

# Distribuição Normal

## Características

- Modelo de dados contínuos.
- Exemplo: Altura de pessoas.
- Formato de sino: valores próximos à média são mais comuns.

## Gráfico

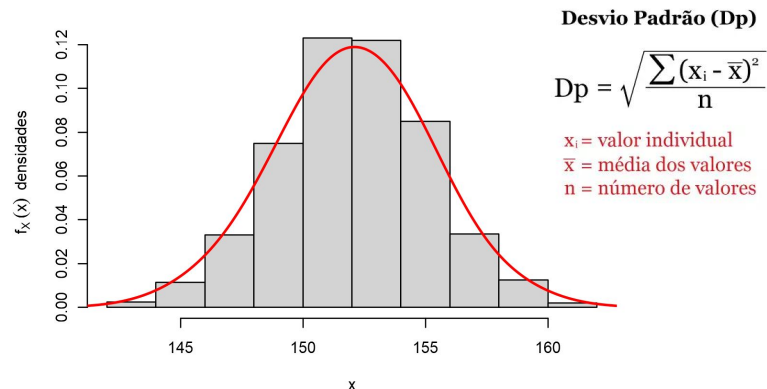
- Larga no meio (média).
- Estreita nas extremidades (valores raros).

Z-score: quão próximo da média.

$x$ : O valor observado (dado que você quer analisar).

$\mu$ : A média da distribuição.

$\sigma$ : O desvio padrão da distribuição.



$$Z = \frac{(x - \mu)}{\sigma}$$

Média ( $\mu$ ) de uma turma em um teste: 70 pontos.

Desvio padrão ( $\sigma$ ): 10 pontos.

Aluno tirou  $x = 85$ .

$$Z = \frac{85 - 70}{10} = 1,5$$

O Z-score é **1,5**, ou seja, o aluno está **1,5 desvios padrão acima da média**.

# Distribuição Binominal

## Características

- Experimentos com dois resultados possíveis (sucesso ou falha).
- Exemplo: Jogar uma moeda 10 vezes.
  - Resultados: "cara" ou "coroa".
- Aplicações: Aprovação em provas, jogos, etc.

$$P(X = k) = \binom{n}{k} \cdot p^k \cdot (1 - p)^{n-k}$$

$P(X = k)$ : Probabilidade de obter exatamente  $k$  sucessos.

$\binom{n}{k}$ : Número de combinações possíveis de  $k$  sucessos em  $n$  tentativas. Calculado como:

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

$p$ : Probabilidade de sucesso em uma única tentativa.

$1 - p$ : Probabilidade de falha.

$k$ : Número de sucessos desejados.

$n$ : Número total de tentativas.

$$P(X = 3) = \binom{5}{3} \cdot (0,5)^3 \cdot (1 - 0,5)^{5-3} \qquad \binom{5}{3} = \frac{5!}{3!(5-3)!} = \frac{5 \cdot 4}{2 \cdot 1} = 10$$

$$P(X = 3) = 10 \cdot (0,5)^3 \cdot (0,5)^2 = 10 \cdot 0,125 \cdot 0,25 = 0,3125$$

A probabilidade de obter exatamente 3 "caras" é **31,25%**.

# Distribuição de Poisson

$$P(X = k) = \frac{\lambda^k \cdot e^{-\lambda}}{k!}$$

## Características

- Modela eventos raros em intervalos de tempo ou espaço.
- Exemplo: Quantas pessoas entram em uma loja por hora.
- Gráfico:
  - Poucos eventos ocorrem com frequência.
  - Valores muito altos ou baixos são raros.

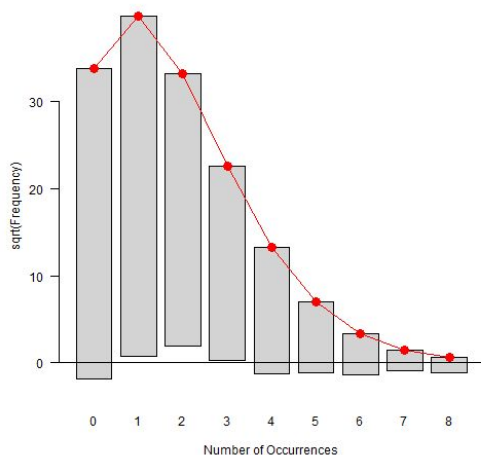
$P(X = k)$ : Probabilidade de ocorrer exatamente  $k$  eventos no intervalo.

$\lambda$ : Média esperada de eventos por intervalo (valor esperado).

$k$ : Número de eventos reais que você quer calcular a probabilidade.

$e$ : Número de Euler (aproximadamente 2,718).

$k!$ : Fatorial de  $k$  ( $k! = k \cdot (k - 1) \cdot (k - 2) \cdot \dots \cdot 1$ ).



Uma pizzeria recebe, em média, **3 pedidos por hora** ( $\lambda=3$ ). Qual é a probabilidade de receber exatamente **5 pedidos** em uma hora ( $k=5$ )?

$$P(X = 5) = \frac{3^5 \cdot e^{-3}}{5!}$$

A probabilidade de receber exatamente **5 pedidos** em uma hora é **10,09%**.

# Importância para Ciências de Dados e IA

## Estatística e Probabilidade na Prática

- Transformam dados brutos em conhecimento útil.
- Estatística Básica:
  - Organização e resumo de dados.
  - Identificação de padrões e variabilidade.
  - Exemplo: Análise de tendências de consumo.
- Estatística Inferencial:
  - Conclusões baseadas em amostras.
  - Validação de modelos.

## Probabilidade:

- Previsão de eventos futuros.
- Aplicações:
  - Algoritmos de aprendizado de máquina.
  - Classificação e tomada de decisões.

# Importância para Ciências de Dados e IA

**Distribuição normal:** para dados contínuos que variam em torno de uma média, como altura ou erro de previsão.

**Distribuição binomial:** para contar quantos acertos ou erros acontecem em várias tentativas, como prever se uma IA acerta ou erra.

**Distribuição de Poisson:** para contar quantas vezes algo raro acontece em certo tempo, como quantas falhas ocorrem por hora.

# Numpy

`np.mean()`, `np.median()`, `np.std()`, `np.var()` - **Estatística descritiva**, usada para explorar o desempenho de modelos e dados de entrada.

```
media = np.mean([10, 20, 30])
```

```
mediana = np.median([1, 2, 3, 4, 5])
```

```
variância = np.var([10, 12, 14])
```

```
desvio = np.std([10, 12, 14])
```

# Numpy

`np.dot()` - **Produto escalar de vetores ou multiplicação de matrizes**, usado em redes neurais e regressão linear

```
w = np.array([0.2, 0.5])
```

```
x = np.array([1, 2])
```

```
resultado = np.dot(w, x) #  $0.2 \cdot 1 + 0.5 \cdot 2 = 1.2$ 
```

# Numpy

`np.random` - **Geração de dados aleatórios**, ideal para criar datasets simulados para treinar modelos.

`np.random.rand()` – números aleatórios entre 0 e 1

`np.random.randint()` – inteiros aleatórios

`np.random.normal()` – dados com distribuição normal

```
# Geração de vetores aleatórios
```

```
x = np.random.rand(5)
```

```
y = np.random.normal(loc=0, scale=1, size=1000)
```



# Numpy

`np.where()` - Criação de decisões condicionais – semelhante a um if vetorial, muito útil para rótulos em classificação.

```
valores = np.array([0.7, 0.85, 0.95])
```

```
classificacao = np.where(valores > 0.9, 'ótimo', 'regular')
```

```
print(classificacao)
```

# Numpy

`np.argmax()` e `np.argmin()` - **Índice do maior ou menor valor**, útil para identificar a classe com maior probabilidade em modelos de classificação.

```
probs = np.array([0.1, 0.3, 0.6])
```

```
classe_preditada = np.argmax(probs) # retorna 2
```

# Numpy

`np.linspace()` e `np.arange()` - **Criação de sequências numéricas**, comuns em visualização, testes ou algoritmos genéticos.

```
valores = np.linspace(0, 1, 5) # [0. , 0.25, 0.5, 0.75, 1. ]
```

`np.sum()`, `np.max()`, `np.min()` - **Operações agregadas** essenciais para cálculo de perdas, métricas e análise de desempenho.

```
dados = np.array([[1, 2], [3, 4]])
```

```
print(np.sum(dados)) # soma total: 10
```

```
print(np.max(dados)) # maior valor: 4
```

# Scipy

Módulo	Função	Aplicação
<code>scipy.stats</code>	<code>norm</code> , <code>binom</code> , <code>poisson</code>	Distribuições probabilísticas
<code>scipy.stats</code>	<code>mode</code> , <code>ttest_ind</code>	Estatísticas e testes
<code>scipy.optimize</code>	<code>minimize</code>	Minimização de funções
<code>scipy.linalg</code>	<code>inv</code> , <code>eig</code> , <code>solve</code>	Álgebra linear
<code>scipy.integrate</code>	<code>quad</code> , <code>dblquad</code>	Integração numérica
<code>scipy.cluster</code>	<code>kmeans</code> , <code>vq</code>	Agrupamento de dados
<code>scipy.spatial</code>	<code>distance.euclidean</code>	Distância vetorial

# Scipy

`scipy.stats.mode()` – Moda - Determinar o valor mais frequente (usado para rótulos, predição mais comum).

```
from scipy import stats
```

```
valores = [1, 2, 2, 3, 3, 3, 4]
```

```
moda = stats.mode(valores)
```

```
print(moda.mode[0], moda.count[0])
```

# Distribuição Normal

`norm.cdf(x, loc=media, scale=desvio_padrao)`

x é Valor que você quer saber a probabilidade acumulada até ele. Ex: altura < x

```
from scipy.stats import norm
```

```
prob = norm.cdf(valor, loc=media, scale=desvio_padrao)
```

```
from scipy.stats import norm
```

```
media = 0.90
```

```
desvio = 0.02
```

```
# Qual a probabilidade de um modelo ter precisão menor que  
0.88?
```

```
prob = norm.cdf(0.88, loc=media, scale=desvio)
```

```
print(f"Probabilidade de precisão < 0.88: {prob:.4f}")
```

# Distribuição Binominal

n # número de transações  
p # probabilidade de acerto  
k # número de acertos desejados

prob\_binomial = binom.pmf(k, n, p)

```
from scipy.stats import binom
```

# Probabilidade de acertar exatamente 7 de 10 previsões com 80% de chance de acerto

```
prob = binom.pmf(k=7, n=10, p=0.8)
```

```
print(f"P(X = 7): {prob:.4f}")
```

# Distribuição de Poisson

mu    # taxa média de falhas  
k     # número de falhas que queremos

```
from scipy.stats import poisson
```

```
prob_poisson = poisson.pmf(k, mu)
```

# Qual a chance de ocorrerem exatamente 3 erros, sabendo que a média é 2 por hora?

```
prob = poisson.pmf(k=3, mu=2)
```

```
print(f"P(X = 3): {prob:.4f}")
```



# Regressão Linear

A regressão linear é um método de aprendizado supervisionado usado para prever valores contínuos. Ela modela a relação entre uma variável dependente (y) e uma ou mais variáveis independentes (x), assumindo uma relação linear entre elas.

$$y = \beta_0 + \beta_1 x + \varepsilon$$

$$\beta_0 = \bar{y} - \beta_1 \bar{x}$$

- y: variável dependente (o valor que queremos prever).
- x: variável independente (o valor usado para prever).
- $\beta_0, \beta_1$ : coeficientes a serem ajustados.
- $\beta_0$ : **intercepto** (bias), o valor de y quando x=0.
- $\beta_1$ : **coeficiente angular**, que representa a inclinação da linha e a variação de y para cada unidade de mudança em x.

$$\beta_1 = \frac{\sum((x - \bar{x})(y - \bar{y}))}{\sum((x - \bar{x})^2)}$$

**Objetivo:** Minimizar o erro (soma dos resíduos ao quadrado).

**Aplicações:** Previsão de vendas, crescimento populacional, etc.

## Vantagens e Desvantagens:

- Simplicidade e interpretabilidade.
- Limitação em problemas não lineares.

# Em Machine Learning

O modelo recebe um conjunto de dados com pares de entrada (x) e saída (y).

Ele tenta encontrar os coeficientes  $\beta_0$  (intercepto) e  $\beta_1$  (inclinação) que minimizam o **erro** entre os valores previstos e os valores reais.

$$\text{Erro Total} = \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

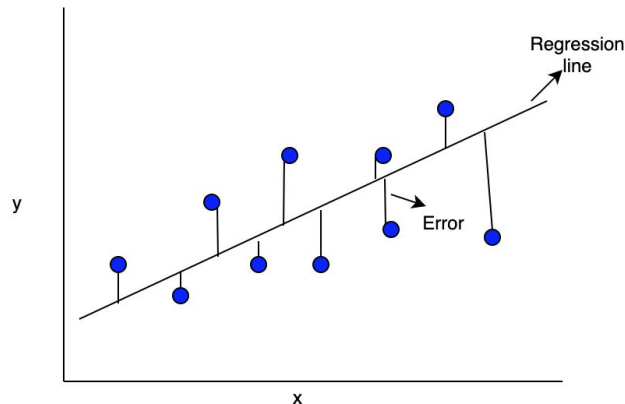
$y_i$  é o valor real,  
 $\hat{y}_i = \beta_0 + \beta_1 x_i$  é o valor previsto pelo modelo.

Após ajustar a equação (ou seja, encontrar  $\beta_0$  e  $\beta_1$ ), o modelo é testado com novos dados (xteste).

Ele usa a equação ajustada para prever  $y_{\text{teste}}$  e compara com os valores reais para verificar a precisão.

**Erro Quadrático Médio (MSE):**

$$MSE = \frac{1}{n} \sum_{i=1}^n (y_i - (\beta_0 + \beta_1 x_i))^2$$



# Exemplo

Prever o preço de uma casa com base na área (em m<sup>2</sup>).

Área (m <sup>2</sup> )	Preço (R\$)
50	150.000
60	180.000
70	200.000
80	230.000
90	250.000

# Exemplo

Prever o preço de uma casa com base na área (em m²).

Área (m²)	Preço (R\$)
-----------	-------------

50	150.000
----	---------

60	180.000
----	---------

70	200.000
----	---------

80	230.000
----	---------

90	250.000
----	---------

$$\bar{x} = \frac{\text{soma dos valores de área}}{\text{número de entradas}} = \frac{50 + 60 + 70 + 80 + 90}{5} = 70$$

$$\bar{y} = \frac{\text{soma dos valores de preço}}{\text{número de entradas}} = \frac{150000 + 180000 + 200000 + 230000 + 250000}{5} = 202000$$

$$\beta_1 = \frac{(50 - 70)(150000 - 202000) + \dots}{(50 - 70)^2 + \dots} = 4000$$

$$\beta_0 = 202000 - (4000)(70) = -80000$$

$$y = -80000 + 4000x$$

Prever o preço para x=75

$$y = -80000 + 4000(75) = 220000$$

# Regressão Logística

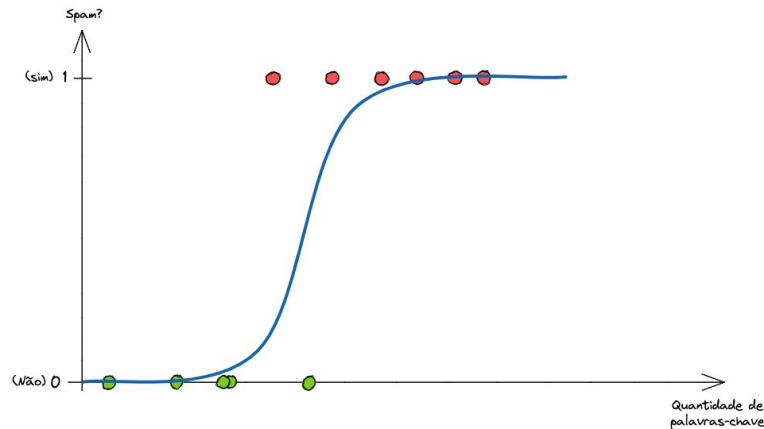
A regressão logística é usada para problemas de classificação, onde a saída é categórica (ex.: sim/não). Diferentemente da regressão linear, que gera valores contínuos, a regressão logística usa a função sigmóide para limitar os valores preditos entre 0 e 1, representando a probabilidade de pertencer a uma classe.

**Fórmula:**

$$P(y = 1|x) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x)}}$$

**$P(y=1|x)$ : probabilidade da classe positiva.**

**Se a probabilidade for maior que 0.5, y é classificado como 1; caso contrário, como 0.**



# Regressão Logística

Na regressão logística, **os parâmetros** ( $\beta_0$  e  $\beta_1$ ) **são encontrados usando um método numérico de otimização**, chamado **Máxima Verossimilhança**.

Em regressão logística: você **ajusta** os parâmetros buscando maximizar a chance dos dados observados acontecerem (**não dá pra calcular direto por fórmula básica**).

Esse processo é feito com **algoritmos iterativos**, como o **Gradiente Descendente**. Esses algoritmos funcionam assim:

1. Começam com valores aleatórios ou nulos para  $\beta_0$  e  $\beta_1$ ;
2. Calculam o quanto esses valores erram nas previsões (usando uma função chamada *log-verossimilhança*);
3. Ajustam os valores de  $\beta_0$  e  $\beta_1$  **um pouquinho por vez**, sempre tentando **reduzir o erro**;
4. Repetem isso centenas ou milhares de vezes, **até encontrar os melhores valores possíveis**.

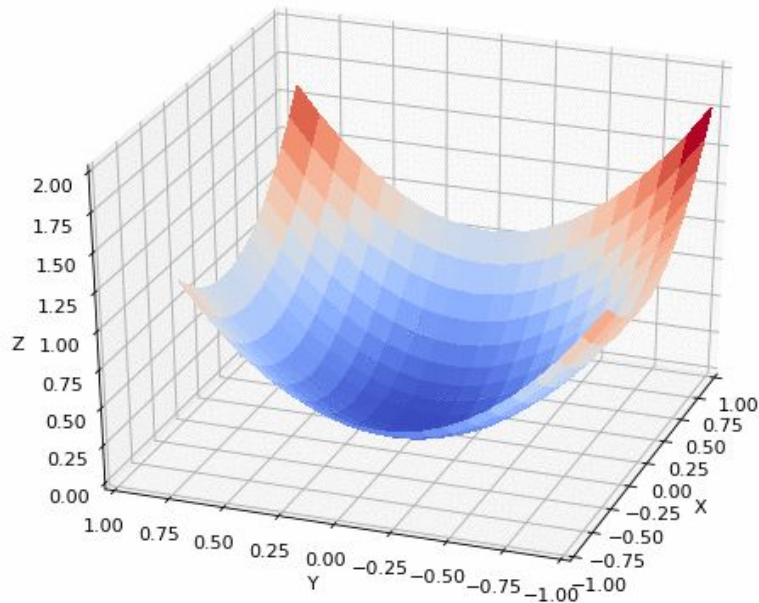
# Gradiente Descendente

A função de erro (ou função de custo) usada na regressão logística é a **log-verossimilhança**:

$$J(\beta) = - \sum_{i=1}^n [y_i \log(p_i) + (1 - y_i) \log(1 - p_i)]$$

Onde  $p_i = \frac{1}{1+e^{-(\beta_0+\beta_1 x_i)}}$  é a probabilidade prevista.

O gradiente descendente testa diferentes combinações de  $\beta_0$  e  $\beta_1$ , calcula o erro para cada uma delas, e vai ajustando os valores até encontrar o ponto onde o erro é o menor possível.



# Exemplo

Classificar  
pacientes como  
"doente" ou  
"saudável" com  
base na idade.

**Idade**

**Diagnóstico (0 = saudável, 1 =  
doente)**

25

0

30

0

35

1

40

1

45

1



Idade

Diagnóstico (0 = saudável, 1 = doente)

## Exemplo

Classificar  
pacientes como  
"doente" ou  
"saudável" com  
base na idade.

25	0
30	0
35	1
40	1
45	1

Previsão para idade  $x=28$

$$P(y = 1|x = 28) = \frac{1}{1 + e^{-(-10+0.25(28))}}$$

$$\beta_0 = -10, \beta_1 = 0.25$$

$$P(y = 1|x = 28) = \frac{1}{1 + e^{-3}} \approx 0.952$$

Se  $P > 0.5$ ,  $y = 1$  (doente). Caso contrário,  $y = 0$  (saudável).

Para  $x = 28$ , o paciente é classificado como **doente**.

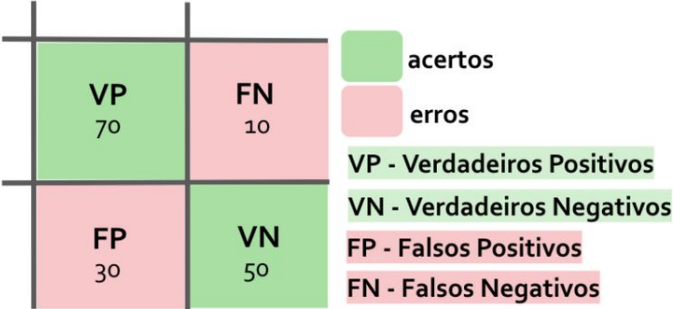
# Árvore de Decisão

Algoritmo	Tipo	Critério de Divisão	Pontos-Chave
ID3	Classificação	Ganho de Informação (Entropia)	Simples, sem poda, não lida com valores contínuos. Base teórica para outros.
C4.5	Classificação	Razão de Ganho	Evolução do ID3. Suporta valores contínuos e faltantes. Faz poda.
CART	Classificação e Regressão	Gini (classificação), MSE (regressão)	Usa apenas divisões binárias. Muito eficiente. Base do Scikit-learn.
CHAID	Classificação	Teste Qui-quadrado	Usa estatística para ramificação múltipla. Comum em pesquisas e análises sociais.

# Matriz Confusão

## CLASSIFICAÇÃO DO MODELO

REAL



Situação real	Previsão do modelo	Res.	Nome técnico
Pessoa <b>tem</b> a doença	Modelo diz que <b>tem</b>	A	<b>Verdadeiro Positivo (TP)</b>
Pessoa <b>não tem</b> a doença	Modelo diz que <b>não tem</b>	A	<b>Verdadeiro Negativo (TN)</b>
Pessoa <b>não tem</b> a doença	Modelo diz que <b>tem</b>	E	<b>Falso Positivo (FP)</b>
Pessoa <b>tem</b> a doença	Modelo diz que <b>não tem</b>	E	<b>Falso Negativo (FN)</b>

# Métricas

## Precision (Precisão):

- Mede **quantas das previsões positivas estavam corretas**.

Alta precisão → poucos falsos positivos.

## Recall (Revocação ou Sensibilidade):

- Mede **quantas das amostras positivas foram detectadas**.

Alto recall → poucos falsos negativos.

## F1-Score:

- Média harmônica entre *precision* e *recall*.
- Equilibra os dois indicadores.

Ideal quando há **dados desbalanceados**.

## Accuracy (Acurácia):

- Percentual de acertos totais (todas as classes).

Pode ser **enganosa em bases desbalanceadas**.

$$\text{Precision} = \frac{\text{Verdadeiros Positivos}}{\text{Verdadeiros Positivos} + \text{Falsos Positivos}}$$

$$\text{Recall} = \frac{\text{Verdadeiros Positivos}}{\text{Verdadeiros Positivos} + \text{Falsos Negativos}}$$

$$F1 = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

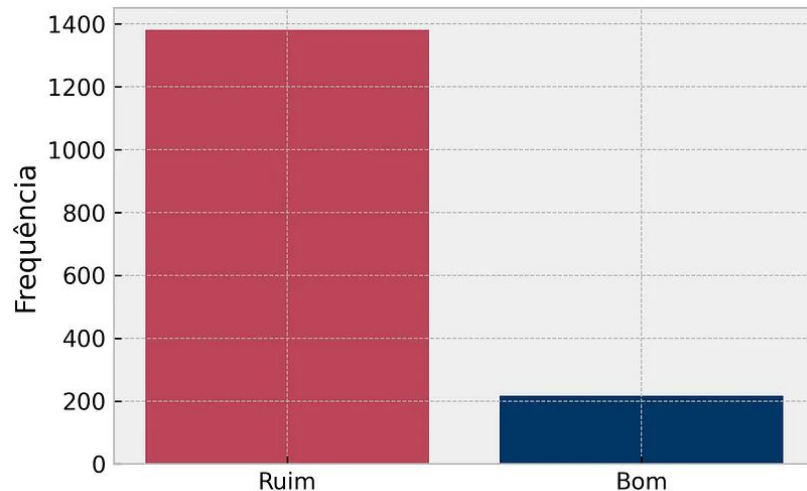
$$\text{Accuracy} = \frac{\text{Acertos Totais}}{\text{Total de Amostras}}$$

# Balanceamento dos Dados

O modelo aprende a **favorecer a classe majoritária**, resultando em alta acurácia, mas **baixa detecção da classe rara**.

## Problemas comuns:

- Falsos negativos altos (ignora eventos importantes).
- Acurácia enganosa.
- Métricas tradicionais (accuracy) se tornam pouco informativas.



# Balanceamento dos Dados

## SMOTE — Synthetic Minority Over-sampling Technique:

- Cria **novas amostras sintéticas** da classe minoritária, em vez de apenas duplicar exemplos existentes.
- Gera pontos *intermediários* entre exemplos reais próximos.

## `class_weight='balanced'` (do Scikit-learn)

- **Não altera os dados.**
- Ajusta **os pesos de cada classe** dentro do cálculo do erro.
- O modelo **pune mais** os erros da classe minoritária.

