

Introdução a Machine Learning

Profa. Dra. Roberta Wichmann

roberta.wichmann@idp.edu.br

Aula 4 – Regressão no Aprendizado Supervisionado: Métodos e Avaliação

Métodos paramétricos e não paramétricos. Conceitos sobre viés e variância.

Conceitos iniciais sobre o método de mínimos quadrados e seleção de modelo.

Técnicas de avaliação de modelos. Conceitos iniciais sobre árvores de decisão e KNN.

Recapitulando a Jornada do Machine Learning

Recapitulando: Fundamentos do Aprendizado de Máquina

- Na sessão anterior, **percorremos o ciclo completo de um projeto de Machine Learning**, desde a formulação do problema até a avaliação do modelo.
- **Definimos o Problema:** Entendemos qual pergunta queremos responder com os dados.
- **Coletamos e Preparamos os Dados:** Reunimos e limpamos as informações essenciais.
- **Escolhemos e Ajustamos Modelos:** Selecionamos um algoritmo e otimizamos seus hiperparâmetros.

Recapitulando a Jornada do Machine Learning

Recapitulando: Fundamentos do Aprendizado de Máquina

- **Evitamos Data Leakage:** Cuidado para não usar informações do futuro no treinamento!
- **Escolhemos as Métricas:** Definimos como medir o desempenho do modelo.
- **Resultado:** Alcançamos um RMSE de 4596 no treino e 4776 no teste, indicando boa generalização, mas com espaço para aprimoramento!

E o que vamos fazer agora?

Recapitulando a Jornada do Machine Learning

Próximos Passos:

- Introduziremos os conceitos que norteiam o **funcionamento interno** dos modelos de regressão.
- **Entenderemos qual a diferença entre a regressão paramétrica e não paramétrica.**
- Vamos explorar em detalhes cada **métrica de avaliação**, bem como os modelos mais utilizados em modelagem.

Mas o que é regressão paramétrica?

Regressão Paramétrica: Simplificando a Realidade

Regressão Paramétrica

- A Regressão Paramétrica assume que a **relação entre as variáveis pode ser descrita por uma fórmula predefinida com um número limitado de parâmetros**.
- Os parâmetros **são os coeficientes que determinam a forma do modelo**.
- É como **tentar encaixar uma peça de um quebra-cabeça em um buraco específico**. Se a peça for a certa, o encaixe é perfeito. Se não for, o resultado pode ser ruim.
- **Exemplo:** Prever o preço de uma casa com base no tamanho (em metros quadrados), assumindo que a relação é linear. Precisamos apenas encontrar os coeficientes da reta.

Como assim coeficientes da reta?

Regressão Paramétrica: Simplificando a Realidade

Exemplos:

- **A Regressão Linear é o exemplo mais famoso de modelo paramétrico.** A abordagem se baseia em determinar uma forma “base” (a equação da reta no exemplo abaixo) fixa que visa explicar a relação entre as variáveis. **O trabalho do modelo é só descobrir os parâmetros que melhor ajustam essa forma aos dados.**
- $\text{Preço} = (\text{inclinação}) \times (\text{tamanho}) + (\text{ponto de partida})$
 - **Inclinação:** mostra o quanto o preço aumenta quando o tamanho da casa aumenta.
 - **Ponto de partida (intercepto):** é o preço quando o tamanho da casa é zero (um valor inicial da linha).
- O desafio do modelo **é encontrar quais os melhores valores da inclinação e do intercepto que ajustam melhor aos dados.**

Regressão Paramétrica: Simplificando a Realidade

Suas vantagens são:

- **Simplicidade:** Fácil de entender e implementar.
- **Interpretabilidade:** Permite identificar a importância de cada variável e sua direção (positiva ou negativa).
- **Eficiência:** Requer menos dados do que modelos não paramétricos.

Como é a interpretação?

Regressão Paramétrica: Simplificando a Realidade

Exemplo: em um modelo de salário

- Salário= $2000 + 500 \times (\text{anos de estudo})$
- Note que o “Salário” em reais é nossa variável target e “anos de estudo” é a covariável.
- **O que acontece a cada aumento de um ano de estudo?**
- Note que a cada um ano no tempo de estudo, há um aumento de em média 500 Reais.
- **O que acontece quando a covariável for 0?**
- Perceba que se os anos de estudo for igual a 0 o salário base é de 2000 Reais.

Regressão Paramétrica: Simplificando a Realidade

Desafios:

- **Rigidez:** A suposição sobre a forma da relação pode ser muito forte e não capturar a complexidade dos dados.
- **Viés:** Se a relação real for não linear, o modelo linear terá um alto viés.
- **Exemplo:** Se a relação entre o tamanho da casa e o preço for exponencial, uma reta não será uma boa aproximação.

Mas o que é viés?

Viés e Variância: O Que São?

Viés e Variância: A Base para Bons Modelos

- Em Machine Learning, nosso objetivo é **construir modelos que façam previsões precisas em dados novos e desconhecidos**, o que chamamos de "generalização".
- Para alcançar esse objetivo, **é fundamental compreender e controlar dois tipos de erros** que podem afetar o desempenho do modelo: **o viés e a variância**.
- A busca por um bom modelo se resume a **encontrar o equilíbrio ideal** entre esses dois tipos de erro, evitando tanto a simplificação excessiva quanto a memorização dos dados.

Entendi, mas o que é o viés?

Viés e Variância: O Que São?

Viés

- Imagine um **arqueiro que sempre mira no mesmo lugar, mas erra o centro do alvo**. Isso acontece porque ele não tem informações suficientes sobre o vento, a distância e outros fatores.
- O viés é um erro sistemático que ocorre quando o **modelo faz suposições muito simplistas sobre os dados**, ignorando padrões importantes e nuances presentes na realidade.
- **Consequências:** Um modelo com alto viés tende a subestimar a complexidade do problema, levando a um ajuste inadequado (underfitting) e a previsões imprecisas.
- **Exemplo:** Usar uma linha reta para modelar uma relação que, na verdade, é uma curva. O modelo sempre errará, mesmo com muitos dados.

Viés e Variância: O Que São?

Variância

- Imagine um arqueiro que tenta se **ajustar a cada pequena variação do vento, tremendo a mão e errando o alvo**. Ele se concentra demais em detalhes irrelevantes, perdendo a visão geral.
- **A variância mede a sensibilidade do modelo a pequenas flutuações nos dados**. Modelos com alta variância "decoram" os dados, incluindo ruídos e valores anômalos.
- **Consequências**: O modelo se torna muito específico para os dados de treinamento, perdendo a capacidade de generalizar para dados novos(overfitting).
- **Exemplo**: Criar uma regra supercomplexa que funciona perfeitamente para um conjunto limitado de dados, mas falha miseravelmente em dados novos.

Viés e Variância: O Que São?

O Segredo é o Equilíbrio

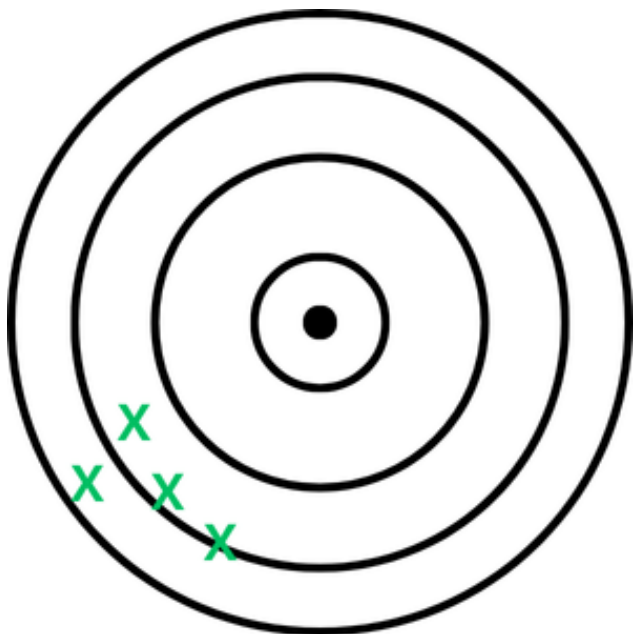
- **O Trade-off:** Diminuir o viés geralmente aumenta a variância, e vice-versa. O objetivo é encontrar o ponto de equilíbrio ideal, onde ambos os erros são minimizados.
- **O Modelo Perfeito:** Um bom modelo é capaz de capturar os padrões importantes nos dados sem se ajustar demais ao ruído. Ele generaliza bem para dados novos e desconhecidos.
- **Como Alcançar o Equilíbrio:** Coletar mais dados e aplicar métodos de validação adequados.

Vamos praticar?

Viés e Variância: O Que São?

VAMOS PRATICAR

- Vamos usar a representação do jogo de Dardos, onde quem acertar mais perto do centro, ganha. Pense em cada dardo como uma previsão do modelo. Classifique se o **modelo tem variância baixa, moderada ou alta, e se o modelo tem viés baixo, moderado e alto.**

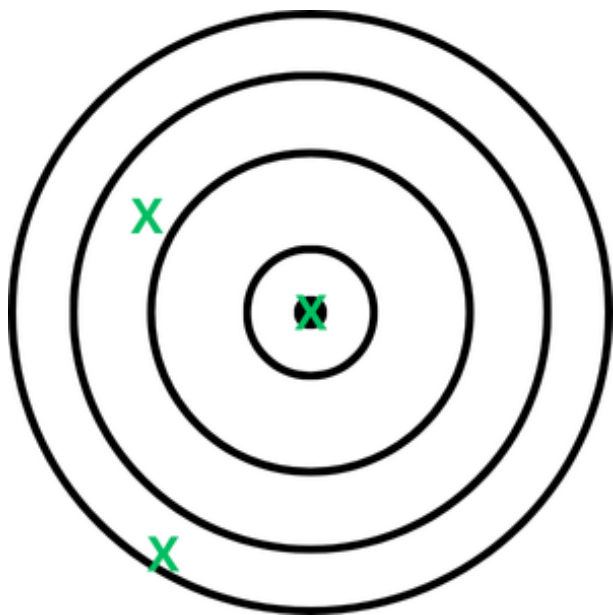


- Como você classifica esse modelo?
- **Resposta:** Viés alto e variância baixo, os pontos estão muito longe do centro.

Viés e Variância: O Que São?

VAMOS PRATICAR

- Vamos usar a representação do jogo de Dardos, onde quem acertar mais perto do centro, ganha. Pense em cada dardo como uma previsão do modelo. Classifique se o **modelo tem variância baixa, moderada ou alta, e se o modelo tem viés baixo, moderado e alto.**

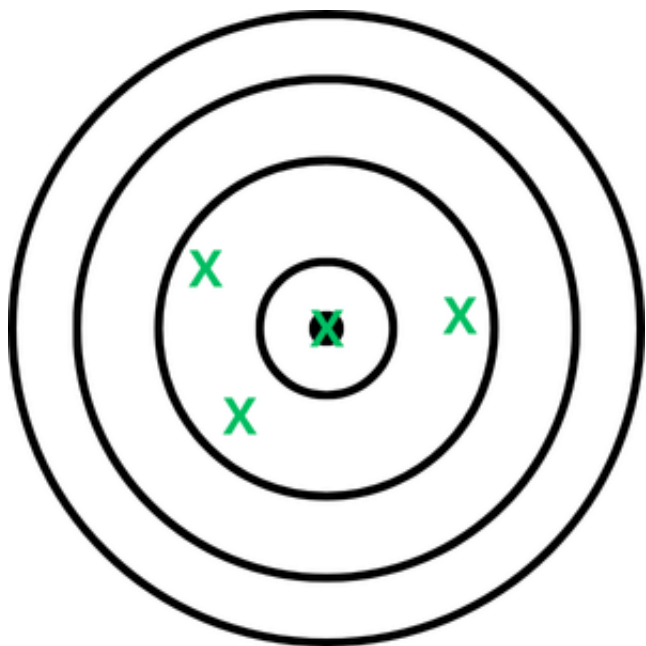


- Como você classifica esse modelo?
- **Resposta:** Variância alta e viés alto, os pontos estão muito longe do centro e muito longe entre si.

Viés e Variância: O Que São?

VAMOS PRATICAR

- Vamos usar a representação do jogo de Dardos, onde quem acertar mais perto do centro, ganha. Pense em cada dardo como uma previsão do modelo. Classifique se o **modelo tem variância baixa, moderada ou alta, e se o modelo tem viés baixo, moderado e alto.**

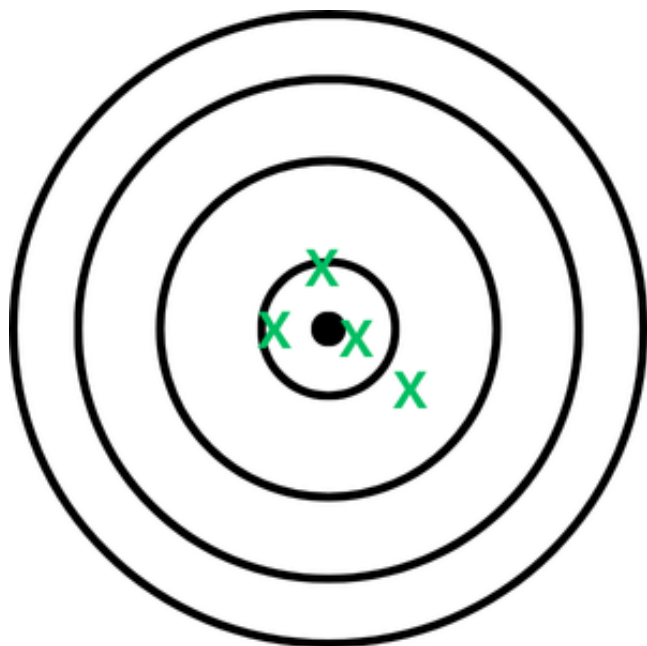


- Como você classifica esse modelo?
- **Resposta:** Variância moderada e viés baixo, os pontos estão próximos entre si e próximos do centro.

Viés e Variância: O Que São?

VAMOS PRATICAR

- Vamos usar a representação do jogo de Dardos, onde quem acertar mais perto do centro, ganha. Pense em cada dardo como uma previsão do modelo. Classifique se o **modelo tem variância baixa, moderada ou alta, e se o modelo tem viés baixo, moderado e alto.**



- Como você classifica esse modelo?
- **Resposta:** Variância baixa e viés baixo, os pontos estão próximos entre si e próximos do centro.

O Aprendizado Paramétrico em Detalhes

Desvendando o Processo de aprendizado

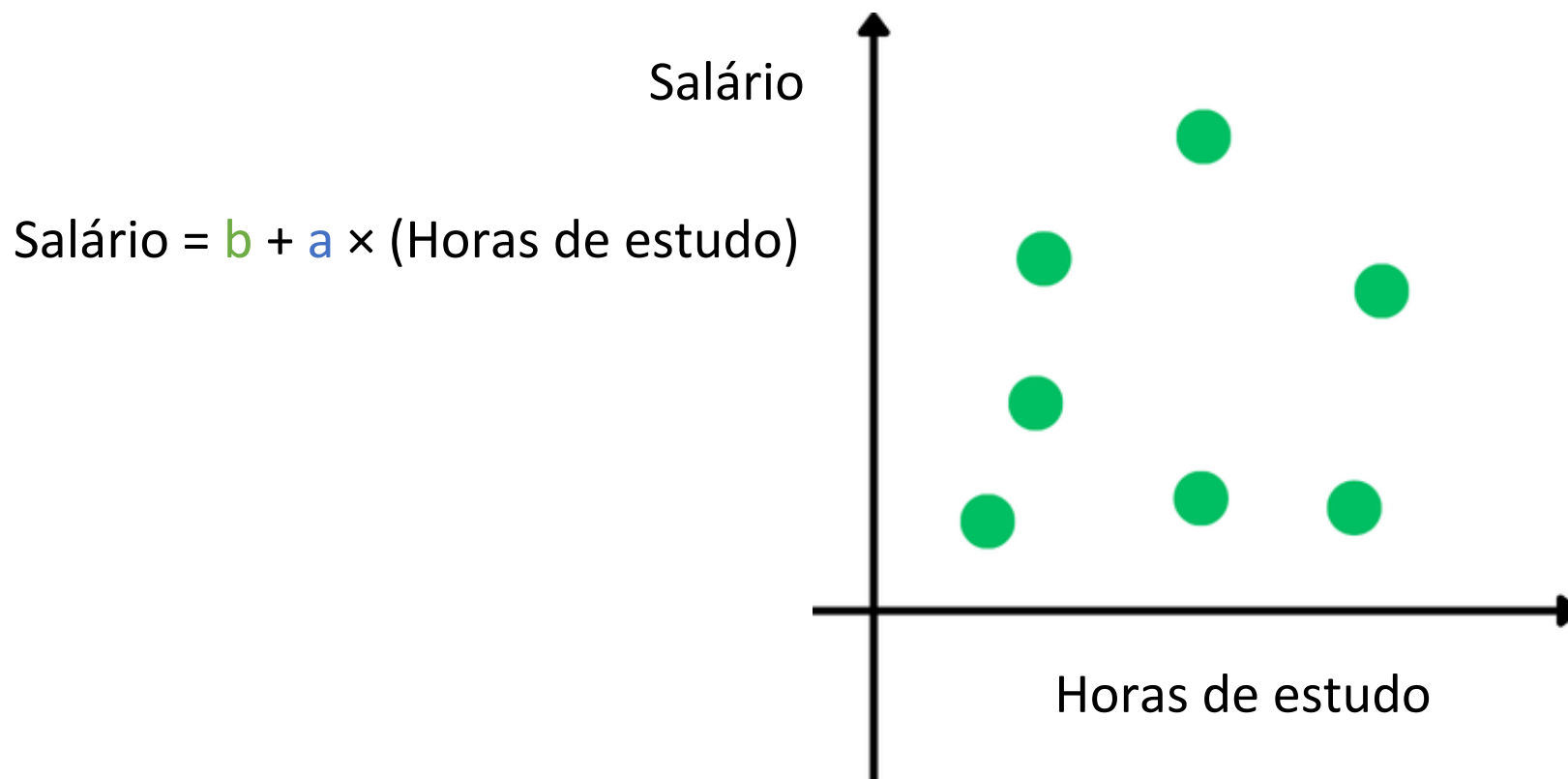
- **O Que Já Sabemos:** Modelos paramétricos assumem uma forma fixa para a relação entre X e Y (ex: $\text{Salário} = 2000 + 500 \times (\text{anos de estudo})$).

Como Eles Aprendem?

- **Definindo a "Perda":** Precisamos de uma forma de medir o quão "ruim" é o ajuste do modelo aos dados. Isso é feito com uma função de custo (ou perda).
- **Ajustando os Controles:** O aprendizado se resume a encontrar os melhores valores para os parâmetros da função, minimizando a função de custo.
- **Pense nos parâmetros como "controles" que ajustam a forma da função.** Por exemplo, na reta $\text{Salário} = b + a \times (\text{Horas de estudo})$, a e b são os parâmetros e a serem estimados e Horas de estudo é a nossa variável.

Mínimos Quadrados: Desvendando o Segredo da "Melhor Linha"

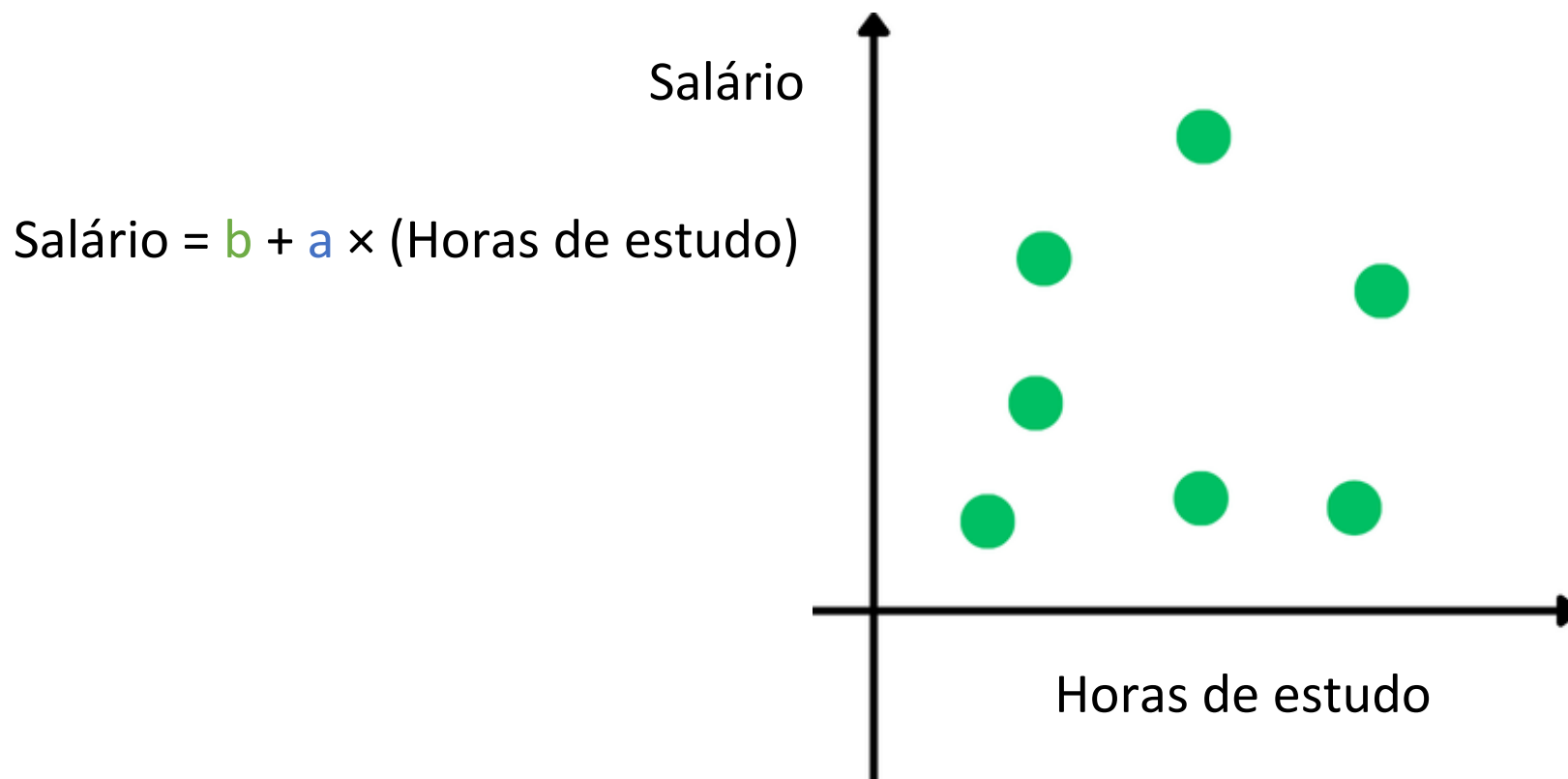
Olhe o gráfico abaixo:



- Temos um monte de pontos espalhados em um gráfico. **Cada ponto é basicamente um indivíduo que possui um determinado valor de horas de estudo e de salário.**

Mínimos Quadrados: Desvendando o Segredo da "Melhor Linha"

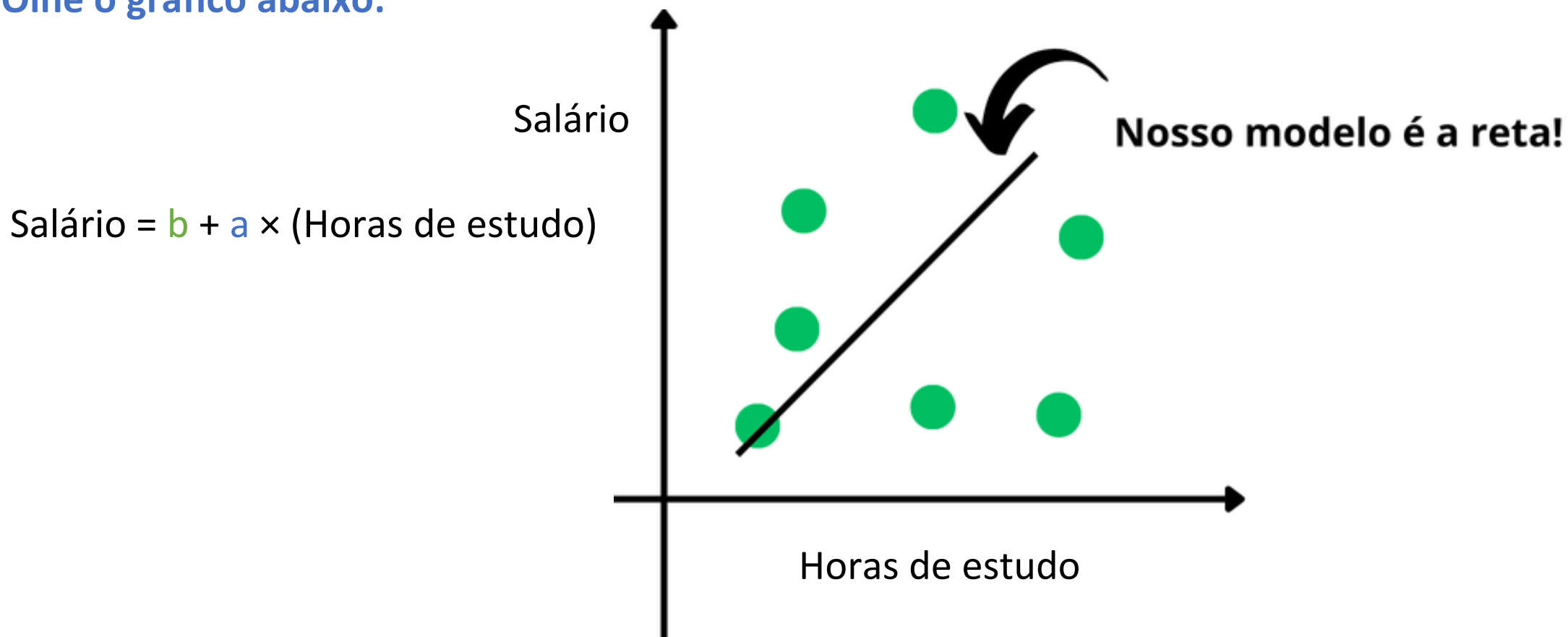
Olhe o gráfico abaixo:



- Queremos encontrar a reta que melhor representa a relação entre esses pontos.

Mínimos Quadrados: Desvendando o Segredo da "Melhor Linha"

Olhe o gráfico abaixo:

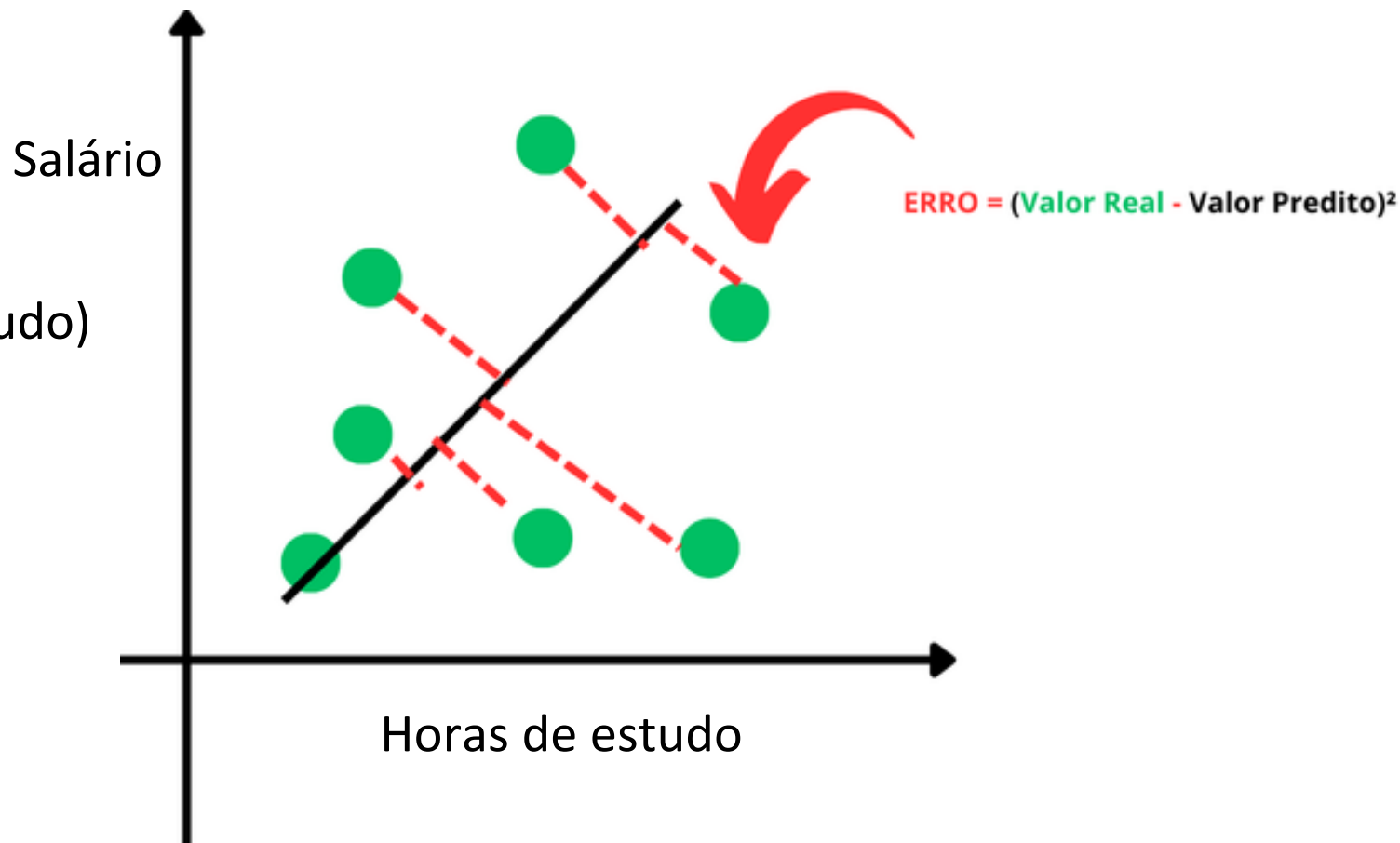


- **Medindo o Erro:** Para cada ponto, medimos a distância até a reta (essa distância é o "erro" de previsão).

Mínimos Quadrados: Desvendando o Segredo da "Melhor Linha"

Olhe o gráfico abaixo:

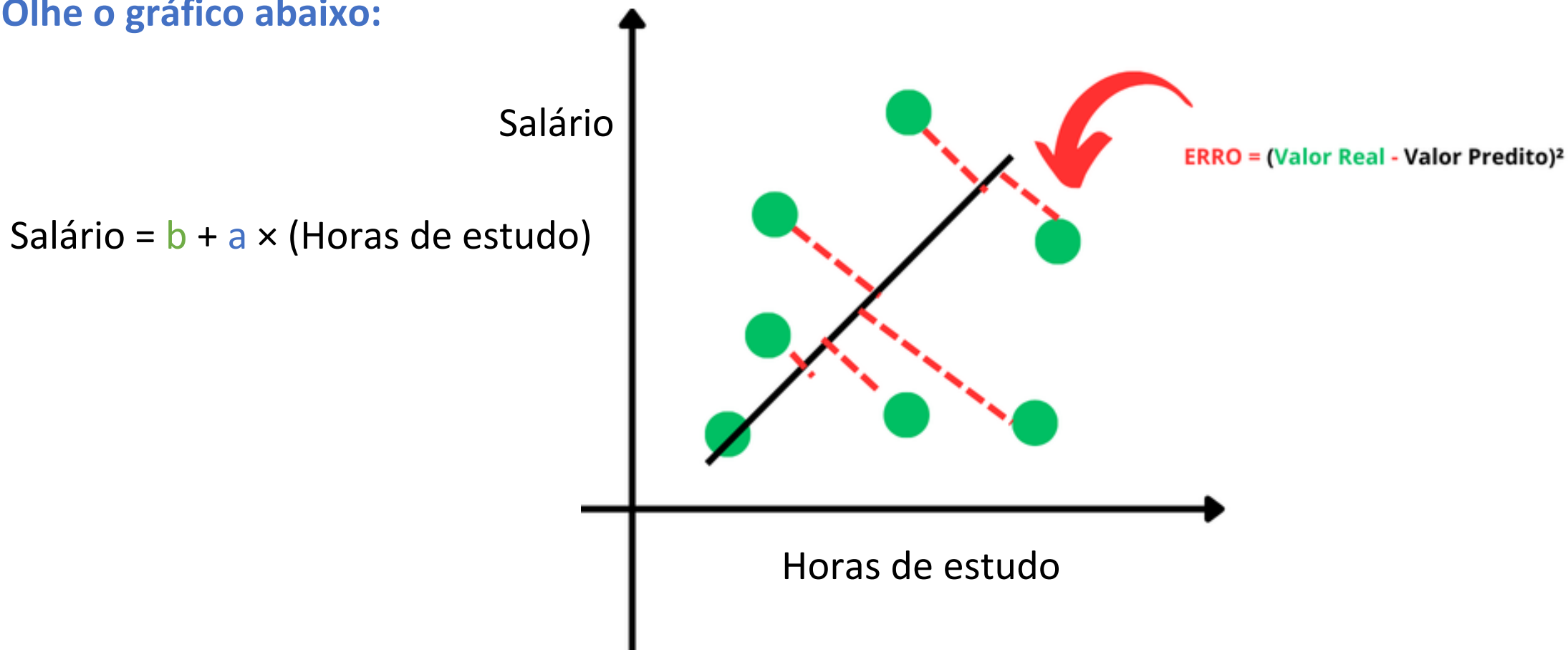
$$\text{Salário} = b + a \times (\text{Horas de estudo})$$



- **Eliminando Sinais:** Elevamos cada distância ao quadrado (isso transforma todos os erros em positivos e evita que se anulem).

Mínimos Quadrados: Desvendando o Segredo da "Melhor Linha"

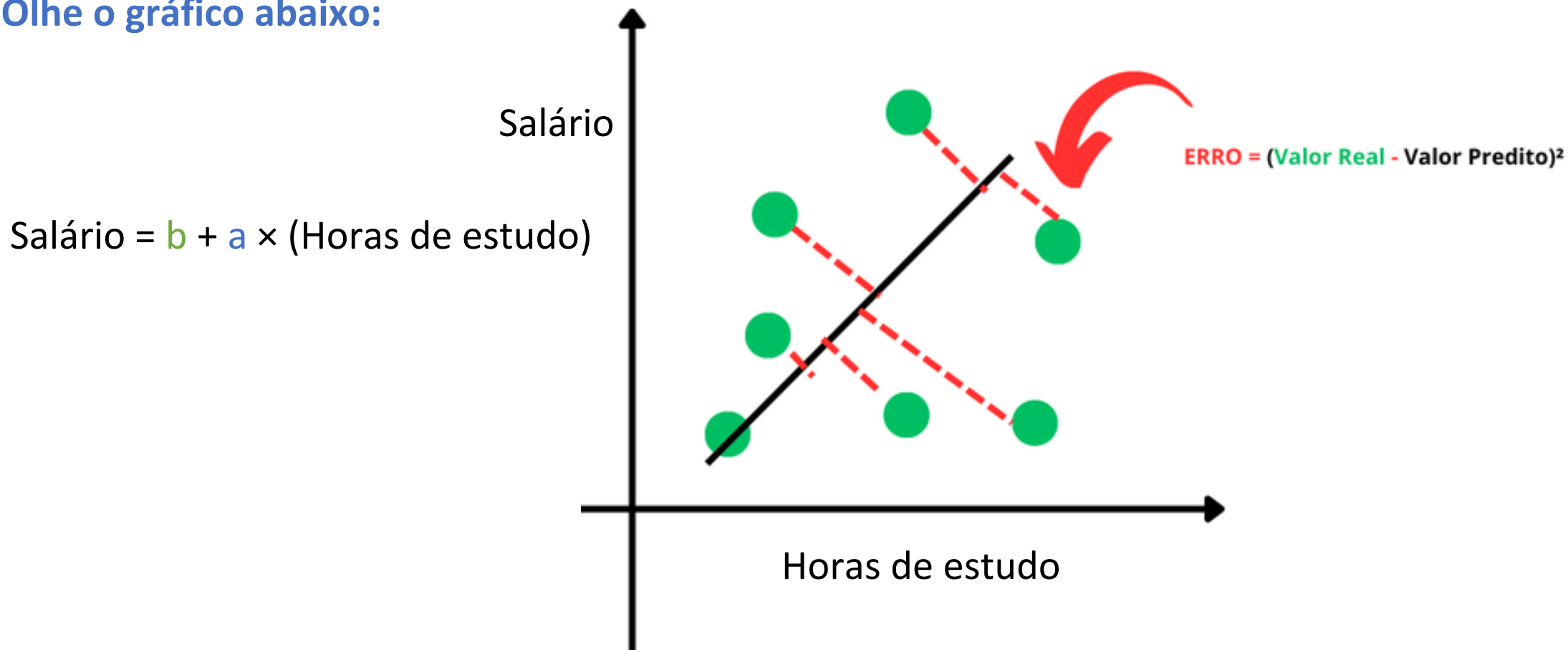
Olhe o gráfico abaixo:



- **Encontrando a "Melhor" Reta:** Ajustamos a posição (nosso b) e a inclinação da reta (nosso a) até que a soma de todos os erros ao quadrado seja a menor possível.

Mínimos Quadrados: Desvendando o Segredo da "Melhor Linha"

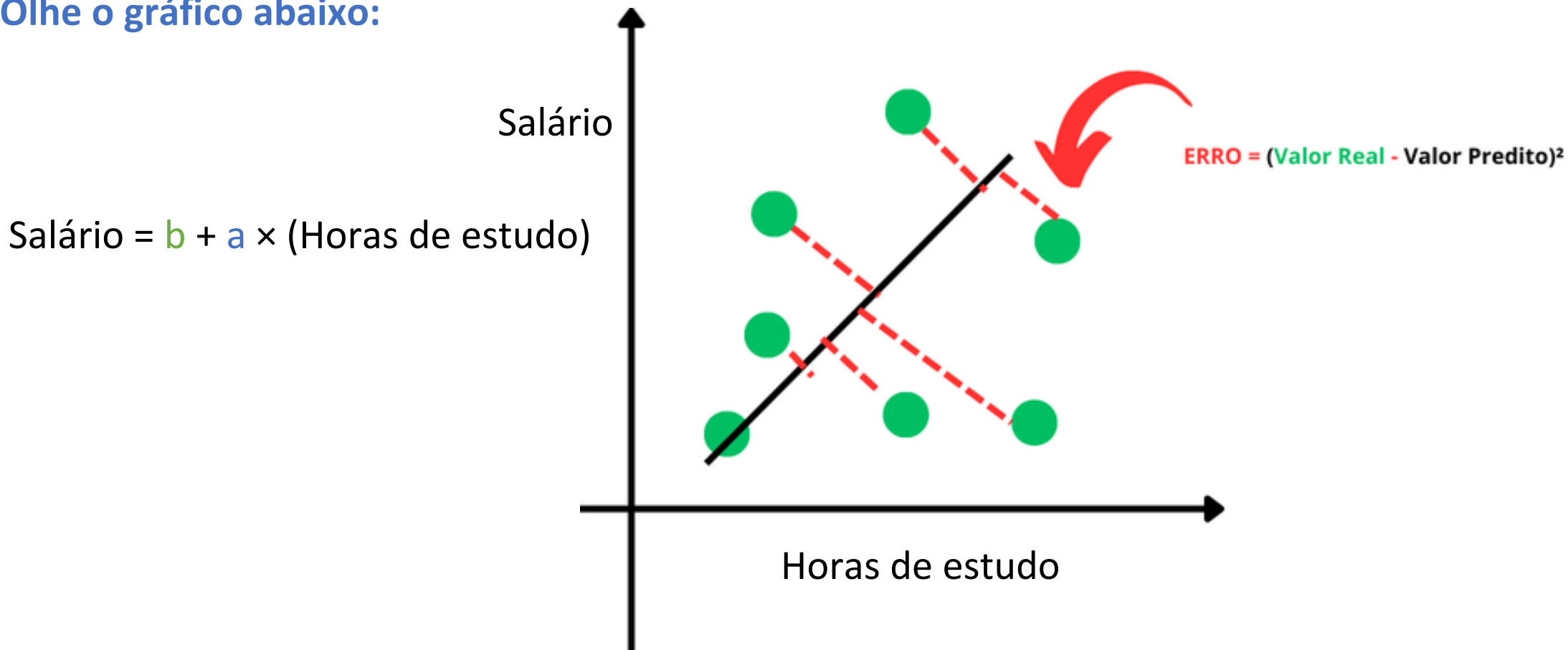
Olhe o gráfico abaixo:



- Imagine que cada ponto é uma pessoa e a reta é uma corda. **Queremos posicionar a corda de forma que todas as pessoas fiquem o mais perto possível dela**, sem que nenhuma precise se esticar demais.

Mínimos Quadrados: Desvendando o Segredo da "Melhor Linha"

Olhe o gráfico abaixo:

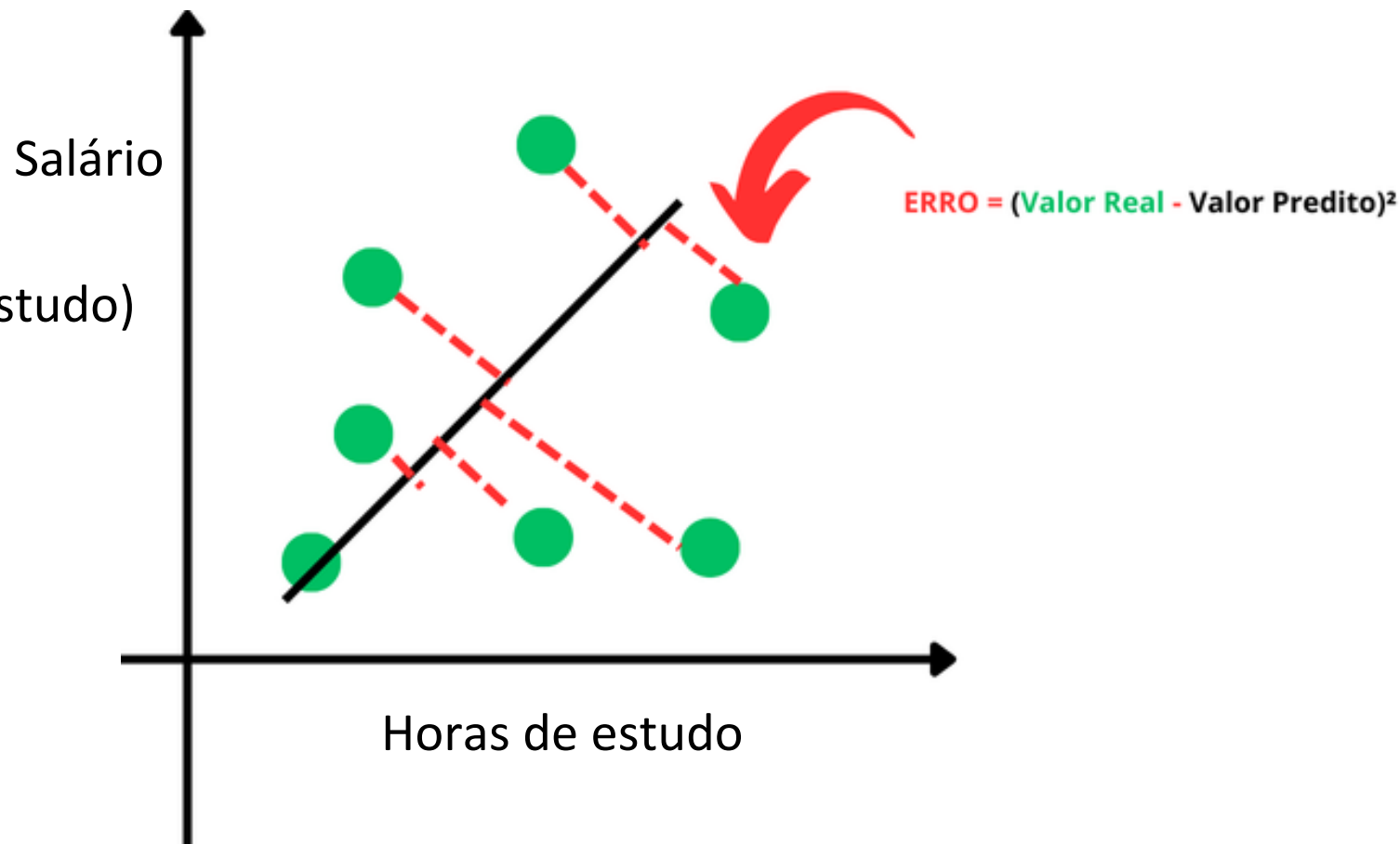


- **Resultado:** A reta que encontramos é a que "melhor" representa os dados, no sentido de minimizar a "força" total que as pessoas precisam fazer para se manterem próximas à corda.

Mínimos Quadrados: Desvendando o Segredo da "Melhor Linha"

Olhe o gráfico abaixo:

$$\text{Salário} = b + a \times (\text{Horas de estudo})$$



E se eu tivesse muitas variáveis, ainda sim poderia usar o método paramétrico?

Seleção de Variáveis

Seleção de Variáveis: Descobrimos o Que Realmente Importa!

- **O Problema:** Em muitos problemas, temos muitas variáveis preditoras. Nem todas são importantes!
- **A Solução:** Selecionar apenas as variáveis que realmente contribuem para a precisão do modelo. Isso torna o modelo mais simples, fácil de entender e com melhor desempenho.
- **Técnicas de Seleção:** Existem várias formas de fazer isso, e vamos explorar algumas das mais populares: Backward e Forward.

O que é Forward?

Seleção de Variáveis

Forward Selection

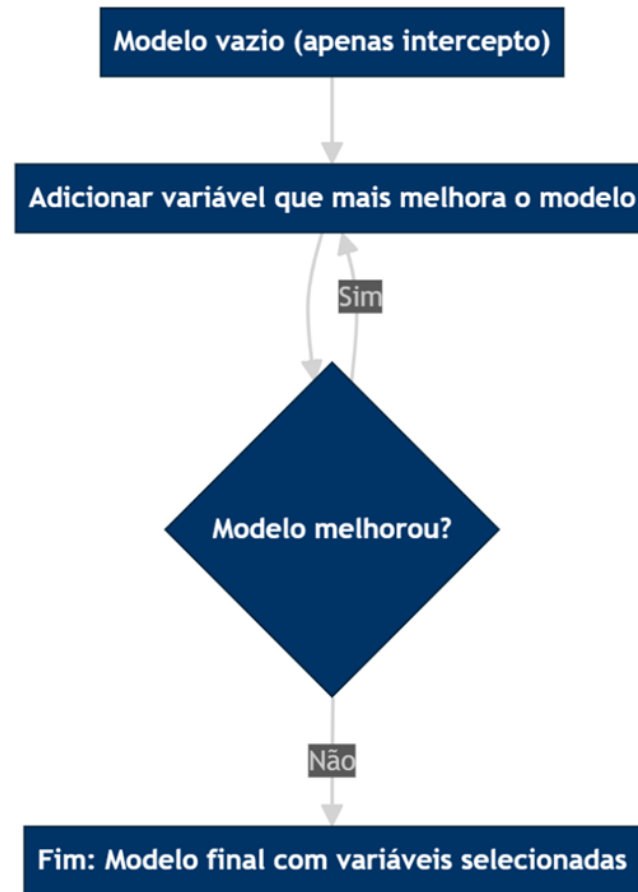
- Imagine construir uma casa. **Começamos com a fundação (a variável mais importante) e adicionamos os outros cômodos (variáveis) um por um**, sempre escolhendo o que mais contribui para a casa ficar completa.

Como funciona?

- **Começamos com um modelo vazio**, apenas com o intercepto (nosso b).
- Adicionamos a **variável que mais melhora o modelo (ex: a que mais diminui o erro)**.
- **Repetimos o processo**, adicionando uma variável por vez, até que a adição de novas variáveis não melhore significativamente o modelo.

Seleção de Variáveis

Forward Selection



Seleção de Variáveis

Backward Elimination

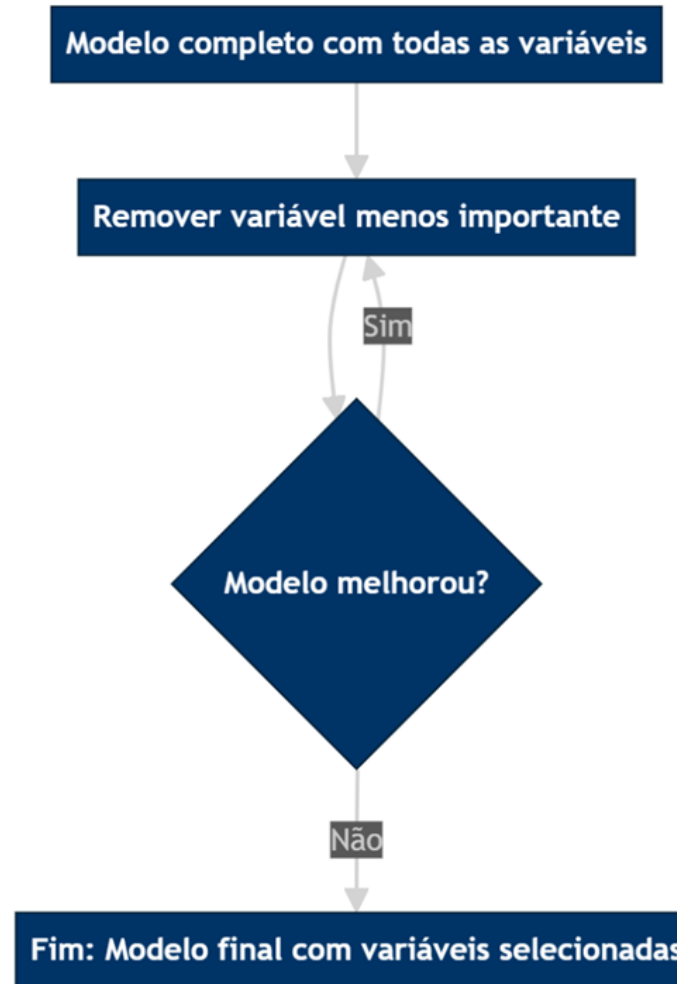
- Imagine que você se mudou para uma casa nova, mas ela veio cheia de coisas velhas. **Começamos com tudo e vamos tirando os objetos um por um até fica apenas com o essencial.**

Como funciona?

- Começamos com um modelo que **inclui todas as variáveis.**
- **Removemos a variável que menos afeta** o desempenho do modelo.
- **Repetimos o processo**, removendo uma variável por vez, até que a remoção de qualquer variável piore significativamente o modelo.

Seleção de Variáveis

Backward Elimination



Métricas de Desempenho

- Vamos analisar as principais métricas de desempenho para os modelos de regressão.

MSE (Mean Squared Error): A Média dos Erros ao Quadrado

- O MSE **mede a média dos quadrados dos erros** (diferenças entre os valores previstos e os valores reais).
- **Sensibilidade a Outliers:** Elevar os erros ao quadrado faz com que o MSE seja muito sensível a outliers (valores muito discrepantes). Um único outlier pode inflar o MSE drasticamente.
- **Amplamente usado como medida para calcular o erro do modelo**, porém a interpretação se torna mais complexa, já que está em termos quadráticos.

Como podemos sanar essa problemática de interpretação?

Métricas de Desempenho

RMSE (Root Mean Squared Error)

- O RMSE é simplesmente a **raiz quadrada do MSE**.
- Está na **mesma unidade da variável alvo** (o que facilita a interpretação).
- **Exemplo:** Se a variável resposta for o preço da casa (em Reais) o modelo tem RMSE = 50.000,0, significa que, **em média, as previsões do modelo estão a cerca de 50 mil reais de distância do valor real**.
- O RMSE mostra, em termos práticos, **o tamanho médio do erro do modelo nas mesmas unidades do problema**.

Métricas de Desempenho

MAE (Mean Absolute Error)

- O MAE mede a **média dos valores absolutos dos erros** (diferenças entre os valores previstos e os valores reais).
- **Resistência a Outliers:** O MAE é mais robusto a outliers do que o MSE e o RMSE, pois não eleva os erros ao quadrado.
- **Exemplo:** Se a variável resposta for o preço da casa (em Reais) o modelo tem RMSE = 40.000,0, significa que, **o modelo erra 40 mil reais nas previsões do preço das casas.**
- O MAE **é uma boa opção quando você quer evitar que outliers influenciem muito** a avaliação do modelo.

Métricas de Desempenho

MAPE (Mean Absolute Percentage Error)

- O MAPE mede o erro médio percentual absoluto. É útil quando a escala da variável alvo é importante.
- **Interpretação:** Um MAPE de 10% significa que, em média, as previsões do modelo estão 10% "distantes" dos valores reais.
- **Exemplo:** Se a variável resposta for o preço da casa (em Reais) o modelo tem MAPE=8%, significa que, **o modelo erra 8% do valor real da casa.**
- **Cuidado:** O MAPE pode ser instável quando a variável alvo tem valores próximos de zero.

Métricas de Desempenho

R^2 (Coeficiente de Determinação)

- O R^2 mede a proporção da variância da variável alvo que é explicada pelo modelo. Varia de 0 a 1.

Interpretação

- **$R^2 = 1$:** O modelo explica 100% da variância (ajuste perfeito).
- **$R^2 = 0$:** O modelo não explica nada da variância (não é melhor do que chutar um valor aleatório).
- **Valores entre 0 e 1:** Indicam a proporção da variância explicada pelo modelo.
- **Limitações:** O R^2 pode ser enganoso em alguns casos, pois sempre aumenta quando adicionamos mais variáveis ao modelo, mesmo que elas não sejam relevantes.

Métricas de Desempenho

AIC e BIC

- **AIC e BIC são formas de comparar modelos**, tentando equilibrar dois pontos: quão bem o modelo se ajusta aos dados e quão simples ele é (quantos parâmetros tem).
- **Valores mais baixos indicam modelos melhores.** A diferença principal é que o AIC tende a favorecer modelos que se ajustam bem, mesmo que sejam um pouco mais complexos, enquanto o BIC penaliza modelos complexos e prefere modelos mais simples e elegantes.
- **No final, a escolha depende do seu objetivo:** modelos simples são mais fáceis de entender, modelos mais complexos podem prever melhor.

Regressão Não Paramétrica

Regressão Não Paramétrica: Mais flexibilidade

- A Regressão Não Paramétrica **não impõe uma forma específica para a relação entre as variáveis**. O modelo se adapta aos dados, capturando relações complexas e não lineares.
- **É como moldar uma argila para criar uma escultura**. A argila se adapta à forma desejada, sem precisar seguir um molde predefinido.
- **Exemplo:** Prever o preço de uma casa com base em diversas características (tamanho, localização, número de quartos, etc.), sem assumir nenhuma relação específica entre elas.

Regressão Não Paramétrica

Vantagens

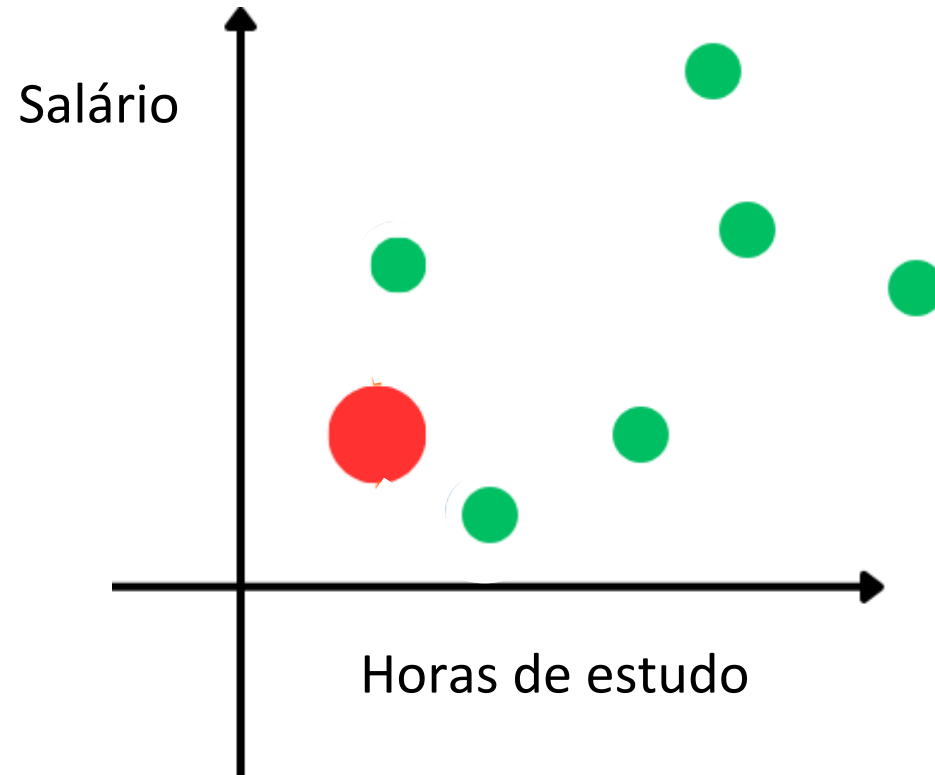
- **Flexibilidade:** Capaz de capturar relações complexas e não lineares.
- **Menor Viés:** Reduz o risco de "errar" na forma da relação entre as variáveis.
- **Exemplo:** Métodos como KNN Regressor (previsão baseada nos vizinhos mais próximos) e Árvores de Regressão (divisão dos dados em regiões) são exemplos de modelos não paramétricos.

Desafios

- **Complexidade:** Mais difícil de implementar.
- **Necessidade de Dados:** Requer um grande número de observações para evitar o sobreajuste (overfitting).

K-Nearest Neighbors (KNN)

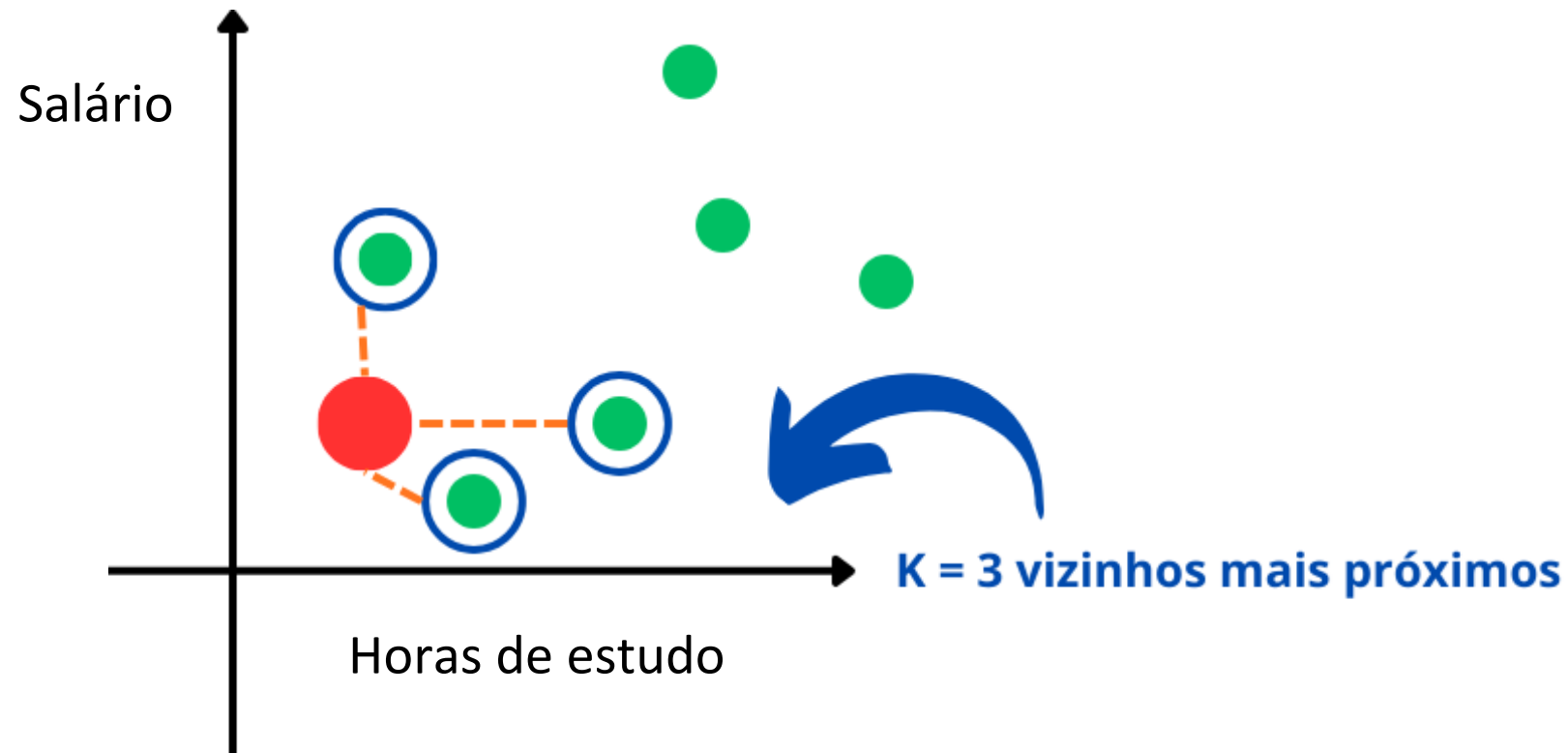
Observe a imagem abaixo:



- Quais são os 3 vizinhos mais próximos?

K-Nearest Neighbors (KNN)

Observe a imagem abaixo:



- Para prever o valor de um **novo ponto**, olhamos para os **K vizinhos mais próximos** e calculamos a **média dos valores deles**, assim o novo valor terá a média dos vizinhos mais próximos.

K-Nearest Neighbors (KNN)

O que ele faz?

- O KNN precisa encontrar as 3 pessoas ($K=3$) nos seus dados que são mais parecidas com essa nova pessoa. O critério é definido pela proximidade em relação às horas de estudo. No caso, você buscaria as 3 pessoas mais próximas nos seus dados históricos.
- Como queremos prever o salário, você calcularia a média dos salários médios das regiões dos 3 vizinhos. Essa média seria a sua previsão para o salário da nova pessoa.

Sobre os hiperparâmetros

- Temos vários tipos de hiperparâmetros que podemos utilizar, fique a vontade para entrar na documentação do Scikit-learn e explorá-los [clikando aqui](#).

Regressão Não Paramétrica

Exemplos de Hiperparâmetros:

K: Número de vizinhos a serem considerados

- **K pequeno:** Modelo mais flexível, mas mais sensível a ruído.
- **K grande:** Modelo mais estável, mas pode ignorar detalhes importantes.

Métrica de Distância: Como medir a "proximidade" entre os pontos

- **Ex:** Distância Euclidiana, Distância de Manhattan.

E o modelo que usamos na aula passada?

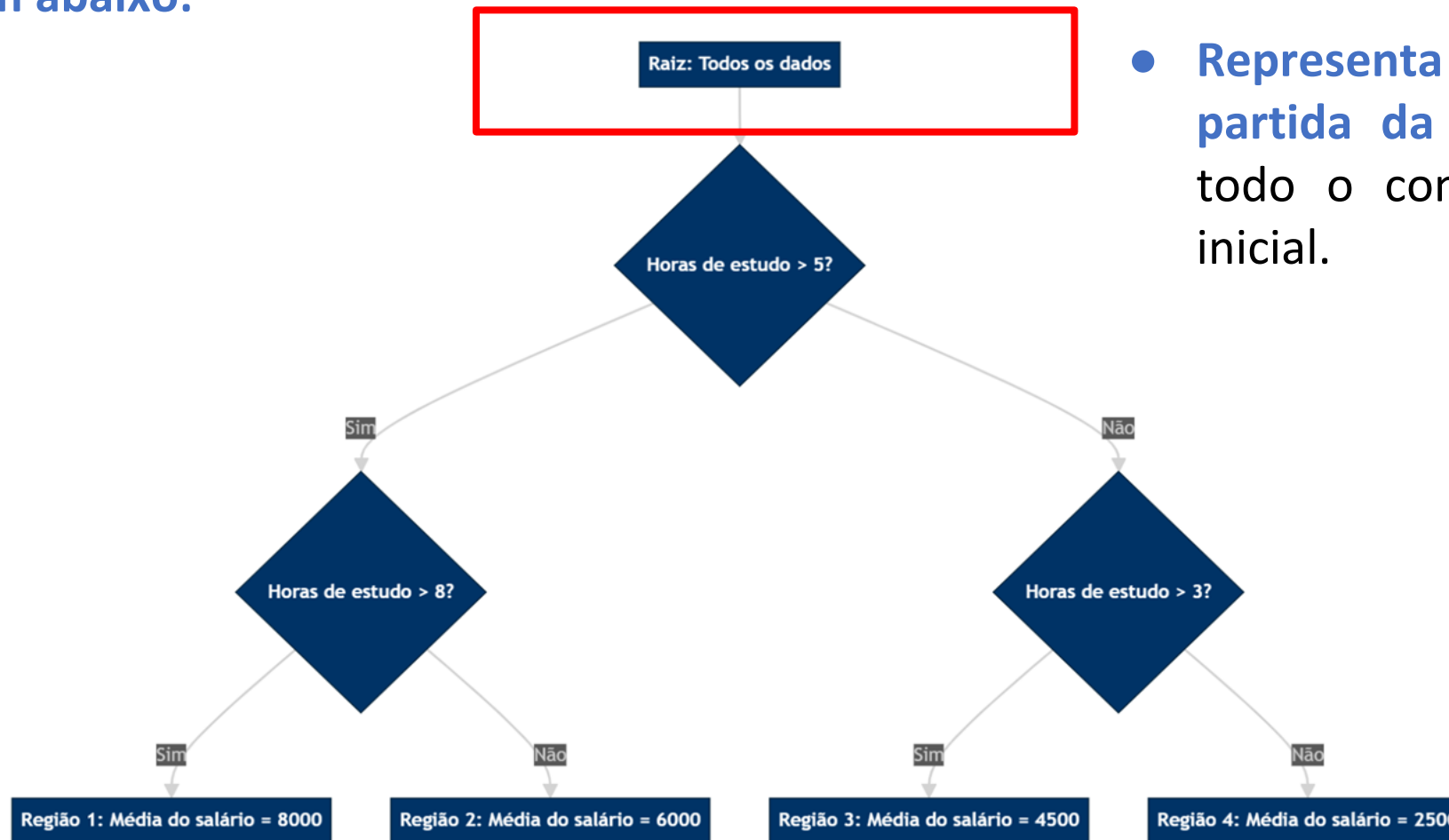
Árvores de Decisão: Intuição inicial

O que ela faz?

- **Dividindo para Conquistar:** A árvore de decisão começa com todos os dados juntos (como uma grande lista de pessoas).
- Então, **ela faz perguntas para dividir essa lista em grupos menores**, por exemplo: "Essa pessoa estudou mais de 5 horas por dia?".
- **Perguntas Estratégicas:** A árvore não faz perguntas aleatórias. Ela escolhe as perguntas que melhor separam as pessoas em grupos com salários mais parecidos.
- Por exemplo, **se a pergunta sobre horas de estudo ajudar a separar pessoas com salários altos e baixos**, ela será usada.

Árvores de Decisão: Intuição inicial

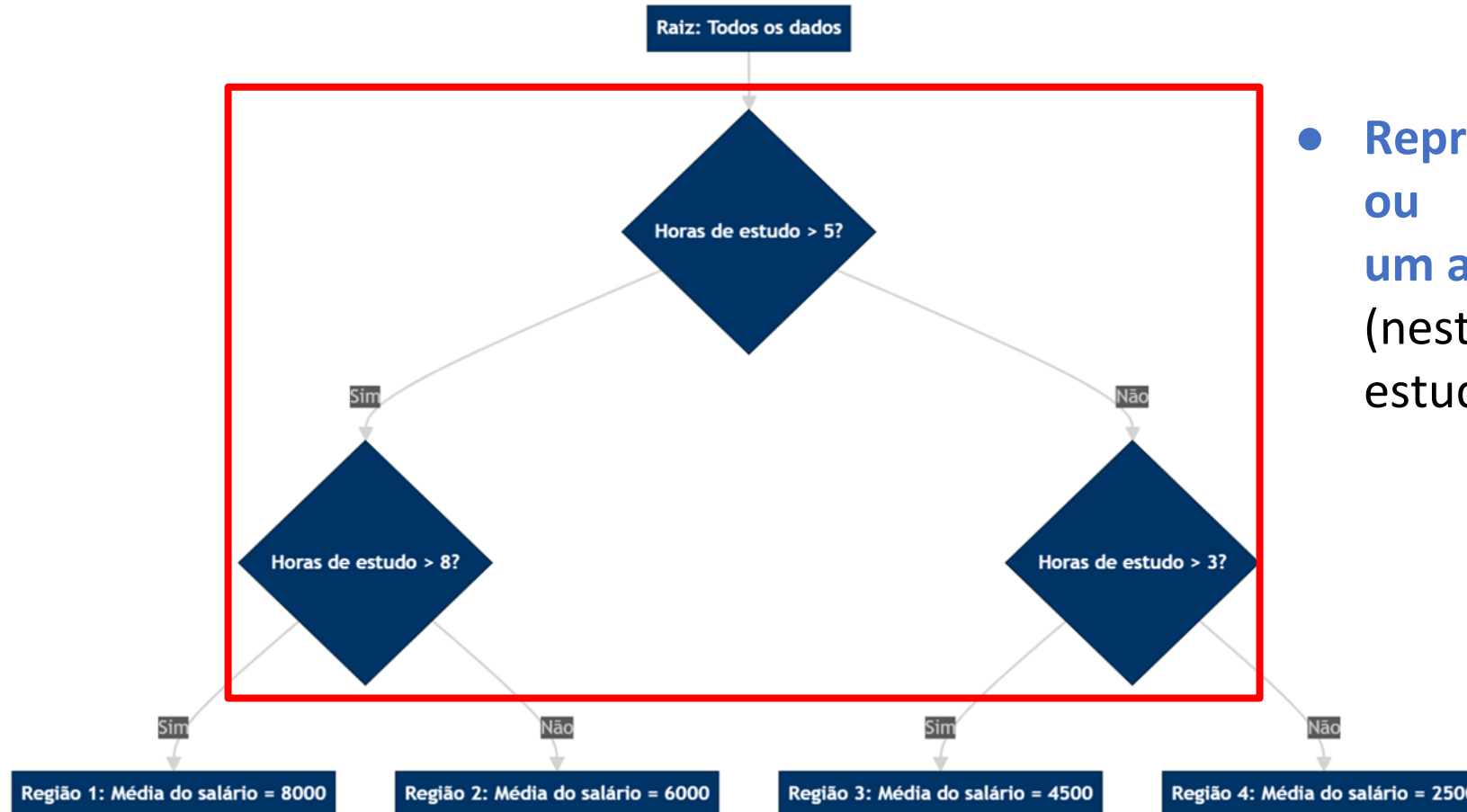
Observe a imagem abaixo:



- Representa o ponto de partida da árvore. Contém todo o conjunto de dados inicial.

Árvores de Decisão: Intuição inicial

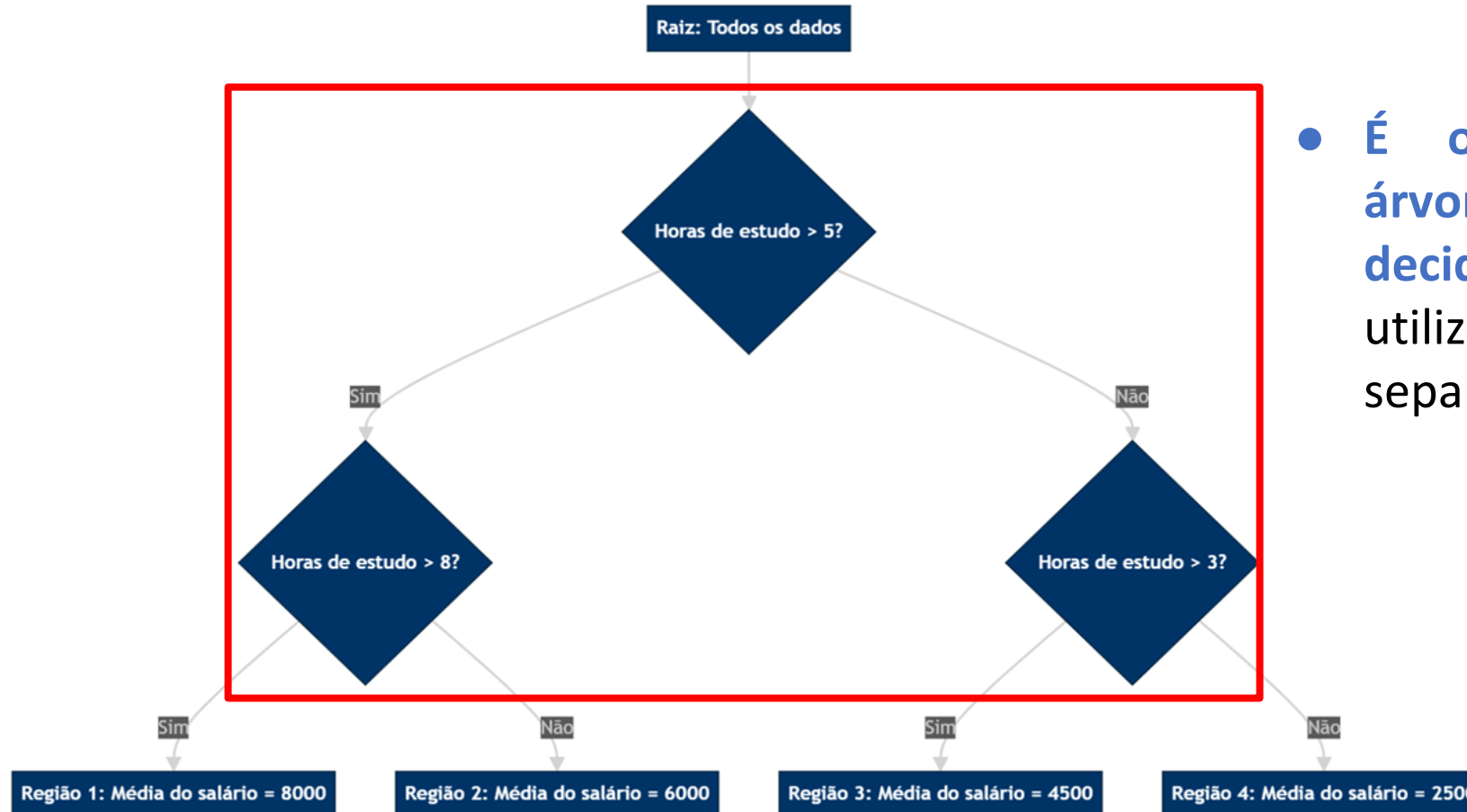
Observe a imagem abaixo:



- Representam testes ou perguntas sobre um atributo dos dados (neste caso, "Horas de estudo").

Árvores de Decisão: Intuição inicial

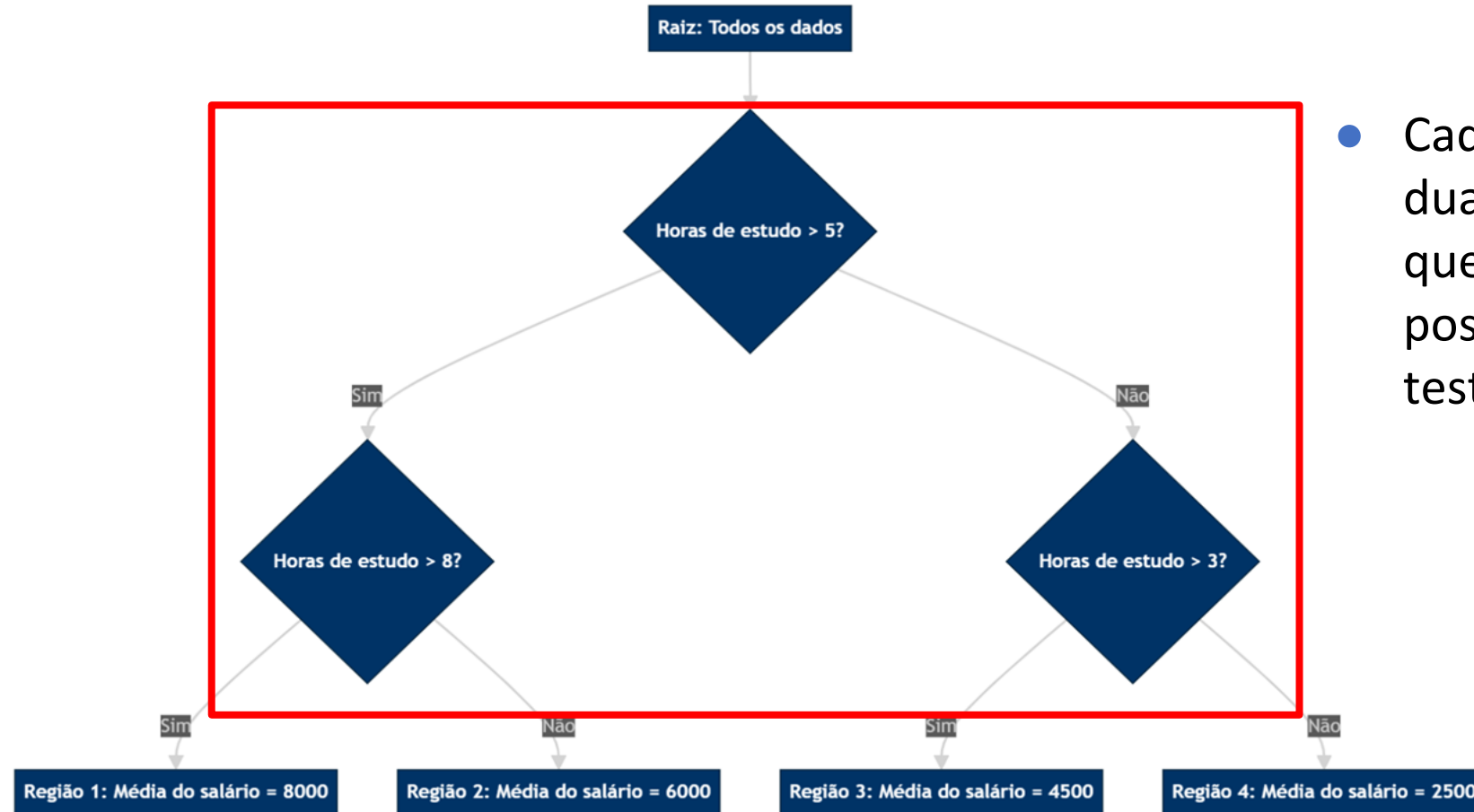
Observe a imagem abaixo:



- É o algoritmo da árvore de decisão que **decide** quais variáveis utilizar e de que forma separar os dados!

Árvores de Decisão: Intuição inicial

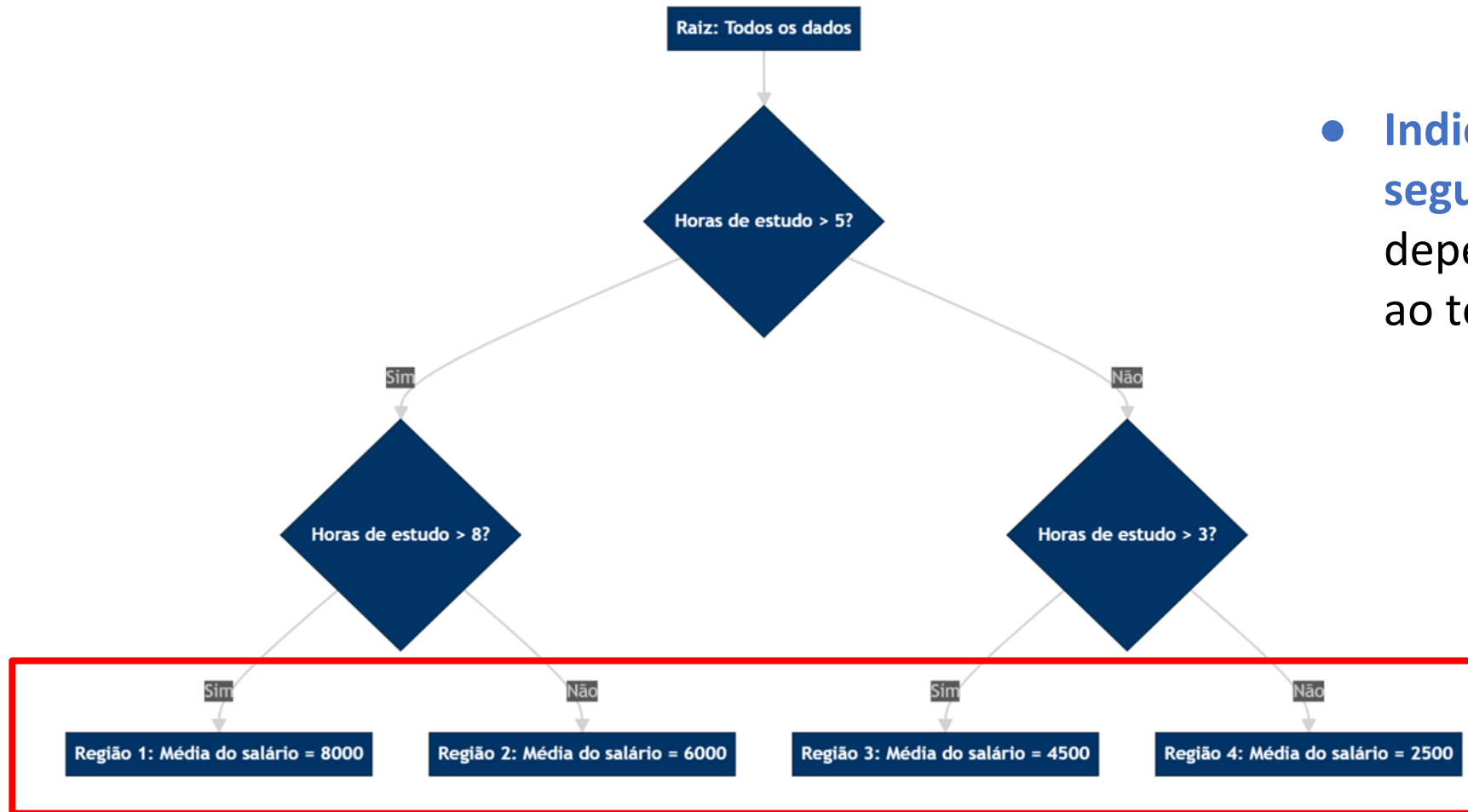
Observe a imagem abaixo:



- Cada nó de decisão tem duas ramificações (arestas) que correspondem às possíveis respostas para o teste (Sim/Não).

Árvores de Decisão: Intuição inicial

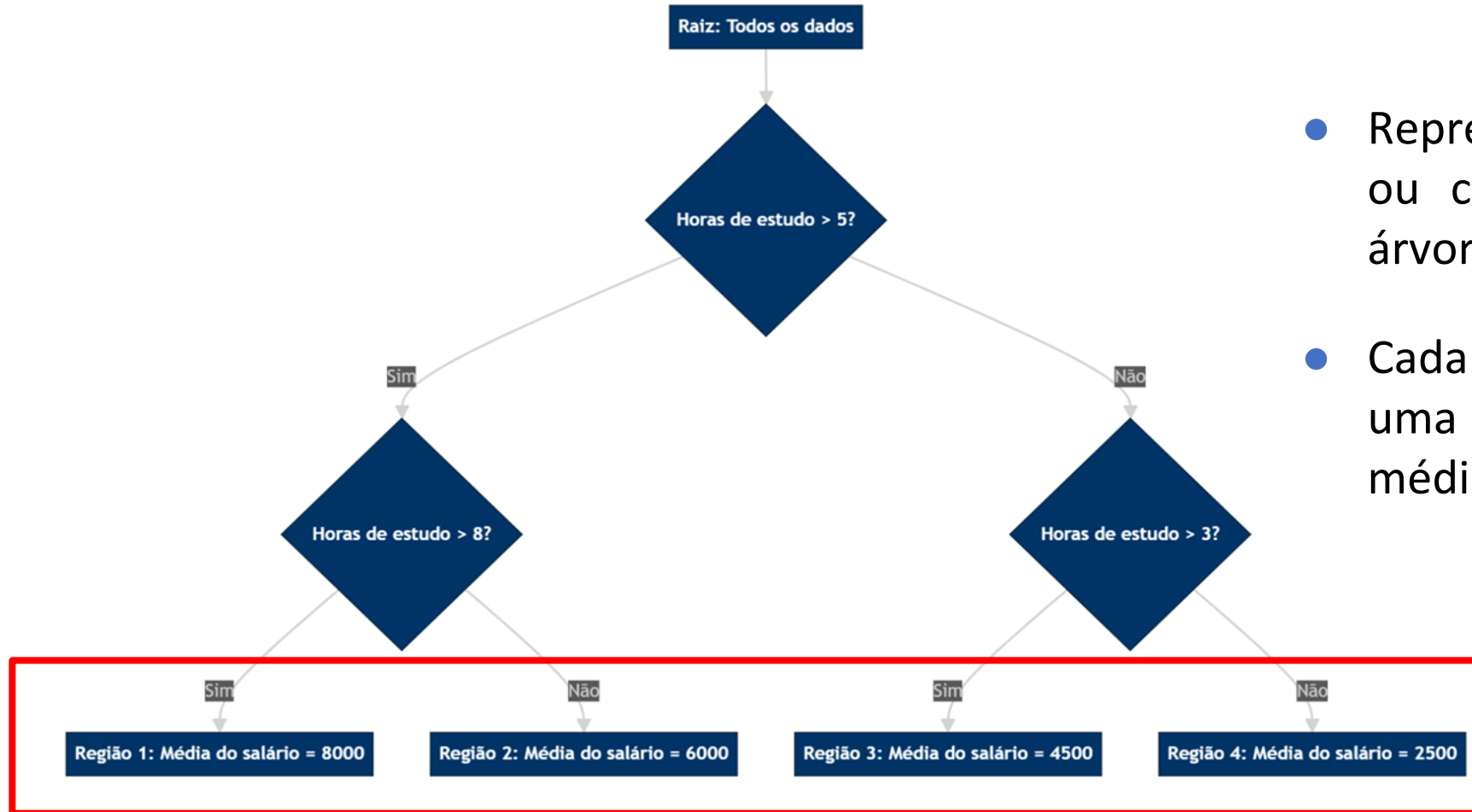
Observe a imagem abaixo:



- Indicam o caminho a seguir na árvore dependendo da resposta ao teste no nó de decisão.

Árvores de Decisão: Intuição inicial

Observe a imagem abaixo:



- Representam as previsões ou classificações finais da árvore.
- Cada folha corresponde a uma região com uma média salarial associada.

Árvores de Decisão: Intuição inicial

Exemplos de Hiperparâmetros:

Profundidade Máxima da Árvore: Controla a complexidade da árvore.

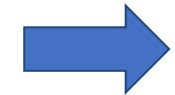
- **Árvores mais profundas:** Podem se ajustar melhor aos dados de treinamento, mas correm o risco de overfitting.
- **Árvores mais rasas:** São mais simples e fáceis de interpretar, mas podem ter baixo desempenho.

Número mínimo de amostras que um nó precisa ter para ser dividido.

- **Número pequeno:** Corre o risco de overfitting, mas pode se ajustar melhor aos dados.
 - **Número grande:** Árvores mais simples, regiões grandes, risco de underfitting.
- Para mais detalhes sobre hiperparâmetros, [clique aqui](#).

Recapitulando: 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
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score # métricas de avaliação do modelo
from sklearn.neighbors import KNeighborsRegressor # modelo KNN

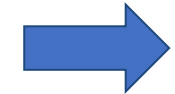
# Carregando dataset
df = pd.read_csv("aula_01_exemplo_01.csv") # Lê o arquivo aula_01_exemplo_01.csv em um DataFrame

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
```

- Aqui, adicionaremos os comandos para conseguir manipular o modelo de KNN e as novas métricas de avaliações que aprendemos nessa aula!

Recapitulando: 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
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score # métricas de avaliação do modelo
from sklearn.neighbors import KNeighborsRegressor # modelo KNN

# 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
```

O código acima não retorna nada pra gente!

Recapitulando: Dividindo os Dados da aula_01_exemplo_01

- Agora vamos deixar claro quais são as variáveis categóricas e quais são as numéricas.


Código

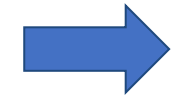
```
[5]: # Definição explícita das colunas por tipo
variaveis_categoricas = ["sex", "smoker", "region"] # listas com nomes das colunas categóricas (devem bater exatamente com os nomes do DataFrame)
variaveis_numericas = ["age", "bmi", "children", "tem_filhos"] # lista das colunas numéricas que serão escalonadas
```

- Perceba que não incluímos a variável target, já que fazemos manipulações de padronizar e transformar apenas nas covariáveis.

O código acima não retorna nada pra gente!

Recapitulando: Dividindo os Dados da aula_01_exemplo_01

- Novamente vamos separar nossos dados de forma que fique claro 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.

Recapitulando: Dividindo os Dados da aula_01_exemplo_01


- Vamos dividir nossos dados em dados de treinamento e dados de teste. Usaremos o comando `train_test_split()`.


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)
```

- Sempre definindo quais são nossas covariáveis e variável target.

Recapitulando: Preparando os Dados da aula_01_exemplo_01

- Vamos recriar 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
```

O código acima não retorna nada pra gente!

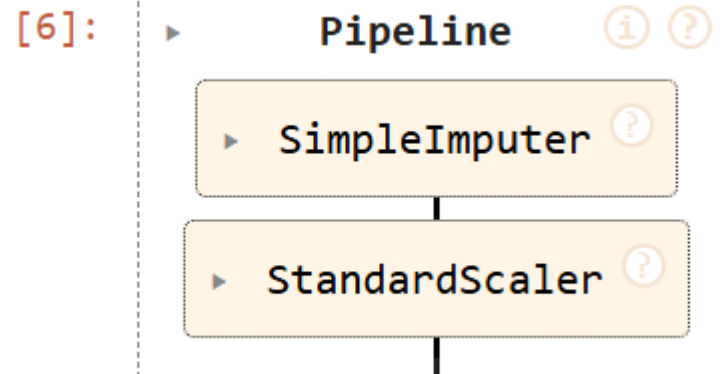
Recapitulando: Preparando os Dados da aula_01_exemplo_01

- Vamos agora recriar a criação das etapas numéricas e categóricas no python.

→
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



- Criando a etapa numérica.

Recapitulando: Preparando os Dados da aula_01_exemplo_01

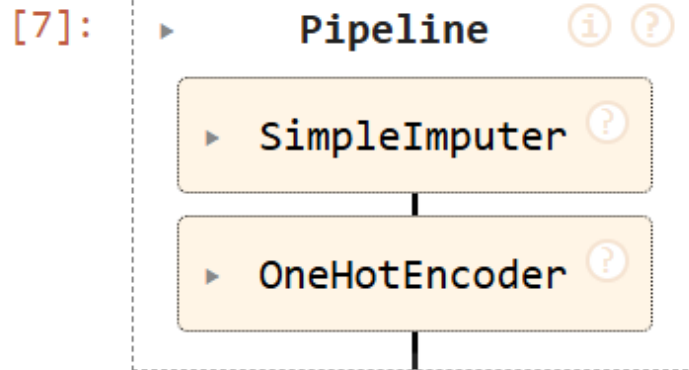
- Vamos agora recriar a criação das etapas numéricas e categóricas no python.

→
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



- Criando a etapa categórica.

Recapitulando: Preparando os Dados da aula_01_exemplo_01

- Agora vamos juntar tudo novamente.


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, categorizador 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
```

- O **ColumnTransformer()** vai servir como um processo de união entre as etapas categóricas e contínuas.

O código acima não retorna nada pra gente!

Modelando os Dados da aula_01_exemplo_01

- Agora vamos criar os novos comandos para inserir os modelos que aprendemos nessa aula. Os comandos anteriores são repetições da aula passada e já foram escritos previamente, basta rodá-los previamente.
- Vamos criar o modelos de árvore de decisão e o KNN. Começaremos pela árvore de decisão.



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

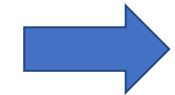


Código

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

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 são os mesmos hiperparâmetros que vimos: “**max_depth**” é a profundidade máxima da árvore e o “**min_samples_split**” é Número mínimo de amostras que um nó precisa ter para ser dividido.

O código acima não retorna nada pra gente!

Modelando os Dados da aula_01_exemplo_01

- Agora vamos usar o Grid Search para encontrar 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
      )
```

- Como o RMSE tem a característica de ser interpretável, vamos seguir utilizando ele como critério para selecionar os melhores hiperparâmetros.

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

- Agora temos o melhor modelo para o caso da árvore de decisão.

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 com as novas métricas.



Código


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

# Métricas de avaliação
mse_treino_arvore = mean_squared_error(y_treino, y_pred_treino_arvore) # Mean Squared Error
rmse_treino_arvore = np.sqrt(mse_treino_arvore) # Root Mean Squared Error
mae_treino_arvore = mean_absolute_error(y_treino, y_pred_treino_arvore) # Mean Absolute Error
r2_treino_arvore = r2_score(y_treino, y_pred_treino_arvore) # R²: proporção da variância explicada pelo modelo
mape_treino_arvore = np.mean(np.abs((y_treino - y_pred_treino_arvore) / y_treino)) * 100 # Mean Absolute Percentage Error (%)
```

- O comando **mean_squared_error()** calcula o MSE. Já o comando **np.sqrt()** calcula a raiz do MSE o que consequentemente faz com que tenhamos o RMSE. O MAE é calculado pelo **mean_absolute_error()** e o r^2 pelo **r2_score()**.

Modelando os Dados da aula_01_exemplo_01

- Por fim, vamos analisar o desempenho do modelo nos dados de treinamento com as novas métricas.


Código

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

      # Métricas de avaliação
      mse_treino_arvore = mean_squared_error(y_treino, y_pred_treino_arvore) # Mean Squared Error
      rmse_treino_arvore = np.sqrt(mse_treino_arvore) # Root Mean Squared Error
      mae_treino_arvore = mean_absolute_error(y_treino, y_pred_treino_arvore) # Mean Absolute Error
      r2_treino_arvore = r2_score(y_treino, y_pred_treino_arvore) # R²: proporção da variância explicada pelo modelo
      mape_treino_arvore = np.mean(np.abs((y_treino - y_pred_treino_arvore) / y_treino)) * 100 # Mean Absolute Percentage Error (%)
```

- como o MAPE não possui uma função própria, temos que criar do zero! primeiro começando de dentro, calculamos o **valor absoluto dos valores reais com os preditos divididos pelos valores reais com o comando `np.abs()`**.

Modelando os Dados da aula_01_exemplo_01

- Por fim, vamos analisar o desempenho do modelo nos dados de treinamento com as novas métricas.


Código

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

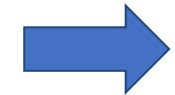
      # Métricas de avaliação
      mse_treino_arvore = mean_squared_error(y_treino, y_pred_treino_arvore) # Mean Squared Error
      rmse_treino_arvore = np.sqrt(mse_treino_arvore) # Root Mean Squared Error
      mae_treino_arvore = mean_absolute_error(y_treino, y_pred_treino_arvore) # Mean Absolute Error
      r2_treino_arvore = r2_score(y_treino, y_pred_treino_arvore) # R²: proporção da variância explicada pelo modelo
      mape_treino_arvore = np.mean(np.abs((y_treino - y_pred_treino_arvore) / y_treino)) * 100 # Mean Absolute Percentage Error (%)
```

- Agora pegamos tudo isso e utilizamos o comando **np.mean()** para calcular a média e **por fim multiplicamos tudo por 100**.

O código acima não retorna nada pra gente!

Modelando os Dados da aula_01_exemplo_01

- Agora vamos mostrar as métricas calculadas.



Código

```
[14]: # Resultados
print("Melhores parâmetros encontrados:", grid_search.best_params_)
print("R² no treino:", np.round(r2_treino_arvore, 3))
print("MSE no treino:", np.round(mse_treino_arvore, 3))
print("RMSE no treino:", np.round(rmse_treino_arvore, 3))
print("MAE no treino:", np.round(mae_treino_arvore, 3))
print("MAPE no treino (%)ate:", np.round(mape_treino_arvore, 2))
```



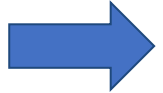
Saída

```
Melhores parâmetros encontrados: {'max_depth': 3, 'min_samples_split': 2}
R² no treino: 0.854
MSE no treino: 21120357.016
RMSE no treino: 4595.689
MAE no treino: 2785.253
MAPE no treino (%): 34.69
```

- O modelo de árvore de decisão explica cerca de 85% da variabilidade dos custos médicos nos EUA ($R^2 = 0,854$). Em média, a previsão difere do valor real em 2.785 dólares (MAE), com RMSE = 4.595 dólares e MAPE = 34,7%, indicando que o modelo captura bem as tendências gerais.

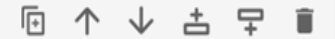
Modelando os Dados da aula_01_exemplo_01

- Agora vamos implementar o modelo KNN.



Código

```
[15]: # criando o modelo  
modelo_knn = KNeighborsRegressor() # criando o modelo KNN; aqui não há aleatoriedade fixa como na árvore
```



- Note que o KNN não tem o **random_state**.

O código acima não retorna nada pra gente!

Modelando os Dados da aula_01_exemplo_01

- Agora vamos criar nossa grade de hiperparâmetros.


Código

```
[16]:  
# Grade de hiperparâmetros  
param_grid_knn = {  
    "n_neighbors": [3, 5, 7, 9, 11], # Número de vizinhos a considerar na média para prever  
    "p": [1, 2] # Tipo de distância: 1 = Manhattan, 2 = Euclidiana  
}
```

- Perceba que são os mesmos hiperparâmetros que discutimos antes!

O código acima não retorna nada pra gente!

Modelando os Dados da aula_01_exemplo_01

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


Código

[17]:

```
# Configurando Grid Search
grid_search_knn = GridSearchCV(
    estimator=modelo_knn,          # modelo KNN a ser otimizado
    param_grid=param_grid_knn,    # Grade de hiperparâmetros
    cv=5,                          # 5-fold cross-validation: o dataset será dividido em 5 partes
    scoring="neg_root_mean_squared_error" # Métrica de comparação: RMSE (quanto menor, melhor)
)

# Treinando com Grid Search
grid_search_knn.fit(x_treino_transformado, y_treino) # usando as covariáveis já preprocessadas e nossa variável target

melhor_modelo_knn = grid_search_knn.best_estimator_ # Melhor modelo KNN encontrado
```

- Vamos continuar utilizando o RMSE.

O código acima não retorna nada pra gente!

Modelando os Dados da aula_01_exemplo_01

- Da mesma forma que a árvore de decisão, vamos calcular as métricas.



Código

```
[18]: # Predição no treino
y_pred_treino_knn = melhor_modelo_knn.predict(x_treino_transformado) # covariáveis de treino

# Avaliação do modelo no treino
mse_treino_knn = mean_squared_error(y_treino, y_pred_treino_knn) # Valores reais, Valores preditos

# Métricas de avaliação
mse_treino_knn = mean_squared_error(y_treino, y_pred_treino_knn) # Mean Squared Error
rmse_treino_knn = np.sqrt(mse_treino_knn) # Root Mean Squared Error
mae_treino_knn = mean_absolute_error(y_treino, y_pred_treino_knn) # Mean Absolute Error
r2_treino_knn = r2_score(y_treino, y_pred_treino_knn) # R²: proporção da variância explicada pelo modelo
mape_treino_knn = np.mean(np.abs((y_treino - y_pred_treino_knn) / y_treino)) * 100 # Mean Absolute Percentage Error (%)
```

- Note que todos os passos se repetem, apenas mudando o modelo!

O código acima não retorna nada pra gente!

Modelando os Dados da aula_01_exemplo_01

- Agora vamos mostrar as métricas calculadas.


Código

```
[19]: # Resultados
print("Melhores parâmetros encontrados:", grid_search.best_params_)
print("R² no treino:", np.round(r2_treino_knn, 3))
print("MSE no treino:", np.round(mse_treino_knn, 3))
print("RMSE no treino:", np.round(rmse_treino_knn, 3))
print("MAE no treino:", np.round(mae_treino_knn, 3))
print("MAPE no treino (%):", np.round(mape_treino_knn, 2))
```


Saída

```
Melhores parâmetros encontrados: {'n_neighbors': 3, 'p': 1}
R² no treino: 0.861
MSE no treino: 20116578.002
RMSE no treino: 4485.151
MAE no treino: 2464.023
MAPE no treino (%): 25.87
```

- **O modelo KNN explica cerca de 86% da variabilidade dos custos médicos nos EUA ($R^2 = 0,861$).** Em média, a previsão difere do valor real em 2.464 dólares (MAE), com RMSE = 4.485 dólares e MAPE = 25,9%, indicando que o modelo captura bem as tendências gerais, embora possa divergir em casos extremos.

Modelando os Dados da aula_01_exemplo_01

- Agora vamos ver como cada modelo se comporta no teste. Começaremos com a árvore de decisão.


Código

```
[20]: # Predição no teste
y_pred_teste_arvore = melhor_modelo_arvore.predict(x_teste_transformado) # covariáveis de teste

# Métricas de avaliação
mse_teste_arvore = mean_squared_error(y_teste, y_pred_teste_arvore) # Mean Squared Error
rmse_teste_arvore = np.sqrt(mse_teste_arvore) # Root Mean Squared Error
mae_teste_arvore = mean_absolute_error(y_teste, y_pred_teste_arvore) # Mean Absolute Error
r2_teste_arvore = r2_score(y_teste, y_pred_teste_arvore) # R²: proporção da variância explicada pelo modelo
mape_teste_arvore = np.mean(np.abs((y_teste - y_pred_teste_arvore) / y_teste)) * 100 # Mean Absolute Percentage Error (%)
```

- Da mesma forma que os anteriores, o que muda é a base de dados!

O código acima não retorna nada pra gente!

Modelando os Dados da aula_01_exemplo_01

- Vamos observar o desempenho.

Código

```
[21]: # Resultados
print("Avaliação do modelo de Árvore de Decisão no teste:")
print("R²:", np.round(r2_teste_arvore, 3))
print("MSE:", np.round(mse_teste_arvore, 3))
print("RMSE:", np.round(rmse_teste_arvore, 3))
print("MAE:", np.round(mae_teste_arvore, 3))
print("MAPE (%):", np.round(mape_teste_arvore, 2))
```

Saída

```
Avaliação do modelo de Árvore de Decisão no teste:
R²: 0.853
MSE: 22812669.852
RMSE: 4776.261
MAE: 2865.638
MAPE (%): 37.68
```

- o modelo de árvore de decisão apresentou $R^2 = 0,853$, indicando que cerca de 85% da variabilidade dos custos médicos nos EUA foi explicada pelo modelo. As previsões apresentam MAE de 2.866 dólares, RMSE de 4.776 dólares e MAPE de 37,7%.

Modelando os Dados da aula_01_exemplo_01

- Agora vamos analisar pro KNN.



Código

```
[22]: # Predição no teste usando o melhor modelo KNN
      y_pred_teste_knn = melhor_modelo_knn.predict(x_teste_transformado) # covariáveis de teste

      # Métricas de avaliação
      mse_teste_knn = mean_squared_error(y_teste, y_pred_teste_knn)           # Mean Squared Error
      rmse_teste_knn = np.sqrt(mse_teste_knn)                               # Root Mean Squared Error
      mae_teste_knn = mean_absolute_error(y_teste, y_pred_teste_knn)         # Mean Absolute Error
      r2_teste_knn = r2_score(y_teste, y_pred_teste_knn)                    # R²: proporção da variância explicada pelo modelo
      mape_teste_knn = np.mean(np.abs((y_teste - y_pred_teste_knn) / y_teste)) * 100 # Mean Absolute Percentage Error (%)
```

- Agora conseguiremos comparar o desempenho dos dois modelos.

O código acima não retorna nada pra gente!

Modelando os Dados da aula_01_exemplo_01

- Vamos observar o desempenho.


Código

```
[23]: # Resultados
print("Avaliação do modelo KNN no teste:")
print("R²:", np.round(r2_teste_knn, 3))
print("MSE:", np.round(mse_teste_knn, 3))
print("RMSE:", np.round(rmse_teste_knn, 3))
print("MAE:", np.round(mae_teste_knn, 3))
print("MAPE (%):", np.round(mape_teste_knn, 2))
```


Saída

```
Avaliação do modelo KNN no teste:
R²: 0.708
MSE: 45388801.136
RMSE: 6737.121
MAE: 3996.595
MAPE (%): 42.63
```

- O modelo KNN apresentou $R^2 = 0,708$, indicando que explica cerca de 71% da variabilidade dos custos médicos nos EUA. As previsões têm MAE de 3.997 dólares, RMSE de 6.737 dólares e MAPE de 42,6%,

Modelando os Dados da aula_01_exemplo_01

- Vamos observar o desempenho.


Código

```
[23]: # Resultados
print("Avaliação do modelo KNN no teste:")
print("R²:", np.round(r2_teste_knn, 3))
print("MSE:", np.round(mse_teste_knn, 3))
print("RMSE:", np.round(rmse_teste_knn, 3))
print("MAE:", np.round(mae_teste_knn, 3))
print("MAPE (%):", np.round(mape_teste_knn, 2))
```


Saída

```
Avaliação do modelo KNN no teste:
R²: 0.708
MSE: 45388801.136
RMSE: 6737.121
MAE: 3996.595
MAPE (%): 42.63
```

Vamos organizar as medidas?

Considerações Finais

Modelo	Dados	R^2	MSE	RMSE	MAE	MAPE (%)
Árvore de Decisão	Treino	0,854	21.120.357	4.595	2.785	34,69
	Teste	0,853	22.812.670	4.776	2.866	37,68
KNN	Treino	0,861	20.116.578	4.485	2.464	25,87
	Teste	0,708	45.388.801	6.737	3.997	42,63

- Note que Para o **KNN** por mais que o **RMSE, MSE, MAE, R^2 e MAPE** sejam os **menores no treino**, quando testamos na base de teste, os valores das medidas aumentam invés de permanecer ou diminuir.

Considerações Finais

Modelo	Dados	R^2	MSE	RMSE	MAE	MAPE (%)
Árvore de Decisão	Treino	0,854	21.120.357	4.595	2.785	34,69
	Teste	0,853	22.812.670	4.776	2.866	37,68
KNN	Treino	0,861	20.116.578	4.485	2.464	25,87
	Teste	0,708	45.388.801	6.737	3.997	42,63

- Agora para a árvore de decisão, demos que as medidas permanecem muito próximas, o que caracteriza como um bom modelo de generalização.
- Portanto, a árvore de decisão pode ser dita como um bom modelo para nossos dados.

Próximos Passos

- Até aqui, trabalhamos com modelos supervisionados de regressão, cujo objetivo foi prever valores numéricos contínuos.
- Exploramos algoritmos como a Árvore de Decisão e o KNN, analisando seu desempenho tanto no conjunto de treino quanto no de teste. Essa etapa permitiu compreender como diferentes métodos se ajustam aos dados e como avaliamos suas previsões.
- Agora, passaremos para o aprendizado supervisionado de classificação, que se concentra em prever categorias em vez de valores numéricos. Enquanto a regressão busca estimar um número, a classificação procura atribuir um rótulo a cada observação. Esse tipo de problema é muito comum em diversas aplicações práticas

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.
- DEMÉTRIO, Clarice Garcia Borges; ZOCCHI, Sílvio Sandoval. Modelos de regressão. Piracicaba: Departamento de Ciências Exatas, ESALQ/USP, 2011. Disponível em: https://www.researchgate.net/publication/266233241_Modelos_de_Regressao. Acesso em: 23 set. 202
- 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.

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.

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.

Regressão no Aprendizado Supervisionado: Métodos e Avaliação

Obrigada!

Profa. Dra. Roberta Wichmann

roberta.wichmann@idp.edu.br



INSTITUTO BRASILEIRO DE ENSINO,
DESENVOLVIMENTO E PESQUISA