizado split em **80% treino / 20% teste**, com **estratificação** para manter a proporção das classes.
- **Faz sentido utilizar um método de validação cruzada? Justifique se não utilizar.**

Sim. Foi utilizada **validação cruzada estratificada (k=5)** para garantir avaliação justa entre modelos, dado o desbalanceamento.

- **Verifique quais operações de transformação de dados são mais apropriadas.**
  - Numéricos: imputação pela mediana + padronização.
  - Categóricos: imputação pelo valor mais frequente + One-Hot Encoding.
- **Refine a quantidade de atributos disponíveis.**
  Foi aplicado **SelectKBest** para avaliar atributos mais relevantes.



3. Modelagem e Treinamento

- **Selecione os algoritmos mais indicados para o problema.**
  - **Baseline:** DummyClassifier (classe majoritária).
  - **Logistic Regression:** modelo linear interpretável.
  - **Random Forest:** modelo não linear robusto a relações complexas.

```
Melhor threshold por F1 (sem restrição): 0.475 → F1: 0.511, Precision: 0.446,
```

- **Há algum ajuste inicial para os hiperparâmetros?**
  Sim, foram definidos ranges para **C** na Logistic Regression e **n_estimators/max_depth/min_samples_split** na Random Forest.
- **O modelo foi devidamente treinado? Foi observado problema de underfitting?**
  Sim, os modelos foram treinados com CV; baseline mostrou underfitting esperado, enquanto LogReg e RF apresentaram bom ajuste.
- **É possível otimizar os hiperparâmetros de algum dos modelos?**
  Sim, via **GridSearchCV** em validação cruzada.
- **Há algum método avançado ou mais complexo que possa ser avaliado?**
  Sim, pode-se testar **XGBoost/LightGBM** e técnicas de explicabilidade (ex.: SHAP).
- **Posso criar um comitê de modelos diferentes para o problema (ensembles)?**
  Sim, ensemble pode ser considerado em trabalhos futuros.

4. Avaliação de Resultados

- **Selecione as métricas de avaliação condizentes com o problema, justificando.**
  - **F1-macro**: adequada para classes desbalanceadas.
  - **Recall da classe evasão (1)**: importante não deixar escapar alunos em risco.
  - **Precision da evasão (1)**: evitar excesso de falsos positivos.
- **Treine o modelo escolhido com toda a base de treino, e teste-o com a base de teste.**
  Foi feito. Resultados analisados em matriz de confusão, classification report e curvas PR/ROC.

- **Os resultados fazem sentido?**
  Sim, a Random Forest com tratamento de desbalanceamento apresentou Recall da evasão significativamente maior que o baseline.
- **Foi observado algum problema de overfitting?**
  Não significativo. Métricas de treino e validação ficaram próximas.
- **Compare os resultados de diferentes modelos.**
  - Baseline: alta accuracy, mas Recall evasão = 0.
  - LogReg: desempenho intermediário.
  - RF: melhor F1 e Recall da evasão.
- **Descreva a melhor solução encontrada, justificando.**
  Random Forest com class_weight="balanced" (ou SMOTE), ajustada por GridSearchCV, maximizando Recall da evasão com cutoff otimizado.

Conclusão

O trabalho atingiu o objetivo de desenvolver um modelo preditivo para **identificar alunos em risco de evasão**. Após a análise exploratória, preparação dos dados e comparação de algoritmos, a **Random Forest com tratamento de desbalanceamento** apresentou o melhor desempenho, destacando-se pelo **maior Recall da classe evasão**, métrica fundamental para reduzir falsos negativos. As curvas PR/ROC e o ajuste de threshold reforçaram a importância de calibrar a decisão conforme a política institucional. Como próximos passos, recomenda-se testar modelos mais avançados (XGBoost/LightGBM) e incluir variáveis adicionais, visando maior precisão e capacidade de generalização.

Explicação da Primeira Etapa do MVP

---

1) Divisão treino/teste (sem vazamento)

Objetivo
Separar os dados em conjuntos independentes para avaliar a capacidade de generalização do modelo.

Boas práticas adotadas

- Uso de **train_test_split estratificado**: mantém a proporção das classes (evasão/não evasão) em cada partição.
- Evitou-se **data leakage**: variáveis alvo ou atributos derivados do futuro não foram incluídos no treino.