Explorando Padrões:
Regressão Linear para Predição de Receita

Objetivos da aula

- Introdução à regressão linear para prever receita de filmes.
- Nesta aula vamos criar um modelo simples de regressão linear para estimar a receita de filmes com base no orçamento. Ao final da aula você será capaz de:
  - Preparar dados para regressão linear
  - Usar **LinearRegression** do scikit-learn
  - Separar dados em treino e teste Aplicar fit() e predict()
  - Interpretar resultados de predição
  - Implementar TDD para validar previsões

Instalando scikit-learn

- **scikit-learn** é a biblioteca padrão para machine learning em Python, incluindo regressão linear: https://scikit-learn.org/stable/

```
# Criar e ativar um ambiente virtual
python -m venv venv
# No Windows
venv\Scripts\activate
# No Mac/Linux
source venv/bin/activate

# Instalação das bibliotecas necessárias
pip install scikit-learn
```

Instalando scikit-learn

- Criamos um ambiente virtual para isolar dependências do projeto.
- Instalamos scikit-learn dentro desse ambiente para treinar modelos de regressão linear e manipular dados de filmes, garantindo que o sistema global do Python permaneça intacto.

Importando bibliotecas

- Importamos NumPy para manipulação de arrays e scikit-learn para regressão e separação de dados.

```
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from filme import Filme
```

Criando os filmes

- Lista de filmes usada como base de dados para treinar o modelo.

```
filmes =  [
    Filme("Matrix", 63000000, 465000000, "Ficção"),
    Filme("Titanic", 200000000, 2200000000, "Drama"),
    Filme("Avatar", 237000000, 2847000000, "Ficção"),
    Filme("Joker", 55000000, 1074000000, "Drama"),
    Filme("Interestelar", 165000000, 677000000, "Ficção")
]
```

Extraindo orçamentos e receitas

- Transformamos os dados em arrays NumPy;
- O método **reshape(-1, 1)** transforma o array unidimensional de orçamentos em uma matriz coluna, que é o formato esperado pelo scikit-learn para as features (X).
- Sem isso, o modelo não consegue interpretar corretamente os dados de entrada.

```
orcamentos = np.array([f.orcamento for f in filmes]).reshape(-1, 1)
receitas = np.array([f.receita for f in filmes])
```

Conceito de regressão linear

- A regressão linear estima a relação entre uma variável dependente (receita) e uma independente (orçamento) como uma linha reta.

$$y = a*x + b$$

- O coeficiente a indica quanto a receita muda em média para cada aumento unitário no orçamento, enquanto b representa a receita esperada quando o orçamento é zero.

Separando dados em treino e teste

- Dividimos os dados em 60% treino e 40% teste para avaliar o modelo em dados que ele nunca viu.
- Os dados de treino são usados para ajustar os parâmetros da linha (a e b), enquanto os dados de teste permitem verificar se o modelo generaliza bem e consegue prever receitas de filmes diferentes dos utilizados no treino, evitando **overfitting**.

```
X_train, X_test, y_train, y_test = train_test_split(orcamentos, receitas, test_size=0.4, random_state=42)
```

Separando dados em treino e teste

- O parâmetro **random_state** define uma semente aleatória para a divisão dos dados.
- Isso garante que a separação entre treino e teste seja reproduzível, permitindo que resultados e testes sejam consistentes entre diferentes execuções do código.

```
X_train, X_test, y_train, y_test = train_test_split(orcamentos, receitas, test_size=0.4, random_state=42)
```

Separando dados em treino e teste

- Em outras palavras:
  - O **treino** ensina o modelo.
  - O **teste** verifica se o modelo aprendeu corretamente e não apenas decorou os dados.

- No nosso caso, como temos 5 filmes e test_size=0.4, 2 filmes vão para teste e 3 para treino.

Criando o modelo

- Inicializamos o modelo de regressão linear.
  - modelo = LinearRegression()

- Esse objeto modelo será treinado com os dados de orçamento e receita, aprendendo os coeficientes da linha (a e b) que melhor ajustam os dados observados.

Treinando o modelo

- O método fit() ajusta a linha que melhor representa a relação entre orçamento e receita nos dados de treino.
  - modelo.fit(X_train, y_train)

```
modelo = LinearRegression()
modelo.fit(X_train, y_train)
```

Treinando o modelo

- Em convenção de machine learning, X (maiúsculo) representa uma matriz de características (features) com várias colunas e linhas, enquanto y (minúsculo) representa um vetor de respostas (target) com os valores que queremos prever.

- Aqui, **X_train** contém os orçamentos dos filmes e **y_train** contém as receitas correspondentes.

Verificando coeficiente e intercepto

- **coef_** indica a taxa de variação da receita em relação ao orçamento (quanto a receita aumenta, em média, para cada unidade a mais de orçamento), enquanto **intercept_** é o valor esperado da receita quando o orçamento é zero, ou seja, o ponto em que a linha cruza o eixo Y.

```
print("Coeficiente (a):", modelo.coef_[0])
print("Intercepto (b):", modelo.intercept_)
```

Predizendo receita de teste

- O método **predict()** utiliza os coeficientes (**coef_** e **intercept_**) aprendidos durante o treino para estimar a receita de novos filmes com base em seus orçamentos.
- Ele gera previsões para cada valor de **X_test**, permitindo comparar os resultados previstos com os valores reais de **y_test**.

```
y_pred = modelo.predict(X_test)
print(y_pred)
```

Comparando valores reais e previstos

- O for abaixo mostra lado a lado os valores reais e os estimados pelo modelo.
- Comparar os valores reais e estimados permite avaliar a precisão do modelo, identificar se ele está subestimando ou superestimando a receita e verificar se o modelo generaliza bem para dados que não foram usados no treino.

```
for real, previsto in zip(y_test, y_pred):
    print(f"Real: {real}, Previsto: {previsto:.0f}")
```

Rodando o código

```python
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from filme import Filme

filmes =  [
    Filme("Matrix", 63000000, 465000000, "Ficção"),
    Filme("Titanic", 200000000, 2200000000, "Drama"),
    Filme("Avatar", 237000000, 2847000000, "Ficção"),
    Filme("Joker", 55000000, 1074000000, "Drama"),
    Filme("Interestelar", 165000000, 677000000, "Ficção")
]

orcamentos = np.array([f.orcamento for f in filmes]).reshape(-1, 1)
receitas = np.array([f.receita for f in filmes])

X_train, X_test, y_train, y_test = train_test_split(orcamentos, receitas, test_size=0.4, random_state=42)

modelo = LinearRegression()
modelo.fit(X_train, y_train)

print("Coeficiente (a):", modelo.coef_[0])
print("Intercepto (b):", modelo.intercept_)

y_pred = modelo.predict(X_test)
print(y_pred)

for real, previsto in zip(y_test, y_pred):
    print(f"Real: {real}, Previsto: {previsto:.0f}")
```

Rodando o código

- **Coeficiente**: Cada 1 milhão de dólares de orçamento aumenta a receita em ~11,5 milhões.
- **Intercepto**: Receita estimada com orçamento zero: ~96,6 milhões. Valores previstos: 2.404M e 2.000M, comparados aos reais: 2.200M e 677M.
- O modelo aproxima os valores, mas superestima o filme com menor orçamento, mostrando limitação de um modelo linear simples.

```
(3.9.9) (base) joaopauloaramuni@MacBook-Pro-de-Joao testes % python main.py
Coeficiente (a): 11.538541535358624
Intercepto (b): 96605918.31589627
[2.40431423e+09 2.00046527e+09]
Real: 2200000000, Previsto: 2404314225
Real: 677000000, Previsto: 2000465272
(3.9.9) (base) joaopauloaramuni@MacBook-Pro-de-Joao testes %
```

Rodando o código

- O coeficiente indica que, em média, cada dólar adicional de orçamento aumenta a receita em ~11,5 vezes.
- Os valores previstos (y_pred) representam a estimativa do modelo para os filmes de teste.
- Comparando real x previsto, vemos que o modelo aproxima os valores mas pode superestimar ou subestimar, especialmente com poucos dados.

```
(3.9.9) (base) joaopauloaramuni@MacBook-Pro-de-Joao testes % python main.py
Coeficiente (a): 11.538541535358624
Intercepto (b): 96605918.31589627
[2.40431423e+09 2.00046527e+09]
Real: 2200000000, Previsto: 2404314225
Real: 677000000, Previsto: 2000465272
(3.9.9) (base) joaopauloaramuni@MacBook-Pro-de-Joao testes %
```

Teste TDD: coeficiente positivo

- Testa se a relação entre orçamento e receita é positiva, como esperado.
- Este teste verifica se o modelo capturou corretamente a tendência esperada de que filmes com maior orçamento tendem a gerar maior receita.
- Garantir que a relação seja positiva valida a coerência do modelo com o comportamento real dos dados e evita previsões inconsistentes.

```python
def test_coeficiente_positivo():
    modelo = LinearRegression().fit(orcamentos, receitas)
    assert modelo.coef_[0] > 0
```

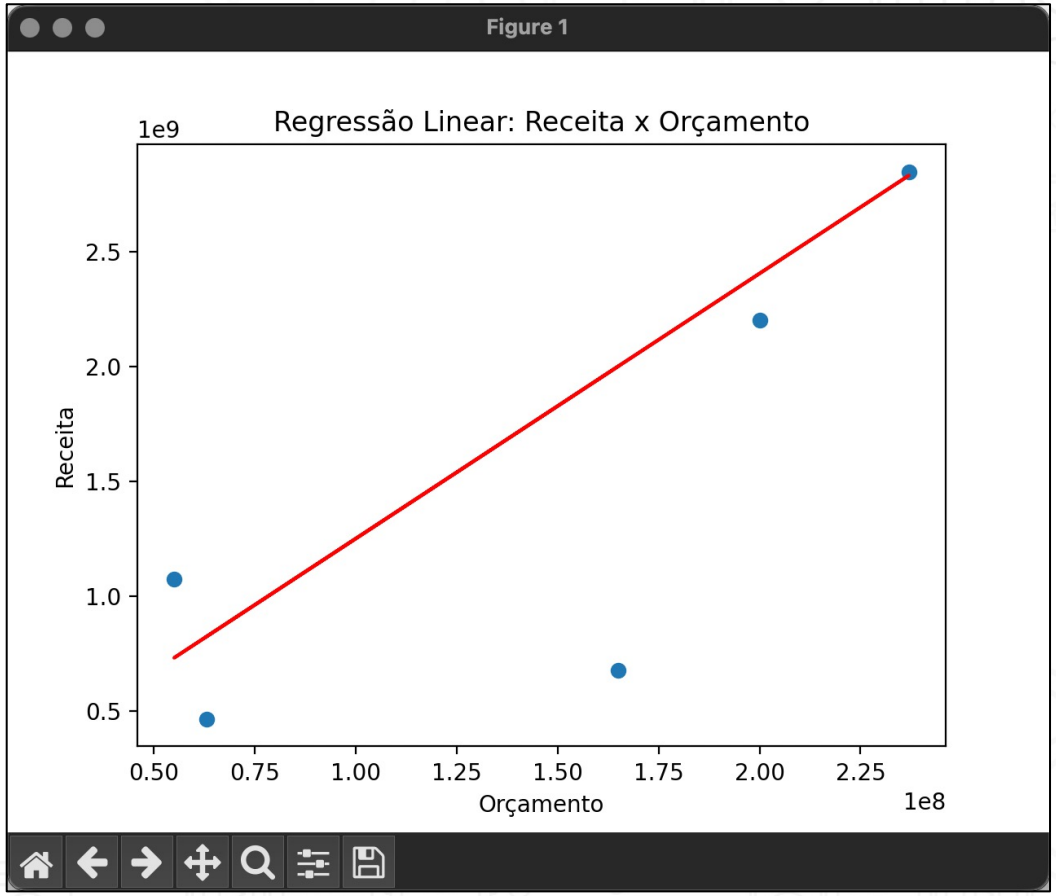
Visualizando a regressão

- Mostra os pontos reais e a linha de regressão ajustada.

```
import matplotlib.pyplot as plt
plt.scatter(orcamentos, receitas)
plt.plot(orcamentos, modelo.predict(orcamentos), color="red")
plt.xlabel("Orçamento")
plt.ylabel("Receita")
plt.title("Regressão Linear: Receita x Orçamento")
plt.show()
```

Visualizando a regressão

- Inclinação positiva indica que filmes com maior orçamento tendem a maior receita.



Ajuste para 70% treino

- Ajusta proporção de treino/teste; mais treino pode melhorar o modelo.
- A proporção entre treino e teste define quantos dados o modelo usa para aprender e quantos para avaliar.
- Aumentar a quantidade de dados de treino pode melhorar a precisão do modelo, enquanto o conjunto de teste garante que a generalização seja corretamente avaliada.

```
X_train, X_test, y_train, y_test = train_test_split(orcamentos, receitas, test_size=0.3, random_state=42)
```

Rodando o código

- Diferença nos resultados: test_size 0.3 vs 0.4
  - Com test_size=0.3 (70% treino / 30% teste), o modelo teve mais dados para aprender e os valores previstos ficaram mais próximos dos reais.
  - Com test_size=0.4 (60% treino / 40% teste), o modelo teve menos dados de treino, o que aumentou o erro nas previsões.

```
(3.9.9) (base) joaopauloaramuni@MacBook-Pro-de-Joao testes % python main.py
Coeficiente (a): 11.538541535358624
Intercepto (b): 96605918.31589627
[2.40431423e+09 2.00046527e+09]
Real: 2200000000, Previsto: 2404314225
Real: 677000000, Previsto: 2000465272
(3.9.9) (base) joaopauloaramuni@MacBook-Pro-de-Joao testes %
```

Rodando o código

- O aumento do conjunto de teste reduziu a precisão das previsões, mostrando como a quantidade de dados de treino influencia diretamente o desempenho do modelo.
- Isso evidencia que modelos preditivos dependem fortemente da quantidade e representatividade dos dados de treino: menos treino pode levar a estimativas mais imprecisas.

```
(3.9.9) (base) joaopauloaramuni@MacBook-Pro-de-Joao testes % python main.py
Coeficiente (a): 11.538541535358624
Intercepto (b): 96605918.31589627
[2.40431423e+09 2.00046527e+09]
Real: 2200000000, Previsto: 2404314225
Real: 677000000, Previsto: 2000465272
(3.9.9) (base) joaopauloaramuni@MacBook-Pro-de-Joao testes %
```

Avaliando modelo com R^2

- O R^2 indica quão bem o modelo explica a variação dos dados reais.
- **score()** retorna o coeficiente de determinação, indicando qualidade do ajuste.
  - R^2: -0.5462661841775076

```
r2 = modelo.score(X_test, y_test)
print("R²:", r2)
```

Interpretando R^2

- R^2 próximo de 1 indica bom ajuste: quanto mais próximo de 1, melhor o modelo explica a variabilidade dos dados.
- No nosso caso, $R^2 \approx -0,55$ significa que o modelo não se ajustou bem: ele é pior que simplesmente usar a média das receitas como previsão.
- Ou seja, o modelo falha em explicar os dados com a quantidade de treino e os filmes usados.

```
r2 = modelo.score(X_test, y_test)
print("R²:", r2)
```

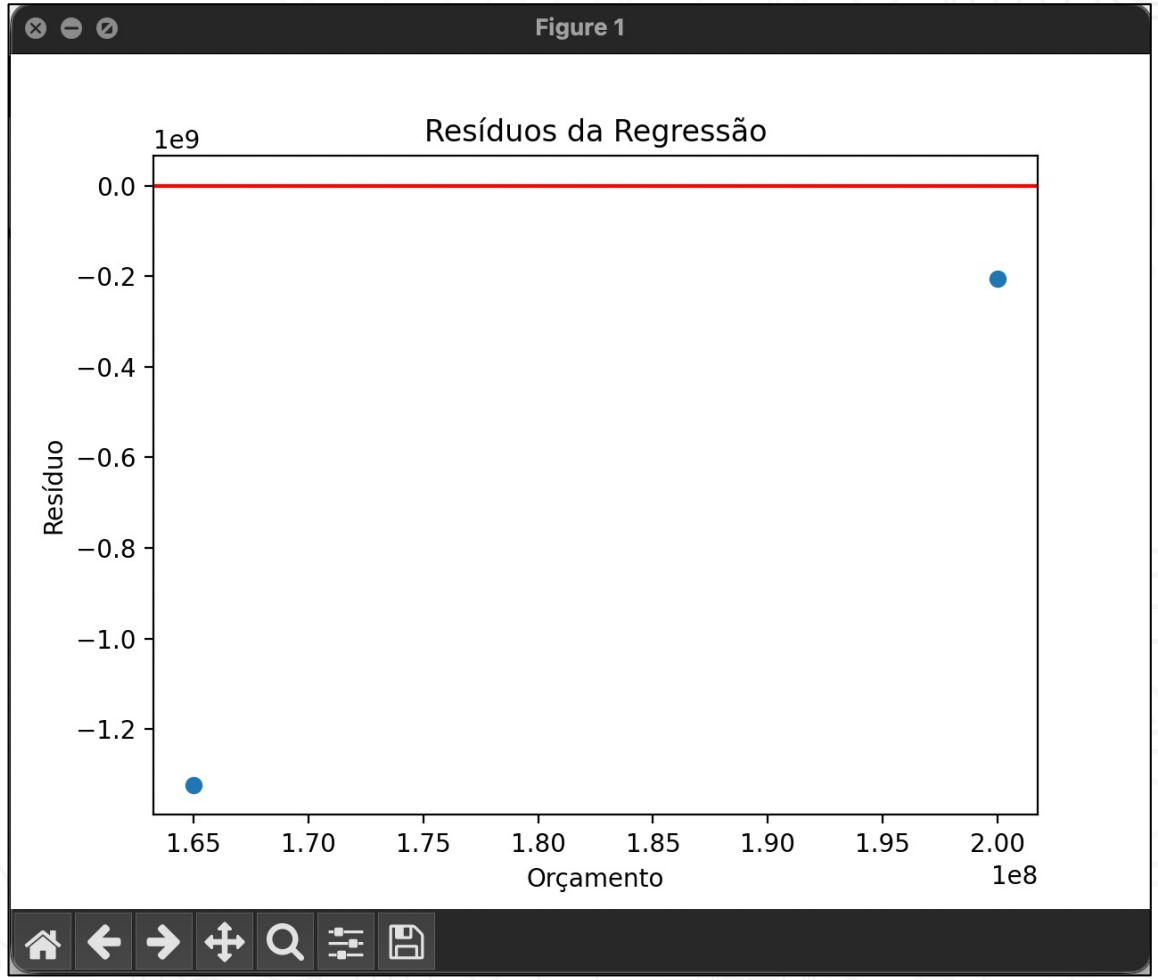
Plotando resíduos

- Visualizando erro de predição; resíduos distribuídos aleatoriamente indicam bom modelo.

```
residuos = y_test - y_pred
plt.scatter(X_test, residuos)
plt.axhline(0, color="red")
plt.xlabel("Orçamento")
plt.ylabel("Resíduo")
plt.title("Resíduos da Regressão")
plt.show()
```

Plotando resíduos

- Considerando test_size=0.3
- Resultado:
  - [-2.04314225e+08 -1.32346527e+09]



Plotando resíduos

- Esses valores são os resíduos, ou seja, a diferença entre os valores reais e os valores previstos pelo modelo:

$$resíduo = real - previsto$$

- No nosso caso:
  - -2,04e+08 → o modelo superestimou a receita do primeiro filme em ~204 milhões.
  - -1,32e+09 → o modelo superestimou a receita do segundo filme em ~1,32 bilhão.

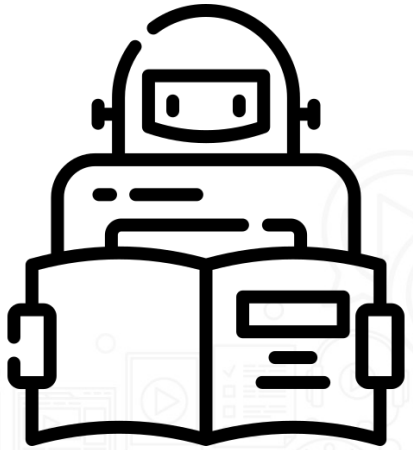
- Ou seja, os valores previstos estão acima dos reais, indicando que o modelo não se ajustou bem aos dados, reforçando o baixo R^2.

Impacto da Qtd de Dados no Modelo

- Quanto maior a quantidade de filmes utilizados para treinar o modelo, mais ele consegue aprender padrões reais da relação entre orçamento e receita.
- Com mais dados de treino:
  - As previsões ficam mais próximas dos valores reais.
  - O coeficiente de determinação (R^2) tende a aumentar.
  - O modelo se torna mais robusto e confiável para novos filmes.

Impacto da Qtd de Dados no Modelo

- Em outras palavras:
  - Mais dados de treino permitem que o modelo capture melhor a relação orçamento x receita, melhorando precisão e confiabilidade das previsões.



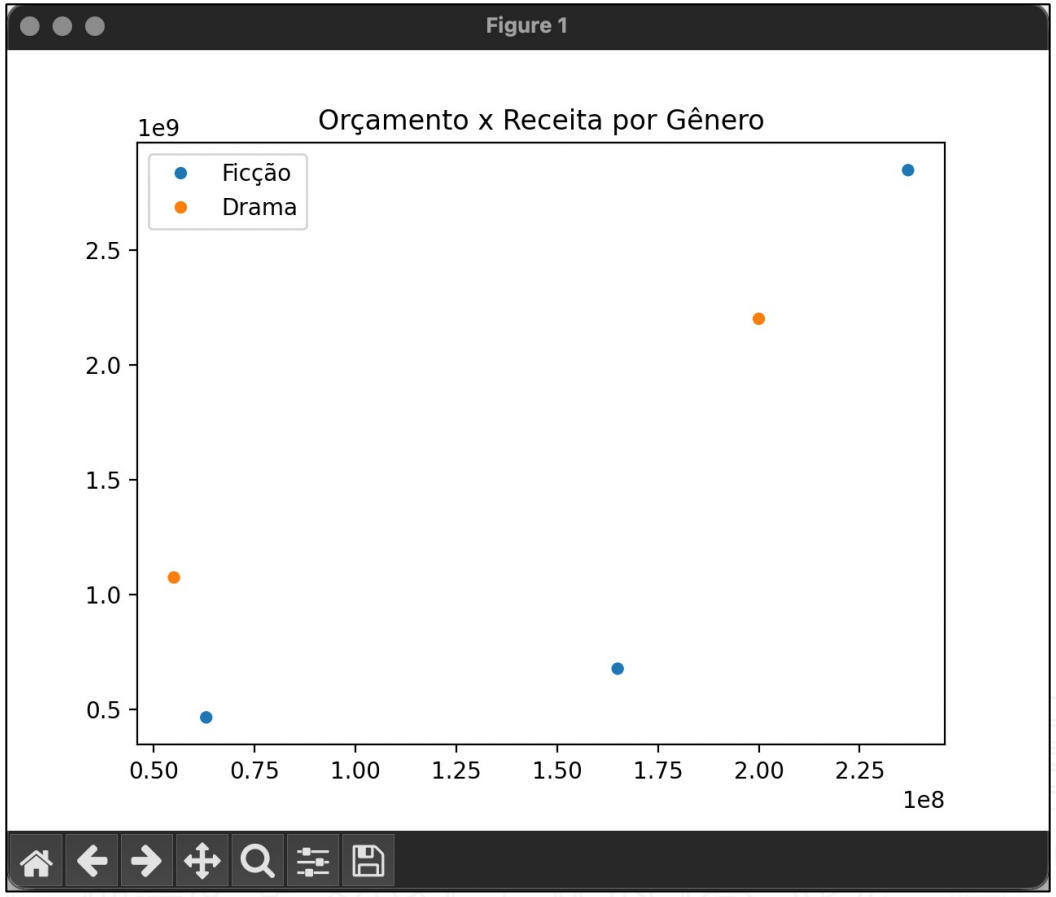
Scatter com cores por gênero

- Visualiza como a receita se distribui entre gêneros diferentes.

```
import seaborn as sns
generos  =  [f.genero for f in filmes]
sns.scatterplot(x=orcamentos.flatten(), y=receitas, hue=generos)
plt.title("Orçamento x Receita por Gênero")
plt.show()
```

Scatter com cores por gênero

- Visualiza como a receita se distribui entre gêneros diferentes.



Criando função de predição

- Encapsula predição em função reutilizável.
- modelo.predict() sempre retorna um array NumPy, mesmo se você passar apenas um valor de orçamento.

```
def prever_receita(orcamento):
    return modelo.predict(np.array([[orcamento]]))[0]
```

Criando função de predição

- Para usar a função **prever_receita**, você simplesmente chama ela passando um valor de orçamento, e ela retorna a receita estimada pelo modelo. Por exemplo:

```
# Suponha que você quer prever a receita de um filme com orçamento de 100 milhões
orcamento_novo_filme = 100_000_000  # 100 milhões
receita_prevista = prever_receita(orcamento_novo_filme)

print(f"Para um orçamento de {orcamento_novo_filme:,}, a receita prevista é {receita_prevista:,.0f}")
```

Comparando previsões e reais

- Mostra lado a lado a receita real e a prevista pelo modelo.

```
import pandas as pd
comparacao = pd.DataFrame({
    "Orcamento": X_test.flatten(),
    "Receita Real": y_test,
    "Receita Prevista": y_pred
})
print(comparacao)
```

Comparando previsões e reais

- Mostra lado a lado a receita real e a prevista pelo modelo.
- O pandas foi usado para organizar os dados de forma tabular, facilitando a comparação entre valores reais e previstos.

Orçamento	Receita Real	Receita Prevista
200000000	2200000000	2.404.314.225
165000000	677000000	2.000.465.270

Encerramento

- Você aprendeu:
  - Preparar dados de filmes para regressão linear
  - Criar modelo com **LinearRegression**
  - Separar dados em treino e teste
  - Treinar e prever valores com **fit()** e **predict()**
  - Avaliar o modelo com R^2 e resíduos
  - Visualizar resultados e interpretar padrões de sucesso
  - Aplicar **TDD** para validar coeficientes e predições

PUC Minas