Lab6

June 20, 2024

1 Data Science Academy

1.1 Matemática e Estatística Aplicada Para Data Science, Machine Learning e IA

1.2 Lab 6

1.2.1 Usando Cálculo e Limite de Funções em Data Science com Linguagem Python

1.3 Instalando e Carregando os Pacotes

[ ]:
```
# Para atualizar um pacote, execute o comando abaixo no terminal ou prompt de␣
 ↪comando:
# pip install -U nome_pacote

# Para instalar a versão exata de um pacote, execute o comando abaixo no␣
 ↪terminal ou prompt de comando:
# pip install nome_pacote==versão_desejada

# Depois de instalar ou atualizar o pacote, reinicie o jupyter notebook.

# Instala o pacote watermark.
# Esse pacote é usado para gravar as versões de outros pacotes usados neste␣
 ↪jupyter notebook.
#pip install -q -U watermark
```

[ ]:
```
pip install -q -U watermark
```

```
Note: you may need to restart the kernel to use updated packages.
```

[ ]:
```
# Imports
import torch
import numpy as np
import sympy
from sympy import symbols, diff
```

[ ]:
```
# Versões dos pacotes usados neste jupyter notebook
%reload_ext watermark
%watermark -a "Data Science Academy"
```

Author: Data Science Academy

1.4 Calculando a Derivada com Linguagem Python

A derivada mede a sensibilidade à mudança da função (valor de saída) em relação a uma mudança na sua entrada.

Em termos mais simples, a derivada de uma função em um ponto específico é a taxa na qual a função está mudando naquele ponto. Isso é frequentemente entendido como a inclinação da linha tangente à função naquele ponto.

Por exemplo, se temos uma função que descreve a posição de um carro em movimento ao longo do tempo, a derivada dessa função em um ponto específico nos dá a velocidade do carro naquele momento.

A derivada de uma função f(x) é normalmente escrita como f'(x) ou df/dx. O processo de encontrar a derivada é chamado de diferenciação.

Se tivermos uma função f(x) = x^n, onde n é um número real, então a derivada desta função é dada por f'(x) = n * x^(n-1).

Há regras especiais (como a regra do produto, regra do quociente e a regra da cadeia) para lidar com funções mais complexas.

Para calcular a derivada de uma função em Python, você pode usar o módulo sympy, que é uma biblioteca Python para matemática simbólica.

Vamos calcular a derivada da função f(x) = x^3 + 2x^2 - x + 1 no ponto x = 1.

Primeiro, você precisará importar o módulo sympy e definir as variáveis simbólicas:

[ ]:
```
# Define a variável simbólica x
x = symbols('x')
```

[ ]:
```
# Define a função
f = x**3 + 2*x**2 - x + 1
```

[ ]:
```
# Calcula a derivada de f
derivada_f = diff(f, x)
```

[ ]:
```
print(f"Derivada da função: {derivada_f}")
```

```
Derivada da função: 3*x**2 + 4*x - 1
```

Isso vai imprimir a derivada da função, que é 3x^2 + 4x - 1.

Agora, para avaliar a derivada no ponto x = 1, você pode substituir x por 1 na derivada:

[ ]:
```
# Avalia a derivada no ponto x = 1
valor = derivada_f.subs(x, 1)
```

[ ]:
```
print(f"Valor da derivada no ponto x=1: {valor}")
```

```
Valor da derivada no ponto x=1: 6
```

Isso significa que a derivada da função no ponto x = 1 é 6.

[ ]:
```
# Define a variável simbólica x
x = symbols('x')
```

[ ]:
```
# Define a função
f = x**2
```

[ ]:
```
# Calcula a derivada de f
derivada_f = diff(f, x)
```

[ ]:
```
print(f"Derivada da função: {derivada_f}")
```

```
Derivada da função: 2*x
```

[ ]:
```
# Avalia a derivada no ponto x = 5
valor = derivada_f.subs(x, 5)
```

[ ]:
```
print(f"Valor da derivada no ponto x=5: {valor}")
```

```
Valor da derivada no ponto x=5: 10
```

1.5 Representação Geométrica da Derivada em Python

- Passo 1: Escolher uma Função

Primeiro, escolheremos uma função para a qual queremos calcular a derivada. Por exemplo, vamos usar a função quadrática f(x)=x^2.

- Passo 2: Plotar a Função

Depois criamos valores randômicos e plotamos a função.

- Passo 3: Escolher um Ponto e Calcular a Inclinação da Secante

Escolheremos um ponto na função e calcularemos a inclinação da linha secante, que passa por esse ponto e outro ponto muito próximo a ele. Essa inclinação é uma aproximação da derivada.

- Passo 4: Diminuir a Distância entre os Pontos

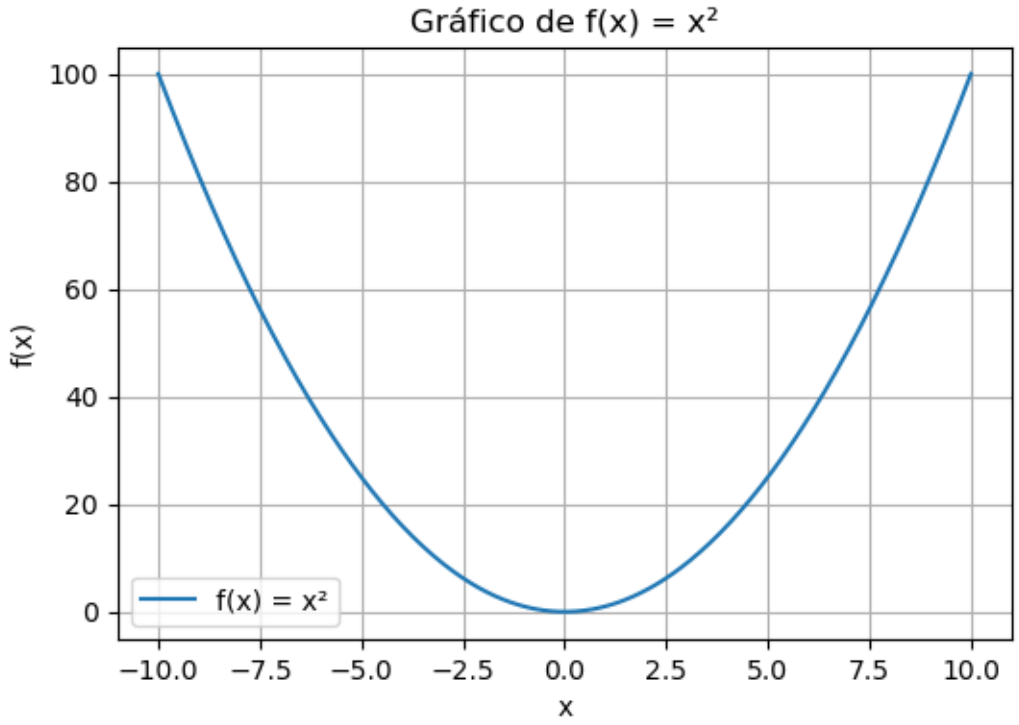
Diminuiremos gradualmente a distância entre os dois pontos e observaremos como a inclinação da linha secante se aproxima da inclinação da linha tangente.

- Passo 5: Calcular a Derivada

Finalmente, mostraremos como a inclinação da linha secante converge para a inclinação da linha tangente, que é a derivada da função no ponto escolhido.

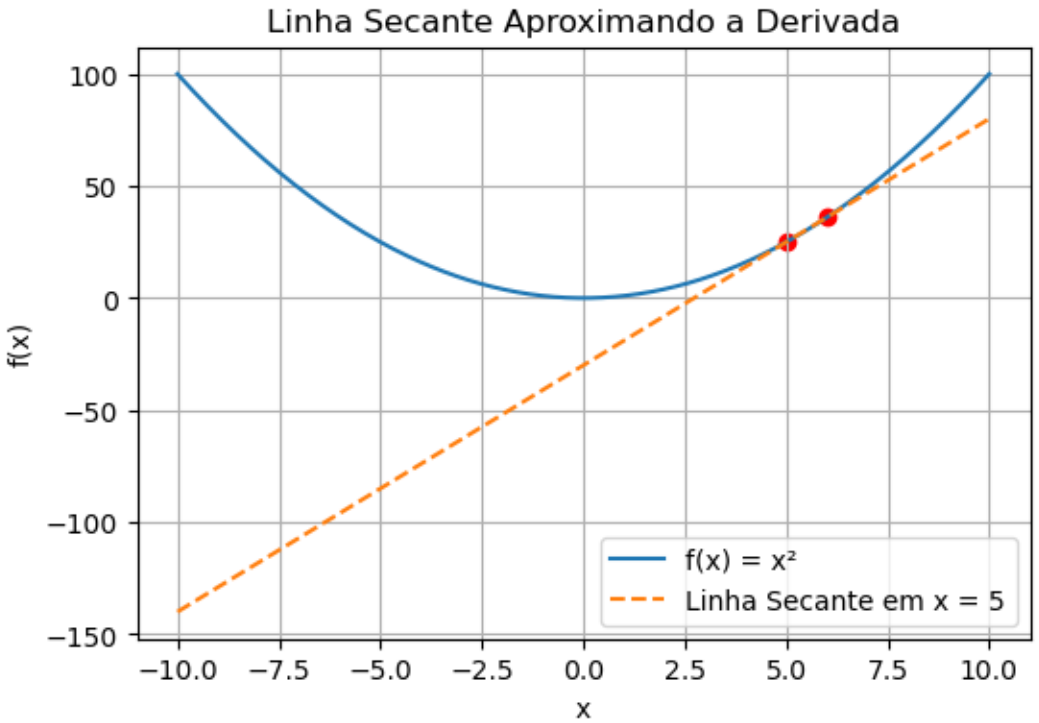
[ ]:
```
# Passos 1 e 2

import numpy as np
import matplotlib.pyplot as plt
```

```python
# Passo 1: Definindo a função
def f(x):
    return x**2

# Passo 2: Plotando a função
x = np.linspace(-10, 10, 400)
y = f(x)

plt.figure(figsize=(6, 4))
plt.plot(x, y, label="f(x) = x²")
plt.title("Gráfico de f(x) = x²")
plt.xlabel("x")
plt.ylabel("f(x)")
plt.legend()
plt.grid(True)
plt.show()
```



Vamos escolher um ponto na curva, por exemplo, x=5, e calcular a inclinação da linha secante entre x=5 e outro ponto próximo, digamos x=6. A inclinação da linha secante é dada pela diferença das funções dividida pela diferença dos x's.

[ ]:
```
# Passo 3: Escolher um ponto e calcular a inclinação da secante

# Ponto escolhido e ponto próximo
x1, x2 = 5, 6

# Calculando a inclinação da secante
secant_slope = (f(x2) - f(x1)) / (x2 - x1)

# Criando a linha secante
secant_line = secant_slope * (x - x1) + f(x1)

# Plotando a função e a linha secante
plt.figure(figsize=(6, 4))
plt.plot(x, y, label="f(x) = x²")
plt.plot(x, secant_line, label="Linha Secante em x = 5", linestyle='--')
plt.scatter([x1, x2], [f(x1), f(x2)], color='red') # Pontos na curva
plt.title("Linha Secante Aproximando a Derivada")
plt.xlabel("x")
plt.ylabel("f(x)")
plt.legend()
plt.grid(True)
plt.show()

secant_slope
```

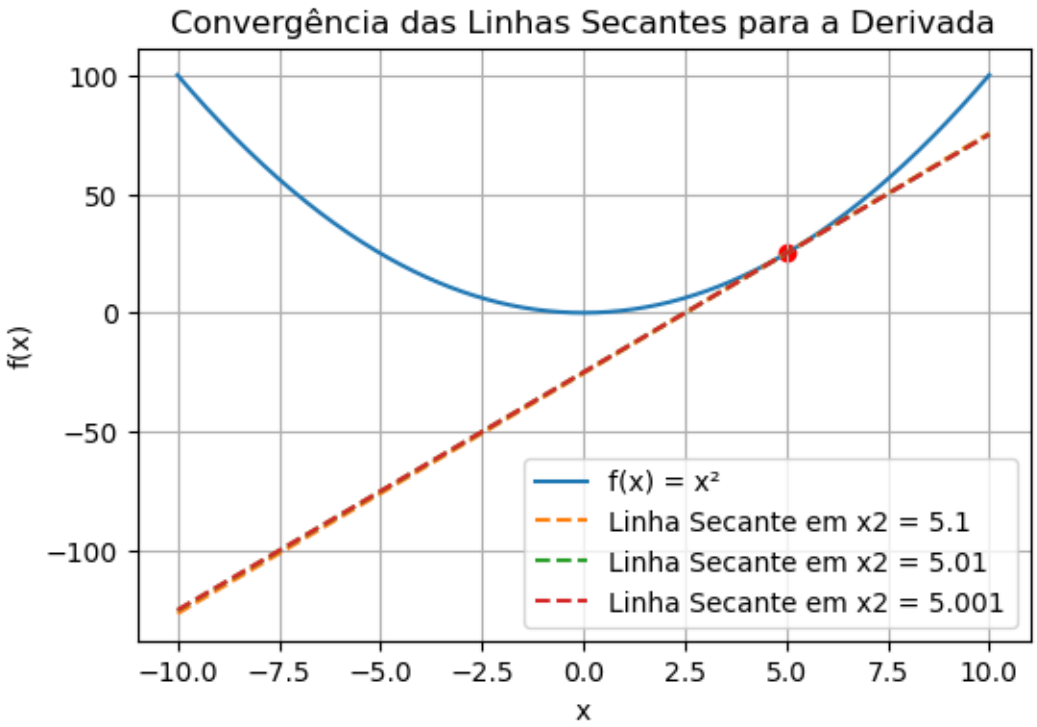


[ ]: 11.0

Agora, vamos diminuir a distância entre os dois pontos para ver como a inclinação da linha secante se aproxima da inclinação da linha tangente (derivada) em x=5. Vamos fazer isso para x = 5.1, 5.01, 5.001 e observar as mudanças.

Para cada novo valor de x, recalcularemos a inclinação da secante e plotaremos as linhas secantes para visualizar a convergência para a linha tangente.

[ ]:
```
# Passos 4 e 5: Diminuindo a distância entre os pontos e visualizando a␣
 ↪convergência

# Novos pontos próximos
x2_values = [5.1, 5.01, 5.001]

# Plotando a função
plt.figure(figsize=(6, 4))
plt.plot(x, y, label="f(x) = x²")

# Calculando e plotando as novas linhas secantes
for x2 in x2_values:
    secant_slope = (f(x2) - f(x1)) / (x2 - x1)
    secant_line = secant_slope * (x - x1) + f(x1)
    plt.plot(x, secant_line, label=f"Linha Secante em x2 = {x2}",␣
 ↪linestyle='--')

# Adicionando o ponto de tangência
plt.scatter([x1], [f(x1)], color='red') # Ponto na curva
plt.title("Convergência das Linhas Secantes para a Derivada")
plt.xlabel("x")
plt.ylabel("f(x)")
plt.legend()
plt.grid(True)
plt.show()
```



O gráfico está mostrando como as linhas secantes se aproximam da linha tangente à medida que a distância entre os pontos diminui. Como você pode ver, à medida que x se aproxima de 5, a inclinação da secante (a linha pontilhada) se aproxima da inclinação da linha tangente no ponto x=5.

Este processo ilustra como a derivada, que é a inclinação da linha tangente, pode ser aproximada por linhas secantes. À medida que a distância entre os pontos na linha secante se torna infinitesimalmente pequena, a inclinação da secante converge para a inclinação da tangente, que é a derivada da função no ponto dado.

1.6 Função Composta - Regra da Cadeia (Chain Rule)

A Regra da Cadeia (ou Chain Rule, em inglês) é uma fórmula para calcular a derivada de uma composição de funções. Em outras palavras, é usada quando temos uma "função dentro de uma função", também conhecida como função composta.

Vamos considerar duas funções, f(x) e g(x). Se temos uma função h(x) que é a composição dessas duas funções, isto é, h(x) = f(g(x)), então a Regra da Cadeia diz que a derivada de h(x) é a derivada de f em relação a g, multiplicada pela derivada de g em relação a x.

Matematicamente, isso é expresso da seguinte maneira:

h'(x) = f'(g(x)) * g'(x)

Essa fórmula nos diz que, para derivar a função composta h(x) = f(g(x)), primeiro derivamos a função externa f em relação à função interna g, e então multiplicamos pelo resultado da derivação

da função interna g em relação a x.

Vamos ilustrar isso com um exemplo:

Suponha que temos h(x) = (3x + 1)^2. Esta é uma composição de f(u) = u^2 e g(x) = 3x + 1. Se quisermos encontrar h'(x), primeiro derivamos f(u) em relação a u para obter 2u, e então substituímos u por g(x) para obter 2*(3x + 1). Depois derivamos g(x) em relação a x para obter 3.

Finalmente, multiplicamos esses dois resultados para obter h'(x) = 2 * (3x + 1) * 3 = 6 * (3x + 1) = 18x + 6.

A Regra da Cadeia é uma ferramenta fundamental no cálculo diferencial e é frequentemente usada quando se lida com funções compostas.

https://www.deeplearningbook.com.br/algoritmo-backpropagation-parte1-grafos-computacionais-e-chain-rule/

Podemos usar a biblioteca sympy para calcular a derivada de uma função composta. Vamos considerar a função h(x) = (3x + 1)^2 como mencionado na explicação acima.

Aqui está o código Python:

[ ]:
```
# Definir a variável simbólica x
x = symbols('x')
```

[ ]:
```
# Definir a função composta h(x)
h = (3*x + 1)**2
```

[ ]:
```
# Calcular a derivada de h(x) usando a regra da cadeia
h_prime = diff(h, x)
```

[ ]:
```
print(f"Derivada de (3x + 1)^2: {h_prime}")
```

```
Derivada de (3x + 1)^2: 18*x + 6
```

Esta é a derivada de h(x) = (3x + 1)^2.

1.7 Regra da Cadeia em Redes Neurais Artificiais

O uso mais comum da regra da cadeia em redes neurais artificiais está na implementação do algoritmo de retropropagação (backpropagation), que é usado para treinar redes neurais.

A retropropagação é um algoritmo que calcula o gradiente da função de perda (loss function) com respeito aos pesos da rede. Esse gradiente é então usado para ajustar os pesos na direção que minimiza a perda. A regra da cadeia é usada para calcular esse gradiente.

O gradiente de uma função é um vetor que contém as derivadas parciais da função em relação a cada uma de suas variáveis. Ele fornece a direção do maior aumento da função e a magnitude desse aumento é dada pelo valor do gradiente naquele ponto.

A regra da cadeia é um teorema no cálculo que permite a diferenciação de funções compostas. No contexto de funções de múltiplas variáveis, a regra da cadeia permite calcular a derivada de uma função composta considerando as derivadas das funções componentes.

Quando se calcula o gradiente de uma função composta usando a regra da cadeia, o que se obtém é uma expressão para a taxa de variação da função composta em relação a cada uma de suas variáveis. Em outras palavras, o gradiente resultante nos dá a direção e magnitude do maior aumento da função composta no espaço de várias dimensões.

Essa informação é extremamente útil em uma série de aplicações, incluindo otimização de funções, onde se deseja encontrar o ponto mínimo ou máximo de uma função, bem como em métodos numéricos e aplicações de Machine Learning, como no treinamento de redes neurais com o método do gradiente descendente.

Vamos implementar uma rede neural simples com apenas um neurônio (também chamado de perceptron) para demonstrar isso. Usaremos a biblioteca PyTorch, que lida automaticamente com a regra da cadeia durante a retropropagação.

Considere Fórmula Matemática: y = x * w

Para criar nosso exemplo, vamos definir os conceitos abaixo:

Cálculo do Gradiente: Em redes neurais, o processo de aprendizado envolve otimizar os parâmetros (ou pesos) da rede para minimizar a função de perda (ou erro). Isso é feito usando técnicas de otimização como o gradiente descendente. Para aplicar essas técnicas, é necessário calcular o gradiente da função de perda em relação a cada parâmetro. O gradiente é essencialmente a taxa de mudança da função de perda com respeito a esses parâmetros, ou seja, a derivada.

Backpropagation: O cálculo do gradiente é realizado através de um processo chamado backpropagation. Para isso, as bibliotecas de aprendizado de máquina mantêm um grafo de computação que registra todas as operações realizadas nos tensores que têm requires_grad definido como True. Quando a função de perda é calculada, o gradiente dessa perda é propagado de volta através do grafo e os gradientes em relação a cada tensor são acumulados.

requires_grad=True: Ao definir requires_grad=True para um tensor no PyTorch, você está informando à biblioteca que deseja que ela calcule os gradientes desse tensor durante a passagem para trás (backpropagation). Normalmente, isso é feito para os parâmetros da rede que você deseja otimizar. Por exemplo, pesos e vieses em uma rede neural teriam requires_grad=True.

Otimização e Atualização de Parâmetros: Durante o treinamento, esses gradientes são usados por otimizadores (como SGD, Adam, etc.) para atualizar os parâmetros da rede na direção que minimiza a função de perda.

[ ]:
```
# Inicializa o tensor de entrada x (dado de entrada)
x = torch.tensor([10.0], requires_grad = True)
x
```

[ ]:
```
tensor([10.], requires_grad=True)
```

[ ]:
```
# Inicializa o tensor de peso w (coeficiente do modelo, exatamente o que modelo␣
↪vai aprender no treinamento)
w = torch.tensor([0.03], requires_grad = True)
w
```

[ ]:
```
tensor([0.0300], requires_grad=True)
```

[ ]:
```
# Inicializa o tensor de saída y (aquilo que queremos prever)
y = torch.tensor([1.0])
y
```

[ ]:
```
tensor([1.])
```

[ ]:
```
# Define a função de ativação como a função identidade (f(x) = x)
funcao_ativacao = torch.nn.Identity()
```

Uma das principais razões para usar funções de ativação é introduzir não linearidade no modelo. Redes neurais são projetadas para aproximar funções complexas e a maioria dos problemas do mundo real que queremos modelar são não lineares por natureza. Sem funções de ativação não lineares, uma rede neural, independentemente de sua profundidade (número de camadas), seria equivalente a um modelo linear e, portanto, incapaz de modelar a complexidade encontrada em tarefas reais como reconhecimento de imagem, processamento de linguagem natural, etc.

[ ]:
```
# Calcula a saída da rede neural
y_previsto = funcao_ativacao(w * x)
```

[ ]:
```
y_previsto
```

[ ]:
```
tensor([0.3000], grad_fn=<MulBackward0>)
```

[ ]:
```
# Define a função de perda como o quadrado da diferença entre
# a saída do modelo (valor previsto) e o alvo (valor real)
funcao_de_erro = torch.nn.MSELoss()
```

[ ]:
```
# Calcula a perda comparando a saída ao alvo
erro = funcao_de_erro(y_previsto, y)
```

[ ]:
```
# Usa retropropagação para calcular os gradientes
erro.backward()
```

[ ]:
```
# Print do gradiente de w
print(f"Gradiente de w: {w.grad}")
```

```
Gradiente de w: tensor([-14.0000])
```

O script Python acima define um neurônio com uma entrada (x) e um peso (w), e usa a regra da cadeia para calcular o gradiente da função de perda com respeito ao peso. O valor de gradiente resultante pode ser usado para atualizar o peso e treinar a rede neural.

1.8 Gradiente Descendente via Operações Matemáticas com Linguagem Python

Uma aplicação comum de derivadas em Data Science é na implementação do algoritmo de gradiente descendente, usado para otimizar funções de custo em modelos de aprendizado de máquina, como regressão linear ou redes neurais.

Aqui está um exemplo de como isso pode ser feito em Python para a tarefa de regressão linear. Para simplificar, vamos considerar uma regressão linear simples com apenas uma variável de entrada.

No exemplo abaixo, o código cria um conjunto de dados aleatório, inicializa os parâmetros da regressão linear de forma aleatória (a variável theta), e então executa o algoritmo de gradiente descendente por um número fixo de iterações.

A cada iteração, o algoritmo calcula o gradiente da função de custo em relação aos parâmetros (gradientes) e, em seguida, atualiza os parâmetros na direção oposta ao gradiente (isso é o que a linha theta = theta - lr * gradientes faz). O tamanho do passo é determinado pela taxa de aprendizado (lr).

Ao final do processo, os parâmetros da regressão linear (ou seja, a inclinação e o intercepto) são armazenados na variável theta. Esses parâmetros minimizam a função de custo e, portanto, representam a melhor linha de ajuste aos dados.

Considere que X é o diâmetro de uma Pizza que um cliente pediu e y a gorjeta dada por um cliente. Conseguimos prever a gorjeta com base no diâmetro da Pizza?

Vamos definir uma relação linear entre X e y:

y = coef1 + (coef2 * X)

[ ]:
```
# Criamos um conjunto de dados sintéticos
X = 2 * np.random.rand(100,1)
y = 4 + 3 * X + np.random.randn(100,1)
```

[ ]:
```
print(X[1:11])
```

```
[[1.39245107]
 [0.41368498]
 [1.27118281]
 [0.43803037]
 [0.34911067]
 [1.85280768]
 [1.94604386]
 [0.54261294]
 [1.51216813]
 [1.34307475]]
```

[ ]:
```
print(y[1:11])
```

```
[[ 8.15693178]
 [ 5.09829621]
 [ 6.51485604]
 [ 6.85547044]
 [ 3.24465427]
 [10.63626207]
 [11.21365138]
 [ 5.81530998]
 [11.25561531]
 [ 6.88774434]]
```

[ ]:
```
# Taxa de aprendizado
lr = 0.1
```

[ ]:
```
# Inicialização dos hiperparâmetros
n_iterations = 1000
m = 100
```

[ ]:
```
# Inicialização dos parâmetros (coeficientes)
theta = np.random.randn(2,1)
```

[ ]:
```
theta
```

[ ]:
```
array([[ 0.91264417],
       [-0.53053333]])
```

[ ]:
```
# Adiciona coluna de 1s a X
X_b = np.c_[np.ones((m, 1)), X]
```

[ ]:
```
X_b[1:11]
```

[ ]:
```
array([[1.        , 1.39245107],
       [1.        , 0.41368498],
       [1.        , 1.27118281],
       [1.        , 0.43803037],
       [1.        , 0.34911067],
       [1.        , 1.85280768],
       [1.        , 1.94604386],
       [1.        , 0.54261294],
       [1.        , 1.51216813],
       [1.        , 1.34307475]])
```

[ ]:
```
# Loop de treino
for iteration in range(n_iterations):

    # Calcula o gradiente (derivada parcial)
    gradientes = 2 / m * X_b.T.dot(X_b.dot(theta) - y)

    # Usa os gradientes para atualizar os coeficientes
    theta = theta - lr * gradientes
```

[ ]:
```
print(gradientes)
```

```
[[ 5.76161341e-14]
 [-4.87949601e-14]]
```

[ ]:
```
print(theta)
```

```
[[3.56948244]
 [3.34982803]]
```

[ ]:
```
theta[0]
```

[ ]:
```
array([3.56948244])
```

[ ]:
```
theta[1]
```

[ ]:
```
array([3.34982803])
```

$$y = coef1 + (coef2 * X)$$

[ ]:
```
# Qual a previsão de gorjeta se o cliente pedir uma Pizza de 14 centímetros de diâmetro?
y = theta[0] + (theta[1] * 14)
```

[ ]:
```
print(y)
```

```
[50.4670748]
```

[ ]:
```
%reload_ext watermark
%watermark -a "Data Science Academy"
```

```
Author: Data Science Academy
```

[ ]:
```
#%watermark -v -m
```

[ ]:
```
#%watermark --iversions
```

1.9 Fim