Introdução a Machine Learning

Profa. Dra. Roberta Wichmann

roberta.wichmann@idp.edu.br

Aula 3 – Etapas na Construção de Modelos de Machine Learning

Entendendo o problema, análise descritiva,coleta e divisão dos dados.

Pré-processamento e transformação dos dados.

Treinamento, aprimoramento do modelo e teste final.

As Etapas da construção de modelos de Machine Learning

O Que é Machine Learning (ML)?

- ML é como **ensinar computadores a aprender com dados**, sem programá-los explicitamente para cada tarefa.
- Pense em **ensinar um cachorro**: mostramos exemplos, corrigimos erros e ele aprende a reconhecer comandos.
- Com o ML, **criamos "modelos" que podem prever, classificar ou tomar decisões com base nos dados que recebem**.

Mas como fazer esses modelos?

idp

As Etapas da construção de modelos de Machine Learning

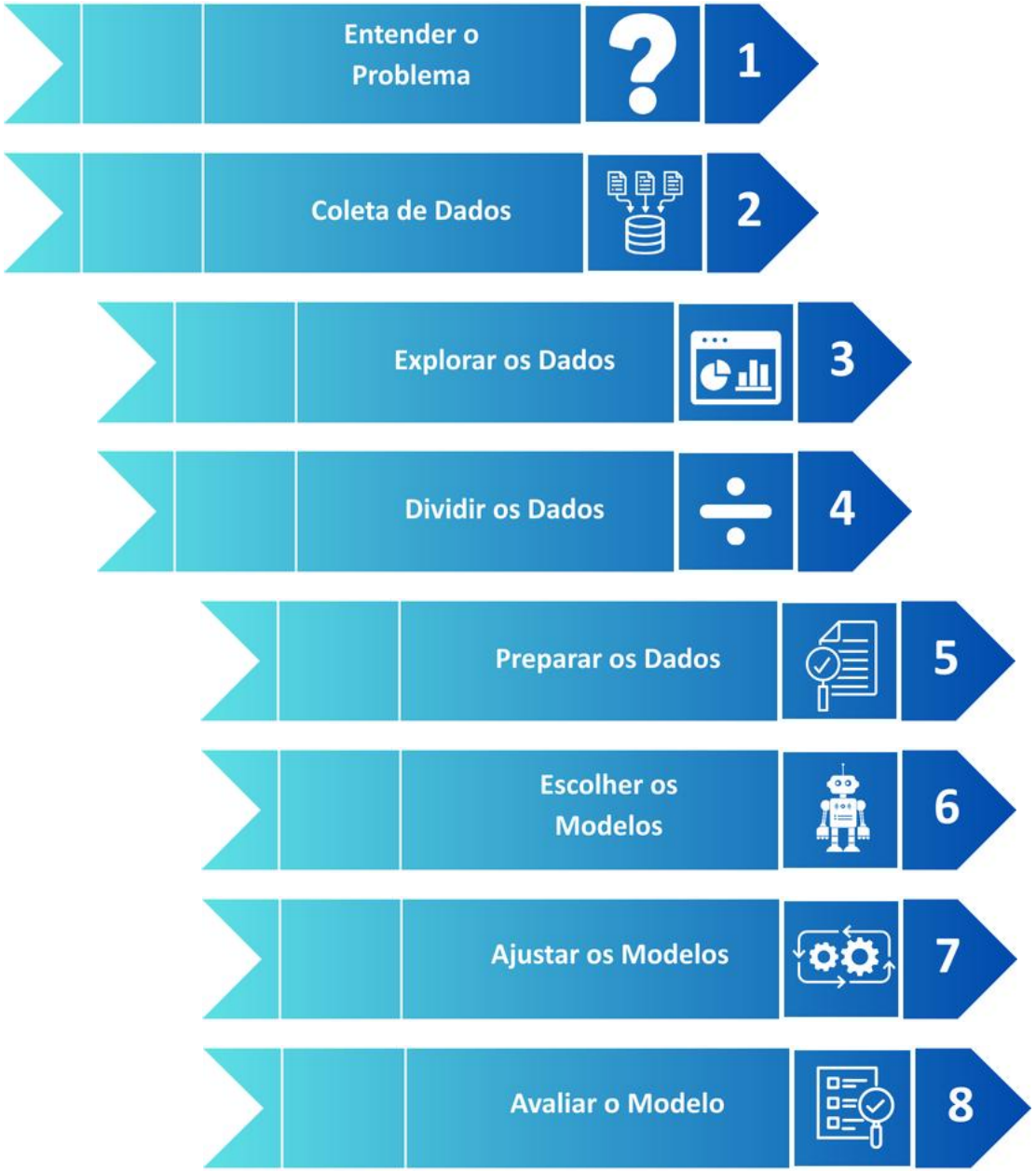
Por Que Este Guia Passo a Passo?

- ML pode **parecer complicado**, mas seguindo um processo estruturado, **fica mais acessível**.
- Vamos entender que criando um caminho claro **dá para transformar um problema real em uma solução de ML eficaz**.
- Vamos **quebrar cada etapa**, desde entender o problema até colocar a solução em prática.

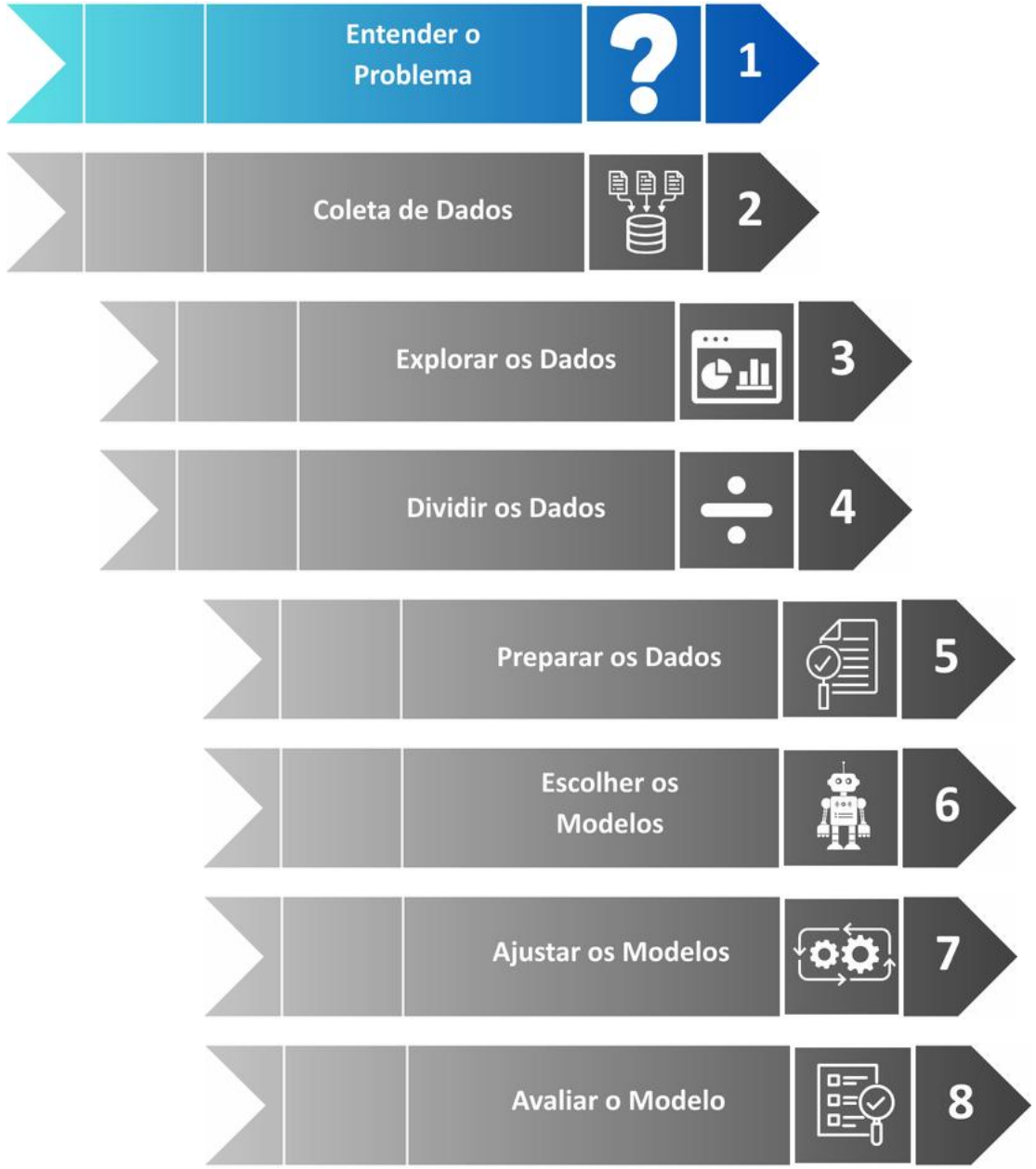
Quais são as etapas?

idp

As Etapas da construção de modelos de Machine Learning



Começando Pelo Início



Entender o Problema

Panorama geral:

- Antes de começar a construir, **precisamos entender o que queremos alcançar.**
- Qual é o **problema que estamos tentando resolver?** Qual é o objetivo final?
- Este passo é crucial para **garantir que nosso projeto de ML esteja alinhado** com as necessidades do negócio.

Vamos relembrar?

Entender o Problema

Qual nosso problema?

- Prever os custos médicos individuais dos beneficiários de seguro de saúde nos EUA.

Quais possíveis variáveis utilizar?

- **Idade** do beneficiário (quanto mais velho, maior o risco de gastos médicos).
- **Sexo** (diferenças biológicas podem afetar custos de saúde).
- Peso ou **índice de massa corporal** (IMC).
- **Região de residência** (custos médicos podem variar por região).

Entender o Problema

Como Enquadrar o Problema Corretamente?

- **É um problema de aprendizado supervisionado** (prever um resultado com base em dados rotulados)?
- **Ou não supervisionado** (encontrar padrões ocultos nos dados)?
- A escolha do **tipo de aprendizado e processamento de dados impacta diretamente o modelo de ML** que vamos construir.

Vamos relembrar?

idp

Entender o Problema

Escopo do Projeto:

- Na base de dados exemplo_01_aula_01, vemos **há algumas colunas que possam vir a nos auxiliar para o nosso problema.**

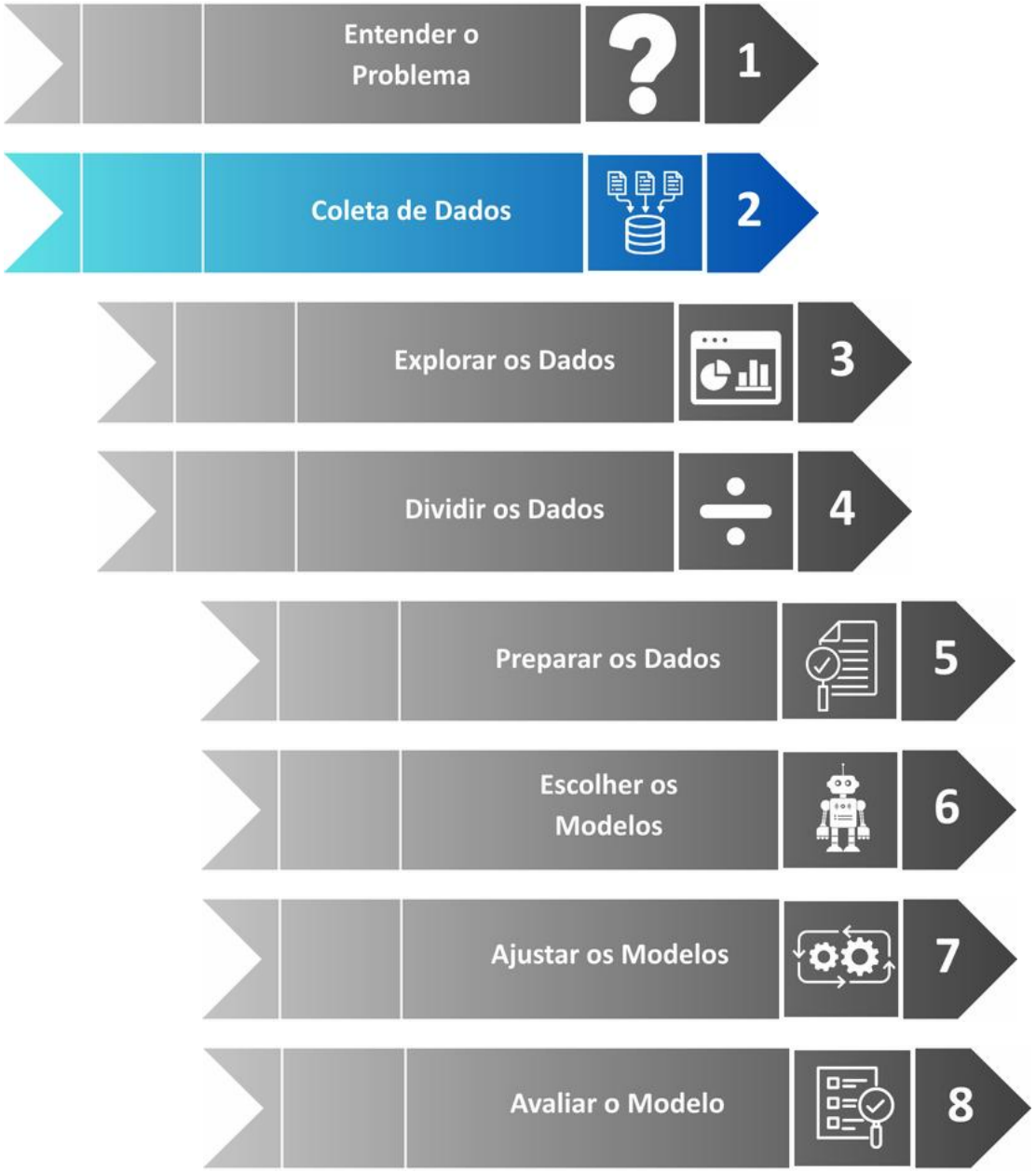
Como queremos prever os custos médicos, qual seria nossa variável target?

- **Resposta:** charges(Custos médicos), ou seja um aprendizado **supervisionado com regressão.**
- As demais variáveis serão nossas **auxiliares para fazer a modelagem preditiva.**

Agora entendemos o que nós vamos fazer!

idp

As Etapas da construção de modelos de Machine Learning



Coletar os Dados

A Coleta dos dados:

- Dados são a matéria-prima do Machine Learning. **Quanto mais dados, melhor o modelo pode aprender.**

- Precisamos identificar **quais dados são relevantes para o nosso problema** e como podemos obtê-los.

- Este passo envolve desde a **busca por fontes de dados até a garantia de que temos permissão para usá-los.**

Lembre-se da LGPD!

idp

Coletar os Dados

Listar e Documentar os Dados Necessários:

- **Quais informações precisamos?** (Dados demográficos? Histórico de compras? Textos?)
- **Onde podemos encontrar esses dados?** (Bancos de dados? Planilhas?Internet?)
- **É fundamental documentar cada fonte de dados,** incluindo sua origem, formato e restrições de uso (**Lembre-se do dicionário de dados!**)

Vamos relembrar?

Coletar os Dados

Listar e Documentar os Dados Necessários:

- Como precisávamos de informações sobre indivíduos, a **primeira escolha foi utilizar o Kaggle.**

- Com os dados coletados, **criamos um dicionário de dados** para nos ajudar a entender como estava a estrutura dos nossos dados.

Vamos olhar o dicionário novamente?

idp

Coletar os Dados

Nome	Descrição das variáveis	Tipo dos dados	Unidade de Medida	Valores Válidos
bmi	Índice de Massa Corporal (IMC)	Numérico Continuo	kg/m²	Valores positivos
sex	Sexo do beneficiário	Categórico Nominal	Não se aplica	male (homem), female (mulher)
age	Idade do beneficiário principal	Numérico Continuo	Anos	valores positivos
children	N° de filhos ou dependentes cobertos pelo seguro	Numérico Discreto	Contagem	valores positivos
smoker	Indica se o beneficiário é fumante	Categórico Nominal	Não se aplica	yes (Sim), no (Não)
region	Região de residência do beneficiário nos Estados Unidos	Categórico Nominal	Não se aplica	northeast, northwest, southeast, southwest
charges	Custos médicos individuais cobrados pelo seguro de saúde	Numérico Continuo	Dólares (USD)	valores positivos

idp

Coletar os Dados

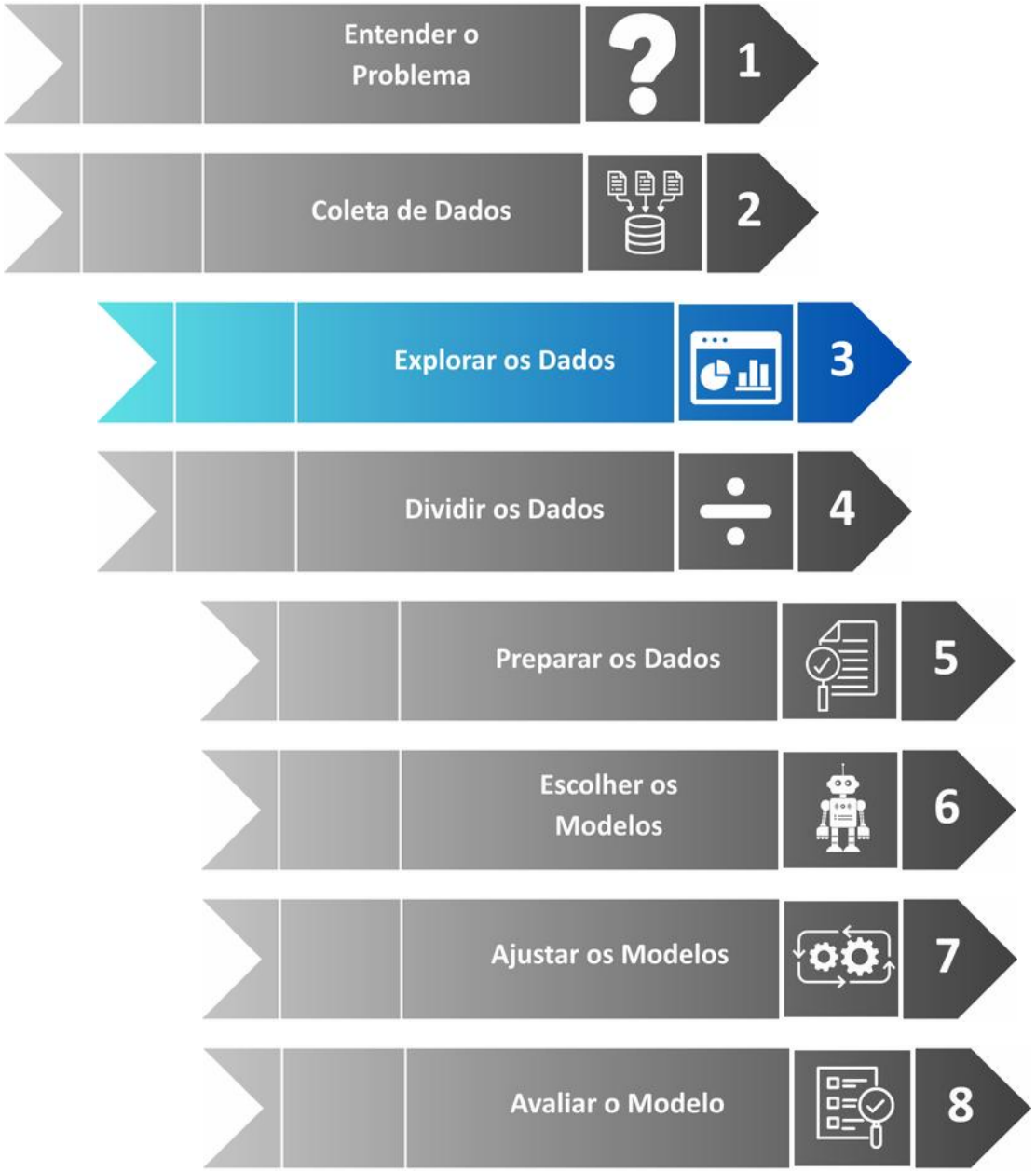
Preparando o Ambiente para os Dados:

- Crie um espaço de trabalho seguro e organizado para **armazenar os dados.**
- Converta os dados para um **formato fácil de manipular** (planilhas, CSV, etc.).
- **Garanta a privacidade dos dados**, removendo informações sensíveis ou anonimizando-as.

Foi o que fizemos com o ANACONDA e o Jupyter Notebook, lembra?

idp

As Etapas da construção de modelos de Machine Learning



Explorar os Dados

Explorar os Dados:

- Agora que temos os dados, precisamos **entender o que eles significam.**
- A exploração de dados nos ajuda a **identificar padrões, valores ausentes, erros e outras características importantes.**
- Este passo **é crucial para tomar decisões** informadas sobre como preparar os dados para o treinamento do modelo.

A exploração é um guia para entender os dados!

Explorar os Dados

Criar um "Diário de Bordo" da Exploração:

- **Registre** todas as suas descobertas e insights.
- Anote **quais atributos são importantes, quais estão faltando, quais precisam ser corrigidos.**
- **Documentar a exploração de dados** garante que você não perca informações valiosas ao longo do processo.

Lembre-se que podemos fazer isso no jupyter notebook!

idp

Explorar os Dados

Visualizando os Dados:

- Use **gráficos e tabelas para entender melhor os dados** (histogramas, diagramas de dispersão, etc.).
- Procure por **correlações entre os atributos** (quais estão relacionados entre si?).
- A visualização de dados **ajuda a identificar padrões e tendências** que seriam difíceis de detectar de outra forma.

Vamos relembrar dos insights que extraímos?

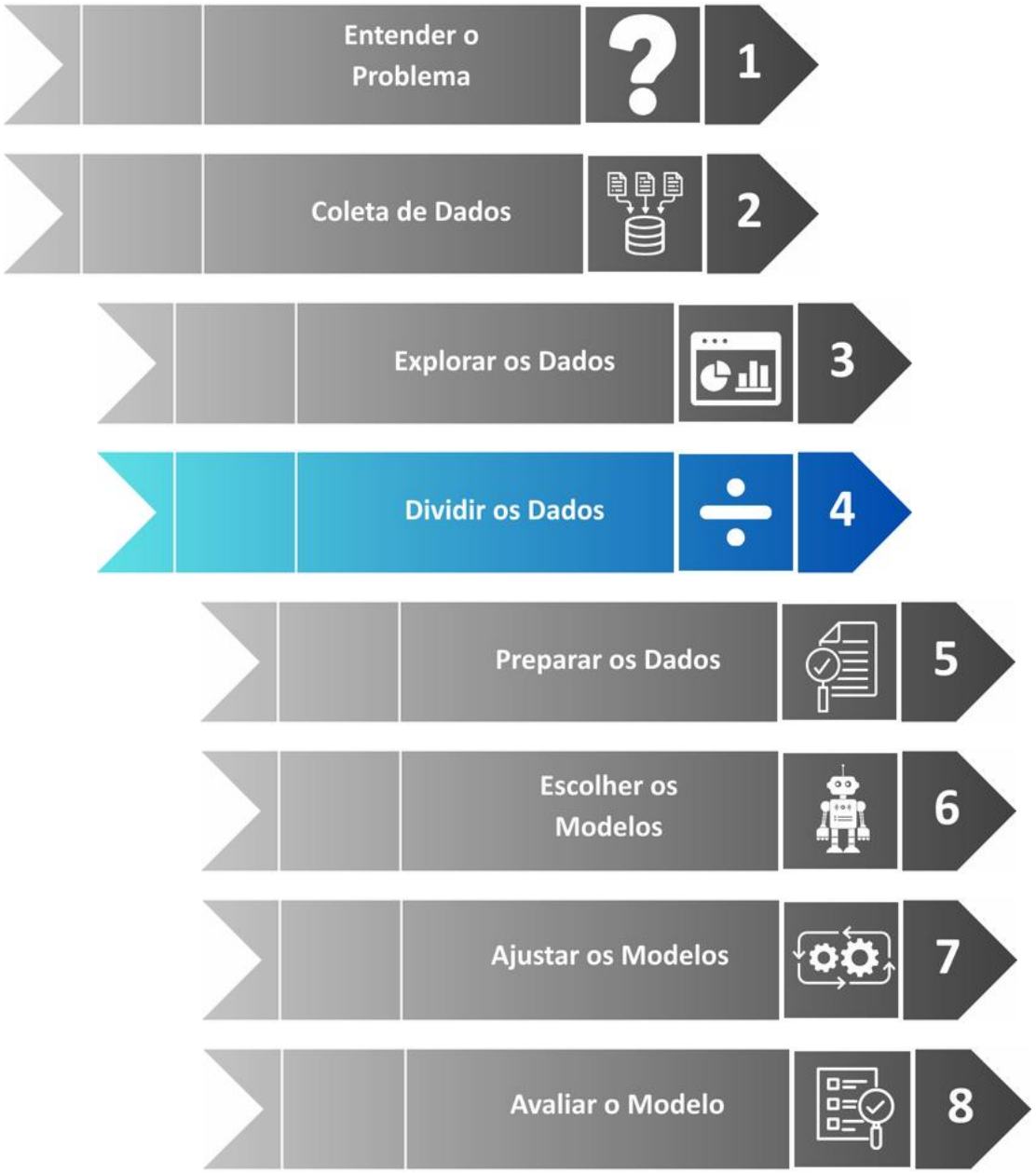
idp

Explorar os Dados

Variáveis Relevantes:

- **idade, se é fumante e região** se destacam como variáveis que possuem alguma relação com os custos.
- O que significa que **elas podem ser variáveis chave que auxiliarão os modelos na predição dos custos.**
- **Variáveis que não pussuem uma forte relação com os custos**, como sexo, podem ser **consideradas para exclusão na etapa de modelagem**, simplificando a criação do modelo, deixando mais fácil de se entender e de se explicar.

As Etapas da construção de modelos de Machine Learning



Dividir os Dados

Dividindo para Conquistar: Treino e Teste

- Imagine que você está **estudando para uma prova**. Você revisa o material, faz exercícios e se prepara.
- Em ML, fazemos algo parecido: **dividimos os dados em duas partes principais: treino e teste.**
- Essa divisão nos **ajuda a garantir que o modelo está realmente aprendendo** e não apenas "decorando" os dados.

A divisão é o passo crucial para começar a modelagem!

Dividir os Dados

Dividindo para Conquistar: Treino e Teste

- A divisão dos dados **sempre respeita alguma proporção** de forma que, a base de treinamento sempre será maior que a de teste.

- Por exemplo, **70% dos dados podem ser usados para treino e os 30% restantes para teste**, ou 80% para treino e 20% para teste.

Geralmente, usa-se a proporção 70/30!

Dividir os Dados

Dados de Treino: A Sala de Aula do Modelo

TREINO

- Os dados de **treino são usados para ensinar o modelo a reconhecer padrões** e fazer previsões.
- É como **mostrar exemplos ao modelo e corrigir seus erros até que ele aprenda a tarefa**.

E os dados de teste?

Dividir os Dados

Dados de Teste: A Prova Final do Modelo

- Os dados de teste **são usados para avaliar o desempenho do modelo depois que ele foi treinado.**
- É como fazer **uma prova para ver se o modelo realmente aprendeu a lição** e consegue generalizar para dados novos.
- O modelo **nunca deve ter visto os dados de teste durante o treinamento**, para garantir uma avaliação justa.

Dividir os Dados

Por Que Separar os Dados?

- Se usarmos os mesmos dados para treinar e testar o modelo, ele **pode simplesmente "decorar" as respostas**.
- Isso significa que ele **terá um bom desempenho nos dados de treino, mas poderá falhar** em dados novos e desconhecidos.
- A divisão em treino e teste **nos ajuda a medir a capacidade de generalização do modelo,** ou seja, sua capacidade de fazer previsões precisas em dados reais.

Existe mais problemas na modelagem?

idp

Dividir os Dados

Data Leakage (Vazamento de dados): O Inimigo Oculto

- **Vazamento de dados** acontece quando informações do futuro ou informações que você não teria na vida real são usadas para treinar o modelo.
- É como "colar" na prova: **o modelo tem acesso a informações que não deveria ter**, o que leva a resultados enganosos e superestimados.
- O Vazamento de dados **pode ser sutil e difícil de detectar, mas seus efeitos podem ser devastadores** para a precisão e confiabilidade do modelo.

Onde posso ver o Vazamento de dados?

idp

Dividir os Dados

Exemplos Comuns de Data Leakage

- **Usar dados futuros para prever o presente:** Imagine prever o preço de ações usando dados do dia seguinte.
- **Incluir informações que não estariam disponíveis no momento da previsão:** Por exemplo, usar o diagnóstico final de um paciente para prever seus sintomas iniciais.
- **Vazar informações do conjunto de teste para o conjunto de treino:** Por exemplo, usar técnicas de pré-processamento e imputação de dados em todo o conjunto de dados antes de dividir em treino e teste.

Como posso evitar isso?

idp

Dividir os Dados

Como Evitar o Data Leakage: Dicas Práticas

- **Tenha um entendimento profundo dos dados:** Conheça a origem dos dados, o significado de cada atributo e as relações entre eles.
- **Divida os dados corretamente:** Separe os conjuntos de treino e teste antes de aplicar qualquer técnica de pré-processamento.
- **Simule cenários do mundo real:** Pense em como o modelo será usado na prática e garanta que ele tenha acesso apenas às informações disponíveis nesse cenário.

Vamos praticar?

idp

Dividindo os Dados da aula_01_exemplo_01

- Vamos criar um novo script para realizar todo o passo a passo de modelagem, mas primeiro vamos carregar nossos dados.

Código

```
[1]: import pandas as pd  # manipulação de tabelas
from sklearn.model_selection import train_test_split, GridSearchCV  # divisão treino/teste + grid search
from sklearn.preprocessing import OneHotEncoder, StandardScaler  # codificação categórica e padronização numérica
from sklearn.impute import SimpleImputer # imputação dos dados
from sklearn.compose import ColumnTransformer  # aplicar transformações diferentes em colunas
from sklearn.pipeline import Pipeline  # encadeia pré-processamento + modelo
from sklearn.tree import DecisionTreeRegressor  # modelo baseado em várias árvores
from sklearn.metrics import mean_squared_error, r2_score  # métricas de regressão
import numpy as np  # biblioteca numérica

# Carregando dataset
df = pd.read_csv("aula_01_exemplo_01.csv")  # lê o arquivo aula_01_exemplo_01.csv em um DataFrame

df['tem_filhos'] = (df['children'] > 0).astype(int)  # Cria uma nova coluna 'tem_filhos' que indica se a pessoa tem filhos (1) ou não (0)
                                                     # (df['children'] > 0) cria uma Series booleana (True/False)
                                                    # .astype(int) converte True para 1 e False para 0

df.head()
```

- Note que **iremos novamente ler nossos dados** com o **read_csv()**.
- Perceba que pela **aula passada, criamos uma nova variável que identifica se um indivíduo possui ou não filhos**.

Dividindo os Dados da aula_01_exemplo_01

- Vamos criar um novo script para realizar todo o passo a passo de modelagem, mas primeiro vamos carregar nossos dados.

Código

```
[1]: import pandas as pd  # manipulação de tabelas
from sklearn.model_selection import train_test_split, GridSearchCV  # divisão treino/teste + grid search
from sklearn.preprocessing import OneHotEncoder, StandardScaler  # codificação categórica e padronização numérica
from sklearn.impute import SimpleImputer # imputação dos dados
from sklearn.compose import ColumnTransformer  # aplicar transformações diferentes em colunas
from sklearn.pipeline import Pipeline  # encadeia pré-processamento + modelo
from sklearn.tree import DecisionTreeRegressor  # modelo baseado em várias árvores
from sklearn.metrics import mean_squared_error, r2_score  # métricas de regressão
import numpy as np  # biblioteca numérica

# Carregando dataset
df = pd.read_csv("aula_01_exemplo_01.csv")  # lê o arquivo aula_01_exemplo_01.csv em um DataFrame

df['tem_filhos'] = (df['children'] > 0).astype(int)  # Cria uma nova coluna 'tem_filhos' que indica se a pessoa tem filhos (1) ou não (0)
                                                     # (df['children'] > 0) cria uma Series booleana (True/False)
                                                     # .astype(int) converte True para 1 e False para 0

df.head()
```

Qual a saída do código?

Dividindo os Dados da aula_01_exemplo_01

- Vamos criar um novo script para realizar todo o passo a passo de modelagem, mas primeiro vamos carregar nossos dados.

Saída

[1]:

	age	sex	bmi	children	smoker	region	charges	tem_filhos
0	19	female	27.900	0	yes	southwest	16884.92400	0
1	18	male	33.770	1	no	southeast	1725.55230	1
2	28	male	33.000	3	no	southeast	4449.46200	1
3	33	male	22.705	0	no	northwest	21984.47061	0
4	32	male	28.880	0	no	northwest	3866.85520	0

- Perceba que agora temos a nossa variável criada na aula passada!

idp

Dividindo os Dados da aula_01_exemplo_01

- Agora vamos separar nossos dados de forma que fique claro no Python o **que é nossa variável target e nossas covariáveis.**

Código

```
[2]: # Separando as covariáveis (X) e target (y)
X = df.drop("charges",  # drop("charges", axis=1) remove a coluna 'charges' (axis=1 indica coluna; axis=0 seria linhas);
            axis=1)  # retorna um novo DataFrame sem 'charges'
y = df["charges"]  # seleciona a coluna 'charges' como Series – esta é a variável alvo que queremos prever

X.head()
```

Saída

[2]:

	age	sex	bmi	children	smoker	region	tem_filhos
0	19	female	27.900	0	yes	southwest	0
1	18	male	33.770	1	no	southeast	1
2	28	male	33.000	3	no	southeast	1
3	33	male	22.705	0	no	northwest	0
4	32	male	28.880	0	no	northwest	0

- Perceba que agora temos a variável **X** que só possui as nossas covariáveis e a variável **y** que possui apenas a variável target.

Dividindo os Dados da aula_01_exemplo_01

- Agora vamos dividir nossos dados em **dados de treinamento contendo 80% dos dados totais** e **dados de teste com 20% dos dados totais**. Usaremos o comando **tra**

Código

```
[4]: # Divisão treino/teste

x_treino, x_teste, y_treino, y_teste = train_test_split( # função para divisão dos dados
    X,                                # covariáveis
    y,                                # variável target
    test_size=0.2,                    # 20% para teste
    random_state=42                   # Reprodutibilidade, como essa função aleatoriza os dados, vamos fixar essa aleatorização
)

print("número de linhas e colunas da base de treino:",x_treino.shape)
print("número de linhas e colunas da base de teste:",x_teste.shape)
```

Saída

```
número de linhas e colunas da base de treino: (1070, 7)
número de linhas e colunas da base de teste: (268, 7)
```

- Note que dentro da função, precisamos dizer primeiro qual nossa base **X** de covariáveis, seguida pela base **y** de target.

Dividindo os Dados da aula_01_exemplo_01

- Agora vamos dividir nossos dados em **dados de treinamento contendo 80% dos dados totais** e **dados de teste com 20% dos dados totais**. Usaremos o comando **tra**

Código

```
[4]: # Divisão treino/teste

x_treino, x_teste, y_treino, y_teste = train_test_split( # função para divisão dos dados
    X,                                # covariáveis
    y,                                # variável target
    test_size=0.2,                    # 20% para teste
    random_state=42                   # Reprodutibilidade, como essa função aleatoriza os dados, vamos fixar essa aleatorização
)

print("número de linhas e colunas da base de treino:",x_treino.shape)
print("número de linhas e colunas da base de teste:",x_teste.shape)
```

Saída

```
número de linhas e colunas da base de treino: (1070, 7)
número de linhas e colunas da base de teste: (268, 7)
```

- Depois definimos o tamanho do teste em **test_size = 0.2**, isto é quero que 20% dos meus indivíduos estejam na base de dados de teste.

Dividindo os Dados da aula_01_exemplo_01

- Agora vamos dividir nossos dados em **dados de treinamento contendo 80% dos dados totais** e **dados de teste com 20% dos dados totais**. Usaremos o comando **tra**

Código

```
[4]: # Divisão treino/teste

x_treino, x_teste, y_treino, y_teste = train_test_split( # função para divisão dos dados
    X,                                # covariáveis
    y,                                # variável target
    test_size=0.2,                    # 20% para teste
    random_state=42                   # Reprodutibilidade, como essa função aleatoriza os dados, vamos fixar essa aleatorização
)

print("número de linhas e colunas da base de treino:",x_treino.shape)
print("número de linhas e colunas da base de teste:",x_teste.shape)
```

Saída

```
número de linhas e colunas da base de treino: (1070, 7)
número de linhas e colunas da base de teste: (268, 7)
```

- **Poderíamos ter dividido os dados em 70% para treinamento e 30% para teste. Porém, considerando que temos apenas 1.338 observações**, a divisão reduziria o conjunto de treinamento, comprometendo a capacidade do modelo de aprender adequadamente.

Dividindo os Dados da aula_01_exemplo_01

- Agora vamos dividir nossos dados em **dados de treinamento contendo 80% dos dados totais** e **dados de teste com 20% dos dados totais**. Usaremos o comando **tra**

Código

```
[4]: # Divisão treino/teste

x_treino, x_teste, y_treino, y_teste = train_test_split( # função para divisão dos dados
    X,                                  # covariáveis
    y,                                  # variável target
    test_size=0.2,                      # 20% para teste
    random_state=42                     # Reprodutibilidade, como essa função aleatoriza os dados, vamos fixar essa aleatorização
)

print("número de linhas e colunas da base de treino:",x_treino.shape)
print("número de linhas e colunas da base de teste:",x_teste.shape)
```

Saída

```
número de linhas e colunas da base de treino: (1070, 7)
número de linhas e colunas da base de teste: (268, 7)
```

- Note também que no python eu posso definir **4 variáveis de uma vez!**

idp

Dividindo os Dados da aula_01_exemplo_01

- Agora vamos dividir nossos dados em **dados de treinamento contendo 80% dos dados totais** e **dados de teste com 20% dos dados totais**. Usaremos o comando **tra**

Código

```
[4]: # Divisão treino/teste

x_treino, x_teste, y_treino, y_teste = train_test_split( # função para divisão dos dados
    X,                                  # covariáveis
    y,                                  # variável target
    test_size=0.2,                      # 20% para teste
    random_state=42                     # Reprodutibilidade, como essa função aleatoriza os dados, vamos fixar essa aleatorização
)

print("número de linhas e colunas da base de treino:",x_treino.shape)
print("número de linhas e colunas da base de teste:",x_teste.shape)
```

Saída

```
número de linhas e colunas da base de treino: (1070, 7)
número de linhas e colunas da base de teste: (268, 7)
```

- **Cuidado!** Comece **SEMPRE** definindo as variáveis segundo a ordem acima, **ou seja primeiro adicione o x_treino, logo em seguida o x_teste, seguido pelo y_treino e por fim o y_teste.**

idp

Dividindo os Dados da aula_01_exemplo_01

- Agora vamos dividir nossos dados em **dados de treinamento contendo 80% dos dados totais** e **dados de teste com 20% dos dados totais**. Usaremos o comando **tra**

Código

```
[4]: # Divisão treino/teste

x_treino, x_teste, y_treino, y_teste = train_test_split( # função para divisão dos dados
    X,                                  # covariáveis
    y,                                  # variável target
    test_size=0.2,                      # 20% para teste
    random_state=42                     # Reprodutibilidade, como essa função aleatoriza os dados, vamos fixar essa aleatorização
)

print("número de linhas e colunas da base de treino:",x_treino.shape)
print("número de linhas e colunas da base de teste:",x_teste.shape)
```

Saída

```
número de linhas e colunas da base de treino: (1070, 7)
número de linhas e colunas da base de teste: (268, 7)
```

- O **random_state** um número que faz a divisão dos **dados em treino e teste sempre da mesma forma**. Assim, mesmo rodando o código várias vezes, você terá os mesmos conjuntos, garantindo que os resultados sejam consistentes e comparáveis.

idp

Dividindo os Dados da aula_01_exemplo_01

- Agora vamos dividir nossos dados em **dados de treinamento contendo 80% dos dados totais** e **dados de teste com 20% dos dados totais**. Usaremos o comando **tra**

Código

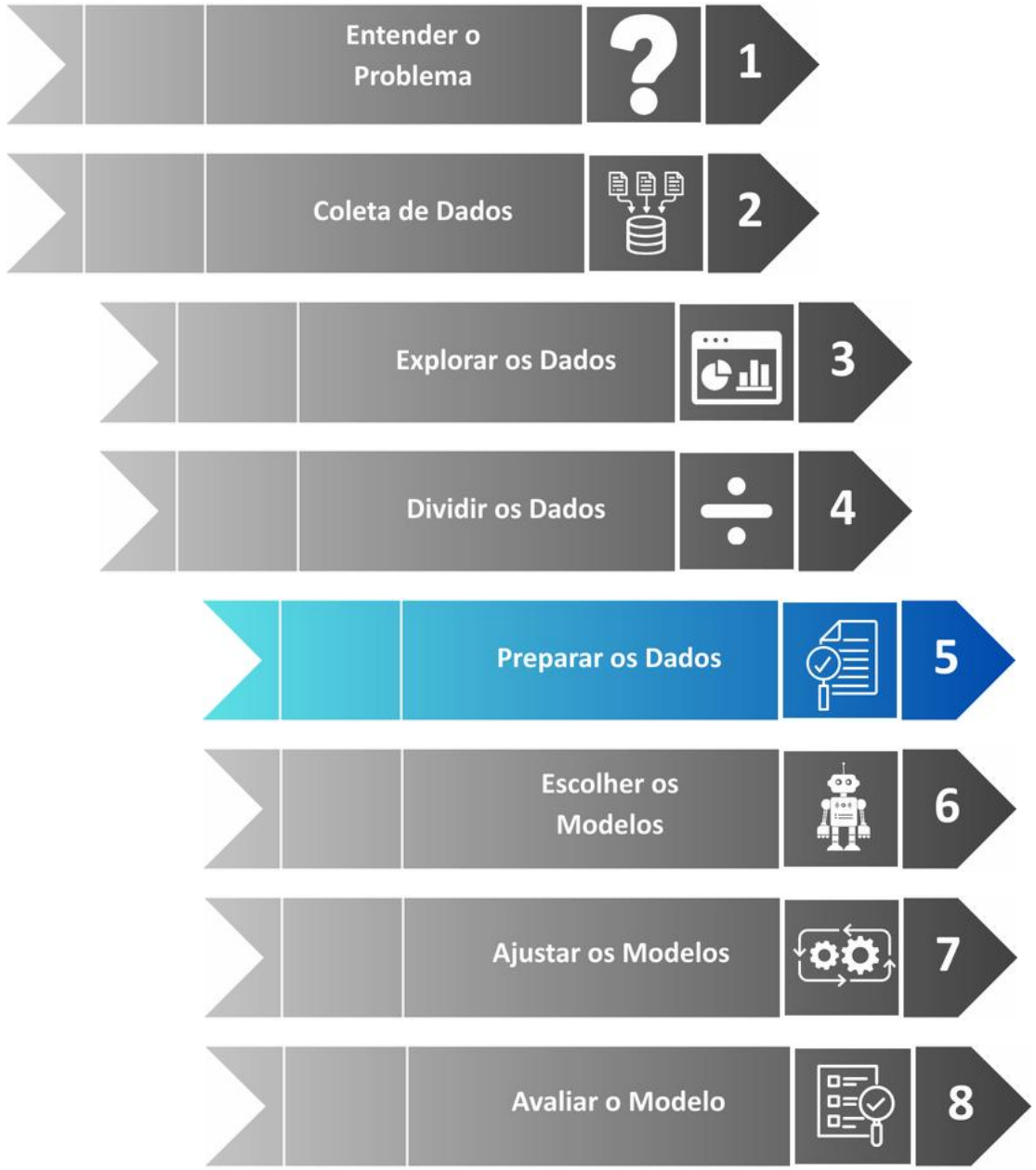
```
[4]: # Divisão treino/teste

x_treino, x_teste, y_treino, y_teste = train_test_split( # função para divisão dos dados
    X,                                # covariáveis
    y,                                # variável target
    test_size=0.2,                    # 20% para teste
    random_state=42                   # Reprodutibilidade, como essa função aleatoriza os dados, vamos fixar essa aleatorização
)

print("número de linhas e colunas da base de treino:",x_treino.shape)
print("número de linhas e colunas da base de teste:",x_teste.shape)
```

Saída

```
número de linhas e colunas da base de treino: (1070, 7)
número de linhas e colunas da base de teste: (268, 7)
```

- Agora conseguimos dividir nossa base de dados! **Assim 20% dos nossos dados (268 indivíduos) estão nas bases de teste (x_teste e y_teste)** e 80% (1070 indivíduos) estão nas bases de treino (x_treino,y_treino).

As Etapas da construção de modelos de Machine Learning



Preparar os Dados

Preparar os Dados (De Matéria Bruta a Ouro)

- **Os dados raramente estão perfeitos**. Precisamos limpá-los, transformá-los e formatá-los para que o modelo possa aprender com eles.

- Este passo é um dos **mais importantes e demorados em um projeto de ML**.

- Uma boa preparação dos dados pode **fazer toda a diferença na precisão** e desempenho do modelo.

E como preparamos os dados?

idp

Preparar os Dados

Limpeza dos Dados

- A limpeza de dados **garante que o modelo não seja influenciado** por informações incorretas ou incompletas.
- Sempre **utilize da análise descritiva preliminar** em busca de inconsistências e valores ausentes.
- **Precisamos de metodologias** para imputar valores ausentes.

Mas o que é imputar?

Preparar os Dados

Imputação de Dados: Preenchendo as Lacunas

- **Imputar dados significa substituir valores ausentes** em um conjunto de dados por valores estimados.
- É como **"tapar os buracos" em uma planilha** para que todas as células tenham um valor.
- Isso é importante porque muitos modelos de Machine Learning **não funcionam bem com dados faltantes**.

Como eu faria para os dados numéricos?

idp

Preparar os Dados

Técnicas Comuns de Imputação

- **Média/Mediana:** Substituir o valor ausente pela média ou mediana dos valores da coluna.

- **Remover Linhas/Colunas:** Excluir as linhas ou colunas que contêm valores ausentes (use com cautela!).

Mas qual escolher?

Preparar os Dados

Escolhendo a Técnica de Imputação Certa

- **Média:** Use quando os dados são simétricos(como o BMI).
- **Mediana:** Use quando os dados são assimétricos (como a variável charges).
- **Categoria Mais Frequente (Moda):** Use para dados categóricos, substituindo o valor ausente pela categoria que aparece com mais frequência.
- A **escolha da técnica deve ser baseada na análise dos dados** e no conhecimento do domínio do problema.

Só precisa imputar?

idp

Preparar os Dados

Escalonamento de Atributos: Nivelando o Jogo

- Imagine que você está **comparando o desempenho de atletas em diferentes esportes**. Um corre em metros, outro nada em segundos, outro arremessa em quilômetros.

- Para comparar de forma justa, **precisamos colocar todos na mesma unidade de medida**! Isso é o que o escalonamento faz com os dados.

- **Escalonar os atributos significa colocar todos na mesma escala**, como transformar tudo para porcentagens.

- Isso evita que **atributos com valores muito grandes "dominem" o aprendizado do modelo**, garantindo que todos os atributos contribuam igualmente.

E como o Escalonamento funciona para os dados categóricos?

Preparar os Dados

One-Hot Encoding: Transformando Categorias em Números

- Muitos modelos de Machine Learning **só conseguem trabalhar com números**.
- Mas e se tivermos **atributos categóricos (cores, nomes, tipos de produto)**?
- É aí que entra o One-Hot Encoding: **transformamos cada categoria em uma coluna binária** (0 ou 1).

Como essa técnica funciona?

idp

Preparar os Dados

Como Funciona o One-Hot Encoding?

- Imagine que temos um atributo **"Cor" com os valores: "Vermelho", "Azul", "Verde"**.
- Criamos **três novas colunas: "Cor_Vermelho", "Cor_Azul", "Cor_Verde"**.
- Se a cor for **"Vermelho", a coluna "Cor_Vermelho" recebe 1** e as outras recebem 0.

Vamos visualizar esse processo?

idp

Preparar os Dados

Como Funciona o One-Hot Encoding?

Base de dados

ID	Cor
Individuo 1	vermelho
Individuo 2	azul
Individuo 3	verde

Base transformada

ID	Cor_vermelho	Cor_azul	Cor_verde
Individuo 1	1	0	0
Individuo 2	0	1	0
Individuo 3	0	0	1

Preparar os Dados

One-Hot Encoding: A Armadilha da Redundância

- Ao usar One-Hot Encoding, **é comum remover uma das colunas criadas**. Por quê?
- Se temos as cores "Vermelho", "Azul" e "Verde", **precisamos apenas de duas colunas**: "Cor_Vermelho" e "Cor_Azul".
- **Se ambas forem 0, sabemos que a cor é "Verde"**. A coluna "Cor_Verde" se torna redundante e pode ser removida.
- Poderíamos remover qualquer outra coluna (**Cor_Vermelho** ou **Cor_Azul**), a escolha é arbitrária e passível de mudança, já que o objetivo é evitar a redundância de informação.

Vamos concertar na base de exemplo?

Preparar os Dados

Como Funciona o One-Hot Encoding?

Base de dados

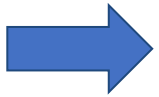
ID	Cor
Individuo 1	vermelho
Individuo 2	azul
Individuo 3	verde

Base transformada

ID	Cor_vermelho	Cor_azul
Individuo 1	1	0
Individuo 2	0	1
Individuo 3	0	0

Preparar os Dados

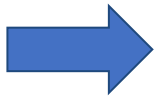
Como Funciona o One-Hot Encoding?



Base de dados

ID	Cor
Individuo 1	vermelho
Individuo 2	azul
Individuo 3	verde

Ou

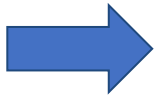


Base transformada

ID	Cor_azul	Cor_verde
Individuo 1	0	0
Individuo 2	1	0
Individuo 3	0	1

Preparar os Dados

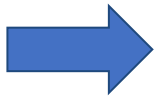
Como Funciona o One-Hot Encoding?



Base de dados

ID	Cor
Individuo 1	vermelho
Individuo 2	azul
Individuo 3	verde

Ou



Base transformada

ID	Cor_vermelho	Cor_verde
Individuo 1	1	0
Individuo 2	0	0
Individuo 3	0	1

Vamos Praticar?

Preparando os Dados da aula_01_exemplo_01

- Vamos criar cada processo de imputação e transformação de dados.

Código

```
[5]: # Pré-processamento

escalonador = StandardScaler() # Escalonador, ele vai padronizar as variáveis numéricas

categorizador = OneHotEncoder(drop="first",  # O categorizador vai Remover a primeira dummy para evitar redundância
                              handle_unknown="ignore") # Ignora categorias desconhecidas em dados de teste

imputador_numerico = SimpleImputer(strategy="median")       # imputador numérico usando mediana

imputador_categorico = SimpleImputer(strategy="most_frequent")  # imputador categórico usando a moda
```

- Como **temos variáveis de tipos diferentes** (idades com valores entre 18 anos e 64 anos, custos em doláres etc), se faz necessário realizar a padronização.
- O comando **StandardScaler()** vai criar o Escalonador que será usado para padronizar nossas variáveis numéricas.

Preparando os Dados da aula_01_exemplo_01

- Vamos criar cada processo de imputação e transformação de dados.

Código

```
[5]: # Pré-processamento

escalonador = StandardScaler() # Escalonador, ele vai padronizar as variáveis numéricas

categorizador = OneHotEncoder(drop="first",  # O categorizador vai Remover a primeira dummy para evitar redundância
                              handle_unknown="ignore") # Ignora categorias desconhecidas em dados de teste

imputador_numerico = SimpleImputer(strategy="median")       # imputador numérico usando mediana

imputador_categorico = SimpleImputer(strategy="most_frequent")  # imputador categórico usando a moda
```

- Como **temos variáveis categóricas** (região, se é fumante ou não por exemplo), vamos ter que categorizar via **One Hot Encoder.**
- O comando **OneHotEncoder()** vai criar o categorizador que será usado para padronizar nossas variáveis categóricas.

Preparando os Dados da aula_01_exemplo_01

- Vamos criar cada processo de imputação e transformação de dados.

Código

```
[5]: # Pré-processamento

escalonador = StandardScaler() # Escalonador, ele vai padronizar as variáveis numéricas

categorizador = OneHotEncoder(drop="first",  # O categorizador vai Remover a primeira dummy para evitar redundância
                              handle_unknown="ignore") # Ignora categorias desconhecidas em dados de teste

imputador_numerico = SimpleImputer(strategy="median")       # imputador numérico usando mediana

imputador_categorico = SimpleImputer(strategy="most_frequent")  # imputador categórico usando a moda
```

- **Note que criamos um imputador para os dados numéricos e categóricos** mesmo não havendo valores ausentes. Por que?
- Isso garante que, **caso novos dados futuros contenham valores ausentes, o pré-processamento funcione automaticamente sem gerar erros.**

Preparando os Dados da aula_01_exemplo_01

- Vamos criar cada processo de imputação e transformação de dados.

Código

```
[5]: # Pré-processamento

escalonador = StandardScaler() # Escalonador, ele vai padronizar as variáveis numéricas

categorizador = OneHotEncoder(drop="first",  # O categorizador vai Remover a primeira dummy para evitar redundância
                              handle_unknown="ignore") # Ignora categorias desconhecidas em dados de teste

imputador_numerico = SimpleImputer(strategy="median")       # imputador numérico usando mediana

imputador_categorico = SimpleImputer(strategy="most_frequent")  # imputador categórico usando a moda
```

- **E se os dados estivessem em uma escala comum? seria necessário padronizar?**

- **Não necessariamente**, a padronização serve para deixar o modelo mais “justo” na hora de prever novos valores. Se os dados já estiverem em uma escala comum, essa etapa poderia ser pulada.

Preparando os Dados da aula_01_exemplo_01

- Vamos criar cada processo de imputação e transformação de dados.

Código

```
[5]: # Pré-processamento

escalonador = StandardScaler() # Escalonador, ele vai padronizar as variáveis numéricas

categorizador = OneHotEncoder(drop="first",  # O categorizador vai Remover a primeira dummy para evitar redundância
                              handle_unknown="ignore") # Ignora categorias desconhecidas em dados de teste

imputador_numerico = SimpleImputer(strategy="median")       # imputador numérico usando mediana

imputador_categorico = SimpleImputer(strategy="most_frequent")  # imputador categórico usando a moda
```

- Note que utilizamos o parâmetro **drop = "first"** para evitar redundância, como aprendemos!
- O comando **handle_unknown = "ignore"** é utilizado para lidar com categorias que nunca foram vistas antes. Ele irá ignorá-las.

idp

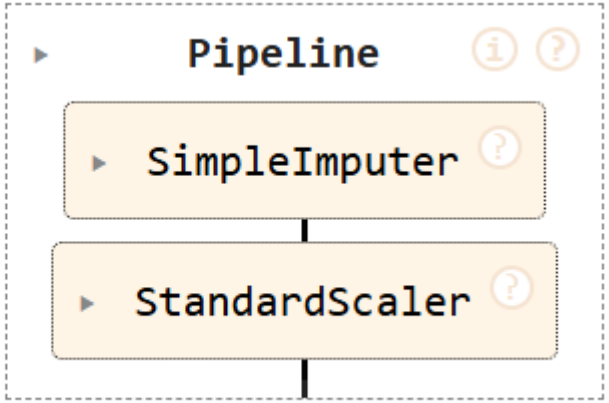
Preparando os Dados da aula_01_exemplo_01

- Agora **vamos juntar esses comandos,** de forma que fique uma etapa para os dados numéricos (imputação e padronização) e outra etapa para os dado categóricos (imputação e categorização).

Código

```
[6]: # Etapas de transformação das variáveis numéricas
etapas_numericas = Pipeline(
    [
        ("imputer", imputador_numerico),     # "imputer" é o nome da etapa → aplica a imputação (substitui valores ausentes pela mediana)
        ("scaler", escalonador)              # "scaler" é o nome da etapa → aplica padronização
    ]
)
etapas_numericas
```

Saída



- Com o comando **Pipeline()**, agora criamos um passo a passo apenas para as variáveis numérica.
- Dentro do Pipeline, adicionamos o **imputador_numerico** e especificamos que é um **“imputer”** (imputador)

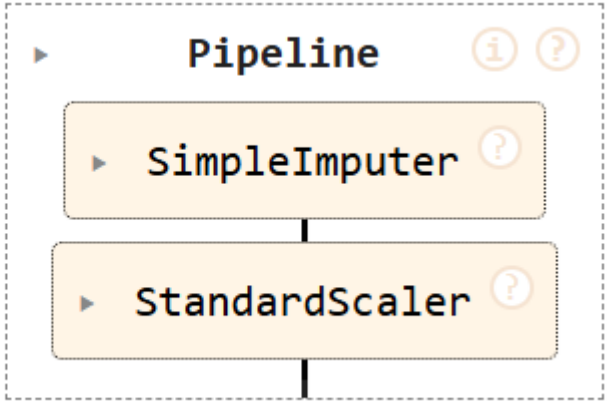
Preparando os Dados da aula_01_exemplo_01

- Agora **vamos juntar esses comandos**, de forma que fique uma etapa para os dados numéricos (imputação e padronização) e outra etapa para os dado categóricos (imputação e categorização).

Código

```
[6]: # Etapas de transformação das variáveis numéricas
etapas_numericas = Pipeline(
    [
        ("imputer", imputador_numerico),     # "imputer" é o nome da etapa → aplica a imputação (substitui valores ausentes pela mediana)
        ("scaler", escalonador)              # "scaler" é o nome da etapa → aplica padronização
    ]
)
etapas_numericas
```

Saída



- Adicionamos também o escalonador e especificamos que é um **"scaler"** (escalonador).
- Assim conseguimos dizer pro python que **queremos imputar e depois escalonar nossos dados numéricos**.

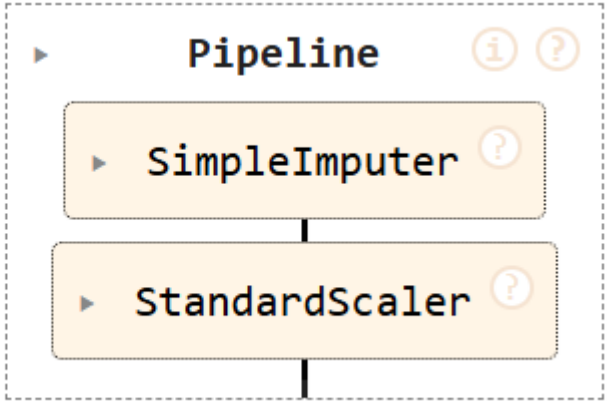
Preparando os Dados da aula_01_exemplo_01

- Agora **vamos juntar esses comandos**, de forma que fique uma etapa para os dados numéricos (imputação e padronização) e outra etapa para os dado categóricos (imputação e categorização).

Código

```
[6]: # Etapas de transformação das variáveis numéricas
etapas_numericas = Pipeline(
    [
        ("imputer", imputador_numerico),     # "imputer" é o nome da etapa → aplica a imputação (substitui valores ausentes pela mediana)
        ("scaler", escalonador)              # "scaler" é o nome da etapa → aplica padronização
    ]
)
etapas_numericas
```

Saída



- **A figura a esquerda nos dá um resumo da estruturação** dentro da nossa etapa implementada.
- Nela conseguimos **visualizar quais são componentes inseridos na etapa**.

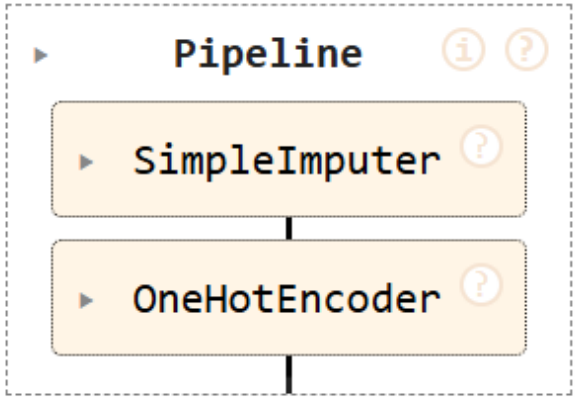
Preparando os Dados da aula_01_exemplo_01

- Vamos fazer a mesma coisa para os dados categóricos.

Código

```
[7]: # Etapas de transformação das variáveis categóricas
etapas_categoricas = Pipeline(
    [
        ("imputer", imputador_categorico), # "imputer" é o nome da etapa → aplica a imputação (substitui valores ausentes pela moda)
        ("encoder", categorizador)         # "encoder" é o nome da etapa → aplica OneHotEncoder
    ]
)

etapas_categoricas
```

Saída



- Com o comando **Pipeline()**, agora criamos um passo a passo apenas para as variáveis categóricas.
- Dentro do Pipeline, adicionamos o **imputador_categorico** e especificamos que é um **"imputer"** (imputador)

Preparando os Dados da aula_01_exemplo_01

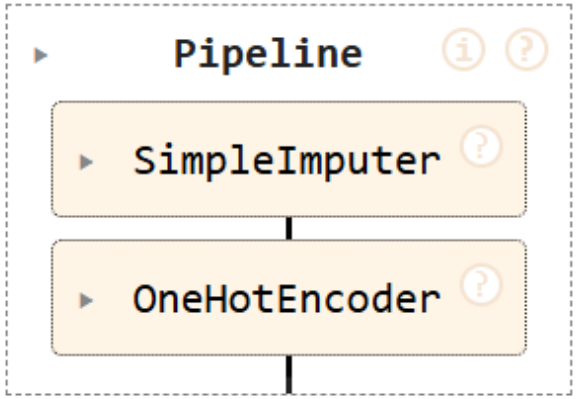
- Vamos fazer a mesma coisa para os dados categóricos.

Código

```
[7]: # Etapas de transformação das variáveis categóricas
etapas_categoricas = Pipeline(
    [
        ("imputer", imputador_categorico), # "imputer" é o nome da etapa → aplica a imputação (substitui valores ausentes pela moda)
        ("encoder", categorizador)         # "encoder" é o nome da etapa → aplica OneHotEncoder
    ]
)

etapas_categoricas
```

Saída

[7]:



- Adicionamos também o categorizador e especificamos que é um "**encoder**" (codificador).
- Assim conseguimos dizer pro python que **queremos imputar e depois categorizar nossos dados categóricos.**

Preparando os Dados da aula_01_exemplo_01

- **Agora vamos juntar tudo em um comando só** e especificar onde as variáveis vão entrar.

Código

```
[8]: # Juntando todo o processamento das variáveis categóricas e numéricas
preprocessador = ColumnTransformer(
    [
        ("num", etapas_numericas,  variaveis_numericas), # Nome da etapa, Escalonador e a lista de variáveis numéricas
        ("cat", etapas_categoricas, variaveis_categoricas) # Nome da etapa, cateogorizador e a lista de variáveis categóricas
    ]
)

x_treino_transformado = preprocessador.fit_transform(x_treino) # aplicando a imputação e transformação nos dados de treino
x_teste_transformado = preprocessador.transform(x_teste) # aplicando a imputação e transformação nos dados de teste
```

- Com o comando **ColumnTransformer()** vamos juntar as duas etapas que criarmos e enfim criar um fluxo onde nossos dados entram brutos e saem padronizados.

- Para cada etapa, vamos especificar o nome (**“num”** para a etapa numérica e **“cat”** para a etapa categórica) e o objeto que foi criado antes (**etapas_numericas** e **etapas_categoricas**)

Preparando os Dados da aula_01_exemplo_01

- **Agora vamos juntar tudo em um comando só** e especificar onde as variáveis vão entrar.

Código

```
[8]: # Juntando todo o processamento das variáveis categóricas e numéricas
preprocessador = ColumnTransformer(
    [
        ("num", etapas_numericas,  variaveis_numericas), # Nome da etapa, Escalonador e a lista de variáveis numéricas
        ("cat", etapas_categoricas, variaveis_categoricas) # Nome da etapa, cateogorizador e a lista de variáveis categóricas
    ]
)

x_treino_transformado = preprocessador.fit_transform(x_treino) # aplicando a imputação e transformação nos dados de treino
x_teste_transformado = preprocessador.transform(x_teste) # aplicando a imputação e transformação nos dados de teste
```

- Por fim, **especificamos quais colunas serão usadas em cada etapa** (lembra que armazenamos antes da divisão dos dados?)

- Com tudo pronto basta utilizar a função **.fit_transform(x_treino)** nos dados de treino, assim ele irá armazenar as informações necessárias para imputar e transformar os dados.

idp

Preparando os Dados da aula_01_exemplo_01

- **Agora vamos juntar tudo em um comando só** e especificar onde as variáveis vão entrar.

Código

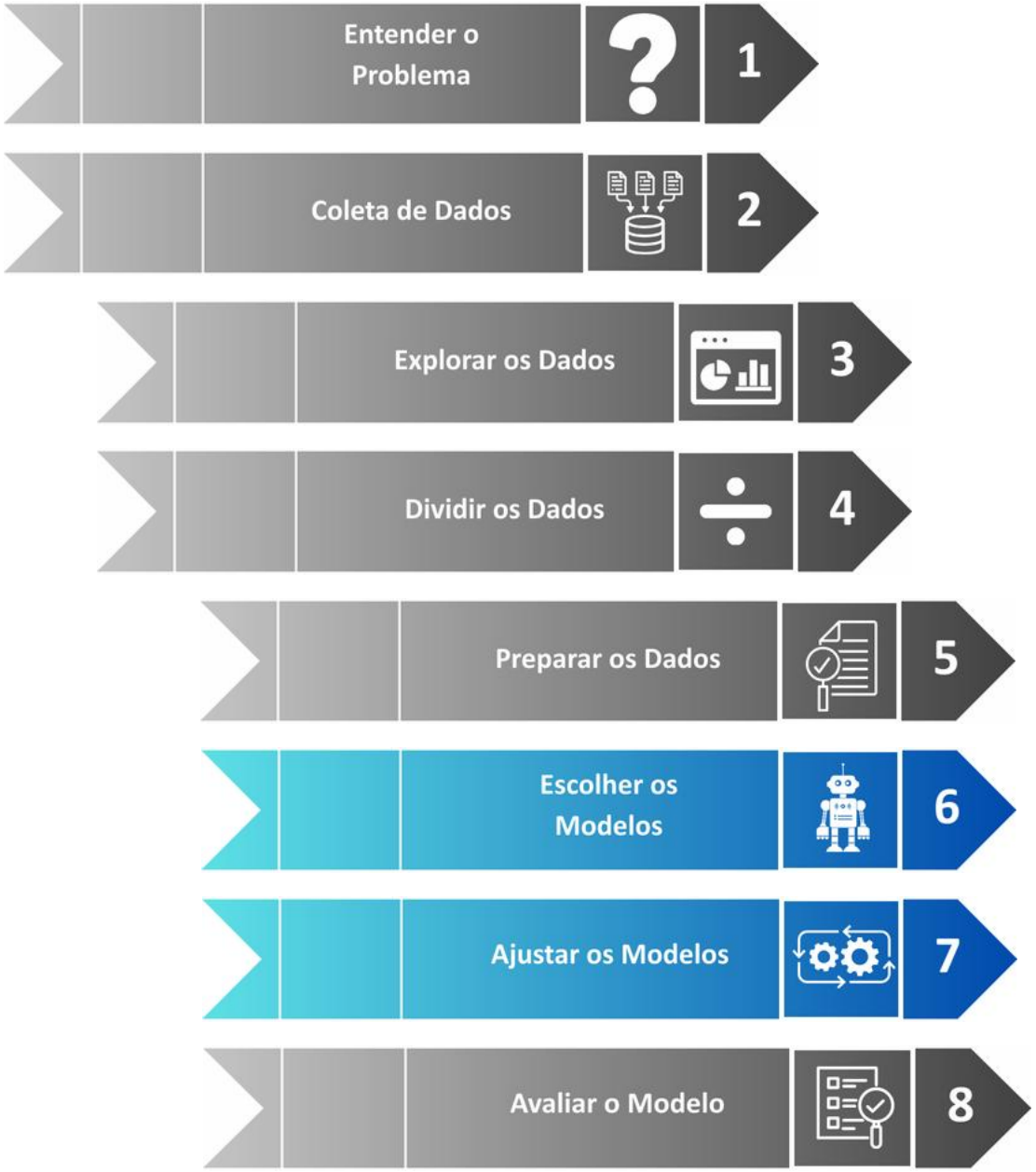
```
[8]: # Juntando todo o processamento das variáveis categóricas e numéricas
preprocessador = ColumnTransformer(
    [
        ("num", etapas_numericas,  variaveis_numericas), # Nome da etapa, Escalonador e a lista de variáveis numéricas
        ("cat", etapas_categoricas, variaveis_categoricas) # Nome da etapa, cateogorizador e a lista de variáveis categóricas
    ]
)

x_treino_transformado = preprocessador.fit_transform(x_treino) # aplicando a imputação e transformação nos dados de treino
x_teste_transformado = preprocessador.transform(x_teste) # aplicando a imputação e transformação nos dados de teste
```

- **Cuidado!** Lembre-se que se eu adicionar informações da base de dados de treino no teste, estaremos **cometendo vazamento de dados!**

- Por isso vamos sempre utilizar o comando **.fit_transform()** no x_treino e na base de dados de teste apenas utilize o **.transform()**, já que as informações já foram extraídas na base de treino

O código acima não retorna nada pra gente!

As Etapas da construção de modelos de Machine Learning



Escolher e Ajustar os Modelos

Escolher Modelos:

- Existem muitos **modelos de Machine Learning diferentes**, cada um com suas próprias características e vantagens.
- Precisamos **experimentar diferentes modelos para encontrar aquele que melhor se adapta ao nosso problema** e aos nossos dados.
- Este passo **envolve tanto a escolha dos modelos quanto a melhor escolha dos hiperparâmetros**.

Mas o que são Hiperparâmetros?

idp

Escolher e Ajustar os Modelos

Hiperparâmetros: As Configurações dos Modelos

- Modelos de Machine Learning têm **"configurações" internas** que controlam como eles aprendem.
- Essas **configurações são chamadas de hiperparâmetros**.
- Ajustar os hiperparâmetros **é como afinar um instrumento musical** para obter o som perfeito.
- Hiperparâmetros diferentes podem **levar a modelos com desempenhos muito diferentes**.

Precisamos encontrar a combinação ideal de hiperparâmetros para cada modelo!

idp

Escolher e Ajustar os Modelos

Exemplo: Cozinhar o Bolo Perfeito

- Imagine que você está **tentando fazer um bolo perfeito**.Você tem a receita (o modelo), mas precisa ajustar algumas coisas:
- **Tempo de forno:** Se deixar muito tempo, o bolo queima. Se deixar pouco tempo, fica cru.
- **Quantidade de açúcar:** Se colocar muito açúcar, fica enjoativo. Se colocar pouco, fica sem graça.
- Ajustar esses "hiperparâmetros" é **essencial para obter o bolo perfeito**! Em ML, é a mesma coisa: encontrar os valores ideais para os hiperparâmetros garante o melhor desempenho do modelo.

mas como temos tantos modelos, como eu posso encontrar esses melhores hiperparâmetros?

Escolher e Ajustar os Modelos

Grid Search: Explorando Todas as Possibilidades

- O Grid Search é uma forma de **encontrar os melhores hiperparâmetros** para um modelo.
- Definimos **uma "grade" de valores possíveis para cada hiperparâmetro que queremos ajustar**.
- O Grid Search **testa todas as combinações possíveis de valores da grade**, treinando e avaliando o modelo para cada combinação.

Vamos entender com um exemplo!

idp

Escolher e Ajustar os Modelos

Como Funciona o Grid Search

- Imagine que temos dois hiperparâmetros: A (com valores 1, 2, 3) e B (com valores 4, 5).
- **Quantas combinações possíveis podemos fazer?**
- O Grid Search testa as seguintes combinações: **(1, 4), (1, 5), (2, 4), (2, 5), (3, 4), (3, 5)**.
- Para cada combinação, treinamos o modelo e avaliamos seu desempenho usando validação cruzada.

O que é validação cruzada?

idp

Escolher e Ajustar os Modelos

Validação Cruzada: Testando o Modelo em Diferentes "Provas"

- A validação cruzada é uma **técnica para avaliar o desempenho do modelo de forma mais robusta e confiável**.
- Em vez de **dividir os dados apenas em treino e teste, dividimos os dados de treino em várias partes** (folds).
- Usamos **uma parte para teste e as outras para treino**, repetindo o processo para cada parte.

Vamos entender com um exemplo!

idp

Escolher e Ajustar os Modelos

Validação Cruzada: Testando o Modelo em Diferentes "Provas"

- Vamos dividir os **dados de treino** em k = 5 folds (pedaços).

Escolher e Ajustar os Modelos

Validação Cruzada: Testando o Modelo em Diferentes "Provas"

- Na primeira repetição, **vamos colocar o pedaço “5” como base de teste e os restantes como base de treinamento**.

- Após a divisão, **avaliamos o modelo com base nessa primeira configuração**.

Escolher e Ajustar os Modelos

Validação Cruzada: Testando o Modelo em Diferentes "Provas"

- Na segunda repetição, vamos colocar o pedaço “4” como teste e os demais como treino. Repetimos esse processo até que todos os pedaços sejam eventualmente usados no treino e no teste.

Repetição 1	1 - Treino	2 - Treino	3 - Treino	4 - Treino	5 - Teste
Repetição 2	1 - Treino	2 - Treino	3 - Treino	4 - Teste	5 - Treino
Repetição 3	1 - Treino	2 - Treino	3 - Teste	4 - Treino	5 - Treino
Repetição 4	1 - Treino	2 - Teste	3 - Treino	4 - Treino	5 - Treino
Repetição 5	1 - Teste	2 - Treino	3 - Treino	4 - Treino	5 - Treino

Escolher e Ajustar os Modelos

Por Que a Validação Cruzada é Importante?

- Ela nos dá uma **estimativa mais precisa do desempenho do modelo**.
- Ajuda a **evitar o overfitting** (quando o modelo "decora" os dados de treino).
- Permite **comparar diferentes modelos e configurações de hiperparâmetros de forma mais justa**.

Mas o que é o overfitting?

Escolher e Ajustar os Modelos

Overfitting: Decorando a Prova em Vez de Aprender

- Overfitting acontece quando o modelo aprende os dados de treino "de cor", incluindo o ruído e os detalhes irrelevantes.
- É como um aluno que decora as respostas da lista de exercícios, mas não entende a matéria e se dá mal na prova.
- O modelo tem um ótimo desempenho nos dados de treino, mas um desempenho ruim em dados novos (não vistos).

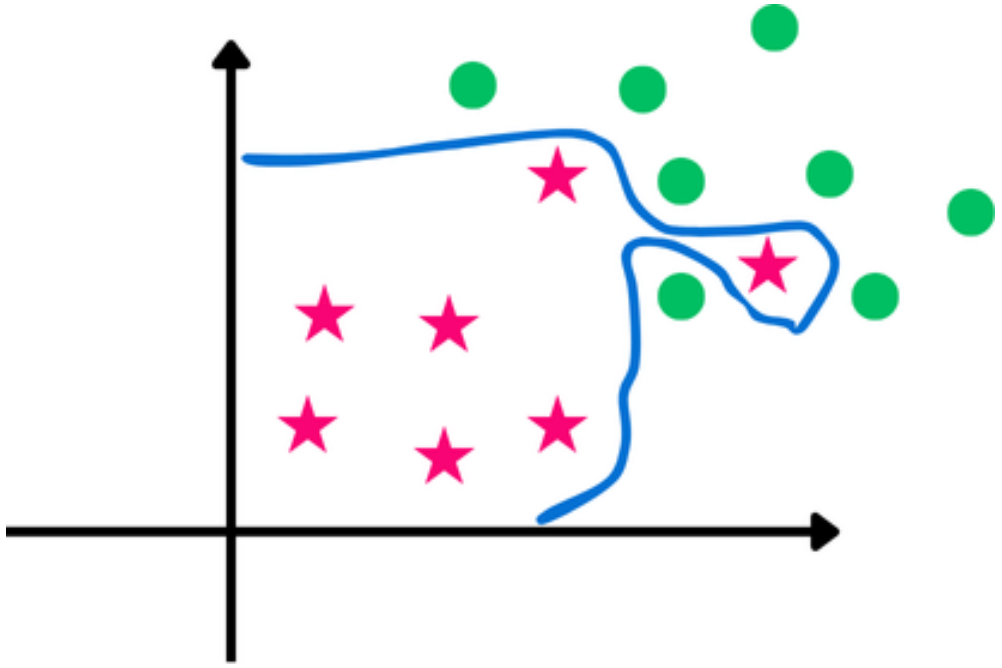
Vamos entender como o Overfitting funciona!

idp

Escolher e Ajustar os Modelos

Exemplo 1:

- Imagine que essa imagem representa um modelo tentando separar "estrelas" de "bolinhas".

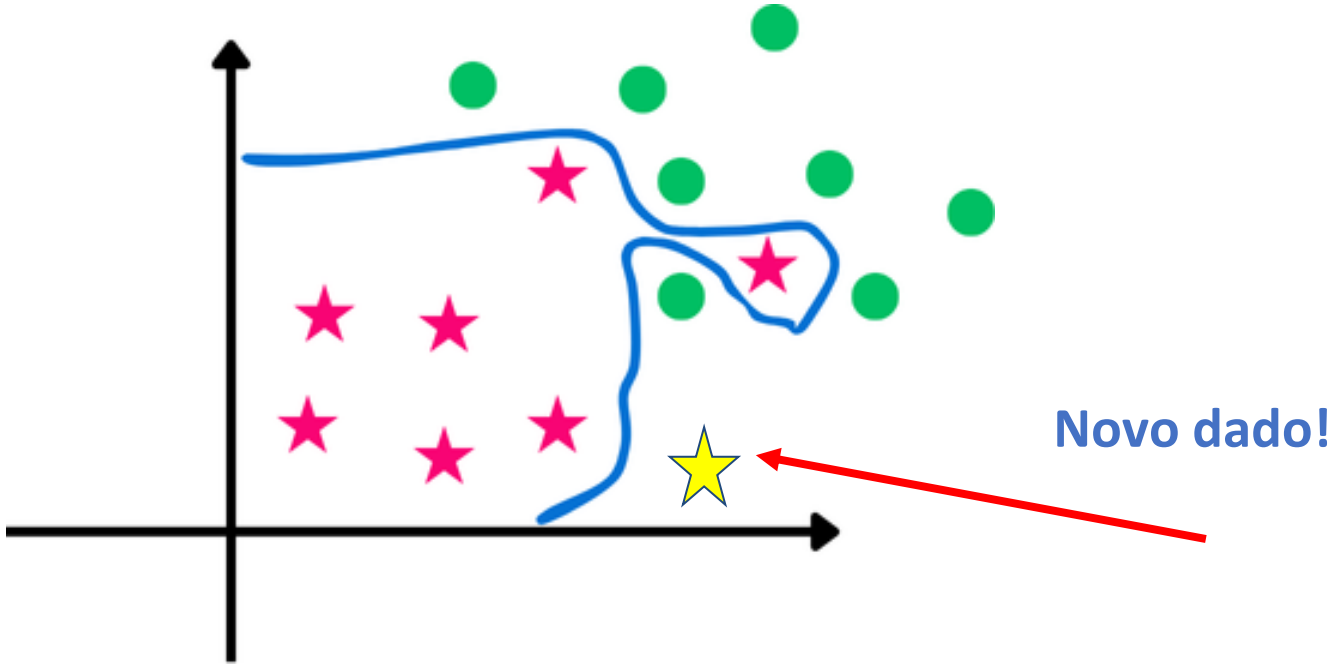


- O modelo criou uma linha super complexa que se adapta perfeitamente aos dados de treino.

Escolher e Ajustar os Modelos

Exemplo 1:

- Mas e se aparecerem novas "estrelas" e "bolinhas" em posições diferentes?



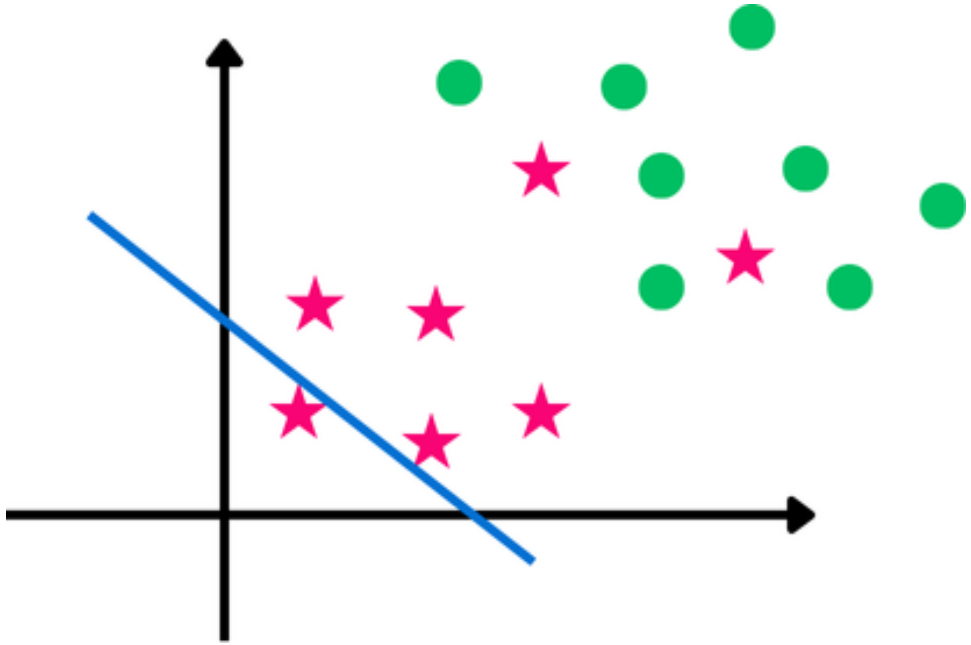
  - O modelo provavelmente terá um desempenho ruim, porque "decorou" os dados de treino em vez de aprender os padrões reais.

Vamos para o cenário 2!

Escolher e Ajustar os Modelos

Exemplo 2:

- Agora olhe esse outro modelo para classificar as “estrelas” e as “bolinhas”. **O que você acha?**



- O modelo está classificando praticamente todas as “estrelas” em “bolinhas”.

Esse cenário damos os nome de Underfitting!

Escolher e Ajustar os Modelos

Underfitting: Muito Preguiçoso para Aprender

- Underfitting acontece quando o modelo **não consegue capturar os padrões importantes nos dados.**
- É como um aluno que **não estuda o suficiente e não aprende nem o básico da matéria**.
- O modelo tem um desempenho **ruim tanto nos dados de treino quanto nos dados novos**.

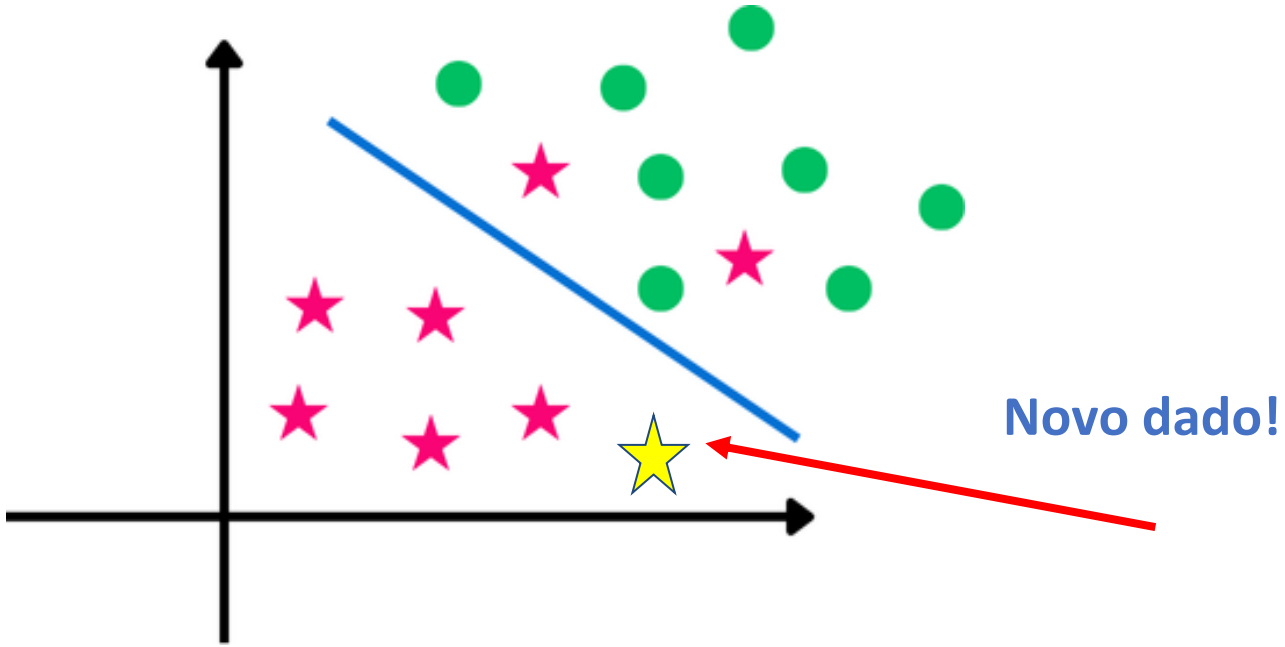
E qual seria o modelo ideal nesse cenário?

idp

Escolher e Ajustar os Modelos

Exemplo 3:

- olhe esse último modelo para classificar as “estrelas” e as “bolinhas”. **O que você acha?**



- O modelo separou bem as estrelas das bolinhas, mesmo errando duas classificações.

Escolher e Ajustar os Modelos

Exemplo 3:

- Queremos modelos que **encontrem o equilíbrio entre aprender os padrões importantes e evitar o overfitting**.

- Um modelo com **boa generalização terá um desempenho razoável nos dados de treino e um bom desempenho em dados novos**.

- É como um **aluno que entende a matéria e consegue aplicar o conhecimento em diferentes situações, mesmo que não acerte todas as questões da prova**.

E como eu consigo saber se o modelo é bom ou ruim?

Escolher e Ajustar os Modelos

Métricas de Desempenho: Medindo o Sucesso do Modelo

- Métricas de desempenho **são como "notas" que usamos para avaliar o quão bem um modelo está funcionando**.

- Elas nos dão uma **medida objetiva do desempenho do modelo** em uma tarefa específica.

- A escolha da **métrica certa depende do tipo de problema** que estamos resolvendo.

Dependendo da abordagem, temos métricas diferentes!

idp

Escolher e Ajustar os Modelos

Métricas de Desempenho: Medindo o Sucesso do Modelo

- Existem **diferentes tipos de problemas de Machine Learning**, e cada um precisa de uma avaliação diferente.
- Se você quer **separar indivíduos em categorias** (como identificar tipos de flores), precisa de testes que façam sentido para categorias.
- Se você quer **prever um número (como o preço de uma casa)**, precisa de testes que avaliem quão perto você chegou do número certo.

Vamos praticar?

idp

Escolher e Ajustar os Modelos

VAMOS PRATICAR

- Analise os cenários abaixo e **diga qual modelo está com Overfitting, Underfitting ou apresenta uma boa Generalização**.

- Para isso vamos **usar uma medida chamada RMSE (Root Mean Squared Error)**. Analisaremos ela com mais detalhes na próxima aula, mas **é uma medida muito utilizada para problemas de regressão**.
- Essa medida **quantifica o erro do modelo, ou seja quanto menor é o RMSE melhor é o modelo**. Mostraremos o RMSE no treinamento e no teste e assim deverão classificar o modelo.

Escolher e Ajustar os Modelos

VAMOS PRATICAR

- Modelo A:
  - RMSE no Treino = 500
  - RMSE no Teste = 6000

- **Qual o diagnóstico do modelo?**
  - **Resposta: Overfitting** - Quanto menor o RMSE, melhor, logo ele tem um desempenho muito bom no treino mas no teste piora.

Escolher e Ajustar os Modelos

VAMOS PRATICAR

- Modelo B:
  - RMSE no Treino = 16000
  - RMSE no Teste = 15000

- **Qual o diagnóstico do modelo?**
  - **Resposta: Underfitting**- Observe que modelo apresenta um **erro muito alto tanto no treinamento quanto no teste**. Comparando com o **modelo anterior (RMSE de 500 no treino e 6.000 no teste)**, observamos um aumento significativo nos valores, indicando que o modelo não está conseguindo capturar os padrões dos dados.

Escolher e Ajustar os Modelos

VAMOS PRATICAR

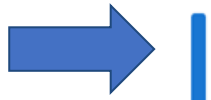
- Modelo C:
  - RMSE no Treino = 1000
  - RMSE no Teste = 1050

- **Qual o diagnóstico do modelo?**
  - **Resposta: Boa Generalização**- Tanto o RMSE do treinamento quanto do teste estão muito próximos, logo consegue prever custos novos com um desempenho razoável.

VAMOS PARA O PYTHON!

Modelando os Dados da aula_01_exemplo_01

- Vamos criar o nosso **modelo de Machine Learning!** Usaremos um modelo muito popular chamado **Árvore de Decisão** (Entraremos em mais detalhes na próxima aula).



Código

```
[9]: # criando o modelo
     modelo = DecisionTreeRegressor(random_state=42) # criando o modelo e fixando a aleatorização
```

- Criamos nosso modelo e armazenamos ele em uma variável chamada **modelo**.

- o **random_state = 42** serve para garantir que, ao rodar o mesmo código várias vezes, o modelo **sempre comece do mesmo ponto e aprenda da mesma forma**. Isso torna os resultados consistentes e reproduzíveis

O código acima não retorna nada pra gente!

Modelando os Dados da aula_01_exemplo_01

- Agora vamos criar nossa grade de hiperparâmetros.

Código

```
[10]: # Grade de hiperparâmetros

param_grid = {
    "max_depth": [2,3,5,7], # Profundidade máxima da árvore
    "min_samples_split": [2,5,10,15,20,25]# Nº mínimo de amostras para dividir um nó
}
```

- O **param_grid** é um dicionário que irá conter dois hiperparâmetros que queremos explorar: **max_depth** e **min_samples_split**.

- Cada um **recebe uma lista de possíveis valores para se explorar e o Grid Search irá encontrar qual o melhor par de hiperparâmetros** para o modelo de árvore de decisão.

Modelando os Dados da aula_01_exemplo_01

- Agora vamos criar nossa grade de hiperparâmetros.

Código

```
[10]: # Grade de hiperparâmetros

param_grid = {
    "max_depth": [2,3,5,7], # Profundidade máxima da árvore
    "min_samples_split": [2,5,10,15,20,25]# Nº mínimo de amostras para dividir um nó
}
```

- Note que **esses valores são colocados manualmente por nós!** Para identificar os possíveis valores, é necessário analisar a documentação do modelo que você quer aprimorar. Clique aqui para ver.

O código acima não retorna nada pra gente!

Modelando os Dados da aula_01_exemplo_01

- Agora vamos usar o Grid Search para encontra os melhores hiperparâmetros.

Código

```
[11]: # Configurando Grid Search
grid_search = GridSearchCV(
    estimator= modelo,              # modelo a ser otimizado
    param_grid=param_grid,          # Grade de hiperparâmetros
    cv=5,                           # 5-fold cross-validation
    scoring="neg_root_mean_squared_error"  # Métrica de comparação: RMSE
)
```

- Note que ele precisa que a gente diga qual é o modelo que queremos aprimorar com o **estimator = modelo**.

- Precisamos também dizer qual nossa grade de parâmetros com o **param_grid = param_grid** (não tem problema se for o mesmo nome!)

Modelando os Dados da aula_01_exemplo_01

- Agora vamos usar o Grid Search para encontra os melhores hiperparâmetros.

Código

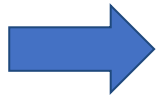
```
[11]: # Configurando Grid Search
grid_search = GridSearchCV(
    estimator= modelo,              # modelo a ser otimizado
    param_grid=param_grid,          # Grade de hiperparâmetros
    cv=5,                           # 5-fold cross-validation
    scoring="neg_root_mean_squared_error"  # Métrica de comparação: RMSE
)
```

- Também é necessário dizer quantos folds (pedaços) vamos utilizar na validação cruzada com **cv = 5** e também qual a medida de avaliação **scoring = "neg_root_mean_squared_error"**

O código acima não retorna nada pra gente!

Modelando os Dados da aula_01_exemplo_01

- Agora vamos usar o Grid Search nos nossos dados de treino usando o comando **.fit().**



Código

```
[12]: # Treinando com Grid Search
grid_search.fit(x_treino_transformado, y_treino) # usando as covariáveis já preprocessadas e nossa variável target

melhor_modelo = grid_search.best_estimator_  # Melhor modelo encontrado
```

- Assim ele vai **pesquisar o melhor par de hiperparâmetros e colocá-los em um novo modelo melhorado.**

- para salvar o novo modelo basta criar uma nova variável **melhor_modelo** colocar o comando **grid_search.best_estimator_.**

O código acima não retorna nada pra gente!

Modelando os Dados da aula_01_exemplo_01

- Por fim, vamos analisar o desempenho do modelo nos dados de treinamento.

Código

```
[13]: # Predição no treino
y_pred_treino = melhor_modelo.predict(x_treino_transformado) # covariáveis de treino

# Avaliação do modelo no treino
mse_treino = mean_squared_error(y_treino,y_pred_treino) # Valores reais, Valores preditos

# Resultados
print("Melhores parâmetros encontrados:", grid_search.best_params_)
print("Root Mean Squared Error no treino:",np.round(np.sqrt(mse_treino)))
```

Saída

```
Melhores parâmetros encontrados: {'max_depth': 3, 'min_samples_split': 2}
Root Mean Squared Error no treino: 4595.69
```

- o comando .**predict()** faz a predição nos dados de treino (covariáveis) e nos retorna as previsões do modelo.

idp

Modelando os Dados da aula_01_exemplo_01

- Por fim, vamos analisar o desempenho do modelo nos dados de treinamento.

Código

```
[13]: # Predição no treino
y_pred_treino = melhor_modelo.predict(x_treino_transformado) # covariáveis de treino

# Avaliação do modelo no treino
mse_treino = mean_squared_error(y_treino,y_pred_treino) # Valores reais, Valores preditos

# Resultados
print("Melhores parâmetros encontrados:", grid_search.best_params_)
print("Root Mean Squared Error no treino:",np.round(np.sqrt(mse_treino)))
```

Saída

```
Melhores parâmetros encontrados: {'max_depth': 3, 'min_samples_split': 2}
Root Mean Squared Error no treino: 4595.69
```

- o **mean_squared_error()** é um comando para auxiliar no cálculo do RMSE. **Sempre coloque a ordem os valores reais e depois as previsões do modelo.**

idp

Modelando os Dados da aula_01_exemplo_01

- Por fim, vamos analisar o desempenho do modelo nos dados de treinamento.

Código

```
[13]: # Predição no treino
y_pred_treino = melhor_modelo.predict(x_treino_transformado) # covariáveis de treino

# Avaliação do modelo no treino
mse_treino = mean_squared_error(y_treino,y_pred_treino) # Valores reais, Valores preditos

# Resultados
print("Melhores parâmetros encontrados:", grid_search.best_params_)
print("Root Mean Squared Error no treino:",np.round(np.sqrt(mse_treino)))
```

Saída

```
Melhores parâmetros encontrados: {'max_depth': 3, 'min_samples_split': 2}
Root Mean Squared Error no treino: 4595.69
```

- o comando **grid_search.best_params_** vai nos dar o melhor par de hiperparâmetros encontrados.

Modelando os Dados da aula_01_exemplo_01

- Por fim, vamos analisar o desempenho do modelo nos dados de treinamento.

Código

```
[13]: # Predição no treino
y_pred_treino = melhor_modelo.predict(x_treino_transformado) # covariáveis de treino

# Avaliação do modelo no treino
mse_treino = mean_squared_error(y_treino,y_pred_treino) # Valores reais, Valores preditos

# Resultados
print("Melhores parâmetros encontrados:", grid_search.best_params_)
print("Root Mean Squared Error no treino:",np.round(np.sqrt(mse_treino)))
```

Saída

```
Melhores parâmetros encontrados: {'max_depth': 3, 'min_samples_split': 2}
Root Mean Squared Error no treino: 4595.69
```

- o comando **np.sqrt(mse_treino)** vai calcular o RMSE e o comando **np.round()** irá arredondar o valor encontrado.

Modelando os Dados da aula_01_exemplo_01

- Por fim, vamos analisar o desempenho do modelo nos dados de treinamento.

Código

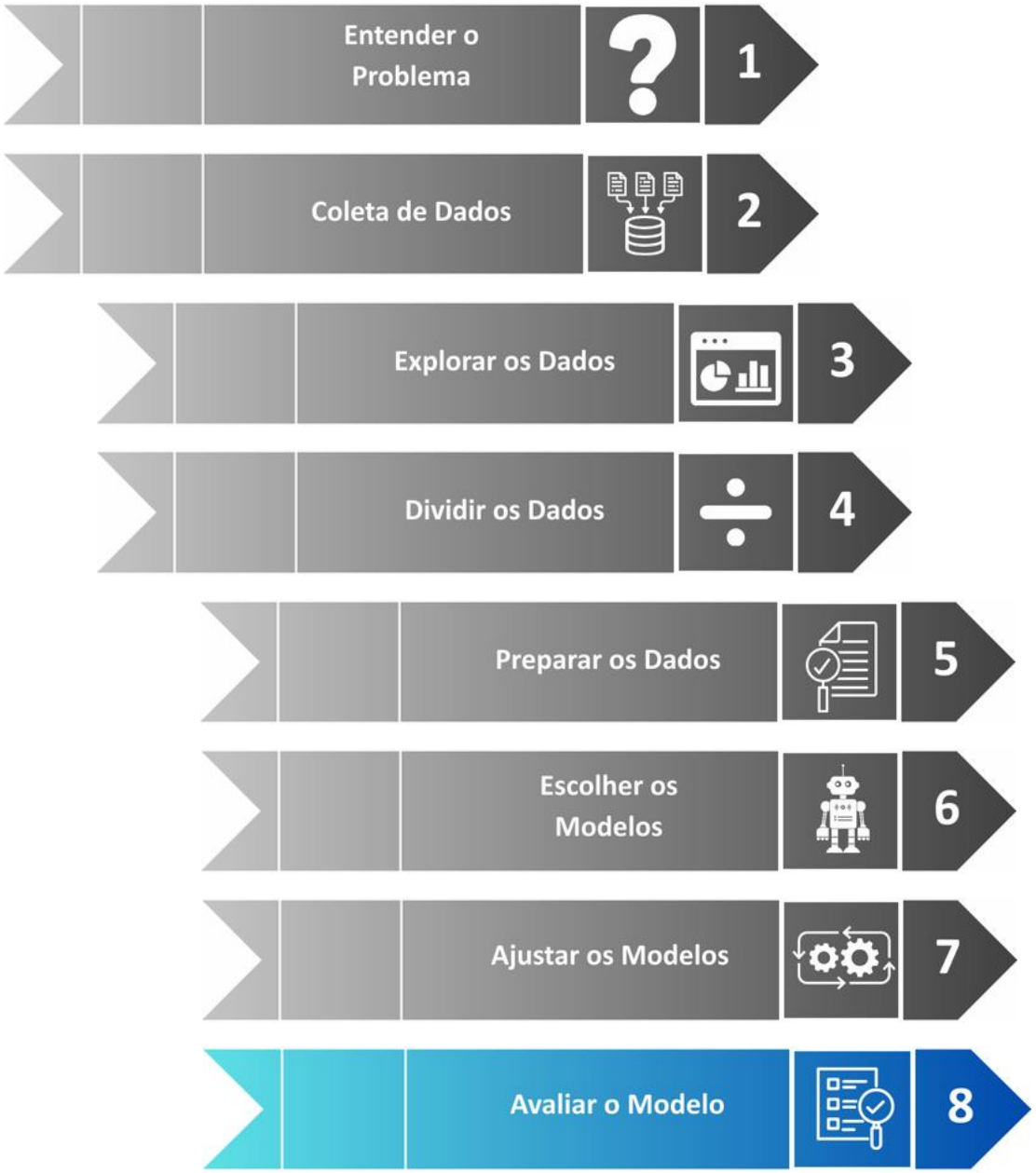
```
[13]: # Predição no treino
y_pred_treino = melhor_modelo.predict(x_treino_transformado) # covariáveis de treino

# Avaliação do modelo no treino
mse_treino = mean_squared_error(y_treino,y_pred_treino) # Valores reais, Valores preditos

# Resultados
print("Melhores parâmetros encontrados:", grid_search.best_params_)
print("Root Mean Squared Error no treino:",np.round(np.sqrt(mse_treino)))
```

Saída

```
Melhores parâmetros encontrados: {'max_depth': 3, 'min_samples_split': 2}
Root Mean Squared Error no treino: 4595.69
```

- Agora podemos calcular o RMSE no teste e analisar o desempenho do nosso modelo!

As Etapas da construção de modelos de Machine Learning



Escolher e Ajustar os Modelos

Chegou a Hora da Verdade: O Teste Final!

- Depois de todo o treinamento e ajuste, **é hora de ver como o modelo se comporta em "condições reais"**.
- É aqui que **entra o conjunto de teste, que guardamos desde o início**. Use-o para calcular as métricas de desempenho finais do modelo.
- Essas métricas **nos dão uma estimativa honesta de quão bem o modelo generaliza** para dados novos e desconhecidos.

Vamos praticar!

Modelando os Dados da aula_01_exemplo_01

- Vamos usar nosso modelo com os dados de teste.

Código

```
[14]: # Predição no teste
y_pred_teste = melhor_modelo.predict(x_teste_transformado) # covariáveis de teste

# Avaliação do modelo
mse_teste = mean_squared_error(y_teste,y_pred_teste) # Valores reais, Valores preditos

# Resultados
print("Root Mean Squared Error no treino:",np.round(np.sqrt(mse_teste)))
```

Saída

```
Root Mean Squared Error no teste: 4776.0
```

- Note que utilizamos os mesmos passos da avaliação do treino. O que mudou foi usar os dados de teste: **x_teste_transformado** e **y_teste**.

- Perceba que o desempenho na **base de teste (4776)** foi muito próxima na **base de treino (4596)**, portanto podemos concluir que nosso modelo está generalizando bem!

Considerações Gerais

Recapitulando:

- **Percorremos um caminho completo no Machine Learning:** definimos o problema, coletamos e preparamos dados, escolhemos e ajustamos modelos.
- **Descobrimos a importância de evitar Data Leakage**, ajustar as "engrenagens" do modelo (hiperparâmetros) e escolher o "teste" certo (métricas).
- E com o modelo final, **chegamos a um RMSE de 4596 no treino e 4776 no teste**. Isso significa que o modelo generalizou bem, mas há espaço para melhorias!
- Na próxima aula, **compreenderemos mais afundo sobre as métricas em regressão e sobre os modelos para esse cenário**.

Disclaimer: propriedade intelectual

Este material foi criado pela professora Roberta Moreira Wichmann e é de sua propriedade intelectual.

É destinado exclusivamente ao uso dos alunos para fins educacionais no contexto das aulas.

Qualquer reprodução, distribuição ou utilização deste material, no todo ou em parte, sem a expressa autorização prévia da autora, é estritamente proibida.

O não cumprimento destas condições poderá resultar em medidas legais.

Referências Bibliográficas

- ABU-MOSTAFA, Yaser S.; MAGDON-ISMAIL, Malik; LIN, Hsuan-Tien. Learning from Data: A Short Course. Pasadena: California Institute of Technology (AMLBook), 2012.

- GERON, Aurélien. Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems. 2. ed. Sebastopol, CA: O'Reilly Media, 2019.

- HARRIS, C. R. et al. Array programming with NumPy. Nature, v. 585, p. 357–362, 2020. DOI: 10.1038/s41586-020-2649-2.

- HASTIE, Trevor; TIBSHIRANI, Robert; FRIEDMAN, Jerome. The elements of statistical learning: data mining, inference, and prediction. 2. ed. New York: Springer, 2009.

- KAPOOR, Sayash; NARAYANAN, Arvind. Leakage and the reproducibility crisis in machine-learning-based science. Patterns, v. 4, n. 9, 2023.

idp

Referências Bibliográficas

- IZBICKI, Rafael; DOS SANTOS, Tiago Mendonça. Aprendizado de máquina: uma abordagem estatística. Rafael Izbicki, 2020.
- MORETTIN, Pedro Alberto; SINGER, Júlio da Motta. Estatística e ciência de dados. 2. ed. Rio de Janeiro: LTC, 2022.
- PEDREGOSA, F. et al. Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research, v. 12, p. 2825–2830, 2011.
- PYTHON SOFTWARE FOUNDATION. Python Language Reference. Disponível em: https://docs.python.org/3/reference/index.html. Acesso em: 10 set. 2025.
- THE PANDAS DEVELOPMENT TEAM. pandas-dev/pandas: Pandas. Zenodo, 2024. Disponível em: https://doi.org/10.5281/zenodo.10537285. Acesso em: 10 set. 2025.

idp

Referências Bibliográficas

- VON LUXBURG, Ulrike; SCHÖLKOPF, Bernhard. Statistical Learning Theory: Models, Concepts, and Results. In: GABBAY, D. M.; HARTMANN, S.; WOODS, J. H. (eds.). Handbook of the History of Logic, vol. 10: Inductive Logic. Amsterdam: Elsevier North Holland, 2011. p. 651–706. DOI: 10.1016/B978-0-444-52936-7.50016-1.

idp

Etapas na Construção de Modelos de Machine Learning

Obrigada!

Profa. Dra. Roberta Wichmann

roberta.wichmann@idp.edu.br

INSTITUTO BRASILEIRO DE ENSINO,
DESENVOLVIMENTO E PESQUISA