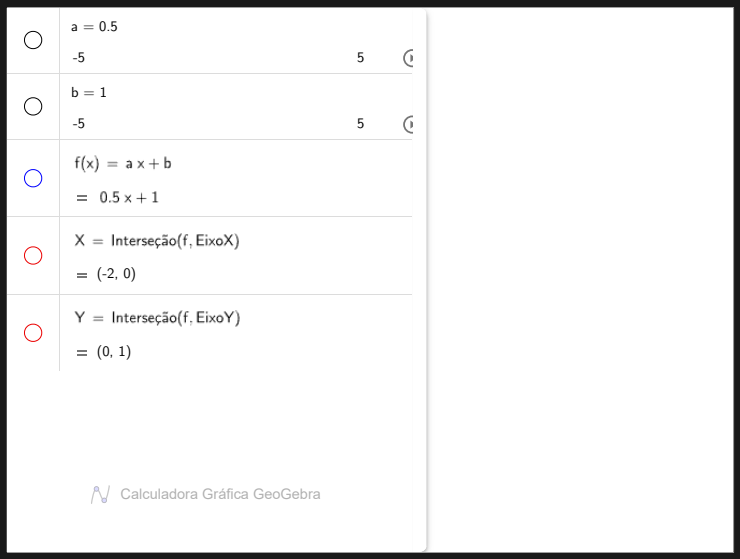
Aula 7 - Fundamentos Matemáticos da Regressão Linear

A Mecânica da Regressão

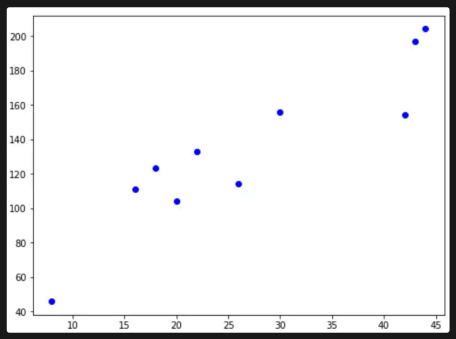
- Queremos ajustar uma linha aos dados, na forma: $y = ax + b$
- Isso é chamado de **regressão linear simples (de uma só feature)**:
  - `x` : feature (variável explicativa)
  - `y` : alvo (variável a ser prevista)
  - `a` : coeficiente angular (inclinação)
  - `b` : intercepto (ponto em que a linha cruza o eixo y)
- `a` e `b` são **os parâmetros do modelo** que queremos aprender.

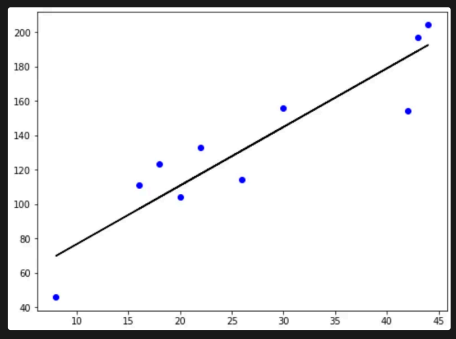


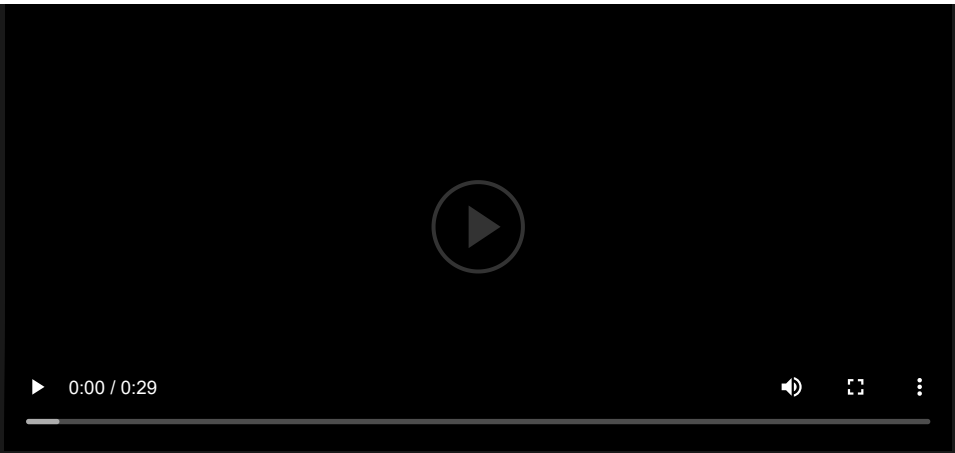
- Para isso, usamos uma **função de erro**, também chamada de **função de perda** ou **função de custo**.

Função de Perda – Conceito Visual

- Queremos que a linha fique **o mais próxima possível das observações**.
- Ou seja, minimizar a **distância vertical** entre a linha ajustada e os pontos de dados.

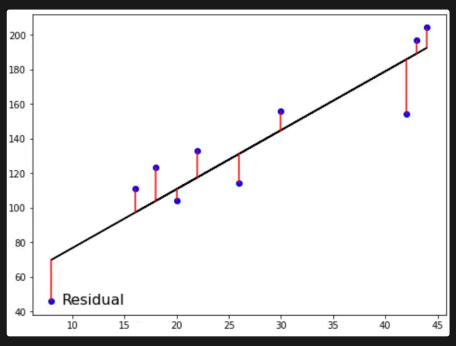


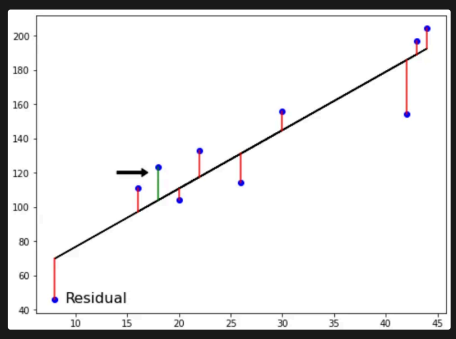




Função de Perda – Cálculo da Distância

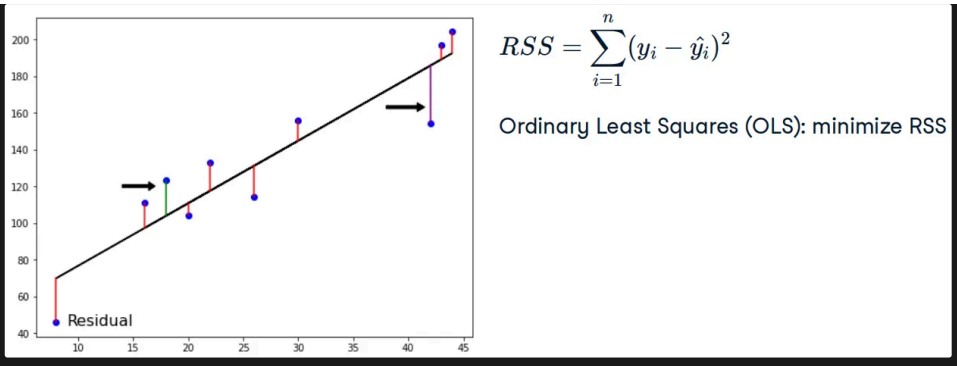
- Para cada observação, calculamos a **distância vertical até a linha**.
- Essa distância é chamada de **resíduo**.
- Se apenas somarmos os resíduos, os positivos e negativos podem se anular.





Mínimos Quadrados Ordinários (OLS)

- Para evitar o cancelamento dos resíduos, **elevamos ao quadrado cada resíduo**.
- A soma dos quadrados dos resíduos é chamada de **RSS** (Residual Sum of Squares).
- O método de regressão que **minimiza o RSS** é chamado de **Mínimos Quadrados Ordinários (OLS)**.



Regressão Linear com Múltiplas Variáveis

- Com duas features (x_1, x_2), a equação se torna: $y = a_1x_1 + a_2x_2 + b$
  - $y = a_1x_1 + a_2x_2 + by = a_1x_1 + a_2x_2 + b$
- Para `n` features, usamos **regressão linear múltipla**:
  - Coeficientes: `a₁, a₂, ..., aₙ`
  - Intercepto: `b`
  - $y = a_1x_1 + a_2x_2 + a_3x_3 + \ldots + a_nx_n + b$
- No `scikit-learn`, os dados devem estar organizados com uma variável para as **features (X)** e uma para o **alvo (y)**.

Regressão Linear com Todas as Features

- Vamos prever **níveis de glicose no sangue** usando todas as features do dataset de diabetes.
- Passos:
  - Importar `LinearRegression` do `sklearn.linear_model`
  - Dividir os dados em treino e teste
  - Instanciar o modelo
  - Treinar com `.fit()`
  - Prever com `.predict()`
- Obs.: O `LinearRegression` do scikit-learn **usa OLS internamente**.
- Lista

```
from sklearn.model_selection import train_test_split from sklearn.linear_model import
LinearRegression X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42) reg_all = LinearRegression() reg_all.fit(X_train, y_train) y_pred =
reg_all.predict(X_test)
```

`test_size=0.3`

- Isso significa que **30% dos dados vão ser usados para teste**, e o restante (70%) para treino.
  - A ideia é **avaliar o modelo em dados que ele nunca viu**.
  - Exemplo: Se você tem 100 amostras, 70 vão para o treino e 30 para o teste.

`random_state=42`

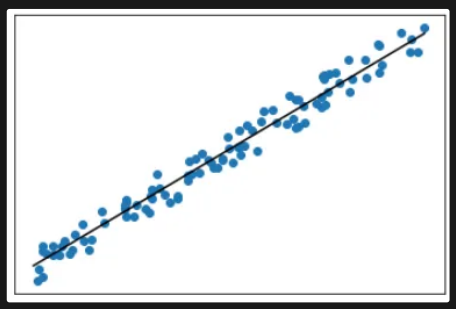
- Isso define **a semente do gerador de números aleatórios** usado para embaralhar e dividir os dados.
  - Usar um valor fixo como `42` garante **reprodutibilidade**: toda vez que rodar o código, a divisão treino/teste será **idêntica**.
  - Por que `42` ? É uma piada recorrente na programação (referência ao livro *O Guia do Mochileiro das Galáxias*, onde 42 é "a resposta para tudo"), mas **você pode usar qualquer número inteiro**.

💡 Se tirar o `random_state`, cada execução do código pode fazer uma divisão diferente dos dados — o que atrapalha a comparação de resultados entre modelos.

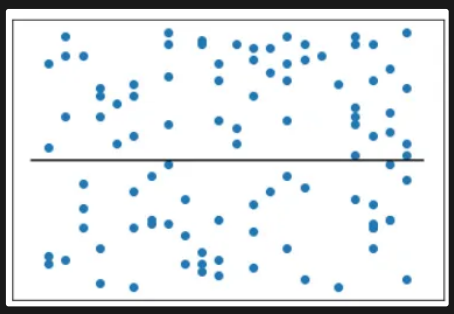
Coeficiente de Determinação (R^2)

- Métrica padrão para regressão linear.
- Indica quanto da **variância da variável-alvo** é explicada pelas features.
- Varia de **0 a 1**:
  - `R² = 1` → previsão perfeita
  - `R² = 0` → modelo não explica nada
- Exemplo: dois gráficos comparando **R^2 alto** e **R^2 baixo**.

R^2 alto



R^2 baixo



R^2 no scikit-learn

- Usamos o método `.score(X_test, y_test)` do modelo.
- Exemplo: um modelo que explica **apenas 35%** da variância da glicose.

```
reg_all.score(X_test, y_test) // 0.35632876407827
```

Erro Quadrático Médio (MSE) e RMSE

- Outra métrica: **MSE (Mean Squared Error)**
  - Média dos quadrados dos resíduos
  - Mesma unidade do alvo, mas elevada ao quadrado

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

- Para obter erro na mesma escala, usamos a **raiz quadrada** do MSE → **RMSE**

$$RMSE = \sqrt{MSE}$$

RMSE no scikit-learn

- Importamos `mean_squared_error` de `sklearn.metrics`
- Chamamos `mean_squared_error(y_test, y_pred, squared=False)`
- Resultado: **erro médio de cerca de 24 mg/dL** nos níveis de glicose

```
from sklearn.metrics import mean_squared_error import numpy as np mse =
mean_squared_error(y_test, y_pred) rmse = np.sqrt(mse) print(f'mse: {mse}, rmse: {rmse}')
```

Vamos Praticar!

- Agora é sua vez de **construir e avaliar um modelo de regressão linear múltipla!**

Exercício

Ajuste e Previsão com Regressão Linear

Agora que você já viu como a **regressão linear** funciona, sua tarefa é criar um modelo de **regressão linear múltipla** usando **todas as features** do dataset `sales_df`, que já foi **pré-carregado** para você.

Prévia do Dataset (`sales_df`)

tv	radio	social_media	sales
13000.0	9237.76	2409.57	46677.90
41000.0	15886.45	2913.41	150177.83

Objetivo

Você irá:

- Treinar um modelo com os dados de treino
- Fazer previsões de **vendas** com os dados de teste

Instruções

- Crie `X`, um array contendo os valores de **todas as features** do `sales_df`, e `y`, contendo todos os valores da coluna "**sales**".
- Instancie um modelo de **regressão linear**.
- Ajuste (treine) o modelo usando os **dados de treinamento**.
- Crie `y_pred`, fazendo **previsões de vendas** com os dados de **teste**.

```
# Create X and y arrays X = sales_df.____("____", axis=____).____ y =
sales_df["____"].____ X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=42) # Instantiate the model reg = ____ # Fit the model to the
data ____ # Make predictions y_pred = reg.____(____) print("Predictions: {}, Actual
Values: {}".format(y_pred[:2], y_test[:2]))
```

Exercício

Desempenho da Regressão

Agora que você ajustou um modelo (`reg`) usando todas as features do `sales_df` e fez previsões dos valores de vendas, é possível avaliar o desempenho utilizando algumas **métricas comuns de regressão**.

As variáveis `X_train`, `X_test`, `y_train`, `y_test` e `y_pred`, além do modelo ajustado `reg`, foram todas **pré-carregadas** do exercício anterior.

Seu objetivo é descobrir:

- **O quanto as features explicam a variância** dos valores-alvo (das vendas) (R^2); e
- **Avaliar a capacidade do modelo de fazer previsões em dados não vistos** (`RMSE`).

Instruções

✅ Importe a função `mean_squared_error`.

✅ Calcule o **R-quadrado (R^2)** do modelo, passando os dados de teste (features e alvo) para o método apropriado.

✅ Calcule o **erro quadrático médio raiz (RMSE)** utilizando `y_test` e `y_pred`.

✅ Imprima os valores de `r_squared` e `rmse`.

```
# Import mean_squared_error from ____.____ import ____ # Compute R-squared r_squared =
reg.____(____, ____) # Compute RMSE rmse = ____(____, ____, squared=____) # Print the
metrics print("R^2: {}".format(____)) print("RMSE: {}".format(____))
```